

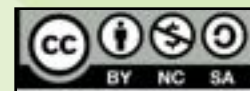


# **CURSO DE INTELIGENCIA ARTIFICIAL MÁLAGA, 30 mayo a 20 julio 2021**

## **ARQUITECTURAS DE DATOS**

**07/06/21 y 08/06/21 - 16:00 – 20:00**

ÍÑIGO SANZ EGURROLA



# Íñigo Sanz Egurrola

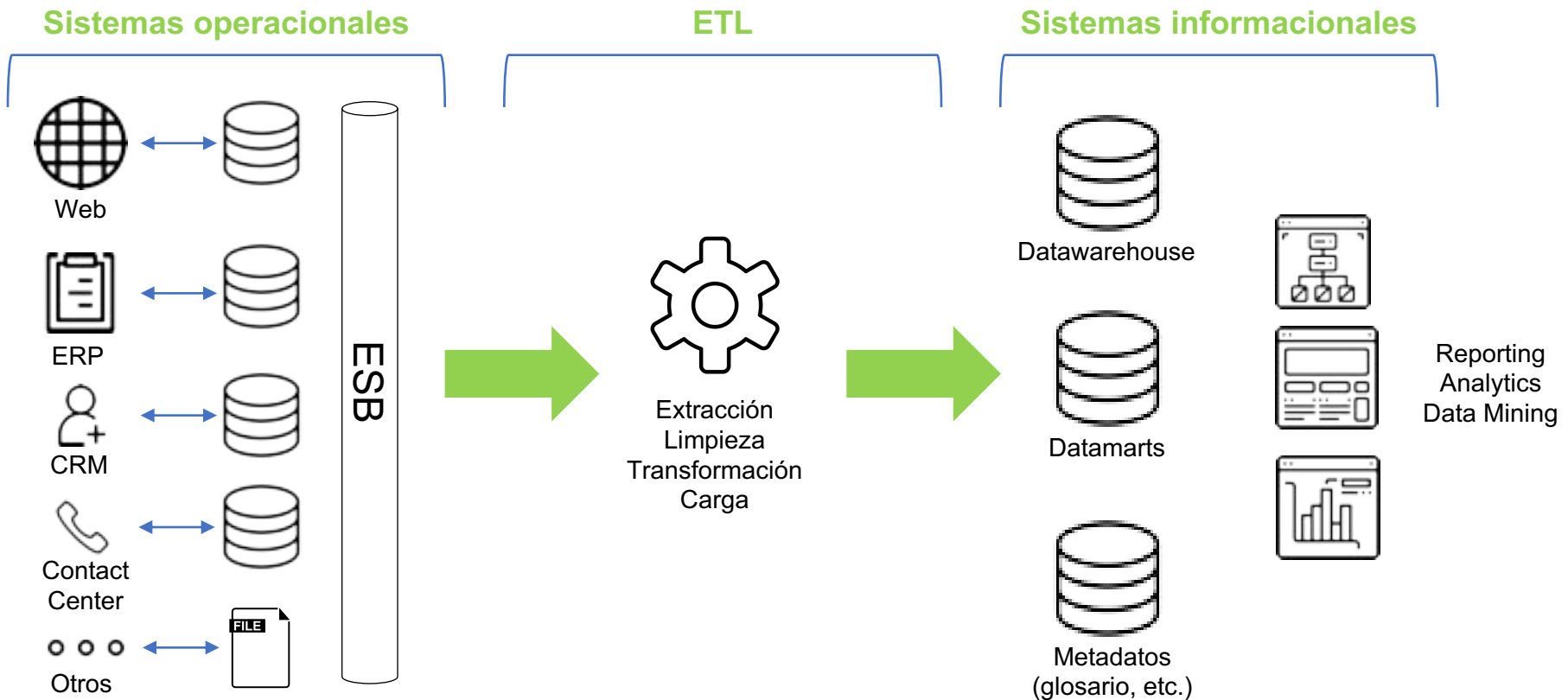
- ✓ Ingeniero en Informática y Máster en Administración de Empresas.
  - ✓ 20 años trabajando en el sector IT como programador, arquitecto o gestor de proyectos o áreas.
  - ✓ Trabajando desde 2013 en tecnologías Big Data.
- Actualmente Director del Programa de Big Data en Indizen Technologies.
- ✓ Contacto:  
[i.sanz@indizen.com](mailto:i.sanz@indizen.com)  
600430897

# Índice

- \_01 Introducción a las plataformas de datos
- \_02 Práctica: SQL y bases de datos relacionales.
- \_03 Ecosistema de tecnologías Big Data
- \_04 Introducción a Hadoop. Distribuciones.
- \_05 Hadoop core: HDFS y YARN.
- \_06 Práctica: manipulando datos en HDFS
- \_07 Ecosistema Hadoop: Hive, Sqoop, Flume, Hue, etc.
- \_08 Práctica: analizando datos en Hadoop.
- \_09 Bases de datos NoSQL: MongoDB, Cassandra, Redis, Neo4j, etc.
- \_10 Práctica: trabajando con MongoDB.
- \_11 Apache Spark.
- \_12 Plataformas cloud: introducción a cloud computing.
- \_13 Plataformas cloud: soluciones Big Data en Cloud.

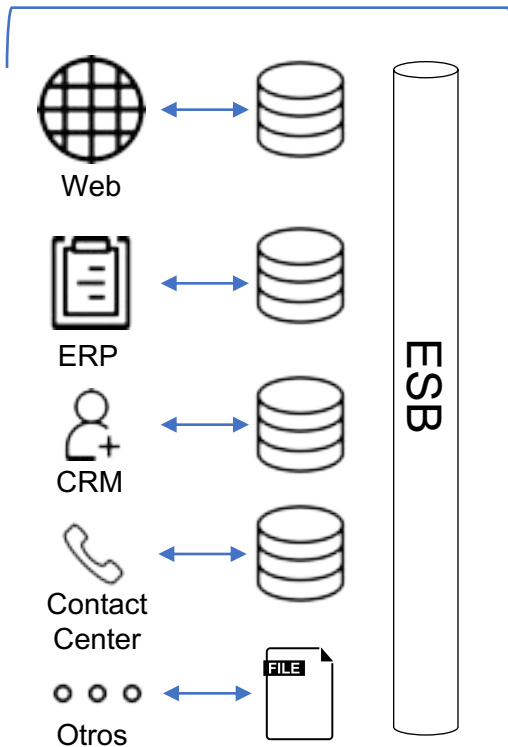
# Introducción a las plataformas de datos

# Arquitecturas de datos tradicionales



# Arquitecturas de datos tradicionales

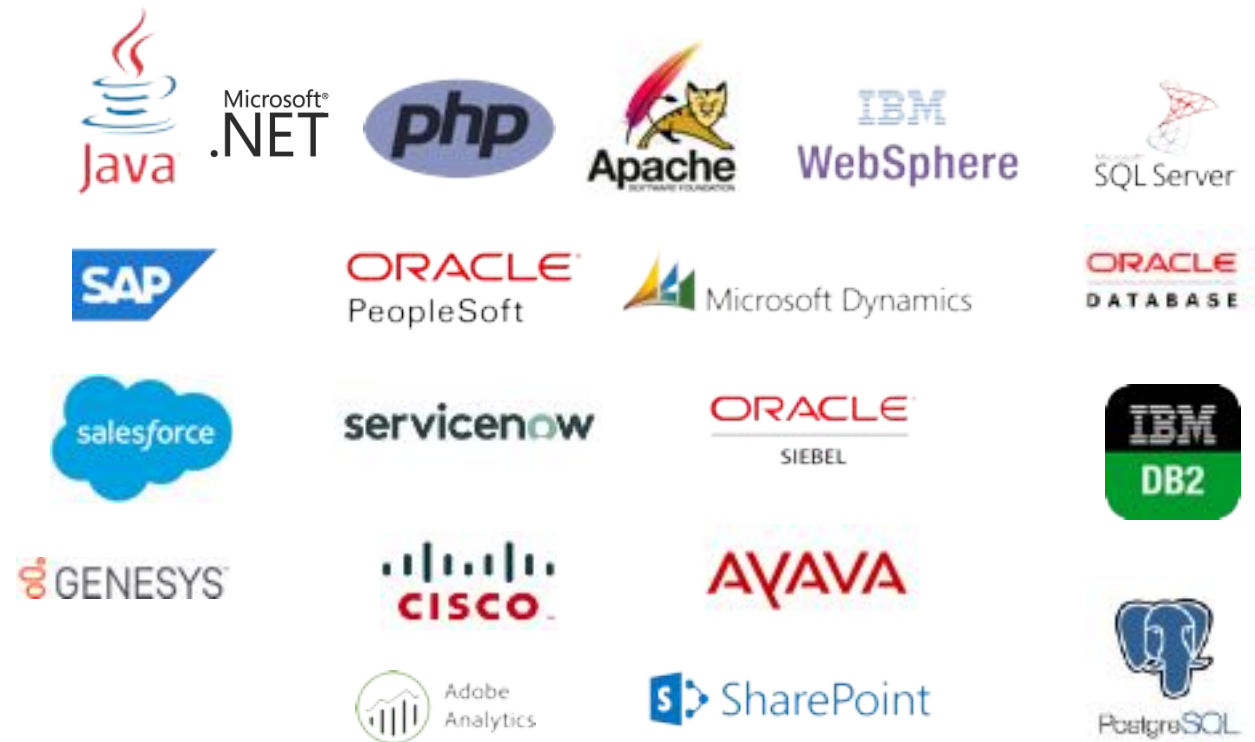
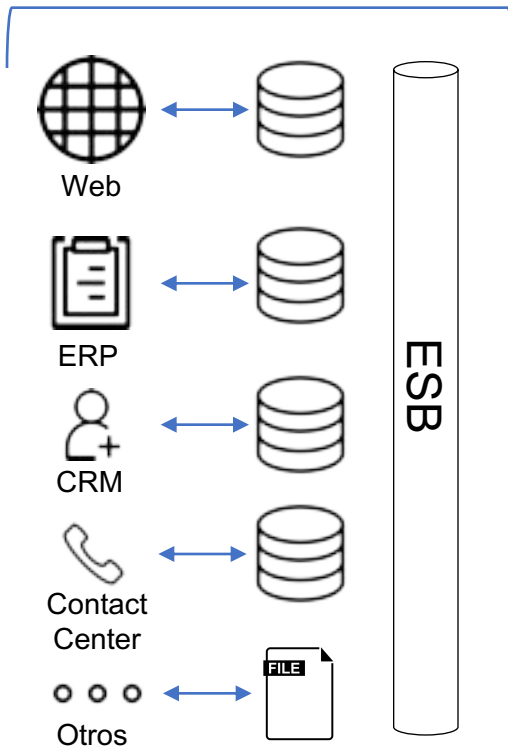
## Sistemas operacionales



- Dan soporte a los **canales** y la **operativa** de la organización.
- Consumidos por clientes, la mayoría de canales requieren **disponibilidad** 24x7.
- Las operaciones suelen ser sobre datos **atómicos** o de entidades aisladas.
- Se apoyan en bases de datos **separadas** donde prima la velocidad de lectura y escritura atómica así como la garantía **transaccional**.
- Habitualmente utilizan **diversas tecnologías** y herramientas, no hay mucha estandarización.

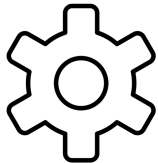
# Arquitecturas de datos tradicionales

## Sistemas operacionales



# Arquitecturas de datos tradicionales

ETL



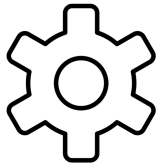
Extracción  
Limpieza  
Transformación  
Carga

- **Recogen** la información de sistemas origen (OLTP o datos externos).
- Realizan operaciones de **limpieza**, **estandarización** o **normalización** de datos.
- **Transforman** los datos realizando combinaciones, cálculos agregados, etc. resultando un formato orientado a la explotación analítica.
- **Cargan** los datos resultantes en los sistemas destino.
- Habitualmente se apoyan en herramientas con un interfaz de diseño amigable y motores de ejecución dedicados.



# Arquitecturas de datos tradicionales

ETL



Extracción  
Limpieza  
Transformación  
Carga



Informatica

ORACLE®

DATA INTEGRATOR

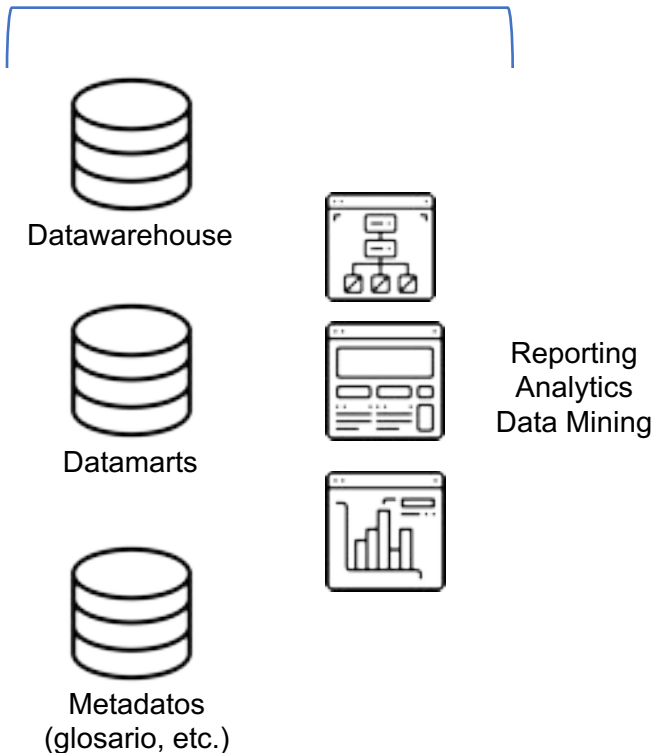
IBM® DataStage



talend

# Arquitecturas de datos tradicionales

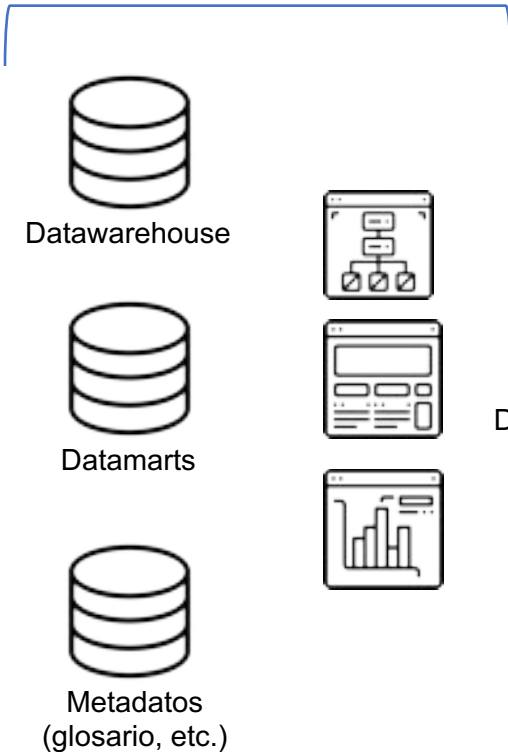
## Sistemas informacionales



- Ofrecen aplicaciones de **ayuda a la toma de decisiones**: informes, cuadros de mando, analítica interactiva o minería de datos.
- Requieren que los **datos estén preparados** en formatos amigables para la explotación analítica.
- Consumidos por analistas o gestores, no suelen necesitar una **disponibilidad** de 24x7.
- Existen más estándares para interoperabilidad, al apoyarse en un conjunto de tecnologías más estándar (principalmente SQL).

# Arquitecturas de datos tradicionales

## Sistemas informacionales



TERADATA

ORACLE

EXADATA



IBM  
COGNOS



Power BI



IBM  
SPSS

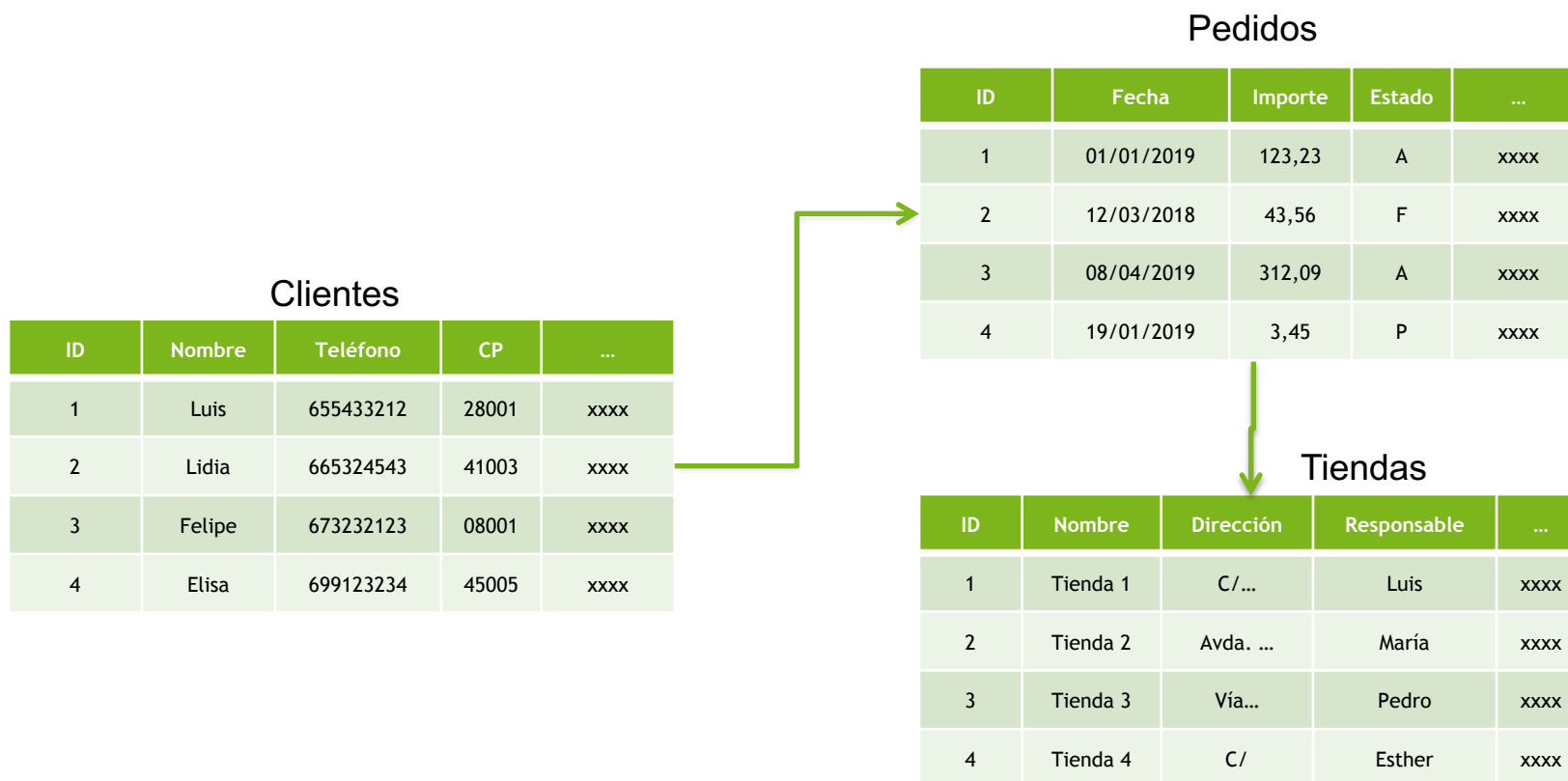
# Arquitecturas de datos tradicionales

La mayor parte de las tecnologías “tradicionales” se apoyan en **bases de datos relacionales** como repositorios de datos donde almacenar y recuperar para cubrir los casos de uso de negocio.

Las bases de datos relacionales tienen las siguientes características fundamentales:

- La mayoría usan un despliegue **centralizado**: la base de datos almacena y se ejecuta en un único servidor.
- Usan un **modelo de datos relacional** para almacenar datos, con esquemas, tablas, registros y campos: son **datos estructurados**.
- Permiten almacenar información tabular, en una tabla sólo se almacenan datos de un **mismo tipo**.
- Las principales soluciones son propietarias con un **coste de licencia alto**.

# Modelo relacional



# Principales elementos de un modelo relacional

- **Tablas:** almacenan un conjunto de elementos que tienen unos mismos atributos (datos).

Pedidos

ID	Fecha	Importe	Estado	ID cliente
1	01/01/2019	123,23	A	2
2	12/03/2018	43,56	F	3
3	08/04/2019	312,09	A	1
4	19/01/2019	3,45	P	4

Clientes

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx

# Principales elementos de un modelo relacional

- **Filas o registros:** cada uno de los elementos que almacena una tabla.

Pedidos

ID	Fecha	Importe	Estado	ID cliente
1	01/01/2019	123,23	A	2
2	12/03/2018	43,56	F	3
3	08/04/2019	312,09	A	1
4	19/01/2019	3,45	P	4

Clientes

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx

# Principales elementos de un modelo relacional

- **Columnas o campos:** cada uno de los atributos que contienen las filas de una tabla. Cada fila tiene un tipo (numérico, alfanumérico, fecha, etc.). Una fila puede tener una columna vacía, pero no puede haber dos filas con valores de distinto tipo para una misma columna.

Pedidos

ID	Fecha	Importe	Estado	ID cliente
1	01/01/2019	123,23	A	2
2	12/03/2018	43,56	F	3
3	08/04/2019	312,09	A	1
4	19/01/2019	3,45	P	4

Clientes

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx



# Principales elementos de un modelo relacional

- **Claves primarias:** debe haber uno campo, o la composición de varios campos que permitan identificar unívocamente a una fila. Este campo/s se llama clave primaria.

Pedidos

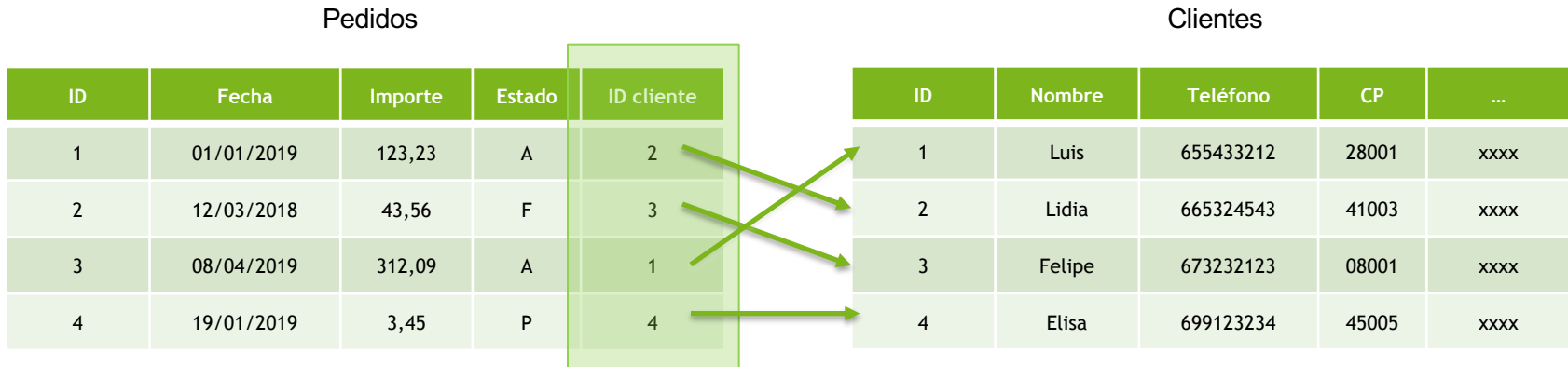
ID	Fecha	Importe	Estado	ID cliente
1	01/01/2019	123,23	A	2
2	12/03/2018	43,56	F	3
3	08/04/2019	312,09	A	1
4	19/01/2019	3,45	P	4

Clientes

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx

# Principales elementos de un modelo relacional

- **Claves foráneas:** una fila de una tabla puede estar relacionada con otra fila de la misma tabla o de otra distinta. Para indicar esa relación, se añade un campo a una tabla con el identificador de la fila asociada.



# Principales elementos de un modelo relacional

- **Esquema:** es una agrupación lógica de tablas.

Clientes

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx

Pedidos

ID	Fecha	Importe	Estado	...
1	01/01/2019	123,23	A	xxxx
2	12/03/2018	43,56	F	xxxx
3	08/04/2019	312,09	A	xxxx
4	19/01/2019	3,45	P	xxxx

Tiendas

ID	Nombre	Dirección	Responsable	...
1	Tienda 1	C/...	Luis	xxxx
2	Tienda 2	Avda. ...	María	xxxx
4	Tienda 4	C/	Esther	xxxx

# Lenguaje de acceso (SQL)

- Las bases de datos relacionales utilizan un lenguaje similar para poder crear o acceder a los datos, SQL.
- Ejemplos:

```
CREATE TABLE pedidos (  
  id INT PRIMARY KEY,  
  fecha VARCHAR(10) NOT NULL,  
  importe DECIMAL(6,2),  
  estado CHAR(1) NOT NULL,  
  id_cliente INT);
```

```
CREATE TABLE clientes (  
  id INT PRIMARY KEY,  
  nombre VARCHAR(25) NOT NULL,  
  telefono VARCHAR(9) NOT NULL,  
  cp CHAR(6) NOT NULL);
```

```
INSERT INTO clientes (id, nombre, telefono, cp) VALUES (1, "Juan  
Pérez", "669123456", "28001");  
INSERT INTO clientes (id, nombre, telefono, cp) VALUES (2, "Eva  
López", "669123456", "28001");  
  
INSERT INTO pedidos (id, fecha, importe, estado, id_cliente) VALUES  
(1, "01/01/2020", 122.14, 'E',1);  
INSERT INTO pedidos (id, fecha, importe, estado, id_cliente) VALUES  
(2, "02/01/2020", 321.98, 'E',1);  
INSERT INTO pedidos (id, fecha, importe, estado, id_cliente) VALUES  
(3, "02/01/2020", 49.95, 'E',2);  
  
UPDATE pedidos SET estado='S' WHERE id=1;  
DELETE FROM pedidos WHERE id=45;  
  
SELECT * FROM pedidos WHERE id=1;  
SELECT * FROM pedidos ORDER BY fecha;  
SELECT * FROM pedidos WHERE importe>100 AND id_cliente=1;  
SELECT fecha, count(*) FROM pedidos GROUP BY fecha;  
SELECT fecha, sum(importe) FROM pedidos GROUP BY fecha;
```

# Práctica: SQL y bases de datos relacionales.

# Prácticas en SQL

- Entrar en <https://sqliteonline.com/>
- Dada la siguiente tabla:

Nombre de la empresa	Facturación (M€)	Empleados	Beneficio (M€)
COMPañIA ESPAñOLA DE PETROLEOS SA	23.857	700	833
MERCADONA SA	23.343	80.000	622
NATURGY ENERGY GROUP SA	23.035	3.000	1.796
REPSOL PETROLEO SA	21.270	2.500	502
ENDESA SA	19.258	3.000	180
INDUSTRIA DE DISEñO TEXTIL SA	18.261	60.000	10.418

- Crear una tabla llamada “Empresas” y las columnas que aparecen en la tabla de empresas (nombre, facturación, empleados, beneficios).
- Introducir las 6 empresas.
- Hacer una consulta para obtener todos los registros insertados.

# Prácticas en SQL

```
CREATE TABLE empresas (  
    nombre VARCHAR(200),  
    facturación REAL,  
    empleados INTEGER,  
    beneficios DECIMAL(15,2)  
);
```

# Prácticas en SQL

```
INSERT INTO empresas (nombre,facturación,empleados,beneficios)
values ('COMPAÑIA ESPAÑOLA DE PETROLEOS SA', 23857, 700, 833);
```

```
INSERT INTO empresas (nombre,facturación,empleados,beneficios)
values ('MERCADONA SA', 23343, 80000, 622);
```

```
INSERT INTO empresas (nombre,facturación,empleados,beneficios)
values ('NATURGY ENERGY GROUP SA', 23035, 3000, 1796);
```

```
INSERT INTO empresas (nombre,facturación,empleados,beneficios)
values ('REPSOL PETROLEO SA', 21270, 2500, 502);
```

```
INSERT INTO empresas (nombre,facturación,empleados,beneficios)
values ('ENDESA SA', 19258, 3000, 180);
```

```
INSERT INTO empresas (nombre,facturación,empleados,beneficios)
values ('INDUSTRIA DE DISEÑO TEXTIL SA', 18261, 60000, 10418);
```



# Prácticas en SQL

```
SELECT * FROM empresas
```

```
SELECT * FROM empresas ORDER BY beneficios DESC
```

```
SELECT * FROM empresas WHERE empleados < 10000 ORDER BY  
beneficios DESC
```

```
SELECT * FROM empresas WHERE empleados < 10000 AND nombre  
like 'REP%' ORDER BY beneficios DESC
```

```
SELECT nombre, (beneficios/facturación)*100 AS margen FROM  
empresas ORDER BY margen DESC
```

# Ecosistema de tecnologías Big Data

# Beneficios del modelo relacional

El modelo relacional y las bases de datos relacionales tienen características muy beneficiosas:

- **Estandariza** un modelo de representación de los datos que permite la **integración** de sistemas y aplicaciones.
- Ofrece un **lenguaje** general y muy potente para obtención y manipulación de datos (SQL).
- Garantiza un nivel de **transaccionalidad** muy elevado (ACID).

Sin embargo, también tiene unas limitaciones...

# ¿Escalabilidad?

**Cientes**

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx

**Pedidos**

ID	Fecha	Importe	Estado	...
1	01/01/2019	123,23	A	xxxx
2	12/03/2018	43,56	F	xxxx
3	08/04/2019	312,09	A	xxxx
4	19/01/2019	3,45	P	xxxx

**Tiendas**

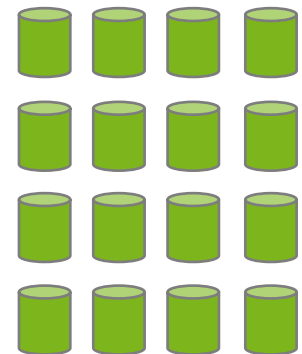
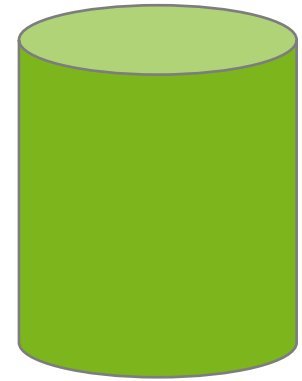
ID	Nombre	Dirección	Responsable	...
1	Tienda 1	C/ ...	Luis	xxxx
2	Tienda 2	Avda. ...	María	xxxx
3	Tienda 3	Vía...	Pedro	xxxx
4	Tienda 4	C/	Esther	xxxx

**¿Qué pasaría si cada tabla tuviera millones o miles de millones de filas?**

# Escalabilidad

Dos opciones para manejar un gran volumen de datos:

- **Escalado vertical:** máquinas de mayor capacidad.
- **Escalado horizontal:** posiblemente dividiendo los datos en bloques y gestionando un bloque en cada máquina (sharding).



# Problemática

## Problemática del modelo relacional tradicional:

- Tiene dificultades para **escalar**.
- El modelo habitualmente no coincide con el modelo de las aplicaciones (**impedance mismatch**).
- La garantía **ACID** tiene dificultades para ejecutarse **sobre entornos distribuidos**.
- El esquema debe ser definido antes de introducir los datos (**schema-on-write**), y los cambios de esquema suelen ser costosos.

Las arquitecturas de datos tradicionales tienen dos problemas principales:

- El **coste** se dispara cuando el volumen de datos es grande.
- La adaptación a los **cambios** o a fuentes de datos diversas es difícil

# Origen de las tecnologías Big Data

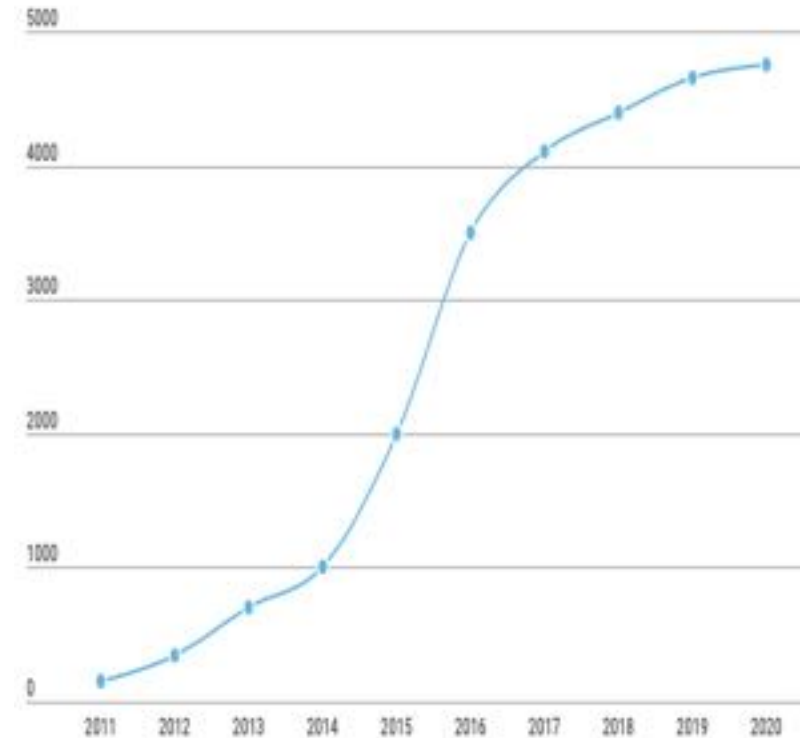
Las tecnologías tradicionales han sido suficientes para la mayor parte de los casos de uso, y de hecho suelen ser las más apropiadas para la mayor parte de los casos de uso actuales.

Sin embargo, a principios del siglo XXI, con la irrupción de internet, empiezan a emerger compañías que requieren manejar un volumen de datos muy grande y diverso, como es el caso de Google o Yahoo.

Para estas empresas, las tecnologías tradicionales no son válidas, principalmente por coste, porque su beneficio por usuario es muy bajo.

Estos fabricantes empiezan a desarrollar **tecnologías “Big Data”**, capaces de almacenar y procesar un gran volumen de datos a un coste razonable.

# Ecosistema de tecnologías Big Data





# Tipos de tecnologías Big Data

Tipos de tecnologías “Big Data”:

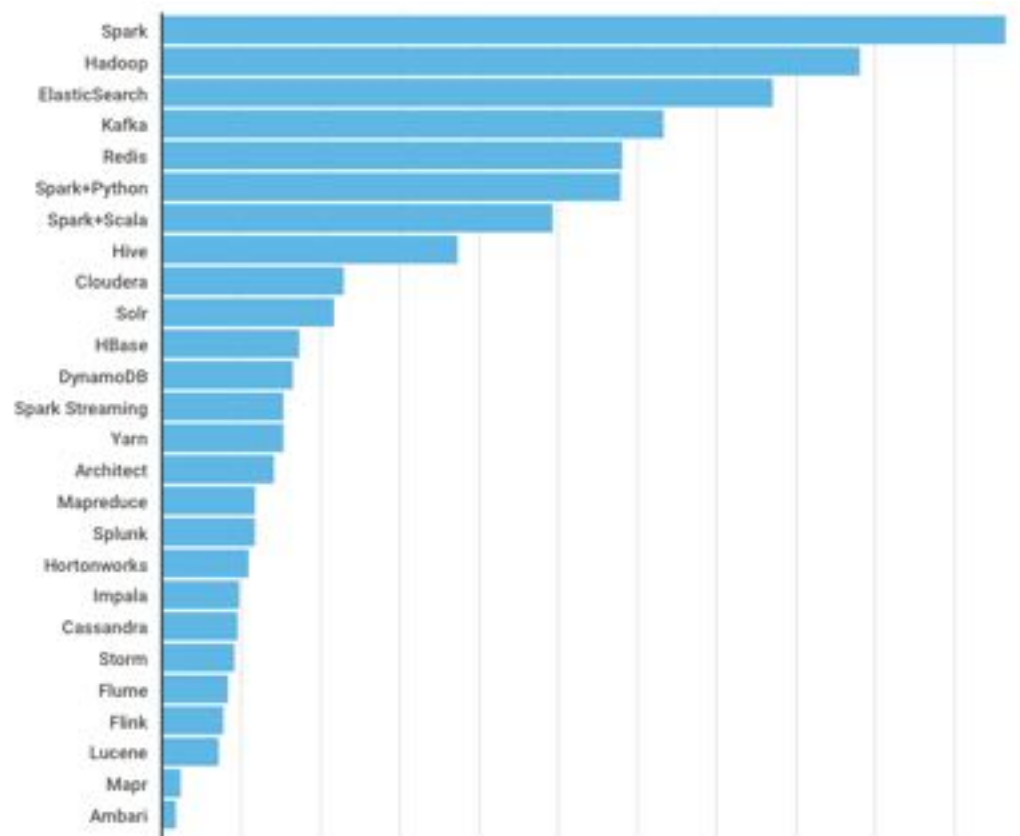
- **Plataformas:** Hadoop, Stratio, etc.
- **Bases de datos NoSQL:** MongoDB, Cassandra, DynamoDB, Hbase, etc.
- **Bases de datos NewSQL:** SAP Hana, Clustrix, NuoDB, etc.
- **MPPs escalables:** Teradata, Vertica, Exasol, etc.
- **Frameworks de procesamiento de datos:** Spark, Flink, etc.
- **Soluciones verticales:** Splunk, ELK, Solr, etc.
- **Transformación de datos:** Talend, Pentaho, Trifacta, etc.
- **Gobierno de datos:** Waterline, Informatica, etc.
- **Acceso a datos:** Impala, Hive, Drill, Presto, etc.
- **Soluciones cloud:** Hadoop-as-a-Service, servicios PaaS de datos, etc.
- ...

# Características habituales

La mayoría de tecnologías Big Data tienen las siguientes características:

- **Sistemas distribuidos:** se despliegan en más de un servidor, repartiendo el trabajo a realizar entre todos los servidores. El escalado tiene un coste lineal.
- **Software opensource:** la mayor parte del código es abierto, lo que no significa que sea “gratis totalmente”, aunque sí más barato.
- **Variedad de datos y flexibilidad:** permiten gestionar cualquier tipo de datos y adaptarse rápidamente a cambios.
- **Hardware commodity y cloud-ready:** no requieren unos servidores específicos, incluso pueden ejecutarse en servidores cloud.

# Uso real de tecnologías Big Data



# Introducción a Hadoop. Distribuciones.

# Apache Hadoop



Es una **plataforma opensource** que ofrece la capacidad de **almacenar** y **procesar**, a “bajo” **coste**, grandes **volúmenes** de datos, sin importar su **estructura**, en un entorno **distribuido**, **escalable** y **tolerante a fallos**, basado en la utilización de **hardware commodity** y en un paradigma acercamiento del **procesamiento a los datos**.

# Historia



**10/2003**

## **GFS**

Google publica un paper sobre su Sistema de almacenamiento escalable denominado Google File System.



**12/2004**

## **MapReduce**

Google publica un paper sobre su implementación del paradigma MapReduce para procesamiento masivo en paralelo



**01/2006**

## **Subproyecto Hadoop de Nutch**

Doug Cutting y Mike Cafarella fueron los creadores de Hadoop como subproyecto de Nutch (buscador).

Cutting se unió a Yahoo, que proporcionó recursos (€) para impulsar Hadoop.



**01/2008**

## **Apache Top-level**

Hadoop es elevado a proyecto top-level de Apache.

# Ecosistema



Los componentes core son **HDFS** y **YARN**:

- HDFS es un sistema de ficheros (capa de **almacenamiento**)
- YARN es un gestor de recursos (capa de **procesamiento**).

Sin embargo, normalmente se identifica el nombre Hadoop con todo el **ecosistema** de componentes independientes que suelen incluirse para dotar a Hadoop de funcionalidades necesarias en proyectos Big Data empresariales, como puede ser la ingesta de información, el acceso a datos con lenguajes estándar, o las capacidades de administración y monitorización.

Estos componentes suelen ser proyectos opensource de Apache.

# Ecosistema



Los principales componentes o proyectos asociados al ecosistema Hadoop son los siguientes:

- Apache Hive
- Apache Hbase
- Apache Pig
- Apache Flume
- Apache Sqoop
- Apache Oozie
- Apache ZooKeeper
- Apache Storm
- Apache Spark
- Apache Kafka
- Apache Atlas, Accumulo, Mahout, Phoenix, Ranger, Zeppelin, etc.





# Ecosistema



Cada componente es un proyecto Apache independiente, lo que impacta, entre otros a:

- Política de **versionado** (periodicidad, identificación, ...).
- **Dependencias** del proyecto con otras versiones de componentes del ecosistema y librerías externas.
- **Roadmap** y estrategia del proyecto.
- Committers / **desarrolladores**.

Por este motivo, realizar una **instalación** de toda una plataforma Hadoop con sus componentes asociados de forma independiente (Hadoop Vanila) resulta muy complicado.

La misma dificultad ocurre para la resolución de **incidencias** que puedan ocurrir en la plataforma cuando se ejecuta en producción.

# Distribuciones



Para solventar las dos dificultades mencionadas, surgen las distribuciones comerciales de Hadoop, que contienen en un único paquete la mayor parte de componentes del ecosistema, resolviendo dependencias, añadiendo incluso utilidades, e incorporando la posibilidad de contratar soporte empresarial 24x7.

Las principales distribuciones que han aparecido son:



# Distribuciones



Para solventar las dos dificultades mencionadas, surgen las distribuciones comerciales de Hadoop, que contienen en un único paquete la mayor parte de componentes del ecosistema, resolviendo dependencias, añadiendo incluso utilidades, e incorporando la posibilidad de contratar soporte empresarial 24x7.

Las principales distribuciones que han aparecido son:

cloudera



# Distribuciones



- **Cloudera:** fue la primera distribución en salir al mercado (2009) y la que ha tenido un mayor número de clientes. Utiliza la mayor parte de componentes de Apache, en algún caso realizando algunas modificaciones, y añade algún componente propietario (Impala, Cloudera Manager, Cloudera Navigator, etc.).
- **Hortonworks:** surgió en 2012 y es una distribución que contiene, sin ninguna modificación, los componentes de Apache sin ninguna modificación.
- **MAPR:** rehizo la mayor parte de componentes utilizando los mismos interfaces pero reimplementando el core para ofrecer un mayor rendimiento.

# Distribuciones



En octubre de 2018 se anunció la fusión de las dos principales distribuciones, Cloudera y Hortonworks, que se hizo efectiva durante 2019, lo que generó un movimiento sin precedentes en el mercado de las tecnologías Big Data.



Asimismo, en 2019, MAPR, ante la imposibilidad para obtener fondos que financiaran su actividad, cesó su actividad, vendiendo todo su portfolio a HPE.

Desaparece MAPR y Hortonworks se une a Cloudera

***¿Por qué tan pocas distribuciones?***



# Distribuciones



Además de las distribuciones mencionadas, es necesario añadir las soluciones Hadoop-as-a-Service:

- Amazon Elastic Map Reduce (EMR).
- Microsoft Azure HDInsight (y evoluciones).
- Google Dataproc.

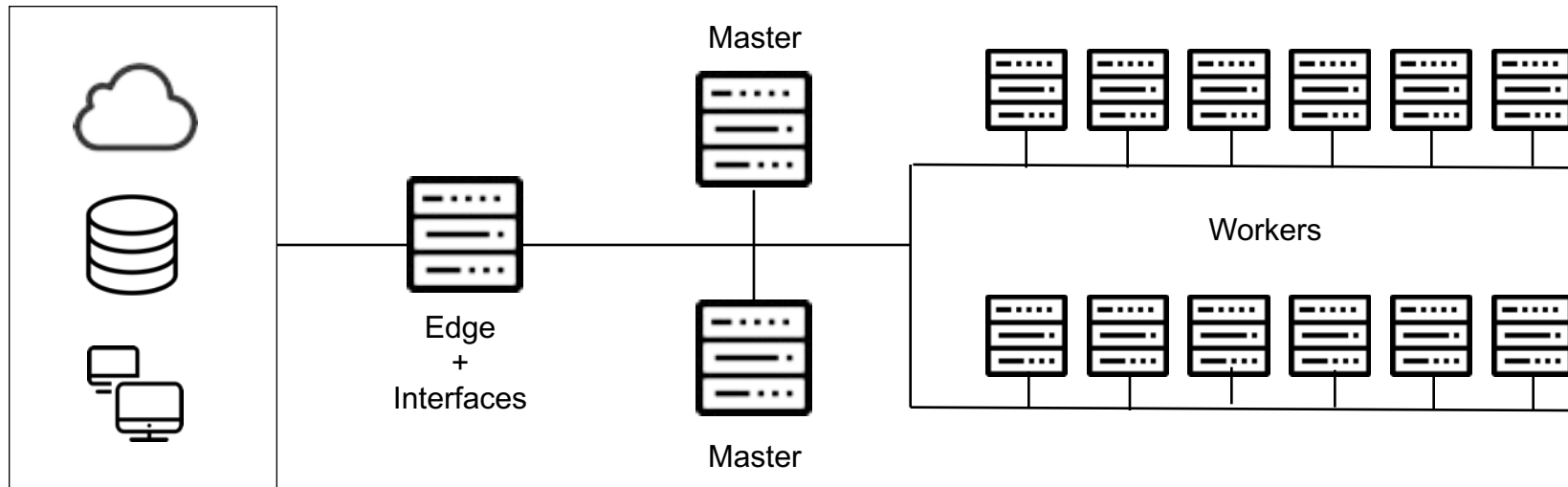
Estas soluciones permiten levantar clusters elásticos en pocos minutos en modalidad pago por uso.

El coste aproximado es de 0,25 - 2 € por nodo y hora.

# Arquitectura



La arquitectura de Hadoop se basa en el uso de un conjunto de nodos “**worker**”, que realizan los trabajos, nodos “**master**” que controlan la ejecución o el almacenamiento de los trabajos/datos, y nodos “**edge**” que hacen de puente entre el clúster y la red exterior y proporcionan interfaces.





# Hardware



A veces el concepto “hardware **commodity**” suele confundirse con “hardware de andar por casa”, cuando lo que hace referencia es a hardware no específico, que no tiene unos requerimientos en cuanto a disponibilidad o resiliencia exigentes.

El hardware típico donde se ejecuta un cluster Hadoop es el siguiente:

- Nodos worker: 256 Gb RAM – 12 HD x 2-3 TB JBOD – 2 CPU x 8 cores.
- Nodos master: 256 Gb RAM – 2 HD x 2-3 TB RAID – 2 CPU x 8 cores.
- Nodos edge: 256 Gb RAM – 2 HD x 2-3 TB RAID – 2 CPU x 8 cores.

# Coste



Para proyectos empresariales, “opensource” no significa “gratis”, ya que se requiere un soporte que es necesario pagar.

El coste habitual de una implantación de Hadoop es el siguiente:

- Hardware: 5–15k € / nodo worker + 3-12k € / nodo master o frontera.
- Soporte: 4-15 k € / nodo al año (según distribución).
- Servicios añadidos de consultoría + implantación: depende de la complejidad de la organización.

Por ejemplo, para un cluster de 20 nodos worker (capacidad aprox. 200 TB) más 4 nodos master más 1 nodo frontera el coste aproximado sería:

- Hardware: 240k €.
- Soporte: 150k €/año.

# ¿Por qué Hadoop?



Hadoop es una **plataforma de almacenamiento y procesamiento** de datos de cualquier tipo y volumen y a un **coste razonable** cuando el volumen de datos es grande.

No es la única tecnología que da respuesta a Big Data, pero sí la que más casuísticas cubre. Normalmente se habla de Hadoop como una plataforma, no como una herramienta.

# Resumen de características



Los datos que se incorporan no necesitan un **esquema prefijado**

Almacena y procesa **cualquier tipo de información**

**Bajo coste** (hardware commodity\* y código open-source)

**Escalabilidad** “ilimitada” y lineal

Enfoque **distribuido**

# Problemática asociada



Requiere nuevos **perfiles escasos**.

Hardware commodity  $\neq$  PC y opensource  $\neq$  gratis.

Nueva pieza en las arquitecturas de gestión de datos, su integración no es sencilla.

El enfoque **distribuido** tiene sus problemas:

- Administración más compleja (N nodos x M componentes)
- Consumo energético, espacio en CPD, ...
- Menor eficiencia que enfoques centralizados.

**Seguridad y gobierno** de datos

**Inmadurez:** tanto de la herramienta como de la organización.

# ¿Cuándo usar Hadoop?



Cuando el volumen de datos es mayor que la capacidad de los sistemas tradicionales (no cabe en una máquina).

Cuando hay un problema de variedad de datos, porque son diversos o porque cambian frecuentemente.

Cuando se requiere una escalabilidad que no pueden ofrecer los sistemas tradicionales, por volumen, por velocidad de proceso, por rendimiento global, y no se requiere un nivel de transaccionalidad elevado.

Cuando se pretende tener una plataforma con la capacidad de almacenamiento y procesamiento de un gran volumen de datos para cubrir diferentes casos de uso (con la misma plataforma).

# ¿Cuándo no usar Hadoop?



Cuando los sistemas tradicionales son capaces de dar soporte a los casos de uso y cuando los formatos/tipos de datos son fijos o no cambian apenas.

Cuando se tiene requisitos de transaccionalidad muy estrictos.

Cuando sólo se requiere resolver un caso de uso “Big Data” muy específico.

# Trabajo de reflexión



Discutir en clase este escenario:

- Actualmente en un banco los datos de los diferentes canales no son compartidos, de manera que cuando un usuario llama al teléfono de atención del cliente para poner una reclamación, si al día siguiente va a la oficina, el director de la oficina no conoce la existencia de dicha reclamación y no puede hacer un tratamiento especial al cliente.
- Asimismo, la información sobre la navegación que hacen los usuarios en la web, al ser un volumen muy grande de información (cada click se almacena por millones de usuarios y páginas vistas) no se procesa. Lo mismo ocurre con otra información como el detalle de los pagos con tarjeta (localización, comercios, etc.), que por su volumetría no se procesa.
- Otra información que maneja el banco, como emails o transcripción de llamadas, por su naturaleza, no son procesadas.



# Trabajo de reflexión



- ¿Tendría algún beneficio desplegar una plataforma Hadoop en el banco?
- En caso de ser beneficioso, ¿qué casos de uso por ejemplo podrían implementarse que ahora no se implementen?

# Trabajo de reflexión



Discutir en clase este escenario:

- Una compañía de intermediación de seguros gestiona una cartera de 300.000 clientes. Para cada cliente almacena información sobre sus datos de contacto y las pólizas que tiene contratadas.
- Sobre estos datos, la dirección quiere tener cuadros de mando en los que poder obtener información sobre evolución de las pólizas contratadas, el desempeño de cada sucursal, etc.
- Además, les gustaría tener modelos predictivos que les permitan adelantarse a la demanda o preveer clientes que podrían abandonar la compañía.

# Trabajo de reflexión



- ¿Tendría algún beneficio desplegar una plataforma Hadoop en la compañía?
- En caso de ser beneficioso, ¿qué casos de uso por ejemplo podrían implementarse que ahora no se implementan?

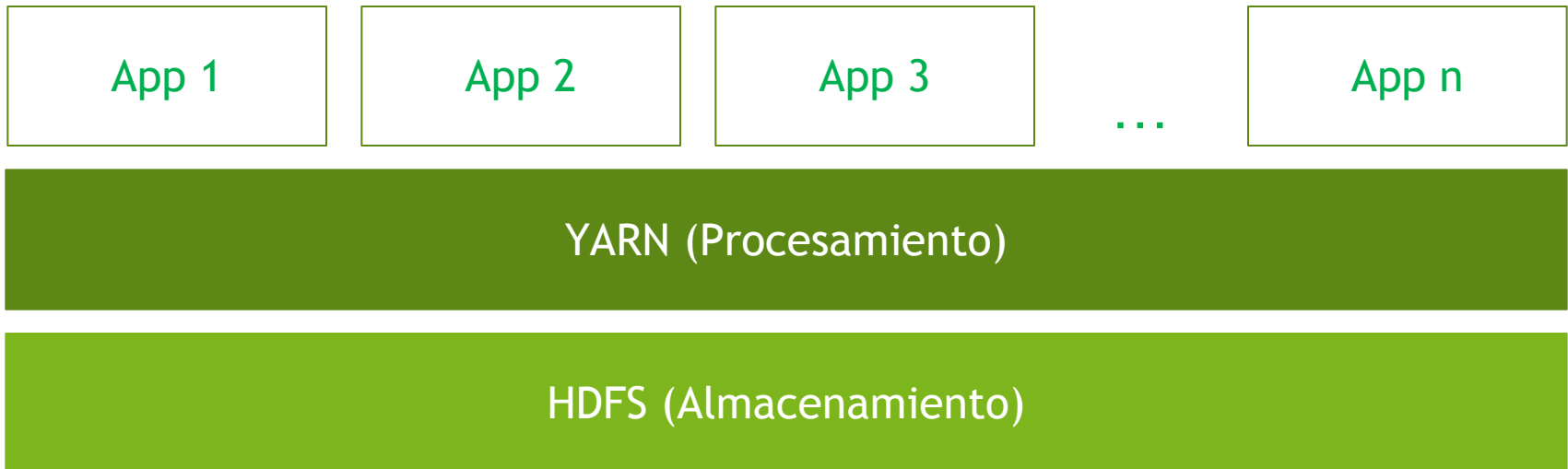
# — 05

# Hadoop core: HDFS y YARN.

# Hadoop core



El core de Hadoop está formado por HDFS y YARN, junto con un conjunto de librerías generales.



# HDFS



**Hadoop Distributed File System**, HDFS, es un sistema de ficheros distribuido diseñado para ejecutarse sobre **hardware commodity** y optimizado para almacenar **ficheros de gran tamaño**.

Tiene capacidad para **escalar horizontalmente** hasta volúmenes de Petabytes y miles de nodos, y está diseñado para poder dar soporte a **múltiples clientes** con acceso concurrente.

No establece **ninguna restricción sobre los datos** que se almacenan en el sistema, ya que éstos pueden ser estructurados, semiestructurados o no disponer de ninguna estructura, como el caso de imágenes o vídeos.

# HDFS



Utiliza **replicación** de los datos entre distintos nodos para sobreponerse a errores y proporciona a los clientes un **interfaz similar a los sistemas de ficheros tradicionales**, con organización de los ficheros en **espacios de nombres** (directorios), y funcionalidades para lectura, búsqueda, listado de ficheros, etc. sin necesidad de conocer la ubicación de los datos o la organización del clúster.

HDFS divide los ficheros en **bloques** de un tamaño por defecto de **128 Mb**, que son **almacenados y replicados** en distintos nodos. Tener un tamaño de bloque tan grande permite un throughput de lectura/escritura elevado, pero se penaliza las operaciones aleatorias.

# HDFS



La arquitectura de HDFS consta de distintos servicios y tipos de nodo, aunque fundamentalmente son:

- **Namenode:** actúa de maestro, manteniendo la metainformación de todo el sistema de ficheros, esto es, el espacio de nombres, la ubicación de los bloques y su replicación. Dispone de un Secondary Namenode para HA.
- **Datanode:** actúan de worker / esclavo, contienen físicamente los bloques pero no gestionan su metainformación, y controlan la validez de los bloques (checksums).

La replicación se realiza de forma automática coordinada por el Namenode. Por defecto los bloques se replican en 3 datanodes, aunque puede modificar el número de réplicas para ficheros / directorios de forma individual.



# HDFS



HDFS soporta operaciones similares a los sistemas Unix:

- Lectura, escritura o borrado de ficheros.
- Creación, listado o borrado de directorios.
- Usuarios, grupos y permisos.

HDFS es un sistema de ficheros append-only, no se permite modificar bloques. Esta filosofía append-only condiciona mucho el funcionamiento de otras aplicaciones que utilizan HDFS como almacenamiento (Hive, HBase, etc.).

# HDFS



Interfaces de acceso:

- Cliente de línea de comandos.
- Java API.
- RestFul API (WebHDFS).
- NFS interface (HDFS NFS Gateway).

```
[root@HDL1 menish]# hdfs dfs -ls /
Found 8 items
drwxrwxrwx   - yarn   hadoop           0 2014-12-27 21:12 /app-logs
drwxr-xr-x   - hdfs   hdfs           0 2014-12-27 20:41 /apps
-rw-r--r--   3 root   hdfs          56 2015-01-05 18:54 /architect
drwxr-xr-x   - mapred hdfs           0 2014-12-27 20:38 /mapred
drwxr-xr-x   - hdfs   hdfs           0 2014-12-27 20:38 /mr-history
-rw-r--r--   3 root   hdfs          56 2015-01-05 18:55 /test
drwxrwxrwx   - hdfs   hdfs           0 2014-12-27 21:11 /tmp
drwxr-xr-x   - hdfs   hdfs           0 2014-12-27 21:11 /user
```

# HDFS



Secuencia de una lectura en HDFS:

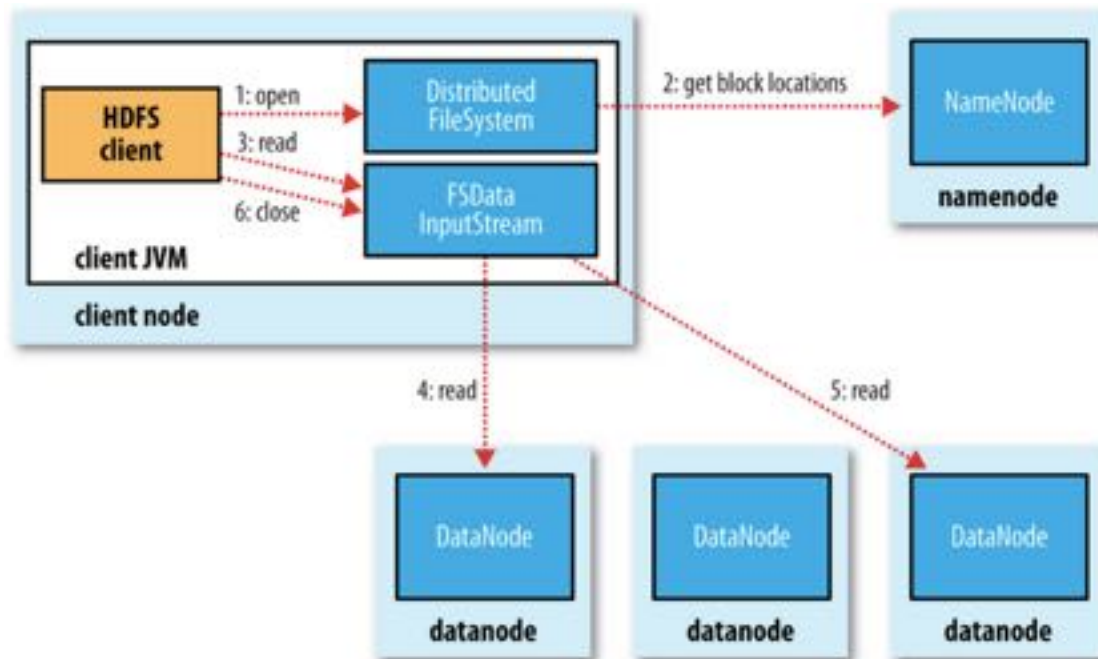


Imagen de Hadoop: The Definitive Guide. <http://shop.oreilly.com/product/0636920033448.do>

# HDFS



Secuencia de una escritura en HDFS:

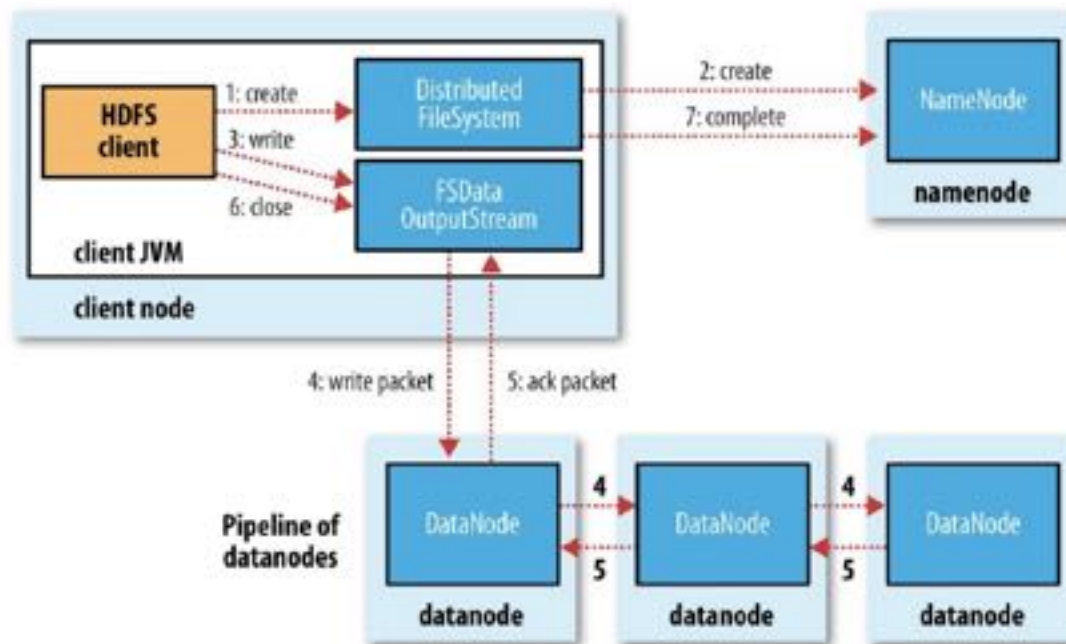


Imagen de Hadoop: The Definitive Guide. <http://shop.oreilly.com/product/0636920033448.do>

# YARN

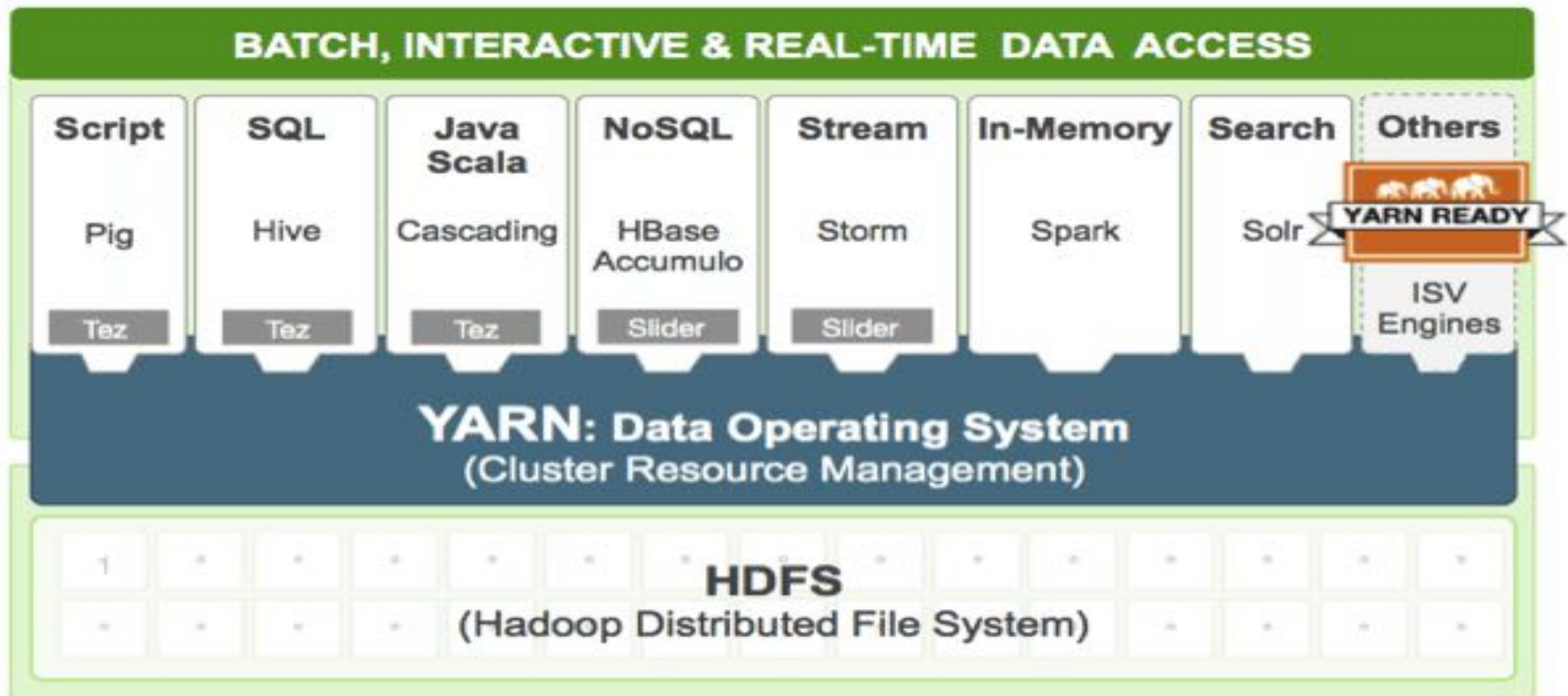


YARN (Yet Another Resource Negotiator) es un gestor de **recursos de computación y memoria** que no estaba presente en las primeras versiones de Hadoop, que usaban únicamente MapReduce como paradigma de computación.

Habilita el procesamiento de los datos almacenados en HDFS de forma concurrente y utilizando tecnologías diferentes de MapReduce (por ejemplo, Storm para event processing).

La mayoría de aplicaciones y frameworks de Hadoop se ejecutan sobre YARN en lo que se refiere a ejecución distribuida (Spark, Storm, Tez, etc.).

# YARN



<https://es.hortonworks.com/apache/yarn/>

# YARN



Principales responsabilidades de YARN:

- Planificar / **organizar** las **tareas y gestionar los recursos** de CPU y memoria.
- Habilitar el uso del cluster para entornos **multi-tenant**, compartidos y seguros.

Las aplicaciones necesitan recursos de CPU y memoria para ejecutarse, que son llamados **Containers**.

YARN se ocupa de la asignación de contenedores a las aplicaciones, estableciendo prioridades, turnos, etc.x

# YARN

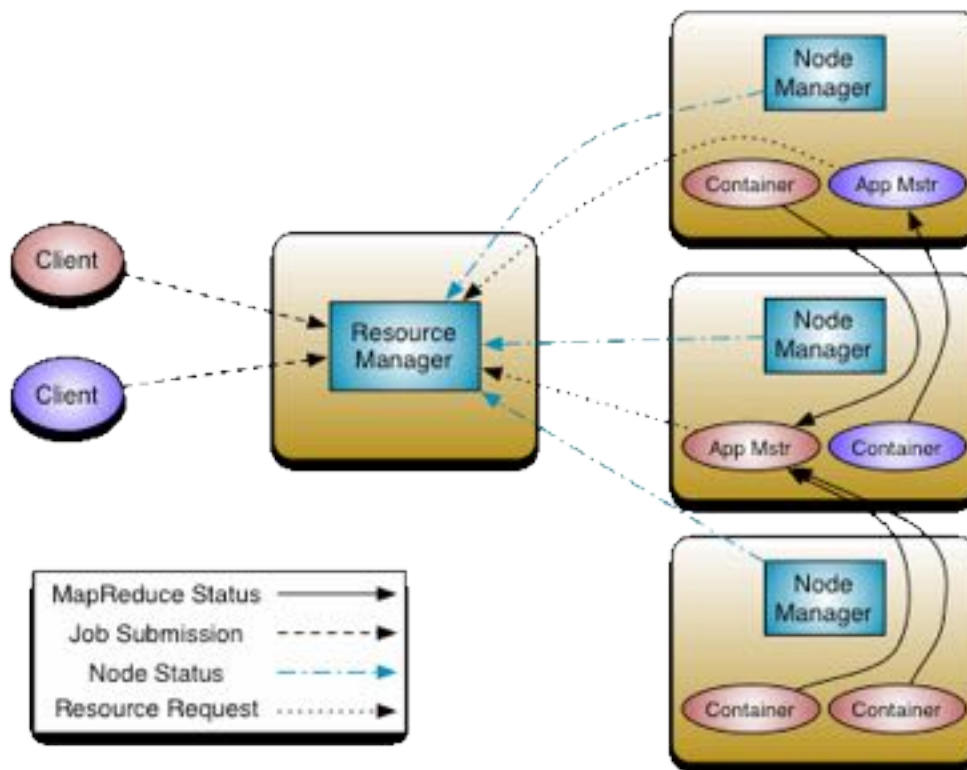


La arquitectura de YARN, de tipo maestro/esclavo, consta de dos tipos de nodos:

- **ResourceManager** (master): maestro de YARN, organiza las tareas y gestiona los recursos disponibles. Dispone de un Scheduler (pluggable) para la organización de tareas. Los **Schedulers** permiten establecer prioridades, reservar recursos, etc. para las aplicaciones. Los principales son:
  - Capacity Scheduler.
  - Fair Scheduler.
  - FIFO Scheduler.
- **NodeManager** (worker): lanza y maneja las aplicaciones en los contenedores de que dispone, monitoriza los recursos (CPU, red y memoria) e informa al ResourceManager periódicamente.



# YARN



# YARN



Algunas características de YARN:

- Es **Multitenancy**: permite acceso simultáneo de distintos motores / aplicaciones sobre los mismos datos.
- **Escalable**: permite incrementar la capacidad de procesamiento añadiendo nuevos nodos worker.
- **Robusto**: ofrece configuraciones de alta disponibilidad para el Resource Manager, que es el único punto único de fallo.
- **Compatibilidad**: es compatible con las aplicaciones que se ejecutan sobre MapReduce 1.
- **Estándar**: unifica el framework y ciclo de vida de ejecución de aplicaciones sobre Hadoop.

# MapReduce



MapReduce es un **paradigma de computación** creado por Google para resolver la creación de índices de búsqueda de páginas.

Fue el motor de procesamiento de Hadoop en sus **primeras versiones**.

Se basa en la capacidad de **dividir** el procesamiento sobre los datos en múltiples nodos y en paralelo, resolviendo la limitación de la computación de grandes volúmenes de datos sobre una única máquina y abstrayendo al desarrollador de las tareas de orquestación de los sistemas de computación distribuida tradicionales.

# MapReduce



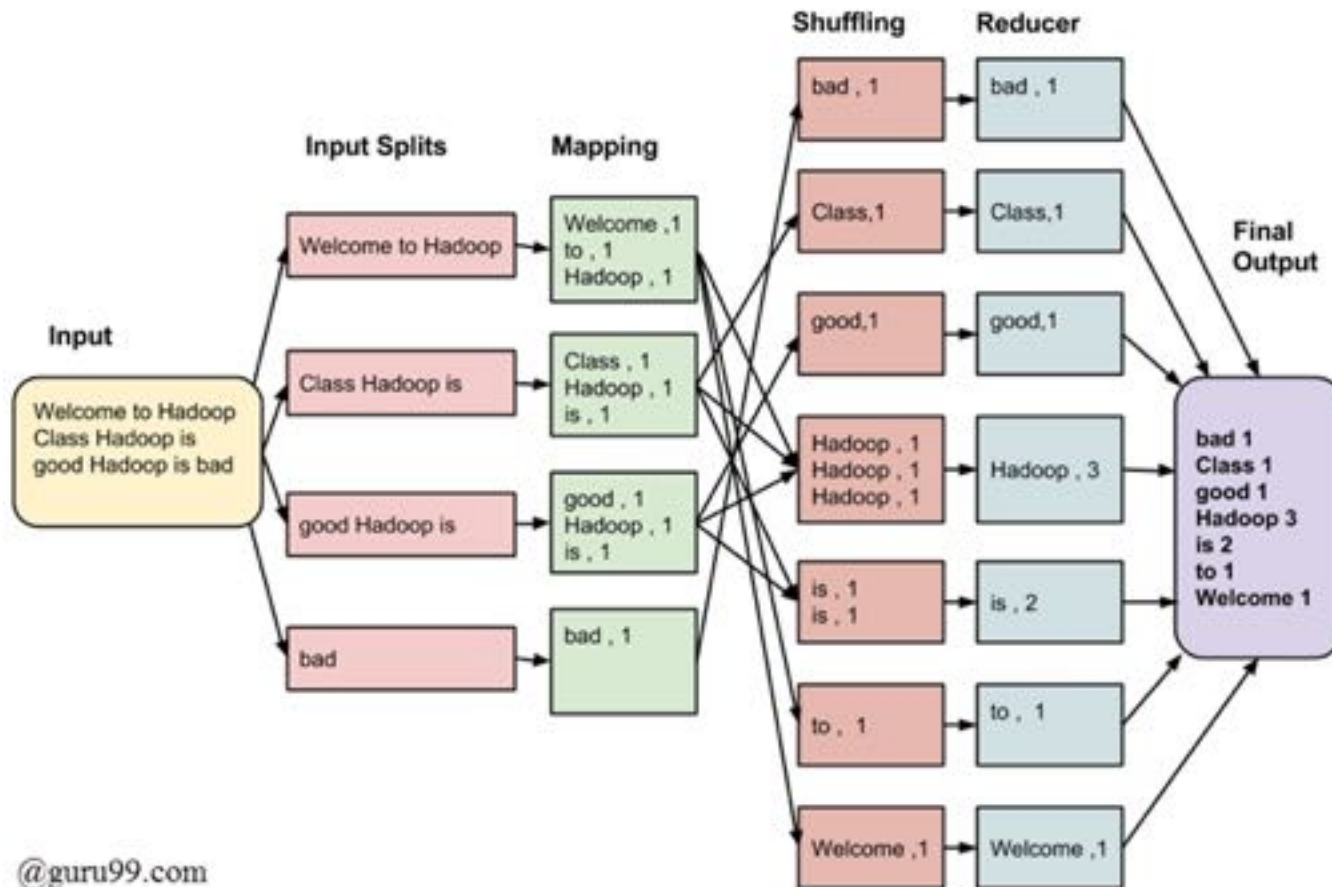
Frente a los sistemas de computación tradicionales, MapReduce ejecuta la lógica **en los nodos donde residen los datos** en lugar de moverlos por la red, lo que mejora el rendimiento en órdenes de magnitud.

MapReduce está orientado a procesamiento **offline** y es **tolerante a fallos**, ya que un nodo del clúster puede fallar o caerse sin que el proceso de computación se aborte.

Permite distintos lenguajes de programación: Java, Python, C++, Perl, etc.

Cada vez se utiliza menos debido a que nuevos frameworks de procesamiento, como Tez o Spark, mejoran su rendimiento en órdenes de magnitud.

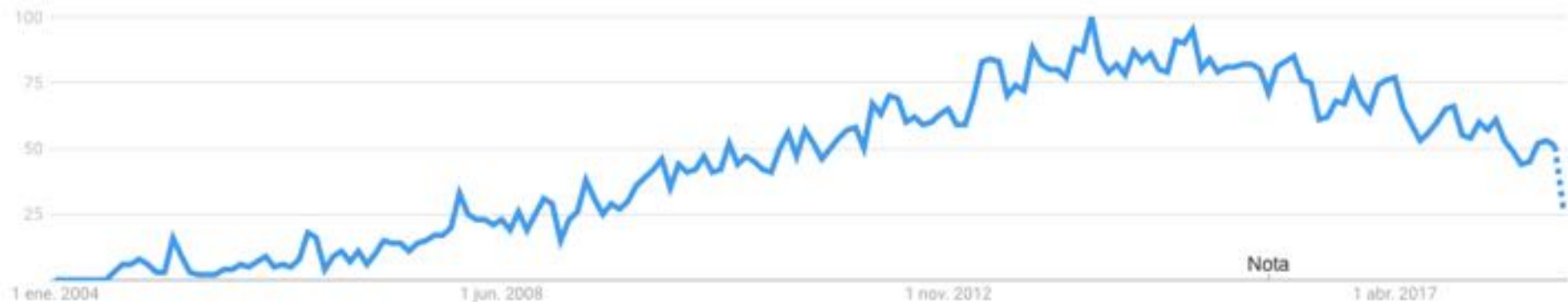
# MapReduce



# MapReduce

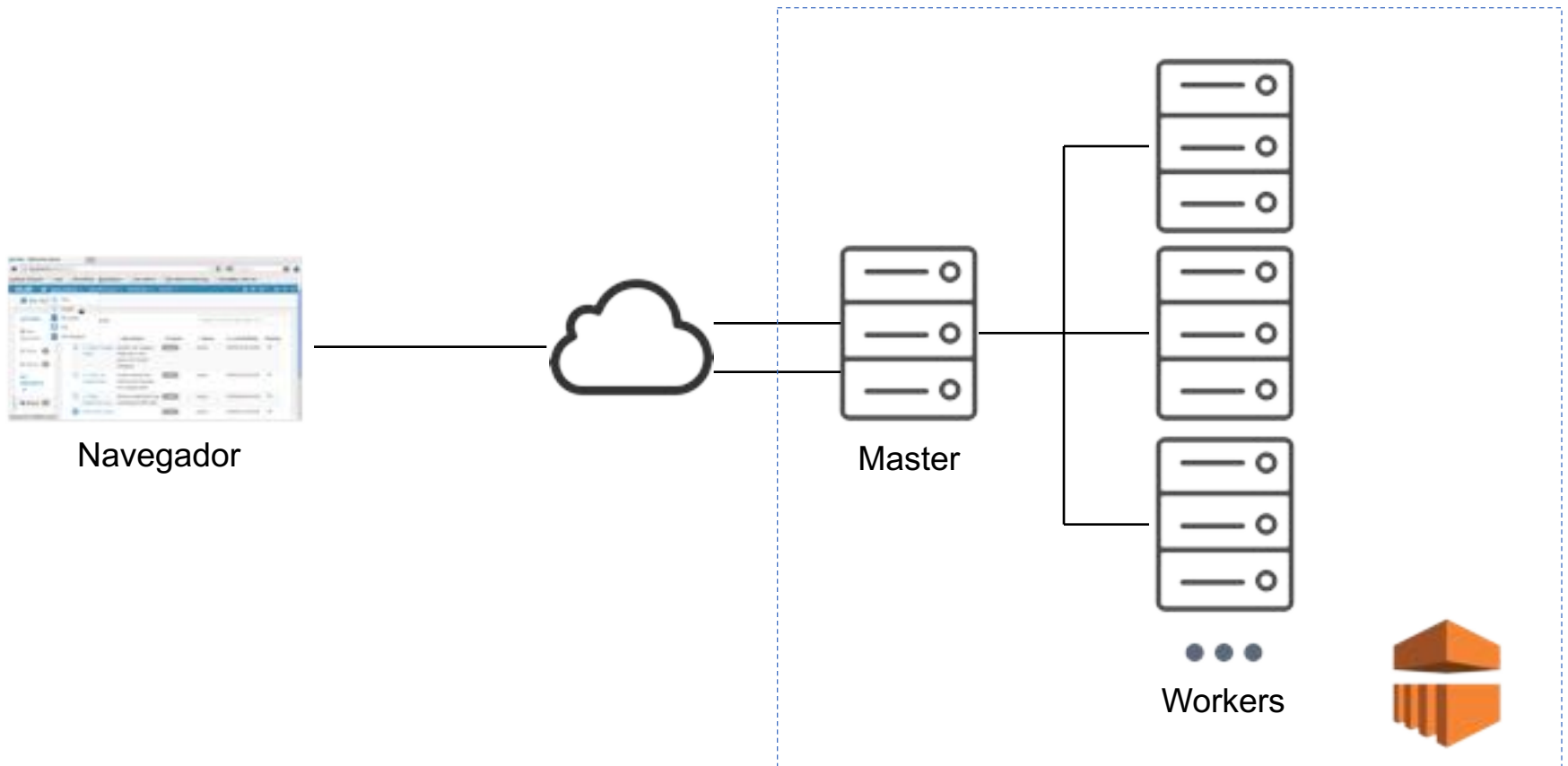


Desde la aparición de Tez, y sobre todo, Spark, está cayendo en desuso.



# Práctica: manipulando datos en HDFS

# Entorno de prácticas





# HDFS “amigable”

**\* TODOS DEBÉIS HABERME ENVIADO VUESTRA DIRECCIÓN IP QUE PODÉIS CONSULTAR EN [www.myip.com](http://www.myip.com) (donde dice “Your IP address is”).**

**Acceder a Hue:**

`https://ec2-*-*-*-*.eu-west-1.compute.amazonaws.com:8888/`

Usuario: hadoop

Password: Hadoop2021!

# HDFS en Hue

1. **Navegar por directorios y crear dos directorios:**  
`/user/hadoop/[vuestras_iniciales]` y  
`/user/hadoop/[vuestras_iniciales]/data`
2. **Descargar el fichero en vuestro PC:**  
`http://www.isenet.es/opendata/equipos\_laliga.csv`
3. **Subirlo al directorio** `/user/hadoop/`  
`/user/hadoop/[vuestras_iniciales]/data`
4. **Ejercicio: crear una carpeta** `/user/hadoop/`  
`/user/hadoop/[vuestras_iniciales]/backup` **y copiar fichero**  
`equipos_laliga.csv`

# Ecosistema Hadoop

# Índice

- \_01 Tez
- \_02 Pig
- \_03 Hive
- \_04 Impala
- \_05 Sqoop
- \_06 Flume
- \_07 Oozie
- \_08 HBase
- \_09 Phoenix
- \_10 Zookeeper
- \_11 Hue y Zeppelin
- \_12 Ambari y Cloudera Manager

# Apache Tez



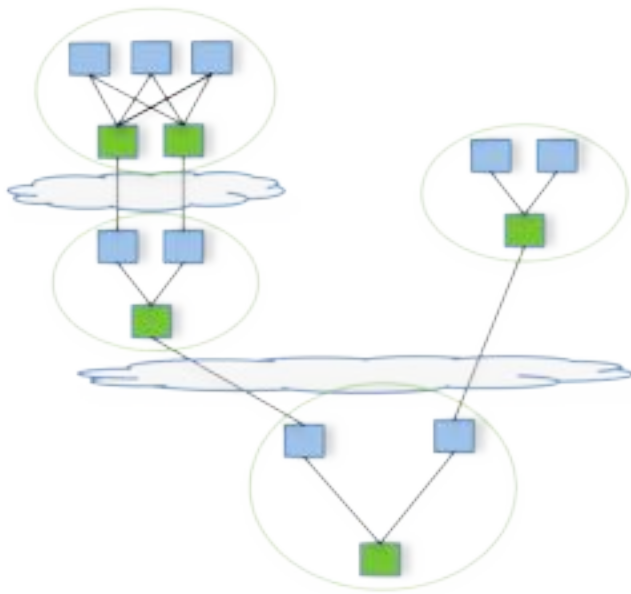
Es un **framework de ejecución** sobre YARN que ofrece un API para que las aplicaciones puedan ejecutarse sin limitarse al paradigma MapReduce, que sólo ofrece operaciones Map y Reduce, con persistencia en disco para cada fase.

Tez fue desarrollado por Hortonworks en colaboración con Microsoft, con el principal objetivo de poder acelerar el procesamiento de consultas SQL sobre Hadoop con Hive.

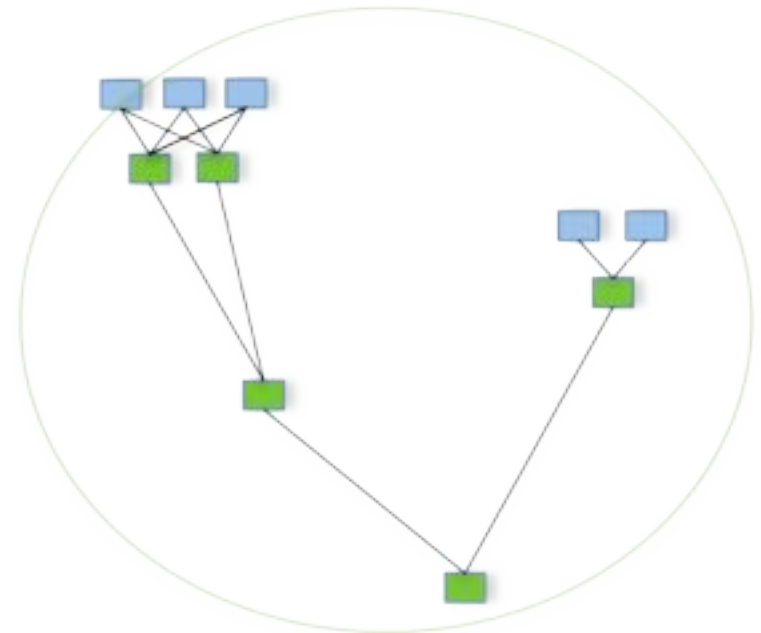
Tez utiliza un modelo de ejecución de grafo acíclico dirigido frente al modelo de fases de MapReduce, y ofrece muchas optimizaciones para poder acelerar un orden de magnitud la ejecución de muchos procesos.

Tiene como inconveniente que ofrece un nivel de fiabilidad menor.

# Apache Tez



Pig/Hive - MR

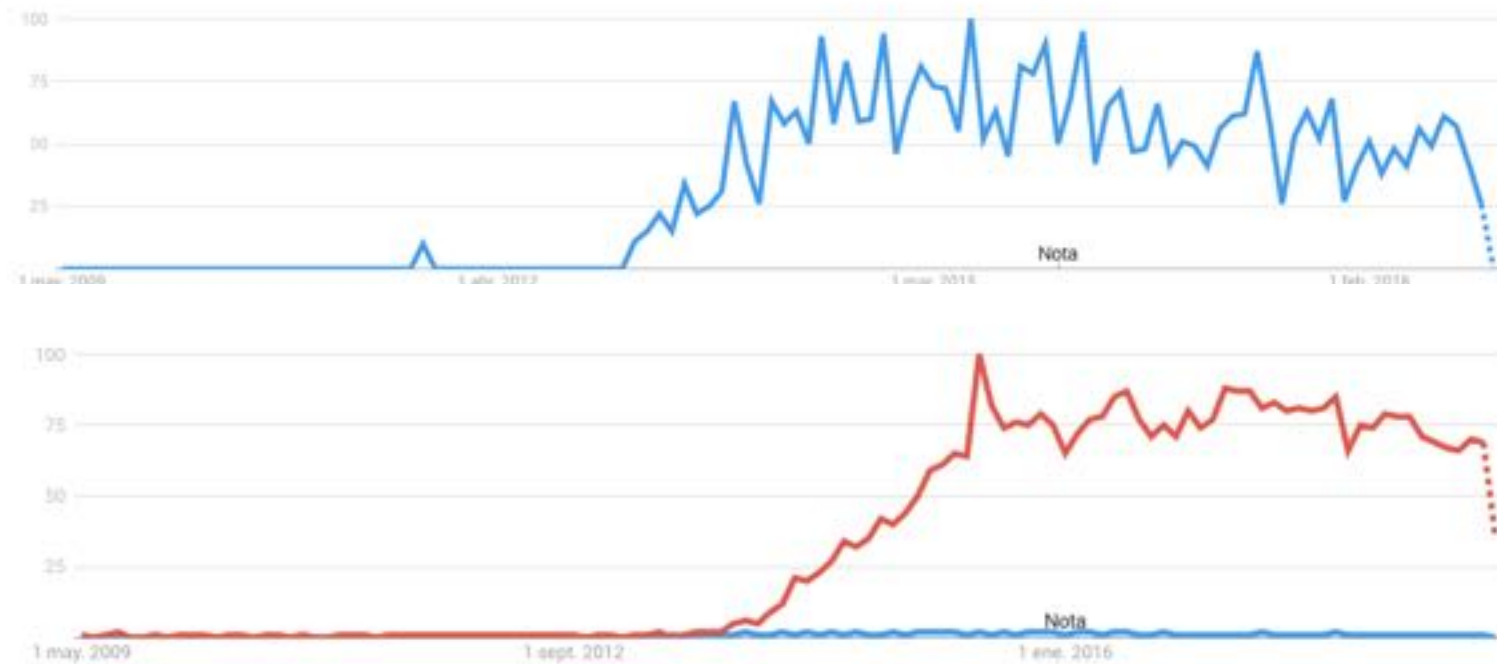


Pig/Hive - Tez

# Apache Tez



Pese al esfuerzo por parte de Microsoft y Hortonworks para su adopción, la realidad es que la aparición de Apache Spark ha lastrado su uso.



# Apache Pig



MapReduce es un framework que requiere mucho desarrollo de código de bajo nivel para poder implementar aplicaciones de procesamiento de datos.

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
}
```

```
public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context
            ) throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

¡Todo esto para contar las palabras de un fichero!



# Apache Pig



Debido a la gran cantidad de código que es necesario generar, así como el nivel de especialización que requiere, la productividad de MapReduce es reducida.

Con el objetivo de poder simplificar la implementación de **programas de procesamiento de datos**, Yahoo desarrolló en 2006 Pig como lenguaje de mayor nivel que pudiera ser utilizado por analistas que no disponen de grandes conocimientos de programación.

# Apache Pig



Pig es un **motor de ejecución** sobre MapReduce (ahora también sobre Tez o Spark) que ofrece un nivel de abstracción mayor.

Utiliza como lenguaje Pig Latin, que tiene similitudes con SQL.

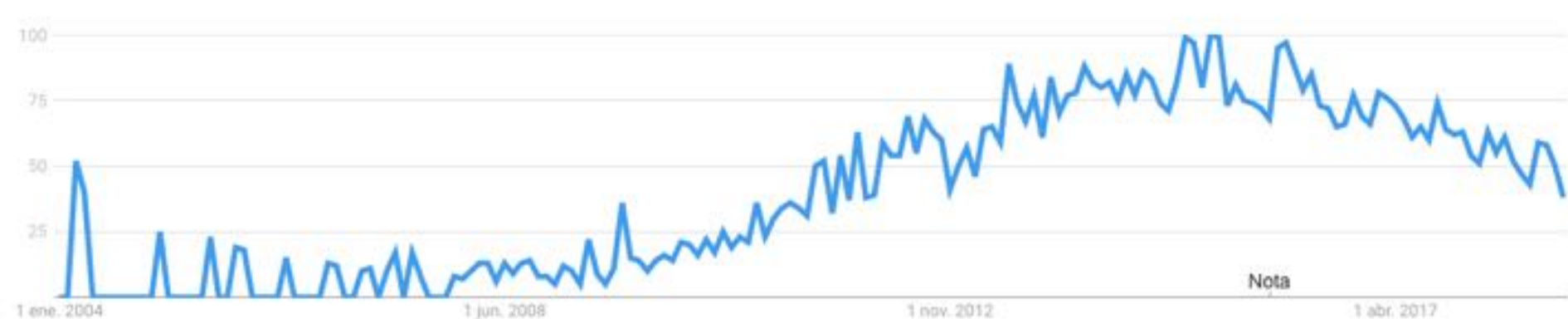
```
input_lines = LOAD '/tmp/file-to-count-words' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count,
group AS word;
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

# Apache Pig



Pig se utiliza principalmente para movimiento o transformación de datos (ETL).

Desde la aparición de Apache Spark, está cayendo en desuso.



# Apache Hive



Hadoop presenta **limitaciones** a la hora de explotar datos en HDFS:

- Necesita **programación** de procesos **MapReduce** para manipular datos (requiere conocimientos de programación).
- Falta de **integración** con herramientas de gestión o explotación de datos.
- **Encarecimiento** de soluciones software por no poder reutilizar código MapReduce. Dificultad de industrialización.

Ante estas dificultades, un grupo de ingenieros de **Facebook** desarrolló Hive como una herramienta que permite simplificar las tarea de analítica con ficheros de HDFS, utilizando un lenguaje similar a SQL para su acceso.

Hive fue incluido como proyecto Apache, lanzando su primera versión estable en 2010.

# Apache Hive



Hive ofrece una visión de los datos en HDFS como un **Datawarehouse** que permite manejar grandes volúmenes de información de forma simple.

Características y funcionalidades:

- Capacidad de definir una **estructura relacional** (tablas, campos, etc.) sobre la información “en bruto” de HDFS.
- **Lenguaje** de consultas HQL, de sintaxis muy similar a SQL incluyendo muchas de las evoluciones de SQL para analítica: SQL-2003, SQL-2011 y SQL-2016.
- **Interfaces estándar** con herramientas de terceros mediante JDBC/ODBC.

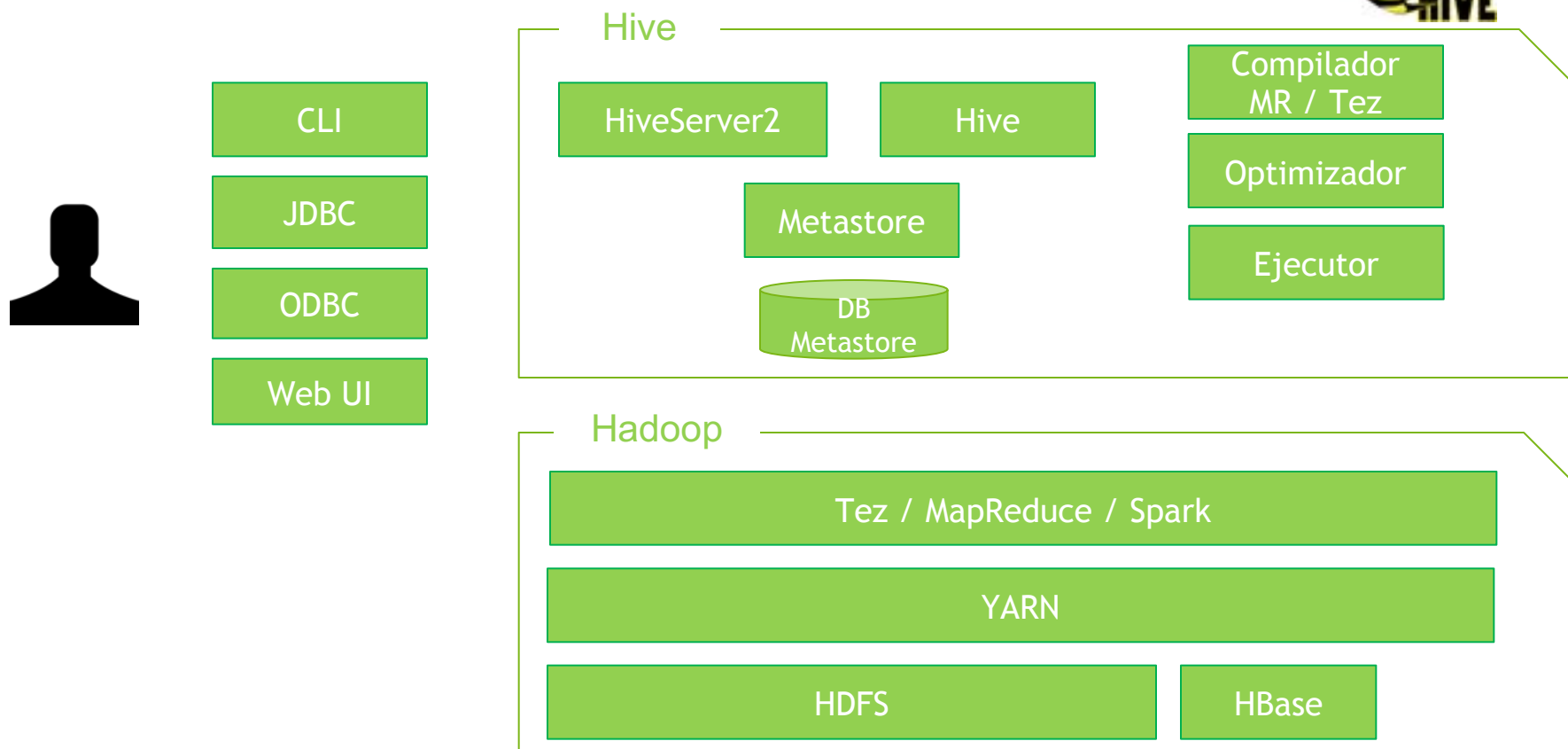
# Apache Hive



## Más características:

- Utiliza **motores** de procesamiento de Hadoop estándar para la ejecución de consultas: MapReduce, Spark o Tez, realizando la traducción automática entre el lenguaje de consultas (HQL) y el código de los programas a ejecutar.
- Mecanismos de **seguridad** en el acceso y modificación de la información.
- Lee ficheros con diferentes **formatos**: texto plano, RF, ORC, HBase, Parquet y otros.
- **Extensible** mediante código a medida (UDF / MapReduce).
- Orientado a **consultas** aunque también ofrece operativa para creación, modificación o borrado de registros.

# Apache Hive. Arquitectura



# Apache Hive. HCatalog



Metastore



- Registro de todos los **metadatos** de Hive y otros componentes que requieren aplicar un modelo sobre datos existentes en HDFS o HBase.
- Por ejemplo, almacena la información de las **tablas** definidas, qué **campos** tienen, con qué **ficheros** se corresponden, etc.
- Permite que **aplicaciones distintas a Hive** puedan usar la definición de tablas definida en Hive, o al revés.

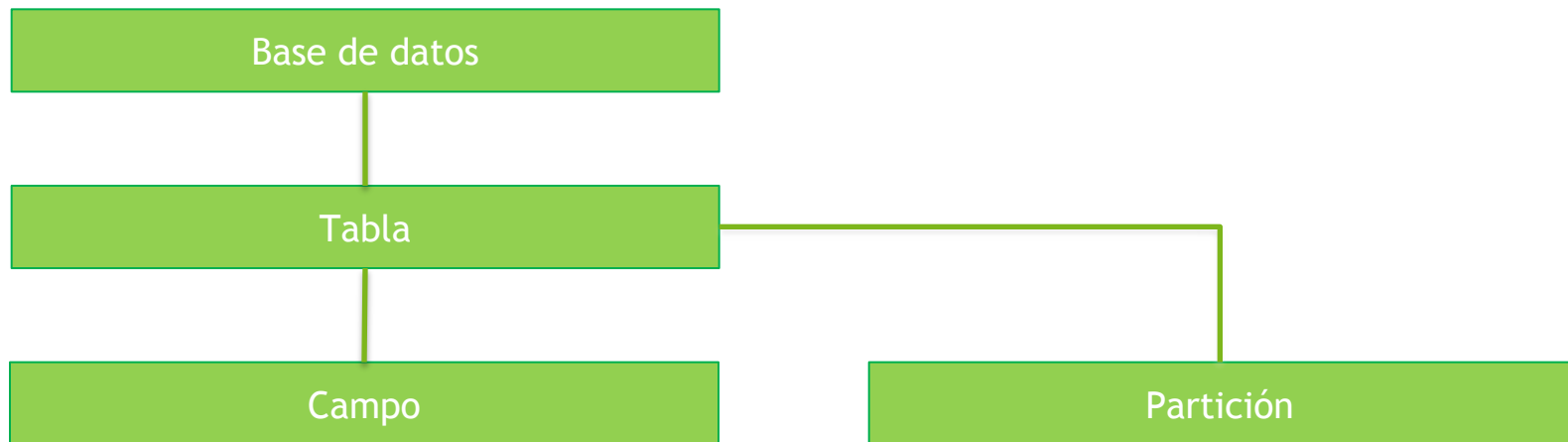


# Apache Hive. HQL



Dos tipos de operaciones: DDL y DML.

- **DDL** (Data Definition Language): permite crear estructuras de datos.
- **DML** (Data Manipulation Language): permite alta, baja, modificación y consulta de datos.



# Apache Hive. HQL



- **Tipos de datos:**
  - Enteros: TINYINT, SMALLINT, INT/INTEGER, BIGINT
  - Decimales: FLOAT, DOUBLE, DECIMAL, NUMERIC.
  - Fecha/hora: TIMESTAMP, DATE, INTERVAL.
  - Cadenas: STRING, VARCHAR, CHAR.
  - Otros: BOOLEAN, BINARY.
  - Compuestos: ARRAY, MAP, STRUCT, UNIONTYPE.

# Apache Hive. HQL



## DDL

- **Bases de datos:**
  - **Creación:** CREATE DATABASE my\_db
  - **Visualización:** SHOW DATABASES
  - **Uso:** USE my\_db
  - **Borrado:** DROP DATABASE my\_db
  - **Modificación:** ALTER DATABASE my\_db SET ...
  - Visualización de **detalle:** DESCRIBE DATABASE my\_db

# Apache Hive. HQL



## DDL

- **Tablas:**

- **Creación:** CREATE TABLE.

```
CREATE TABLE inmuebles (  
    id INT,  
    direccion STRING,  
    provincia STRING,  
    superficie INT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
STORED AS TEXTFILE;
```

- **Visualización:** SHOW TABLES
  - **Borrado:** DROP TABLE inmuebles

# Apache Hive. HQL



- **Modificación:** ALTER TABLE my\_table...

```
ALTER TABLE inmuebles  
ADD COLUMNS (pais STRING COMMENT 'Nombre del país')
```

```
ALTER TABLE inmuebles RENAME TO casas
```

- **Visualización de detalle:**

```
DESCRIBE [EXTENDED|FORMATED] inmuebles
```

# Apache Hive. HQL



## DML

- **Consultas: SELECT.**

```
SELECT id, direccion FROM inmuebles  
WHERE provincia = 'Madrid'  
ORDER BY superficie
```

```
SELECT provincia, count(*) FROM inmuebles  
WHERE superficie > 100  
GROUP BY provincia
```

# Apache Hive. HQL



## DML

- **Consultas:** SELECT.

```
[WITH CommonTableExpression (, CommonTableExpression)*]
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
  FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_list]
  [ORDER BY col_list]
  [CLUSTER BY col_list
   | [DISTRIBUTE BY col_list] [SORT BY col_list]
  ]
[LIMIT [offset,] rows]
```

# Apache Hive. HQL



## DML

- **Consultas:** SELECT. Más operaciones que se permiten:
  - Subconsultas (SELECT FROM SELECT).
  - JOIN de tablas.
  - UNION de tablas.
  - Operadores (UDFs).



# Apache Hive. HQL



## DML

- **Cargar datos:** LOAD DATA .

```
LOAD DATA LOCAL INPATH './data/inmuebles.csv'  
OVERWRITE INTO TABLE inmuebles
```

- **Inserción múltiple:** INSERT OVERWRITE.

```
INSERT OVERWRITE TABLE inmuebles SELECT a.* FROM  
herencias a;
```

```
INSERT OVERWRITE LOCAL DIRECTORY '/data/local_out'  
SELECT a.* FROM inmuebles a
```

# Apache Hive. HQL



## DML

- **Inserción simple: INSERT.**

```
INSERT INTO TABLE inmuebles  
VALUES (23, 'Calle Sol 23, 'Madrid', 'Madrid', 234)
```

- **Actualización: UPDATE.**

```
UPDATE inmuebles SET provincia = 'A Coruña'  
WHERE provincia = 'La Coruña'
```

# Apache Hive. HQL



## DML

- **Borrado:** DELETE.

```
DELETE FROM inmuebles WHERE provincia = 'Madrid'
```

# Impala



Las primeras versiones de Hive ofrecían un rendimiento pobre para consultas on-line, que requerían latencias de segundos.

Impala fue desarrollado inicialmente por Cloudera como alternativa o complemento a Hive, permitiendo prácticamente la **misma funcionalidad**, es decir, ofrecer un **lenguaje SQL-like (HQL) sobre datos almacenados en HDFS o HBase**.

La primera versión beta fue lanzada en **octubre de 2012** y era incluida en la distribución comercial de Cloudera, denominada Cloudera CDH.

En 2015 fue **donada a Apache SF** para su inclusión como proyecto, siendo elevada a proyecto top-level en noviembre de 2017.

# Impala



## Características y funcionalidades:

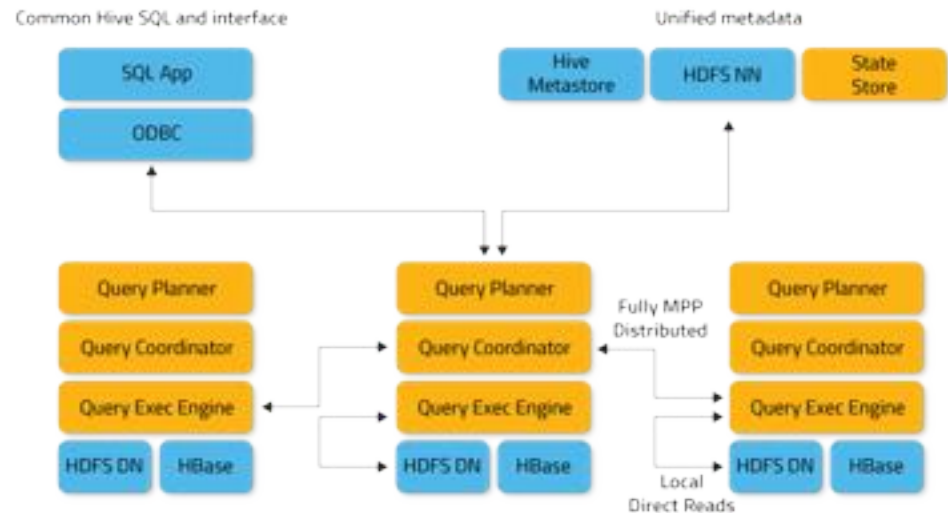
- El **rendimiento** de Impala se debe a que está implementada en código de más bajo nivel de abstracción que Hive (**C++**), y que dispone de un amplio conjunto de optimizaciones en la ejecución de consultas.
- Ofrece el mismo **interfaz** ODBC y lenguaje de consultas de Hive: **HQL**.
- Está orientado a **consultas** típica de **BI / BA**, es decir, consultas complejas que requieren agrupaciones, subconsultas, funciones analíticas, etc.
- Ofrece el mismo nivel de **segurización** de Hive, es decir, autenticación mediante Hadoop / Kerberos, y autorización mediante Sentry.

# Impala



- Soporta **almacenamiento en HBase y HDFS**, permitiendo para éste formatos txt, LZO, SequenceFile, Avro, RCFile y Parquet.
- Para la gestión de metadatos utiliza el mismo almacén/servicio que Hive: **HCatalog / Metastore**.

- Arquitectura:



# HBase



Dos de las principales características de HDFS (sistema de almacenamiento de Hadoop) son la **inmutabilidad** y el gran **tamaño de bloques**.

Estas dos características dan a HDFS un **gran ancho de banda** en lecturas o escrituras masivas, pero dificultan las operaciones **CRUD** de datos atómicos o de pequeño tamaño, que son las operaciones que habitualmente se demandan para dar servicio a las aplicaciones **operacionales** de cualquier compañía.

HBase surge para dar soporte (escalable) a estas operaciones:

- Acceso **aleatorio** a la información.
- Operaciones de **actualización y borrado** de datos.

**HBase es una base de datos NoSQL de modelo clave-valor sobre Hadoop.**

# HBase



## Características:

- Modelo **no relacional** con almacenamiento **columnar**.

Familia columnas: Datos personales			Familia columnas: Ubicación	
ID (Row key)	Nombre	Apellidos	Provincia	CP
12345678Z	José Luis	Pérez García	Sevilla	41001
33434123E	María	Sol Gómez	Córdoba	14002
87676545S	Esteban	Pino López	Madrid	28003

- Las filas tienen un **row-key** que las identifica y organiza, la definición del row-key es un aspecto clave del diseño.



# HBase



## Características:

- Sus principales operaciones son las siguientes:

### Listar las tablas

- `list`

### Crear una tabla

- Crear una tabla de nombre 'testTable' y familia de columnas 'cf'

`create 'testTable','cf'`

### Insertar datos en una tabla

- Insertar en rowA, columna 'cf:columnName' con valor 'val1'

`put 'testTable', 'rowA',  
'cf:columnName','val1'`

### Obtener datos de una tabla

- Obtener rowA de la tabla 'testTable'

`get 'testTable','rowA'`

### Leer el contenido de una tabla (iterar sobre la tabla)

`scan 'testTable'`

### Borrar una tabla

`disable 'testTable'`

`drop 'testTable'`

# HBase



## Características:

- Toda la información de HBase es almacenada en forma de **array de bytes**.
- No ofrece un **lenguaje de acceso** como SQL, sino que ofrece un API, mediante Thrift y Avro o mediante un servicio HTTP RestFul.
- Su modelo de escalado se basa en **sharding**: trocear las tablas y almacenar cada fragmento por separado en HDFS.
- **Tolerante a fallos** mediante replicación a nivel de HDFS, alta disponibilidad, consistencia a nivel de operaciones sobre filas.
- Usa **memoria** para cachear bloques y de esta forma no requerir leer de HDFS para operaciones habituales.

# Phoenix



HBase, que podría considerarse la base de datos operacional sobre Hadoop, tiene como una de sus principales desventajas el **interfaz y lenguaje de acceso**, que lo hacen **difícil de integrar** con soluciones de terceros.

Phoenix es una capa superior a HBase que permite ofrecer un **interfaz SQL** mediante JDBC sobre esta.

**Traduce** las sentencias SQL en operaciones de escaneo en HBase con filtros y coprocesadores, por lo que la mayor parte del procesamiento es llevado a cabo dentro de Hbase, no en Phoenix como componente.

# Phoenix



Permite los siguientes comandos SQL:

SELECT  
UPSERT VALUES  
UPSERT SELECT  
DELETE  
DECLARE CURSOR  
OPEN CURSOR  
FETCH NEXT  
CLOSE  
CREATE TABLE

DROP TABLE  
CREATE FUNCTION  
DROP FUNCTION  
CREATE VIEW  
DROP VIEW  
CREATE SEQUENCE  
DROP SEQUENCE  
ALTER  
CREATE INDEX

DROP INDEX  
ALTER INDEX  
EXPLAIN  
UPDATE STATISTICS  
CREATE SCHEMA  
USE  
DROP SCHEMA  
GRANT  
REVOKE

# Sqoop



Es una herramienta diseñada para **transferir datos entre Hadoop y repositorios relacionales**, como bases de datos o sistemas mainframe.

Permite, por ejemplo, **importar** datos existentes en una tabla de Oracle y almacenarlos en HDFS en un fichero CSV para poder procesarlo y **exportar** su resultado para su explotación en Oracle.

Consiste en un **programa de línea de comandos** que traduce las órdenes en programas MapReduce que son lanzados en el clúster.

# Sqoop

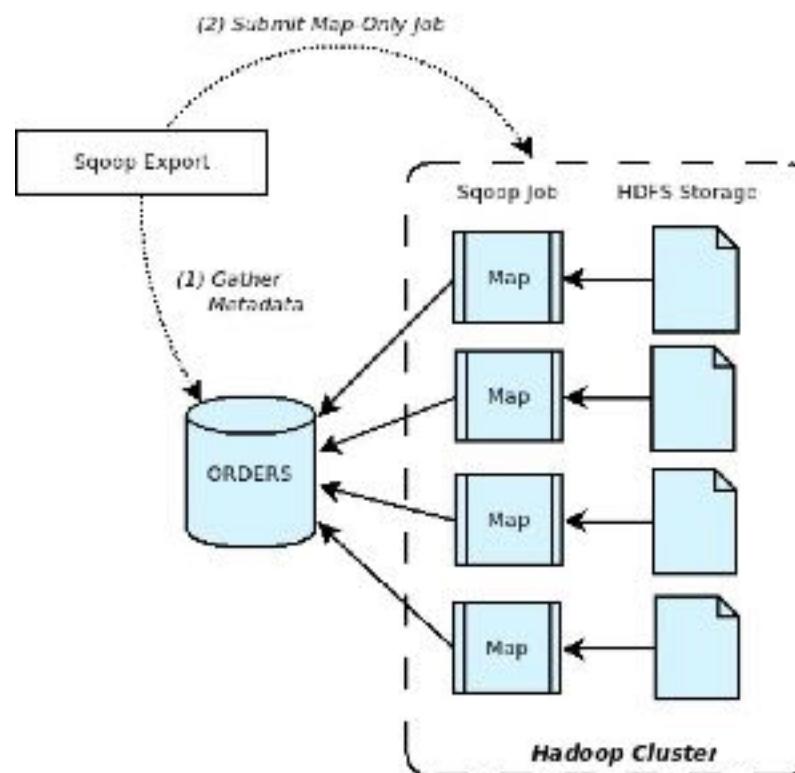
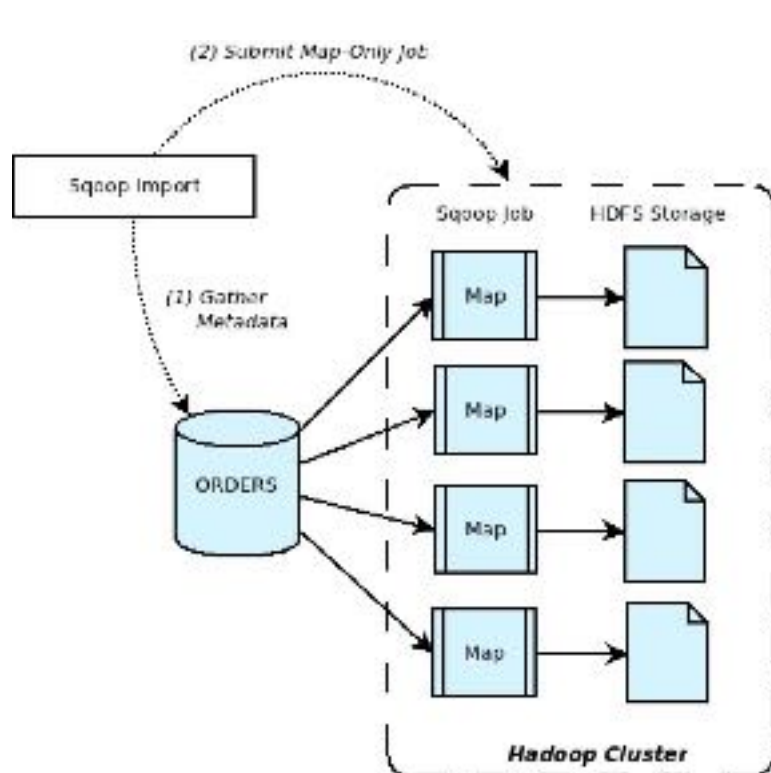


Algunos comandos:

- **sqoop-import:** importa una tabla de una base de datos relacional en HDFS.  
`sqoop import --connect jdbc:mysql://db.foo.com/corp  
--table EMPLOYEES --hive-import`
- **sqoop-export:** exporta un conjunto de ficheros de HDFS a una base de datos relacional.

```
sqoop export --connect jdbc:mysql://db.example.com/foo  
--table bar  
--export-dir /results/bar_data
```

# Sqoop



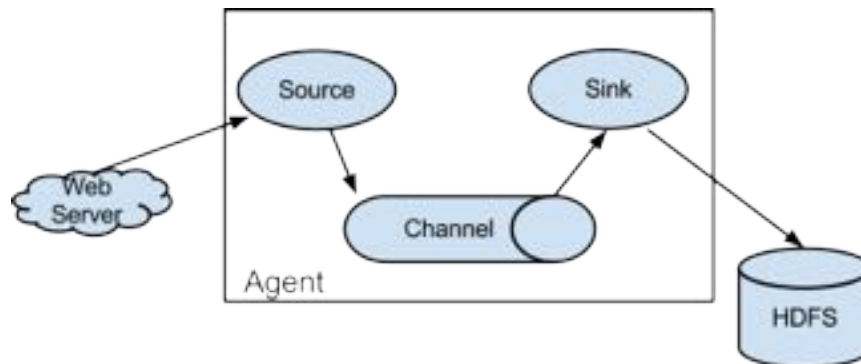
# Flume



Flume es un servicio distribuido y confiable para **recoger, agregar y mover datos** generados de forma continua y atómica, como es el caso de los logs.

Normalmente se usa para recoger y almacenar en Hadoop datos provenientes de sistemas de log, social media, IoT, emails, etc.

Su arquitectura es la siguiente:





# Flume

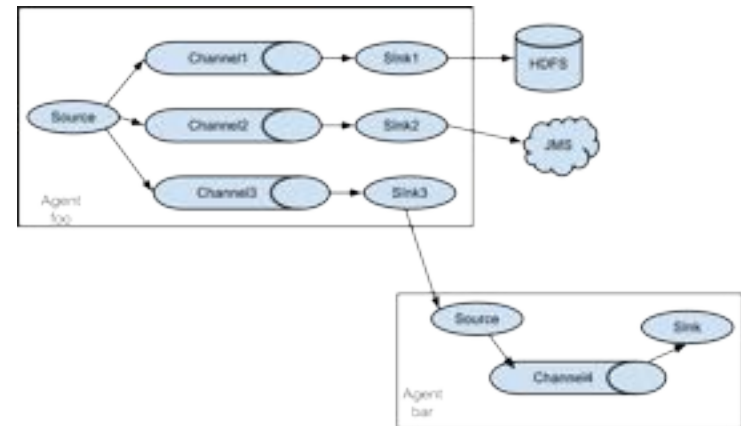


Sus componentes principales son:

- **Fuentes** / Sources: consume datos de una fuente de datos externa. Al recibir un evento/dato, el source lo almacena en uno o varios canales.
- **Canal**: almacena los datos hasta que otro agente de Flume lo consume.
- **Sumidero** / Sink: recoge los eventos del canal, los procesa y los envía a un repositorio externo, como puede ser HDFS.

Flume **garantiza la entrega** de todos los datos recibidos por las fuentes, y ofrece capacidad de recuperación ante caídas.

Los flujos de Flume pueden configurarse, pudiendo montar topologías complejas.



# Flume



Algunos tipos de fuente, canales y sumideros ofrecidos por Flume “de caja”:

- **Fuentes:** Avro, Thrift, Exec, Spooling directory, Taildir, Twitter, Kafka, NetCat, Syslog, HTTP o Custom.
- **Canales:** memoria, JDBC, Kafka, File o Custom.
- **Sumideros:** HDFS, Hive, Logger, Avro, Thrift, IRC, Hbase, Kafka, HTTP, File o Custom.

# Oozie



Facilita el lanzamiento de flujos de trabajo (workflows) cuando se cumplen determinadas condiciones. Ejemplo:

1. Un sistema X genera un fichero CSV con los datos de los pedidos del día anterior.
2. El fichero es enviado a Hadoop.
3. Al llegar el fichero, es necesario guardarlo para su historificación, descargar de la base de datos de CRM el maestro de clientes y generar un tablón resumido con las ventas por categorías, códigos postales, etc.
4. Al terminar el proceso, es necesario enviar un email al director de operaciones para notificarle de que ya puede acceder a la herramienta de informes para ver las ventas del día anterior.

\* Los pasos 3 y 4 serían controlados por Oozie.

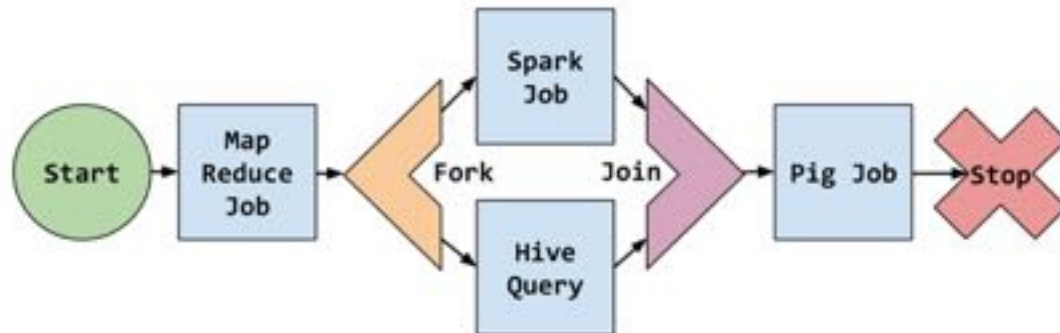
# Oozie



Oozie permite diseñar **flujos de trabajo**. Un flujo de trabajo está formado por:

- Condiciones de **inicio**: temporal o existencia de datos.
- Mecanismos de **control**: bifurcaciones, decisiones, uniones.
- **Acciones**: MapReduce, Hive, Sqoop, DistCp, Spark, Pig, SSH, email.
- Estados de **fin y error**.

El diseño de los workflows se hace mediante XML, y la gestión de flujos es realizada mediante una aplicación web que arranca Oozie.



# Zookeeper



Hadoop es una plataforma distribuida, lo que significa que existen N nodos/servidores que actúan como un único sistema.

Esto requiere que todos los nodos tengan la misma información sobre la configuración de la plataforma para poder actuar de la misma manera.

Zookeeper es un servicio centralizado que **mantiene y gestiona la información compartida** por todos los nodos de una forma confiable, tolerante a fallos, y altamente consistente.

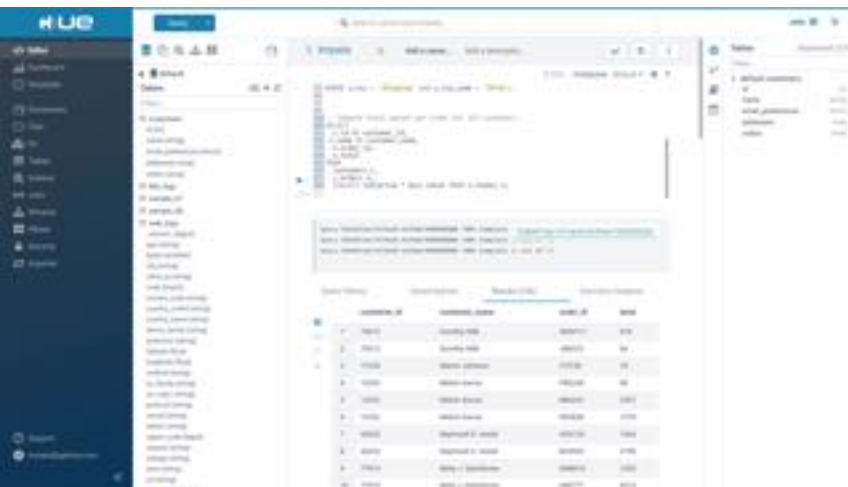
Es una **pieza técnica** que en principio no ofrece funcionalidad al usuario, pero es necesaria para la correcta ejecución de toda la plataforma.

# Hue



Hue es un **interfaz web** que permite trabajar con Hadoop de un modo sencillo para navegar por el espacio de nombres de HDFS, lanzar consultas a Hive/Impala, etc. visualizar las tareas en ejecución, etc.

Todas las distribuciones de Hadoop incluyen este componente, ya que **facilita enormemente el uso de Hadoop** por parte de usuarios de negocio o técnicos.



# Hue



Ofrece, entre otras, las siguientes funcionalidades y capacidades:

- Acceso **segurizado** mediante usuario y contraseña.
- **Editor SQL/HQL** para Hive e Impala, además de para cualquier base de datos SQL (MySQL, Oracle, Phoenix, etc.).
  - Visualización de tablas, bases de datos.
  - Ejecución de consultas y visualización del histórico de consultas.
  - Visualización de resultados.
- **Dashboards** para visualización de datos tomando como fuentes las consultas creadas en los editores.
- **Explorador de datos** que permite navegar por los sistemas de ficheros de HDFS, Amazon S3, así como las tablas y bases de datos en Hive o Hbase.
- Explorador de **tareas** y **Jobs** lanzados.
- Diseño/definición de **flujos** de trabajo (Oozie).

# Zeppelin



Hue es un interfaz sencillo para poder realizar consultas, navegar por los datos, etc. Es un interfaz útil, pero no ofrece toda la potencia que un Data Scientist requiere para su trabajo.

Apache Zeppelin permite cubrir ese gap, ya que es una **herramienta web** para **notebooks**, es decir, una herramienta web que permite **interactuar con los datos** mediante la creación de historias mediante **código** (Spark) o **consultas** (Hive), generando **gráficos**, **tablas**, elementos **interactivos**, que además pueden ser **publicados**, **compartidos** o desarrollados en modo **colaborativo**.

Zeppelin suele ser la primera herramienta de uso por parte de los Data Scientist, aunque los proyectos requieren fases posteriores para **industrializar** estos desarrollos.



# Zeppelin



Permite incorporar intérpretes para distintos lenguajes (R, Scala, etc.), incorporando “de caja” los siguientes:

- Lenguaje Python.
- API Spark.
- SQL / Hive / Impala.



# Ambari



Hadoop, como plataforma distribuida, tiene como una de sus principales dificultades la gestión o monitorización de la cantidad de servicios que se ejecutan en un cluster formado por distintos servidores físicos.

Ambari es una herramienta que permite:

- **Instalar** un cluster Hadoop mediante un wizard, seleccionando la topología del cluster, los servicios a instalar, su configuración, etc.
- **Gestionar los servicios** de un cluster: arrancar, parar o reconfigurar.
- **Gestionar la seguridad**: políticas de autorización, autenticación, etc.
- **Monitorizar** el cluster: recogida y visualización de métricas, definición y gestión de alarmas, etc.

La gestión de un cluster Hadoop sin una herramienta como Ambari es inmanejable.

# Cloudera Manager

Cloudera desarrolló una herramienta de gestión y administración de Hadoop que fue incorporada a sus distribuciones.

Cloudera Manager es **muy similar a Ambari** en cuanto a funcionalidades y manejo. De hecho, Cloudera Manager fue desarrollada 3 años antes que Ambari, aunque su principal limitación es que no está liberada como código libre, siendo una herramienta propietaria de Cloudera, a diferencia de Ambari.

# Trabajo de reflexión



Discutir en clase:

- ¿Qué componentes de Hadoop creéis que son indispensables para una organización que quiere utilizar Hadoop como Data Lake? Hacer un ranking de los 5 más necesarios.
- Para un caso de uso en el que se pretende obtener información de las base de datos de CRM y ERP y combinarlo con los logs de los servidores web para combinarlos y crear un cuadro de mando para que se explote por parte de la dirección, ¿qué componentes de Hadoop creéis que serán necesarios?
- ¿Puede Hive sustituir a la base de datos de un CRM? En caso de no poder, ¿por qué motivo/s?
- ¿Puede HDFS sustituir a un sistema de ficheros compartido? En caso de que no pueda, ¿por qué motivo/s?

# Práctica: analizando datos en Hadoop.

# Recordatorio

**\* TODOS DEBÉIS HABERME ENVIADO VUESTRA DIRECCIÓN IP QUE PODÉIS CONSULTAR EN [www.myip.com](http://www.myip.com) (donde dice “Your IP address is”).**

**Acceder a Hue:**

`https://ec2-*-*-*-*.eu-west-1.compute.amazonaws.com:8888/`

Usuario: hadoop

Password: Hadoop2021!

# Recordatorio

1. **Navegar por directorios y crear dos directorios:**  
`/user/hadoop/[vuestras_iniciales]` y  
`/user/hadoop/[vuestras_iniciales]/data`
2. **Descargar el fichero en vuestro PC:**  
`http://www.isenet.es/opendata/equipos\_laliga.csv`
3. **Subirlo al directorio** `/user/hadoop/`  
`/user/hadoop/[vuestras_iniciales]/data`
4. **Ejercicio: crear una carpeta** `/user/hadoop/`  
`/user/hadoop/[vuestras_iniciales]/backup` **y copiar fichero**  
`equipos_laliga.csv`

# Hive

1. **Ver las bases de datos creadas:** `show databases`
2. **Crear una base de datos:** `CREATE DATABASE eoi`
3. `show databases`
4. **Seleccionar la base de datos creada en el menú de la izquierda.**



# Hive

## 5. Crear tabla para meter los datos del fichero de La Liga:

```
create table equipos_laliga (  
    posicion int,  
    nombre string,  
    jugados int,  
    puntos int,  
    ganados int,  
    empatados int,  
    perdidos int,  
    goles_favor int,  
    goles_contra int,  
    goles_diferencia int)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
STORED AS TEXTFILE;
```

# Hive

6. `show tables`

**7. Ver todos los registros de la tabla:**

```
select * from equipos_laliga
```

**8. Cargar datos en la tabla desde el fichero:**

```
LOAD DATA INPATH '/user/hadoop/[vuestras_iniciales]/data  
/equipos_laliga.csv'  
OVERWRITE INTO TABLE equipos_laliga
```

9. `SELECT * FROM equipos_laliga`

**10. Quitar la primera línea:**

```
ALTER TABLE equipos_laliga SET TBLPROPERTIES ("skip.header.line.count"="1");
```

# Hive

**10. Contar registros:** `select count(*) from equipos_laliga`

**11. TOP 5 equipos con más puntos:**

```
select * from equipos_laliga order by puntos DESC limit 5
```

**12. ... que empiecen por S:**

```
select * from equipos_laliga  
where nombre like 'S%'  
order by puntos desc limit 5
```

# Hive

## 13. ... y que hayan jugado más de 100 partidos:

```
select * from equipos_laliga  
where nombre like 'S%' AND jugados > 100  
order by puntos desc limit 5
```

## 14. Crear un gráfico del resultado con Hue

# Hive

## Ejercicio:

Sobre el fichero `http://www.isenet.es/opendata/nombres.csv`

1. Crear una tabla en Hive para sus datos.
2. Cargar los datos del fichero.
3. Realizar las siguientes consultas:
  - Nº de nombres que aparecen en el fichero.
  - TOP 5 nombres por ocurrencias.
  - TOP 5 nombres de mujer por ocurrencias.
  - TOP 5 nombres de mujer que empiecen por P por ocurrencias.
  - Crear un gráfico con los 20 nombres más comunes.

# Bases de datos NoSQL

# Recordemos: modelo relacional

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx

ID	Fecha	Importe	Estado	...
1	01/01/2019	123,23	A	xxxx
2	12/03/2018	43,56	F	xxxx
3	08/04/2019	312,09	A	xxxx
4	19/01/2019	3,45	P	xxxx

ID	Nombre	Dirección	Responsable	...
1	Tienda 1	C/ ...	Luis	xxxx
2	Tienda 2	Avda. ...	María	xxxx
3	Tienda 3	Vía...	Pedro	xxxx
4	Tienda 4	C/	Esther	xxxx

# Recordemos: beneficios del modelo relacional

El modelo relacional y las bases de datos relacionales tienen características muy beneficiosas:

- **Estandariza** un modelo de representación de los datos que permite la **integración** de sistemas y aplicaciones.
- Ofrece un **lenguaje** general y muy potente para obtención y manipulación de datos (SQL).
- Garantiza un nivel de **transaccionalidad** muy elevado (ACID).

Sin embargo, también tiene unas limitaciones...



# Recordemos: ¿escalabilidad?

Clientes

ID	Nombre	Teléfono	CP	...
1	Luis	655433212	28001	xxxx
2	Lidia	665324543	41003	xxxx
3	Felipe	673232123	08001	xxxx
4	Elisa	699123234	45005	xxxx

Pedidos

ID	Fecha	Importe	Estado	...
1	01/01/2019	123,23	A	xxxx
2	12/03/2018	43,56	F	xxxx
3	08/04/2019	312,09	A	xxxx
4	19/01/2019	3,45	P	xxxx

Tiendas

ID	Nombre	Dirección	Responsable	...
1	Tienda 1	C/ ...	Luis	xxxx
2	Tienda 2	Avda. ...	María	xxxx
3	Tienda 3	Vía...	Pedro	xxxx
4	Tienda 4	C/	Esther	xxxx

**¿Qué pasaría si cada tabla tuviera millones o miles de millones de filas?**

# Recordemos: problemática

## Problemática del modelo relacional tradicional:

- Tiene dificultades para **escalar**.
- El modelo habitualmente no coincide con el modelo de las aplicaciones (**impedance mismatch**).
- La garantía **ACID** tiene dificultades para ejecutarse **sobre entornos distribuidos**.
- El esquema debe ser definido antes de introducir los datos (**schema-on-write**), y los cambios de esquema suelen ser costosos.

# NoSQL

**NoSQL** suele utilizarse para referirse a un conjunto de bases de datos que comparten ciertas características:

- **No** son **relacionales**.
- Su esquema es dinámico (**schema-less**).
- **Escalables** horizontalmente y basadas en un modelo de despliegue distribuido.
- Relajan los requisitos de **transaccionalidad** para poder escalar.



**NO**SQL  
Not Only

# NoSQL

No es la panacea, tiene **desventajas**:

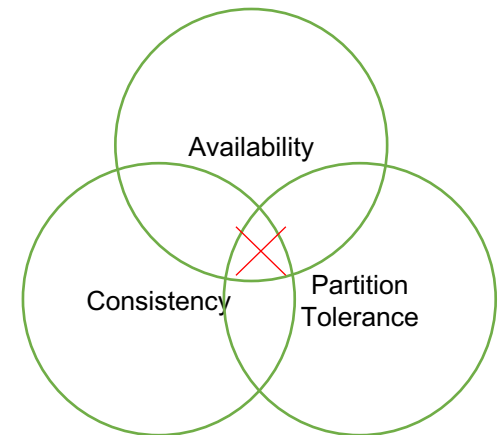
- En su mayoría no son transaccionales (no son **ACID**), ya que relajan los requisitos de fiabilidad para obtener escalabilidad.
- No tienen un lenguaje estándar de acceso como **SQL**.
- No suelen permitir enlazar distintos objetos (no hay **JOINS**).
- Al no haber una definición estándar del esquema en la propia base de datos, son las aplicaciones las que deben conocer el **modelo de datos** para su uso.

Sus principales **beneficios** suelen ser:

- Los datos que se incorporan **no necesitan un esquema prefijado**, lo que permite una mejor adaptación a los cambios (que siempre hay).
- Almacena y procesa **cualquier tipo** de información.
- Bajo **coste** (hardware commodity y código opensource).

# Teorema CAP

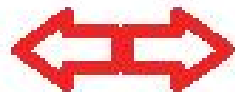
- **Consistency:** todos los clientes ven los mismos datos siempre.
- **Availability:** no hay periodos muertos sin capacidad para leer o escribir.
- **Partition Tolerance:** el sistema funciona aunque haya una desconexión de parte de los nodos de la red.



# Consistencia de datos

**RDMS**  
**ACID**

**A**tomic  
**C**onsistent  
**I**solated  
**D**urable



**NoSQL**  
**CRUD**

**B**asically **A**vailable  
**S**oft state  
**E**ventually consistent

# Tipos de bases de datos NoSQL

NoSQL es un término amplio donde se concentran distintos tipos de bases de datos.

Los principales tipos son:

- Clave valor:
  - Columnares.
  - Orientadas a documento.
  - Diccionarios simples.
- Orientadas a grafos.



# Orientadas a documento

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
```



# MongoDB

- Base de datos NoSQL opensource con licencia GNU AGPL v3.0.
- Implementado en C++.
- Desarrollado por la empresa MongoDB Inc. (antes 10gen), que además ofrece una versión Enterprise, así como servicios de consultoría, formación y soporte:
  - Compañía con casi 700 empleados y +150M€ de inversión.
  - Sede principal en New York.

# NoSQL & MongoDB

- MongoDB es una base de datos NoSQL de tipo documental y CP en cuanto a su posicionamiento en el teorema CAP.

	Type	Examples
Increasing Data Complexity ↓	Key-Value Store	 
	Wide Column Store	 
	Document Store	 
	Graph Store	 



# NoSQL & MongoDB

- Almacena documentos JSON:

```
{  
  name : "Juan Pérez Romay",  
  title : "Big Data Architect",  
  phone : "+343.21.112.1213",  
  age : 40,  
  location : "Madrid, Spain",  
  emails : ["j.perez@indizen.com", "jperezro@gmail.com"]  
}
```

- Permite operaciones de alta, baja, modificación y consulta sobre los documentos almacenados (acceso por documento o por atributo).

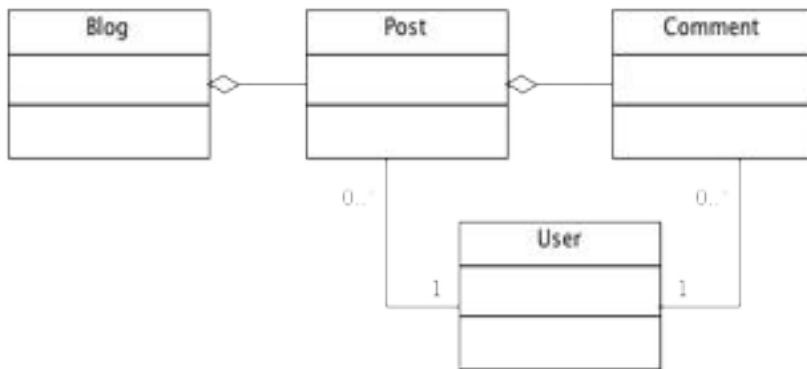
# NoSQL & MongoDB

- Posicionamiento en trade-off funcionalidad – rendimiento:



# MongoDB. Modelado

- **Schema-less:** permite introducir datos sin realizar definición previa ni mantener una estructura concreta
  - { nombre : “Juan”, altura : 180, hijos: [“Luis”, “Ana”] }
  - { nombre : “Belén”, estado\_civil : “soltera” }
- **Minimiza el problema de impedancia** al trabajar con modelos orientados a objetos con persistencia E-R:

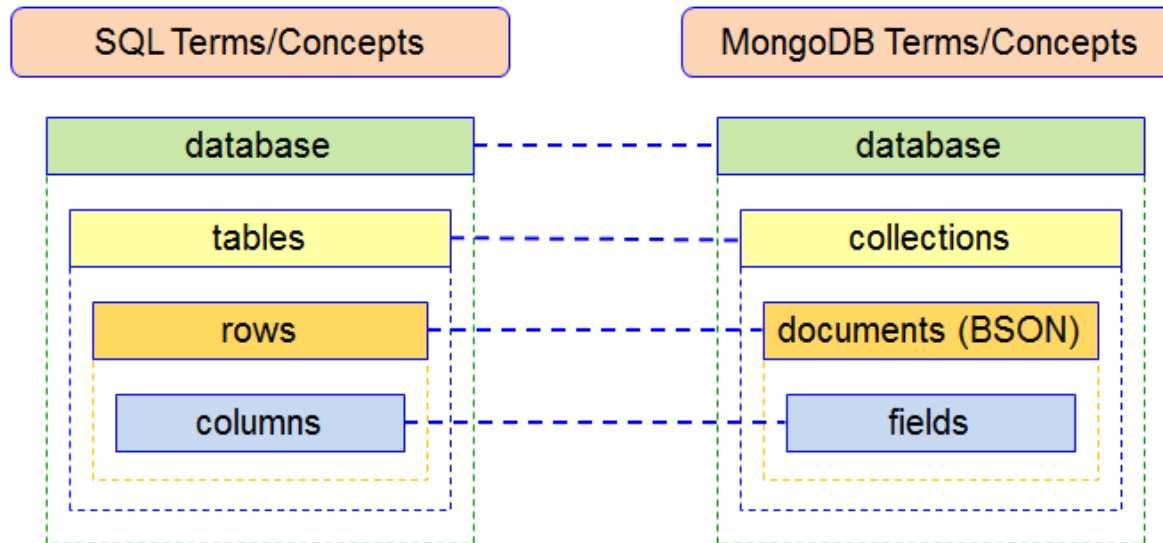


```

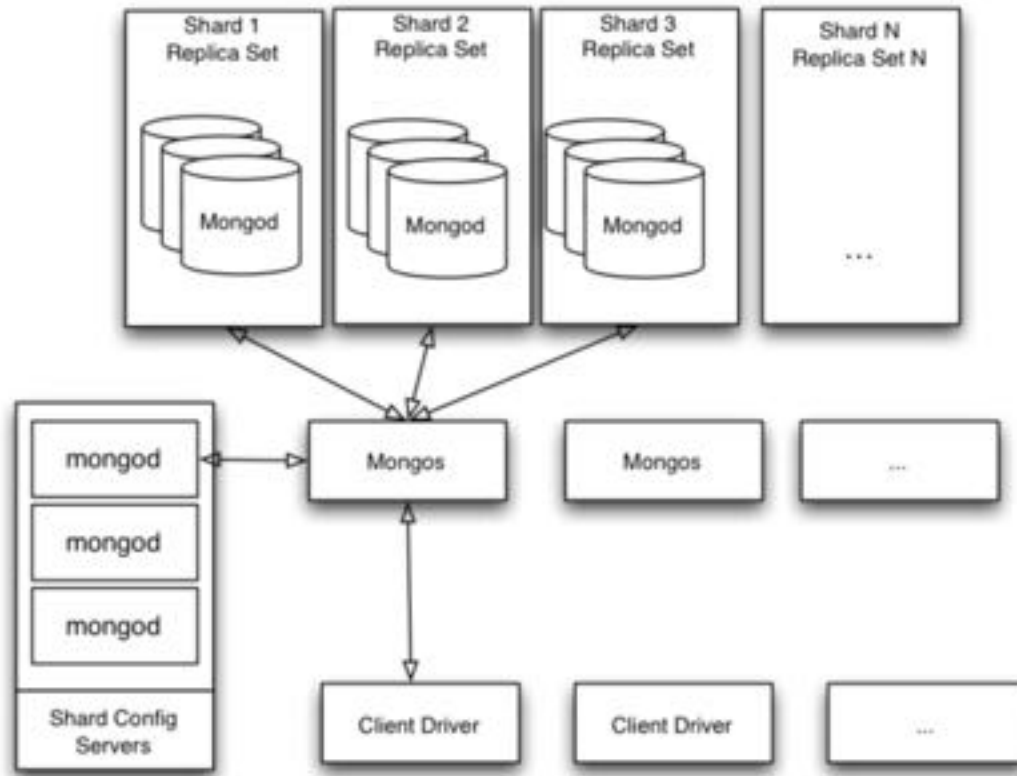
{ "_id" : XXX, "title" : "Technology Blog",
  "posts" : [
    {
      "_id" : YYY, "title" : "Tech Post #1",
      "content" : "Blah blah blah...", "author_id" : ZZZ,
      "comments" : [
        {
          "_id" : NNN, "title" : "I disagree!",
          "content" : "You're not very smart", "author_id" : AAA
        },
        {
          "_id" : MMM, "title" : "I agree!",
          "content" : "You're very smart",
          "author_id" : BBB
        }
      ]
    }
  ]
}
  
```

# MongoDB. Organización

- MongoDB se organiza en bases de datos, colecciones y documentos:



# MongoDB. Arquitectura



# MongoDB. Confiabilidad

- MongoDB garantiza **atomicidad** en las actualizaciones sobre un documento.
- No es posible **actualizar múltiples documentos** en una única operación atómica.
- Proporciona **consistencia eventual**. Las lecturas se realizan sobre el nodo primario.
- El nivel de garantía de escritura (**write-concern**) se configura para cada operación.

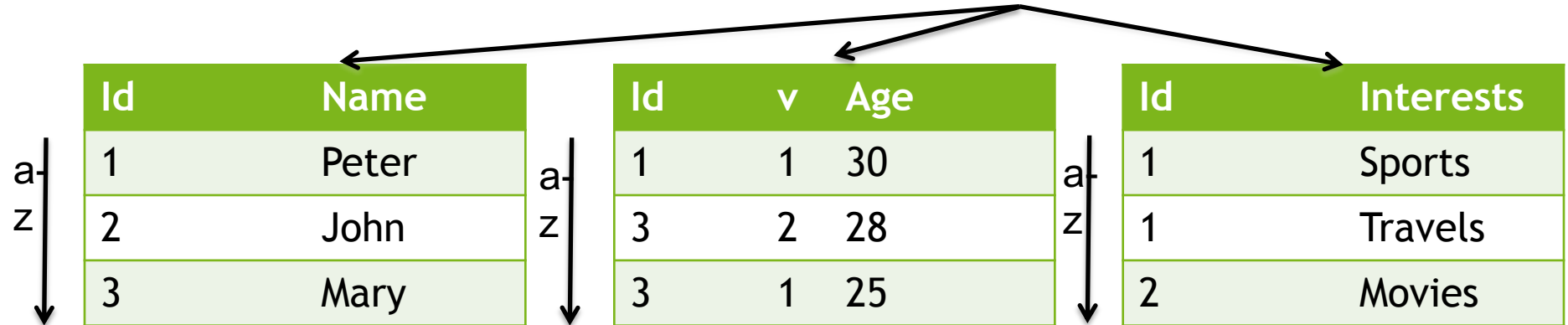


# MongoDB. Escalabilidad

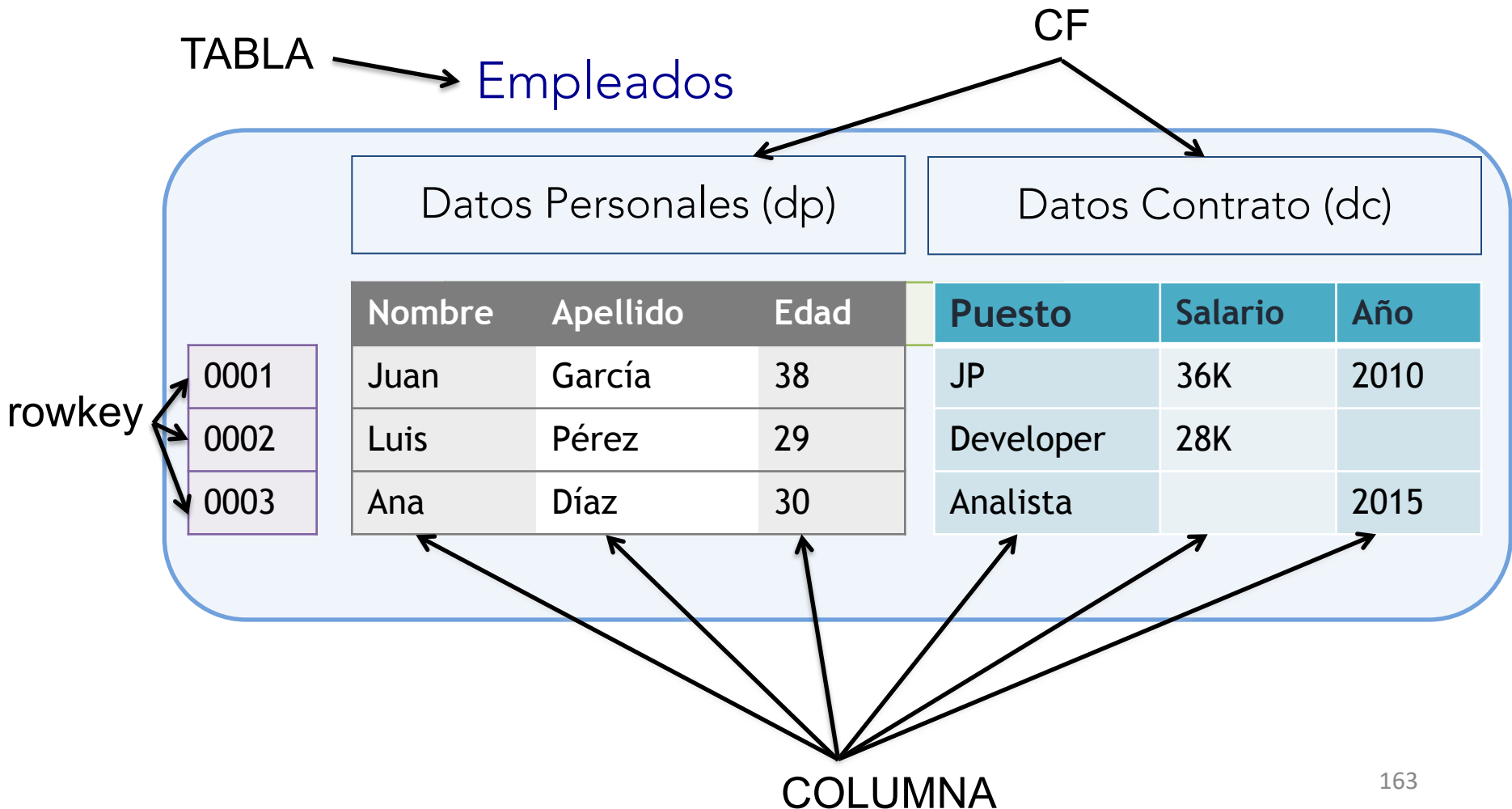
- La escalabilidad se consigue mediante **sharding**: dividir los datos (de una colección) en chunks, que son procesados por nodos independientes.
- El mecanismo de sharding **transparente** para las aplicaciones, incluso al añadir nodos al cluster (rebalanceo automático).
- La división de los datos se realiza definiendo una **shard-key**:
  - Es un atributo de los documentos (\*), no tiene por qué ser el `_id`.
  - Su elección es importante ya que es muy costoso cambiarlo.
  - Debe tener alta cardinalidad y aleatoriedad para garantizar un reparto de carga equilibrado entre nodos.
  - Tipos de sharding: por rango, hash, o definido por el usuario.

# Bases de datos NoSQL orientadas a columnas

Id	Name	V	Age	Interests
1	Peter	1	30	Sports, Travels
2	John	1	(null)	Movies
3	Mary	1	25	(null)
3	Mary	2	28	(null)

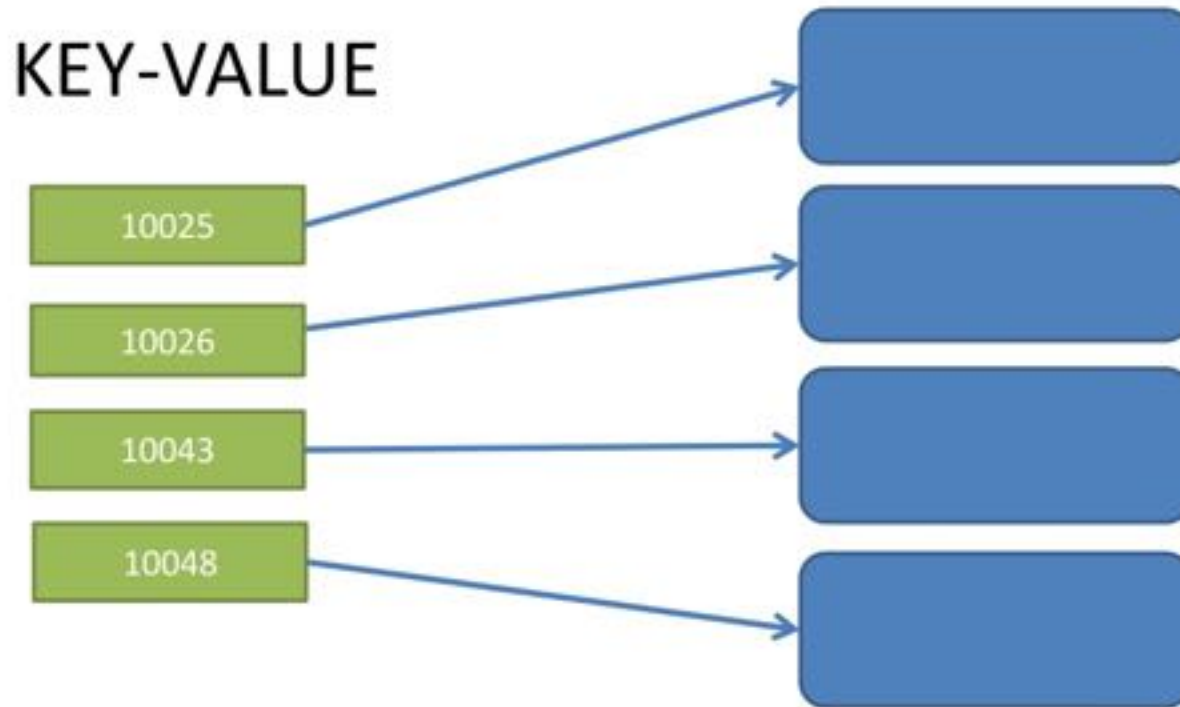


# Bases de datos NoSQL orientadas a columnas



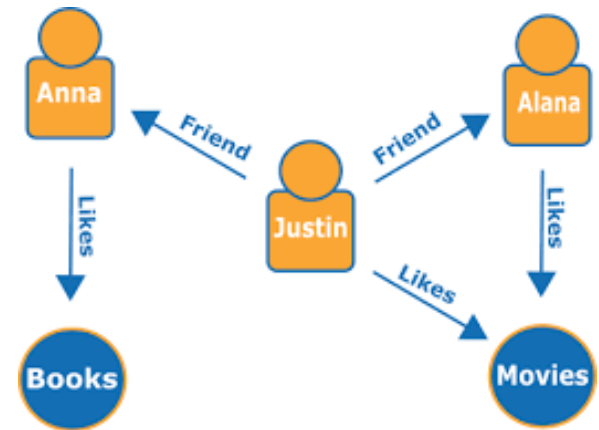
# Bases de datos NoSQL

## Clave-valor



# Bases de datos NoSQL orientadas a grafos

- Modelo de datos: nodos y relaciones
- Características:
  - Modelado directo en forma de grafo.
  - Excelente rendimiento en datos interconectados y no tabulares.
  - Transaccionalidad en relaciones.



# Trabajo de reflexión

Discutir en clase:

- Si las bases de datos NoSQL son mucho más flexibles y escalables que las bases de datos tradicionales, ¿por qué no se deberían utilizar en estos casos de uso? (Si crees que se deberían utilizarse siempre, argumentar el motivo).
- En escenarios de alta volumetría y variedad de datos, ¿cuándo se debería utilizar una base de datos NoSQL y cuándo Apache Hadoop?

# Práctica: trabajando con MongoDB.

# MongoDB online

1. **Acceder a la siguiente URL:** <https://mongoplayground.net/>
2. **En la izquierda, construir el JSON que representa los datos de la siguiente tabla (4 primeras filas):**

Nombre de la empresa	Facturación (M€)	Empleados	Beneficio (M€)
COMPAÑIA ESPAÑOLA DE PETROLEOS SA	23.857	700	833
MERCADONA SA	23.343	80.000	622
NATURGY ENERGY GROUP SA	23.035	3.000	1.796
REPSOL PETROLEO SA	21.270	2.500	502
ENDESA SA	19.258	3.000	180
INDUSTRIA DE DISEÑO TEXTIL SA	18.261	60.000	10.418



# MongoDB online

```
[
  {
    "nombre": "Compañía española de petróleos",
    "facturacion": 23857,
    "empleados": 700,
    "beneficio": 833
  },
  {
    "nombre": "Mercadona SA",
    "facturacion": 23343,
    "empleados": 80000,
    "beneficio": 622
  },
  {
    "nombre": "Naturgy Eneergy Group SA",
    "facturacion": 23035,
    "empleados": 3000,
    "beneficio": 1796
  },
  {
    "nombre": "Repsol Petróleo SA",
    "facturacion": 21270,
    "empleados": 2500,
    "beneficio": 502
  }
]
```

# MongoDB online

## 1. Consultar todos los documentos:

```
db.collection.find()
```

## 2. Buscar los documentos que tienen un atributo “nombre” con valor “Mercadona SA”:

```
db.collection.find({  
  "nombre": "Mercadona SA"  
})
```

## 3. Buscar los documentos que tienen un atributo “facturación” mayor que 23000:

```
db.collection.find({  
  "facturacion": {  
    $gt: 23000  
  }  
})
```

# Apache Spark

# Índice

- \_01 Introducción
- \_02 Historia
- \_03 Arquitectura
- \_04 Ejecución
- \_05 Beneficios y dificultades
- \_06 Spark Core
- \_07 SparkSQL
- \_08 GraphX
- \_09 MLLib
- \_10 Streaming
- \_11 Apache Storm y Apache Flink

# Apache Spark



Es una **plataforma opensource** de **computación** y un conjunto de **librerías** para **procesamiento** en **paralelo** de **datos** sobre **sistemas distribuidos**.

# Apache Spark



- Es el proyecto de tecnologías Big Data con mayor número de **contribuidores**.
- Es el **estándar de facto** para procesamiento de datos en Big Data.
- Ofrece una **plataforma unificada y de propósito general** para procesamiento Big Data, tanto batch como streaming, ofreciendo funcionalidades de carga de datos, exploración, explotación, transformaciones, etc.
- No ofrece funcionalidades de **persistencia durable**, aunque permite utilizar HDFS, S3 u otros sistemas de almacenamiento.
- Garantiza la **escalabilidad** en operaciones paralelizables.
- Ofrece **tolerancia a fallos**.

# Historia



**2009**

## **AMPLab @ UC Berkeley**

Primeros desarrollos dentro de un proyecto del Proyecto Spark Research por Matei Zaharia.



**2013**

## **Apache Software Foundation**

Donado a la Apache Software Foundation



**2014**

## **Databricks**

Matei Zaharia y otras 6 personas fundan Databricks, una compañía que ofrece soporte y un entorno cloud para Spark.

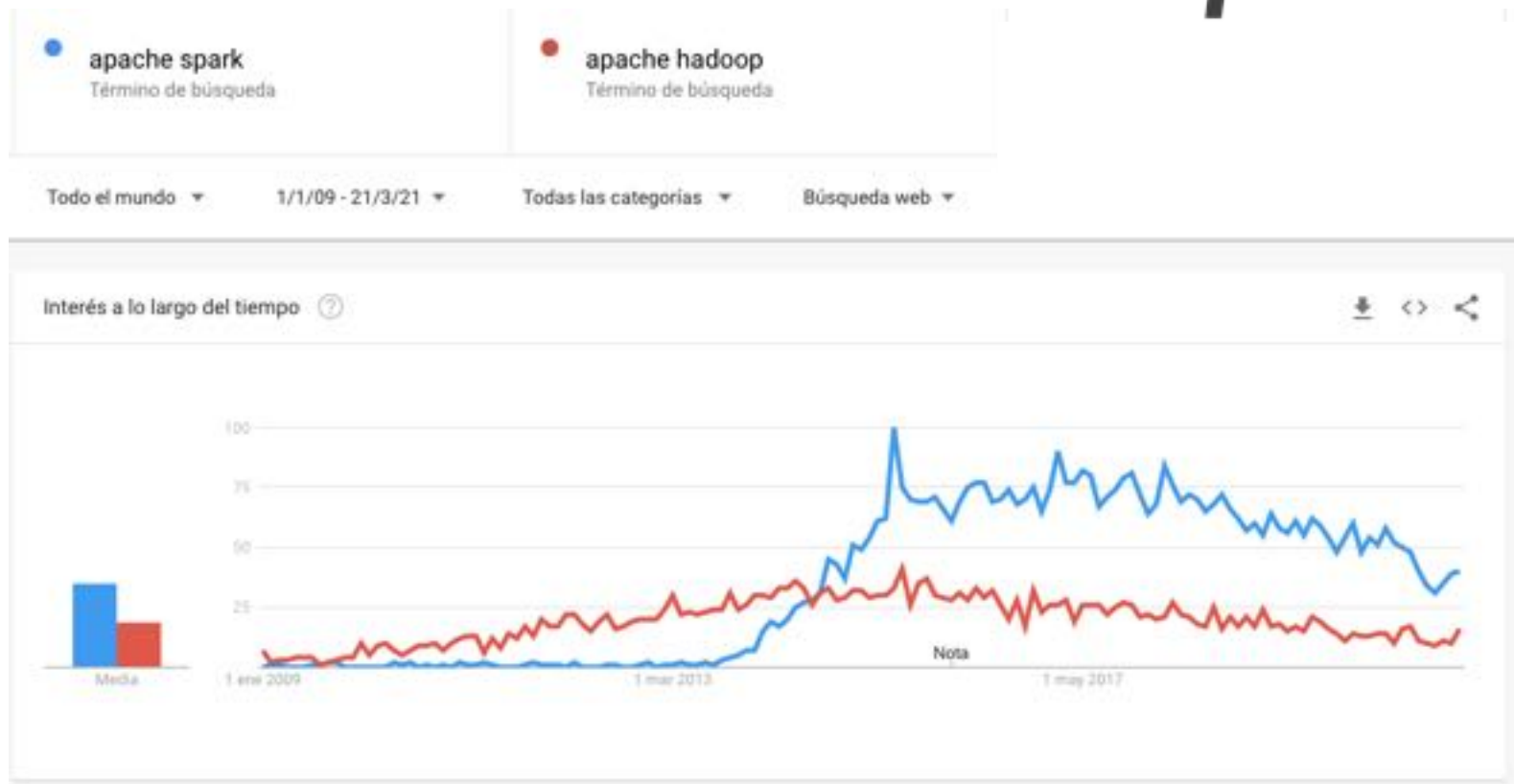


**2014**

## **Apache Top-level**

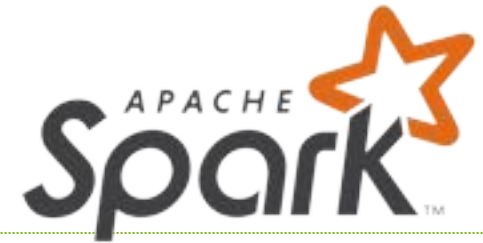
En febrero de 2014 se convirtió a Proyecto Top-Level de Apache.  
Se libera la version 1.0

# Evolución





# Componentes



Spark SQL

Streaming

MLLib

GraphX

SparkR

Spark core

Scala

Java

Python

R

SQL

Stand-alone

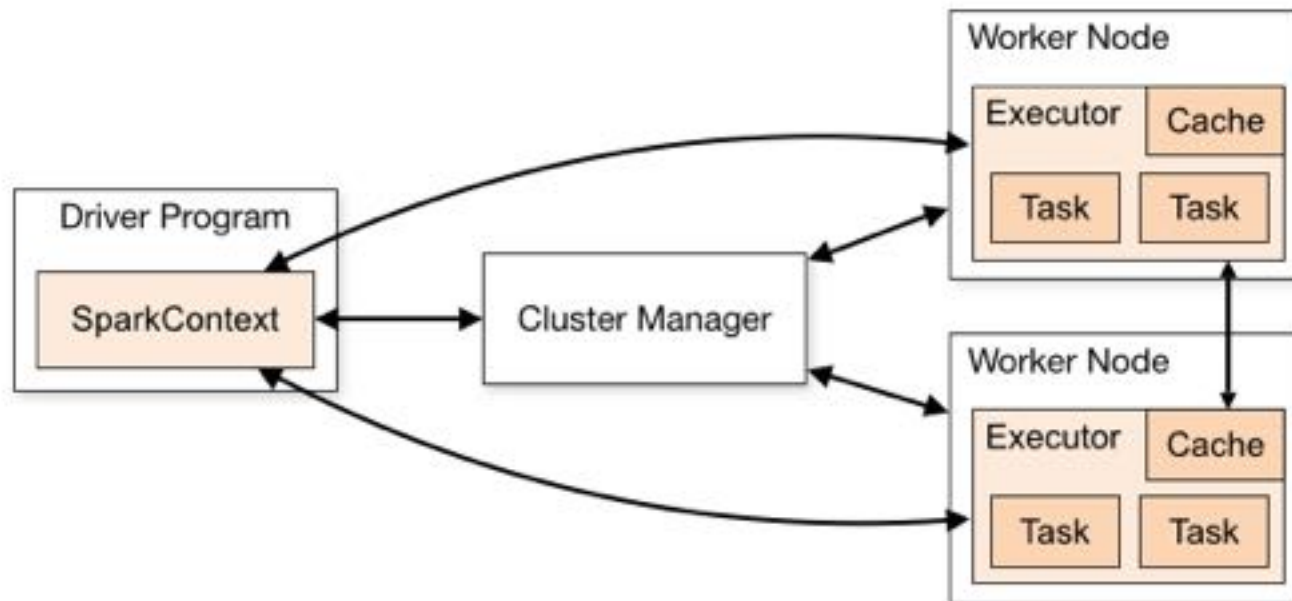
YARN

Mesos

# Arquitectura



Una aplicación Spark se compone de un proceso Driver y de un conjunto de ejecutores.



# Arquitectura



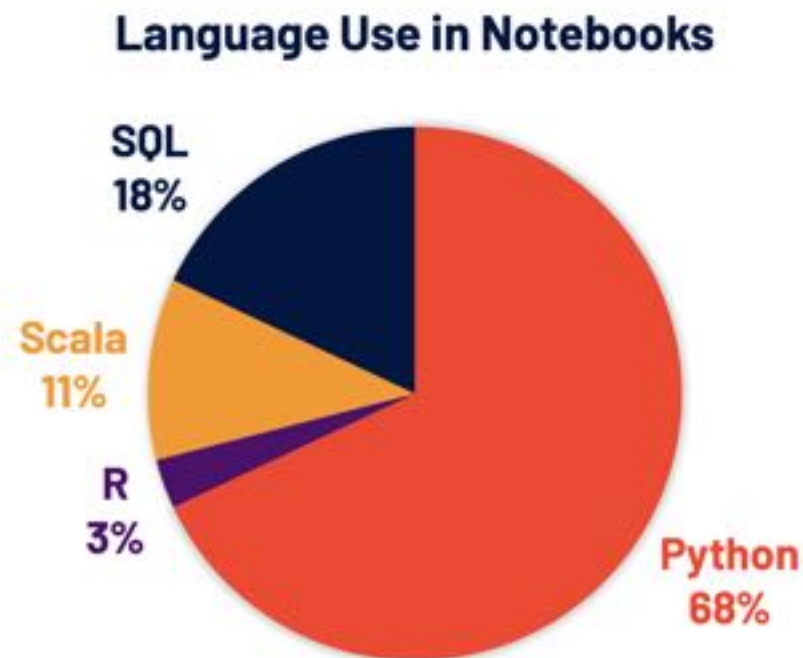
- El **Driver** ejecuta la función main (función de arranque) y se encarga de tres tareas:
  - Mantener la información/**estado** de la aplicación.
  - Responder a las **entradas de datos del usuario**.
  - Analizar, **distribuir** y **planificar** el trabajo de los ejecutores.
- Los **ejecutores** se ocupan de la ejecución de las tareas / trabajos que el driver les asigna y de informar al driver del estado de estos trabajos.
- El **gestor del cluster** controla el entorno de ejecución donde en el que se ejecutan las aplicaciones, disponibilizando recursos para los ejecutores, mediante su conexión con el sistema de procesamiento, que puede ser YARN, Mesos o en modos stand-alone y local.

# Lenguajes



Spark ofrece APIs en los siguientes lenguajes:

- **Scala:** Spark está escrito principalmente en Scala, siendo éste el lenguaje “por defecto”.
- Java.
- Python.
- SQL.
- R.



# Ejecución



Mediante el ejecutable spark-submit se pueden lanzar programas Spark:

```
spark-submit --help
Usage: spark-submit [options] <app jar | python file> [app arguments]
[...]
Options:
  --master MASTER_URL      spark://host:port, mesos://host:port, yarn, or local.
  --deploy-mode DEPLOY_MODE  whether to launch the driver program locally ("client") or on one of
                             the worker machines inside the cluster ("cluster") (Default: client).
  --class CLASS_NAME        Your application's main class (for Java / scala apps).
  --name NAME               A name of your application.
  --jars JARS               Comma-separated list of local jars to include on the driver and executor
                             classpaths.
  --packages                Comma-separated list of maven coordinates of jars to include on the driver
                             and executor classpaths. [...] groupId:artifactId:version.
  --exclude-packages, --repositories [...]
  --py-files PY_FILES       Comma-separated list of .zip, .egg, or .py files to place on the PYTHONPATH
                             for Python apps.
  --files FILES             Comma-separated list of files to be placed in the working directory of each
                             executor.
```

# Ejecución



<code>--conf PROP=VALUE</code>	Arbitrary spark configuration property.
<code>--properties-file FILE</code>	Path to a file from which to load extra properties. [ ... ]
<code>--driver-memory MEM</code>	Memory for driver (e.g. 1000M, 2G) (Default: 1024M).
<code>--driver-java-options</code>	Extra Java options to pass to the driver.
<code>--driver-library-path</code>	Extra library path entries to pass to the driver.
<code>--driver-class-path</code>	Extra class path entries to pass to the driver.
<code>--executor-memory MEM</code>	Memory per executor (e.g. 1000M, 2G) (Default: 1G).

[ ... ]

Spark standalone with cluster deploy mode only:

<code>--driver-cores NUM</code>	Cores for driver (Default: 1).
---------------------------------	--------------------------------

Spark standalone or Mesos with cluster deploy mode only:

<code>--supervise</code>	If given, restarts the driver on failure.
<code>--kill SUBMISSION_ID</code>	If given, kills the driver specified.
<code>--status SUBMISSION_ID</code>	If given, requests the status of the driver specified.

# Ejecución



Spark standalone and Mesos only:

`--total-executor-cores NUM` Total cores for all executors.

Spark standalone and YARN only:

`--executor-cores NUM` Number of cores per executor. (Default: 1 in YARN mode, or all available cores on the worker in standalone mode)

YARN-only:

`--driver-cores NUM` Number of cores used by the driver, only in cluster mode (Default: 1).

`--queue QUEUE_NAME` The YARN queue to submit to (Default: "default").

`--num-executors NUM` Number of executors to launch (Default: 2).

`--archives ARCHIVES` Comma separated list of archives to be extracted into the working directory of each executor.

[ ... ]

# Ejecución



## Ejemplos:

Ejecución local del programa SparkPi con 8 cores asignados:

```
spark-submit
  --class org.apache.spark.examples.SparkPi
  --master local[8]
  /path/to/examples.jar
  100
```



# Ejecución



## Ejemplos:

Ejecución sobre YARN del programa SparkPi:

```
export HADOOP_CONF_DIR=XXX
spark-submit
  --class org.apache.spark.examples.SparkPi
  --master yarn
  --deploy-mode cluster
  --executor-memory 20G
  --num-executors 50
  /path/to/examples.jar
  1000
```

# Ejecución



## Ejemplos:

### Ejecución sobre Mesos de SparkPi:

```
spark-submit
--class org.apache.spark.examples.SparkPi
--master mesos://207.184.161.138:7077
--deploy-mode cluster
--supervise
--executor-memory 20G
--total-executor-cores 100
http://path/to/examples.jar
1000
```

# Beneficios

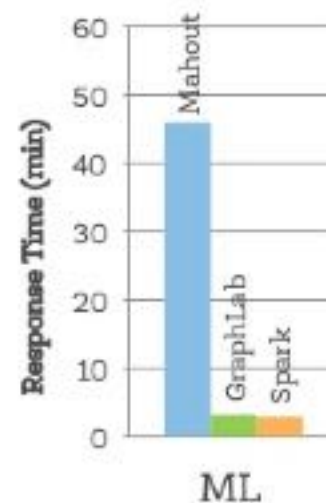
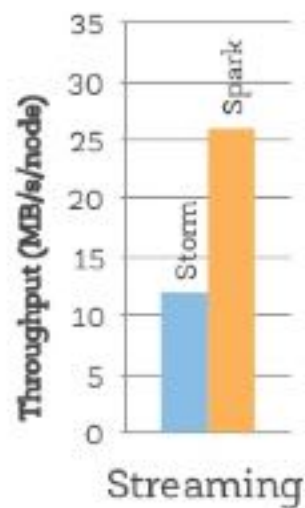
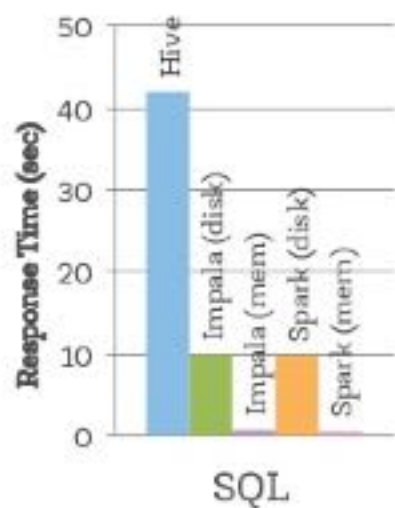


- Utiliza memoria para persistencia efímera de datos, lo que incrementa en órdenes de magnitud su **velocidad** en procesos iterativos frente a frameworks como MapReduce.
- Permite utilizar un mismo **paradigma** y modelo para distintos tipos de procesamiento (batch y streaming).
- Se **integra** con distintos sistemas de persistencia y motores de gestión de datos como Hive, HDFS, etc.
- Ofrece un **API muy rica**: procesamiento de datos, transformaciones, machine learning, grafos, etc.
- Frente a otros paradigmas como MapReduce requiere desarrollar una **cantidad de código muy inferior**.
- +1000 **desarrolladores** contribuyendo en el proyecto.

# Benchmarking



## Performance vs Specialized Systems



# Dificultades



- Está orientado a **perfiles muy técnicos**, con conocimientos sólidos de programación.
- Es **difícil de optimizar** en producción.
- Aunque es una plataforma con un nivel de madurez alto está sufriendo **bastantes cambios** lo que dificulta el manejo de distintas versiones, nuevas funcionalidades, etc.

# Spark Core



- **Spark Core** es la base de Spark. Ofrece funcionalidades para gestión de tareas distribuidas, programación, orquestación, así como funcionalidades básicas de entrada y salida de datos mediante un API en varios lenguajes.
- **Resilient Distributed Dataset (RDD)** es la principal abstracción de datos de Spark Core: es una representación simbólica de una colección elementos con los que se puede trabajar en paralelo.
- Los RDDs son **inmutables**, es decir, no se pueden modificar sus elementos, sino que se crean nuevas copias si es necesario alguna modificación.

# Operaciones



## Existen dos tipos de operaciones sobre RDDs:

- **Transformaciones:** permiten transformar un RDD para generar otro. Las transformaciones no se ejecutan de forma inmediata, sino que se van concatenando para su ejecución lo más tarde posible (al ejecutarse acciones). Por ejemplo, 3 filtrados seguidos de un RDD pueden unirse.
- **Acciones:** persisten o re-distribuyen un RDD. Se ejecutan de forma inmediata. Hay 4 tipos:
  - Ver datos en consola.
  - Redistribuir los datos en los nodos de computación.
  - Recoger datos para mapearlos en objetos nativos cada lenguaje.
  - Escribir en repositorios de datos.

# Operaciones



## Principales transformaciones:

- **map(func):** aplica una transformación (func) a todos los elementos del RDD generando un nuevo RDD.
- **flatMap(func):** aplica una transformación a todos los elementos del RDD pudiendo resultar más de un elemento por cada elemento original del RDD.
- **filter(func):** genera un nuevo RDD resultante de aplicar el filtro func a todos los elementos del RDD original.
- **distinct():** genera un nuevo RDD eliminando duplicados del original.
- **union(rdd):** genera un nuevo RDD que es la unión del original con el enviado como parámetro.



# Operaciones



## Principales acciones:

- **count():** devuelve el número de elementos del RDD.
- **reduce(func):** aplica una función de agregación sobre los elementos del RDD.
- **take(n):** devuelve los n primeros elementos del RDD.
- **collect():** devuelve todos los elementos del RDD (al driver).
- **saveAsTextFile(path):** escribe los elementos de un RDD en un repositorio (filesystem, HDFS, S3, etc.)

# SparkSQL



Es un componente sobre Spark Core para **procesamiento de datos estructurados o semiestructurados**.

Ofrece:

- Un nivel de abstracción de datos estructurados o semi-estructurados superior a RDD: **DataSet** y **DataFrame**.
- Funcionalidades para **acceder o manejar** este tipo de información.
- Soporte a **lenguaje SQL** así como interfaces de acceso desde herramientas relacionales.

Las principales ventajas de SparkSQL es su facilidad de uso y productividad en la construcción de procesos de transformación o consultas.

# DataFrame y DataSet



**DataFrame** es una abstracción que representa tablas de datos (filas y columnas), sobre las que Spark ofrece un API para su manejo y transformación, así como la capacidad de uso de SQL como lenguaje de acceso.

**DataSet** es una abstracción similar a DataFrame aunque con la característica de que las filas tienen unos atributos de tipo establecido (las columnas tienen un tipo previamente definido).

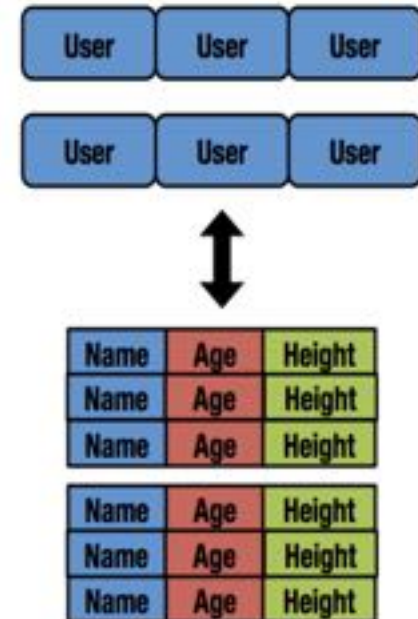
Al igual que RDD, los DataFrames y DataSets son inmutables.

# DataFrame vs RDD

Con RDD los objetos son manejados por los programas Spark de forma opaca, sin una estructura concreta (que sólo conoce el desarrollador).

Con DataFrames, se define un esquema y se permite el uso de lenguaje estándar para el acceso o manejo de los datos.

DataFrame = RDD + esquema.



# Fuentes de datos



SparkSQL permite trabajar con las siguientes fuentes de datos:

## Built-In



## External



# Ejemplo



<https://github.com/databricks/Spark-The-Definitive-Guide/blob/master/data/flight-data/csv/2015-summary.csv>

```
DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count
United States,Romania,15
United States,Croatia,1
United States,Ireland,344
Egypt,United States,15
United States,India,62
United States,Singapore,1
United States,Grenada,62
Costa Rica,United States,588
...
```

# Ejemplo



```
val flightData2015 = spark.read
    .option("inferSchema", "true").option("header", "true")
    .csv("/user/hadoop/eoi_data/2015-summary.csv")

flightData2015.show(5)

flightData2015.createOrReplaceTempView("flight_data_2015")

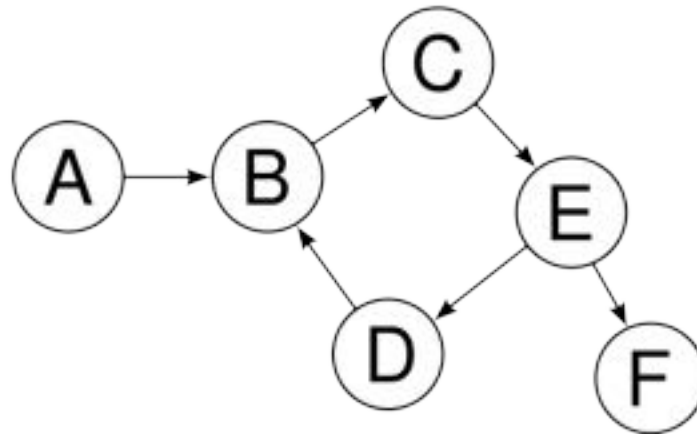
val sqlWay = spark.sql("""SELECT DEST_COUNTRY_NAME,
count(1)
FROM flight_data_2015 GROUP BY DEST_COUNTRY_NAME""")
sqlWay.show(5)
val dataFrameWay =
flightData2015.groupBy("DEST_COUNTRY_NAME").count()
```

# GraphX



Componente de Spark que ofrece un API para manejo de datos como grafos así como funcionalidades de procesamiento de grafos en paralelo.

Un grafo (dirigido) es una estructura que contiene vértices y aristas.





# GraphX



El modelo de abstracción extiende RDD con un **Resilient Distributed Property Graph**: un grafo dirigido, formado por nodos y aristas, con propiedades en ambos elementos.

Las funcionalidades que ofrece son subpragh, joinVertices, mapReduceTriplets, etc.

# MLLib



Es una librería de alto nivel de Spark que ofrece funcionalidades de **Machine Learning**:

- **Algoritmos**: clasificación, clustering, regresión y filtros colaborativos.
- Obtención de **features**: extracción, transformación, reducción de dimensionalidad y selección.
- **Generación de modelos**: entrenamiento, evaluación, aplicación.
- **Persistencia**: almacenamiento y carga de modelos.
- **Utilidades**: funciones de álgebra lineal, estadística, etc.

# Streaming

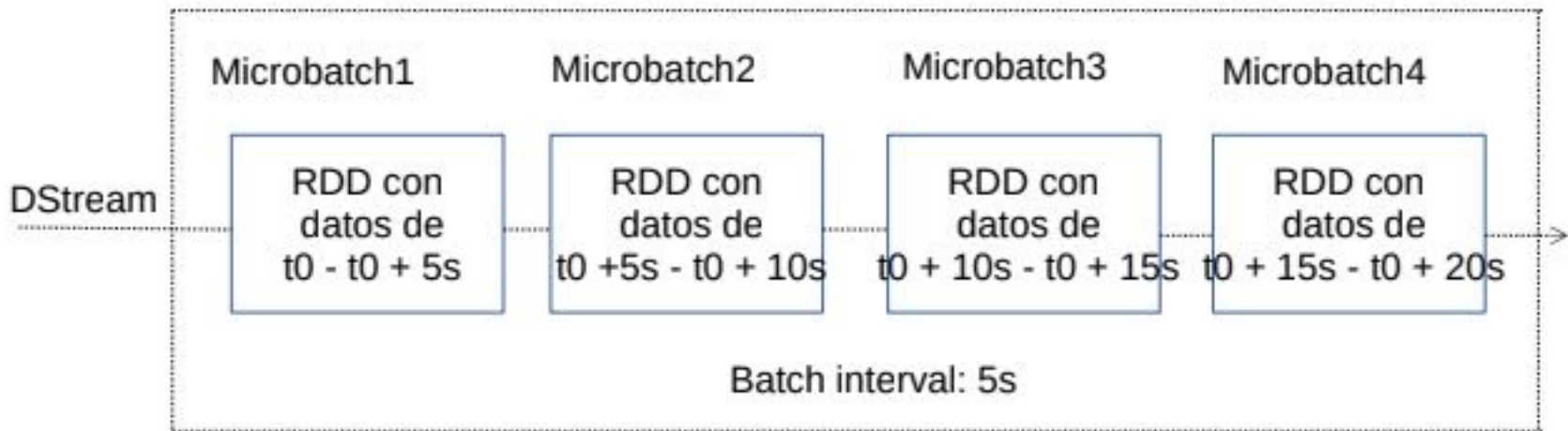


Es el componente de Spark que permite procesar en tiempo near-realtime flujos de datos.

Realiza procesamiento de los datos en **microbatches**, es decir, toma datos durante un periodo de tiempo (batch interval) y lanza un procesamiento en paralelo sobre los datos recogidos.

Se basa en **DStream**, una abstracción construida sobre RDDs que representa un flujo de datos continuo obtenido a partir del stream de entrada o de aplicar transformaciones sobre otros Dstreams.

# Modelo de ejecución



# Transformaciones



Ofrece transformaciones similares a las ofrecidas para RDDs, la principal diferencia es que retornan DStreams:

- `map(func)`
- `flatMap(func)`
- `filter(func)`
- `union(DStream)`
- `count()`
- `reduce(func)`
- `updateStateByKey(func)`

# Otros frameworks de streaming

Apache Spark Streaming tiene como principal inconveniente a la hora de procesar en tiempo real que ofrece una latencia mínima de segundos (batch interval), por lo que no es la mejor tecnología para resolver escenarios que requieren una latencia de milisegundos.

Existen otros dos frameworks de procesamiento real-time que sí cubren escenarios de baja latencia: Apache Storm y Apache Flink.

# Apache Storm



Framework de computación distribuida:

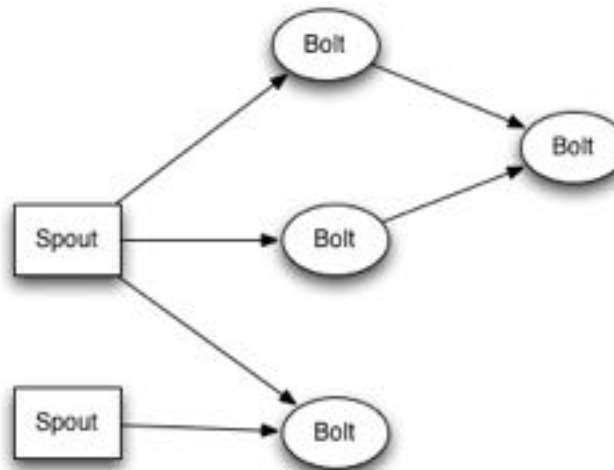
- Ofrece tiempos de latencia de ms, al realizar el procesamiento de cada evento de forma individual y en el momento de su ocurrencia (**one-at-a-time**).
- Garantía “**at least once**”. Con requerimientos “exactly once” requiere utilizar Trident API.
- **Escalable y elástico**: se pueden añadir nodos durante la ejecución sin pérdida de servicio salvo en Zookeeper.
- Permite utilizar distintos **lenguajes** de programación, no sólo Java.
- Puede ser ejecutado sobre **YARN**, y es ofrecido “de caja” por alguna distribución de Hadoop.

# Apache Storm



Se basa en la definición de Spouts y Bolts para definir topologías de procesamiento de streams de tuplas.

Requiere programar cada Bolt y Spout.





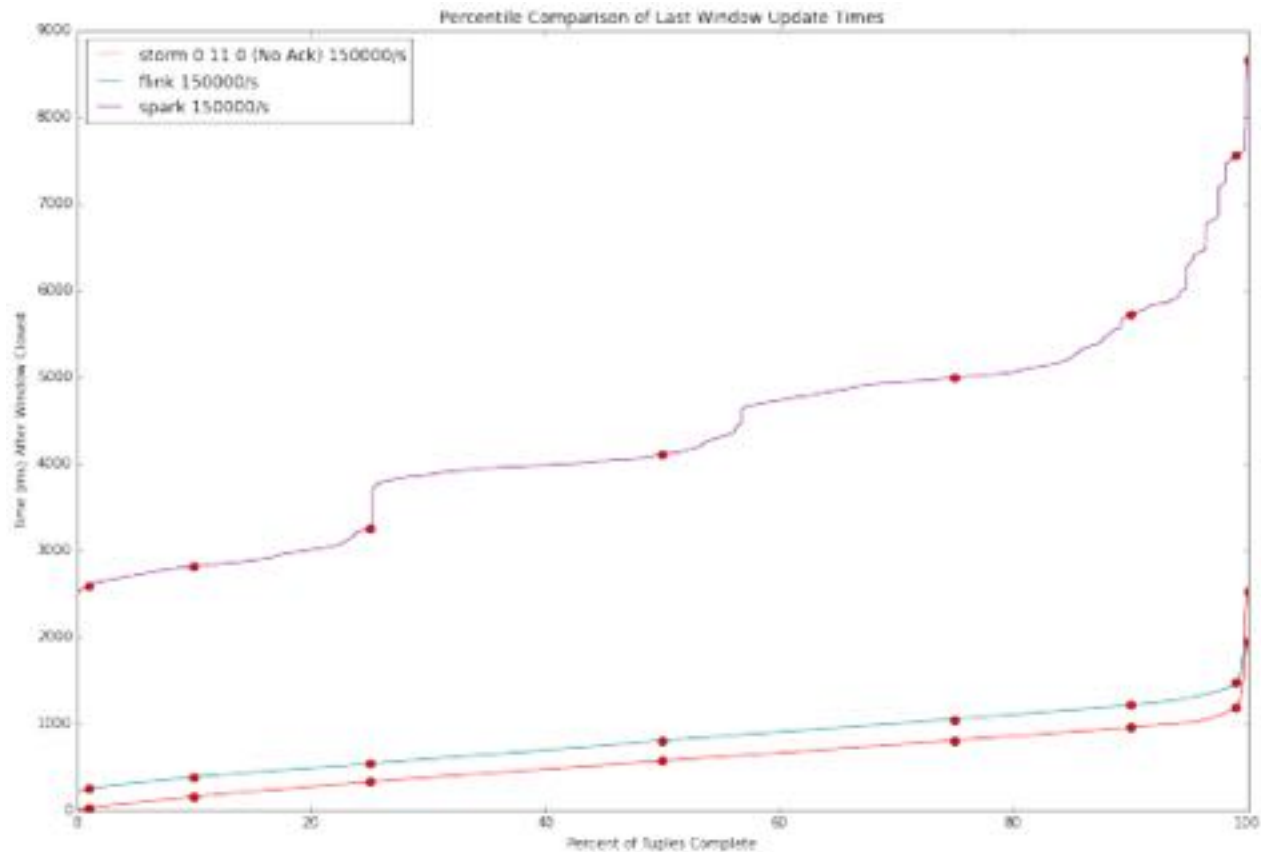
# Apache Flink



Framework de procesamiento distribuido orientado a streaming:

- Ofrece distintos modos de procesamiento: batch, streaming, procesos iterativos, etc.
- El modo de procesamiento es **event-at-a-time**.
- Ofrece un **gran throughput** y **baja latencia**.
- Válido para procesamiento de eventos complejos (**CEP**), que requieren mantenimiento del estado y persistencia (no ofrece repositorio, pero permite conectarlo a Kafka, HDFS, Cassandra, etc.).
- **Tolerante a fallos** y con garantía **exactly-once**.
- Soporta **lenguajes** Java, Scala, Python y SQL.

# Flink vs Spark vs Storm



# Plataformas cloud: introducción a cloud computing.

# Cloud Computing

Gartner define cloud computing como un estilo de computación por la que capacidades de IT son ofrecidas a través de internet **de forma elástica y escalable**.

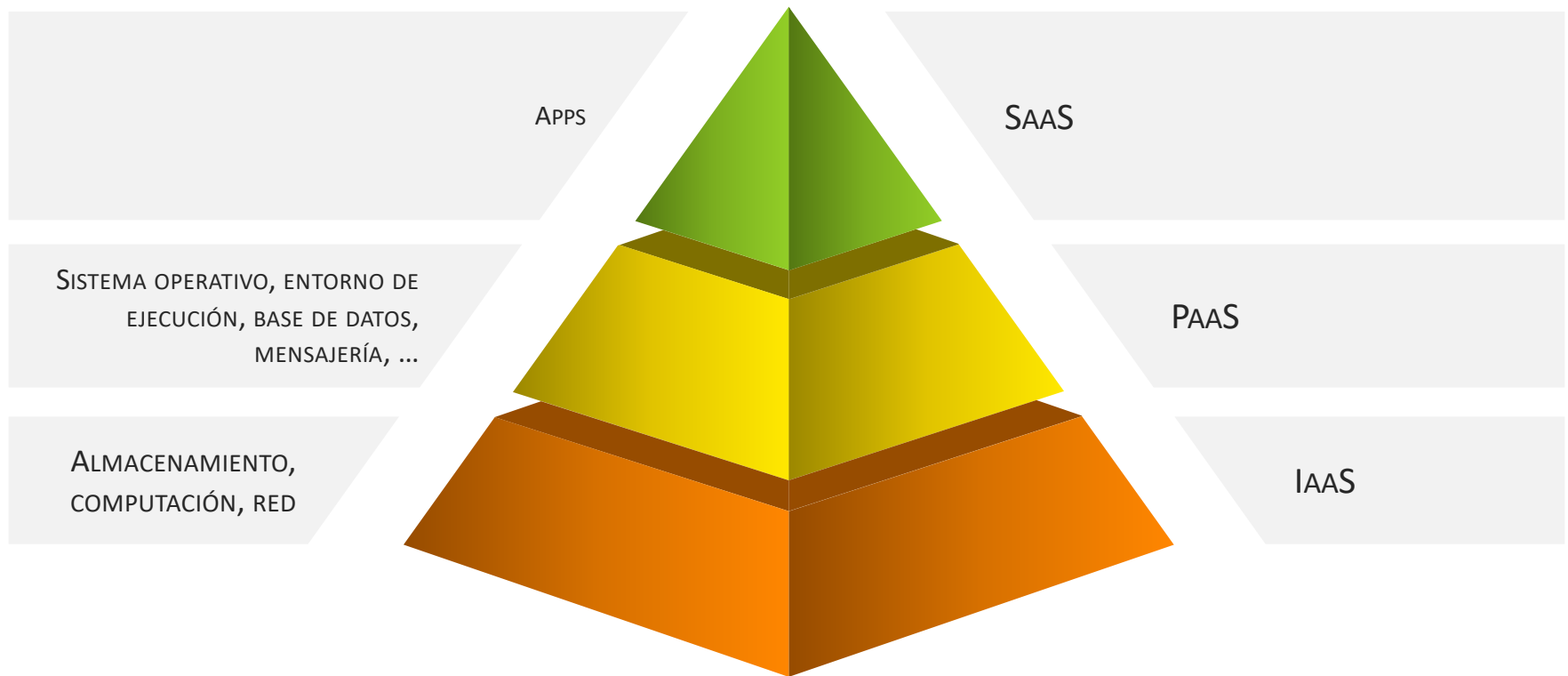
*Computing may someday be organized as a **public utility**  
just as the telephone system is a public utility*  
John McCarthy, 1960



# Características de servicios cloud

- 
- 01** A demanda
  - 02** Pago por uso
  - 03** Escalable
  - 04** Disponible
  - 05** Uniforme

# Niveles de servicios cloud



# Beneficios e inconvenientes

## Beneficios:

- Coste variable.
- Economías de escala.
- Capacidad elástica.
- Reducción del riesgo.
- Agilidad.
- Servicio global.



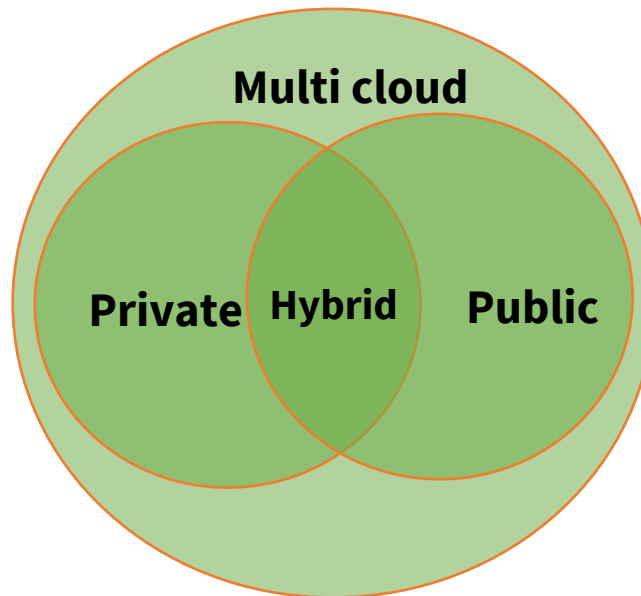
# Estrategias de implementación

## Nube privada

Infraestructura básica dentro del CPD del cliente o en una nube privada. Apropriada para mantener aplicaciones dentro de los "muros" de la empresa.

## Nube híbrida

Las aplicaciones se ejecutan tanto en la nube publica como privada en un entorno mixto.



## Nube publica

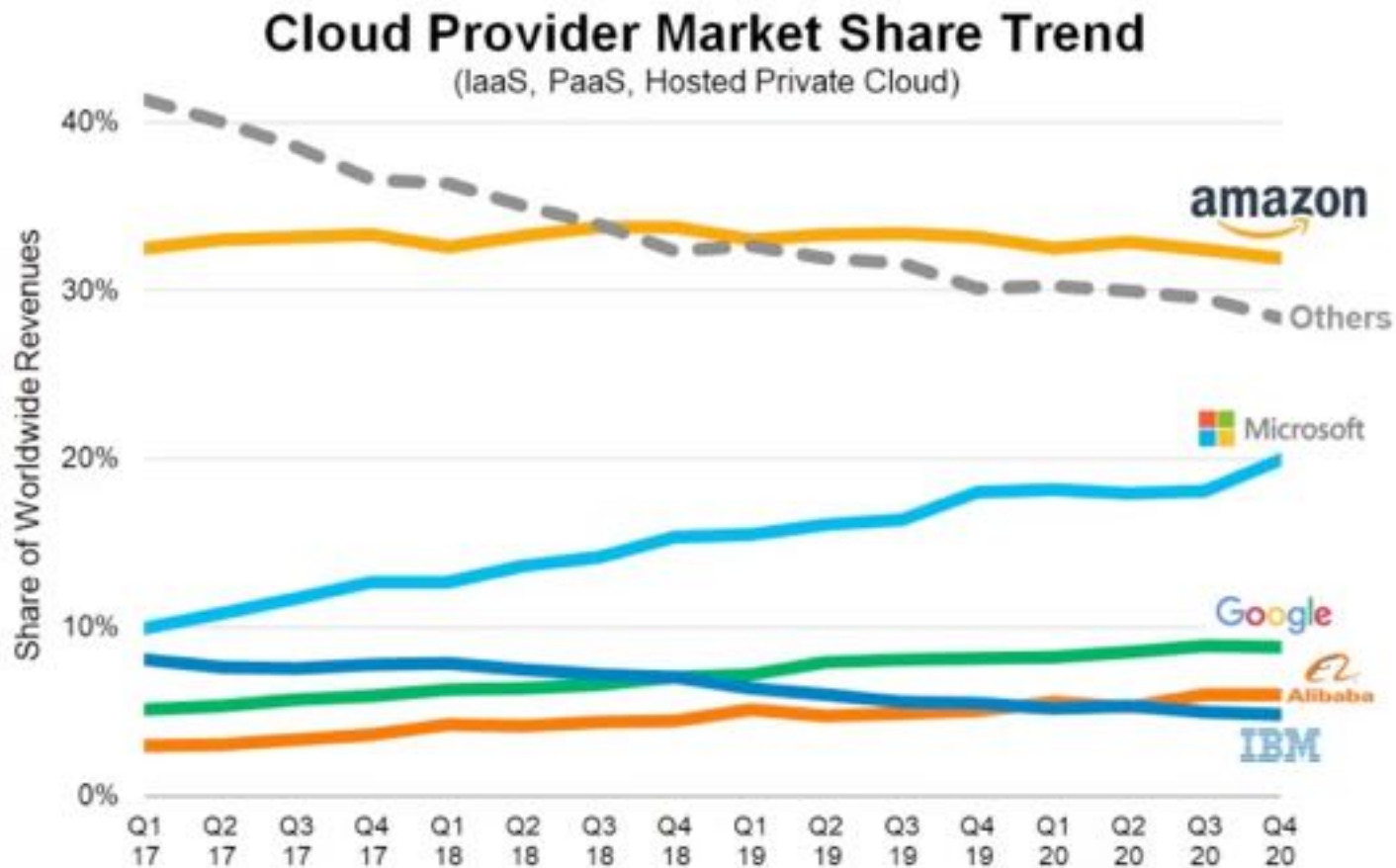
Se ejecuta las aplicaciones en la nube publica usando infraestructuras de AWS, Azure, Google Cloud, etc. Es muy apropiada para pruebas y desarrollo o para montar casos de uso específicos

## Multi nube

Usa el modelo de cloud mas apropiado privada, pública o híbrida para cada tipo de aplicación



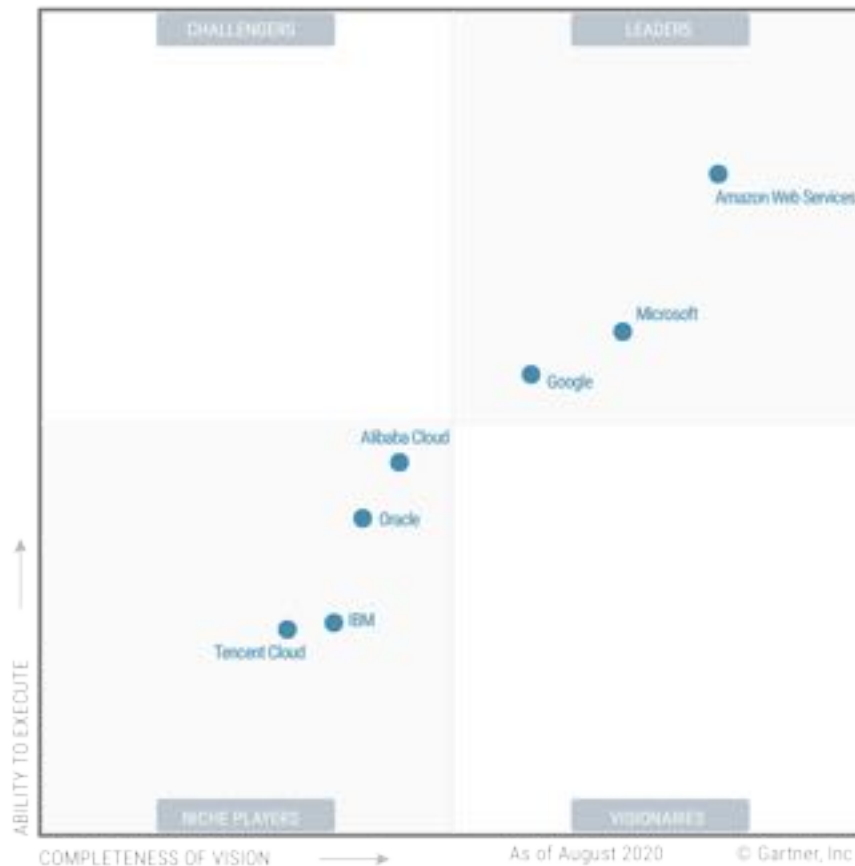
# Principales proveedores de servicios cloud



Source: Synergy Research Group

# Principales proveedores de servicios cloud

Figure 1. Magic Quadrant for Cloud Infrastructure and Platform Services



# Amazon Web Services



Amazon Web Services (AWS) es una suite de servicios cloud de Amazon.

Es líder en este tipo de servicios, aunque es un sector muy competido, donde también ofrece servicios Google, Microsoft, IBM, etc.

Lanzado oficialmente en 2006, dispone de más de 200 servicios.

Cuenta con 25 regiones con una o varias zonas de disponibilidad por región por todo el mundo



# Amazon Web Services. Servicios



## Informática

EC2

Lightsail [↗](#)

Lambda

Batch

Elastic Beanstalk

Serverless Application  
Repository

AWS Outposts

EC2 Image Builder

AWS App Runner



## Contenedores

Elastic Container Registry

Elastic Container Service

Elastic Kubernetes Service

Red Hat OpenShift Service on  
AWS



## Almacenamiento

S3

EFS

FSx

S3 Glacier

Storage Gateway

AWS Backup



## Base de datos

RDS

DynamoDB

ElastiCache

Neptune

Amazon QLDB

Amazon DocumentDB

Amazon Keyspaces

Amazon Timestream

# Amazon Web Services. Servicios



## **Migración y transferencia**

AWS Migration Hub

AWS Application Migration  
Service

Application Discovery Service

Database Migration Service

Server Migration Service

AWS Transfer Family

AWS Snow Family

DataSync



## **Quantum Technologies**

Amazon Braket



## **Redes y entrega de contenido**

VPC

CloudFront

Route 53

API Gateway

Direct Connect

AWS App Mesh

AWS Cloud Map

Global Accelerator

# Amazon Web Services. Servicios



## Herramientas para desarrolladores

CodeStar

CodeCommit

CodeArtifact

CodeBuild

CodeDeploy

CodePipeline

Cloud9

CloudShell

X-Ray

AWS FIS



## Habilitación para clientes

AWS IQ 

[Support](#)

Managed Services

Activate for Startups



## Robótica

AWS RoboMaker



## Cadena de bloques

Amazon Managed Blockchain



## Satélite

Ground Station



# Amazon Web Services. Servicios



## Administración y gobierno

AWS Organizations

CloudWatch

AWS Auto Scaling

CloudFormation

CloudTrail

Config

OpsWorks

Service Catalog

Systems Manager

AWS AppConfig

Trusted Advisor

Control Tower

AWS License Manager

AWS Well-Architected Tool

Personal Health Dashboard 

AWS Chatbot

Launch Wizard

AWS Compute Optimizer

Resource Groups & Tag Editor

Amazon Grafana

Amazon Prometheus

AWS Proton

Incident Manager

# Amazon Web Services. Servicios

## **Servicios multimedia**

Kinesis Video Streams  
MediaConnect  
MediaConvert  
MediaLive  
MediaPackage  
MediaStore  
MediaTailor  
Elemental Appliances &  
Software  
Amazon Interactive Video  
Service  
Elastic Transcoder  
Nimble Studio

## **Análisis**

Athena  
Amazon Redshift  
EMR  
CloudSearch  
Elasticsearch Service  
Kinesis  
QuickSight   
Data Pipeline  
AWS Data Exchange  
AWS Glue  
AWS Lake Formation  
MSK  
AWS Glue DataBrew  
Amazon FinSpace





# Amazon Web Services. Servicios



## **Administración de costos de AWS**

AWS Cost Explorer

AWS Budgets

AWS Marketplace Subscriptions

AWS Application Cost Profiler



## **Móvil**

AWS Amplify

Mobile Hub

AWS AppSync

Device Farm

Amazon Location Service



## **RA y RV**

Amazon Sumerian



## **Integración de aplicaciones**

Step Functions

Amazon AppFlow

Amazon EventBridge

Amazon MQ

Simple Notification Service

Simple Queue Service

SWF

Apache Airflow administrado

# Amazon Web Services. Servicios



## **Informática para usuarios finales**

WorkSpaces

AppStream 2.0

WorkLink



## **Aplicaciones empresariales**

Amazon Connect

Amazon Pinpoint

Amazon Honeycode

Amazon Chime 

Amazon Simple Email Service

Amazon WorkDocs

Amazon WorkMail

Alexa for Business



## **Internet de las cosas**

IoT Core

FreeRTOS

IoT 1-Click

IoT Analytics

IoT Device Defender

IoT Device Management

IoT Events

IoT Greengrass

IoT SiteWise

IoT Things Graph



## **Desarrollo de videojuegos**

Amazon GameLift

# Amazon Web Services. Servicios



## **Machine Learning**

Amazon SageMaker

Amazon Augmented AI

Amazon CodeGuru

Amazon DevOps Guru

Amazon Comprehend

Amazon Forecast

Amazon Fraud Detector

Amazon Kendra

Amazon Lex

Amazon Personalize

Amazon Polly

Amazon Rekognition

Amazon Textract

Amazon Transcribe

Amazon Translate

AWS DeepComposer

AWS DeepLens

AWS DeepRacer

AWS Panorama

Amazon Monitron

Amazon HealthLake

Amazon Lookout for Vision

Amazon Lookout for Equipment

Amazon Lookout para métricas

# Amazon Web Services. Servicios



## ① Seguridad, identidad y conformidad

IAM

Resource Access Manager

Cognito

Secrets Manager

GuardDuty

Inspector

Amazon Macie

AWS Single Sign-On

Certificate Manager

Key Management Service

CloudHSM

Directory Service

WAF & Shield

AWS Firewall Manager

Artifact

Security Hub

Detective

AWS Audit Manager

AWS Signer

AWS Network Firewall

# Microsoft Azure



Azure es una suite de servicios cloud de Microsoft.

Lanzado oficialmente en 2010, dispone de más de 200 servicios.

Cuenta con más de 60 regiones, siendo el proveedor de cloud con mayor número de ellas.

En cuanto a coste, es equivalente o un poco inferior al de AWS.



# Plataformas cloud: soluciones Big Data en Cloud.

# Amazon EMR



- Es un servicio de Amazon Web Services que permite crear **clusters Hadoop a demanda**.
- Utiliza una **distribución propia** de Amazon que permite seleccionar los componentes que van a lanzarse en el cluster (Hive, Spark, etc.).
- Ofrece **elasticidad**: modificar dinámicamente el dimensionamiento del cluster según necesidades.
- Se ejecuta **sobre máquinas EC2** (IaaS).
- **Pago por uso**: el coste asociado es el alquiler de las máquinas por horas más un sobrecoste de aproximadamente el 25%.
- Ejemplo de coste aproximado:
  - 20 nodos con 122 Gb RAM, 16 vCPU: 32 €/h.
  - Para ejecutar una tarea de 10h: 320 €.
  - Con 200 nodos: duración = 1 hora, coste = 320 €.

# Demo: Arranque de un clúster en Amazon EMR

<https://eu-west-1.console.aws.amazon.com/elasticmapreduce/home?region=eu-west-1>



## Bienvenido/a a Amazon Elastic MapReduce

Amazon Elastic MapReduce (Amazon EMR) es un servicio web que permite a empresas, investigadores, analistas de datos y desarrolladores procesar de forma fácil y rentable grandes volúmenes de datos.

Parece que no dispone de ningún clúster. Crear uno ahora:

[Crear clúster](#)

## Cómo funciona Elastic MapReduce

### Cargar



Cargue sus datos y la aplicación de procesamiento en S3.

[Más información](#)

### Crear



Configure y cree el clúster especificando las entradas de datos, los resultados, tamaño del clúster, configuración de seguridad, etc.

[Más información](#)

### Monitorizar



Monitoree el estado y el progreso del clúster. Recupere el resultado en S3.

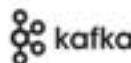
[Más información](#)



# Azure HDInsight



- Es un servicio de Azure que permite crear **clusters Hadoop a demanda**.
- Utiliza la distribución HDP de Hortonworks que permite seleccionar los componentes que van a lanzarse en el cluster (Hive, Spark, etc.).
- Ofrece **elasticidad**: modificar dinámicamente el dimensionamiento del cluster según necesidades.
- Ofrece distintos tipos de clusters para lanzar: Hadoop, Spark, Interactive Query, HBase y Kafka.
- Está cayendo en desuso en favor de otros tipos de soluciones de Azure (Azure Databricks, Azure DataLake Storage, etc.).



# Demo: Arranque de un clúster Azure HDInsight



Microsoft Azure Upgrade Search resources, services, and docs (G+/I)


Home > HDInsight clusters Directorio predeterminado

+ Create Manage view Refresh Export to CSV Open query Feedback Assign tags Delete

Filter for any field... Subscription == all Resource group == all Location == all Add filter

Showing 0 to 0 of undefined records. No grouping

Name ↑	Cluster type ↑	Status ↑	Resource group ↑	Location ↑
--------	----------------	----------	------------------	------------



No HDInsight clusters to display

Create an HDInsight cluster to process massive amounts of data using popular open-source frameworks such as Hadoop, Spark, Hive, LLAP, Kafka, Storm, ML Services, and more.

[Learn more about HDInsight >](#)

Create HDInsight cluster

# Gracias.