Piensa en grande:
Big data para programadores

by @rafbermudez

@rafbermudez

- Rafael Bermúdez Míguez @rafbermudez
- Ingeniero informático y PDGE
- Miembro activo de la comunidad
- Vivo pegado al código
- mail: rafa@rafbermudez.com

wirelessgalicia SOLUCIONES INALAMBRICAS

UNIVERSIDADE DA CORUÑA

GT ARTABRIA GRUPO TECNOLÓGICO ARTABRIA S.L.

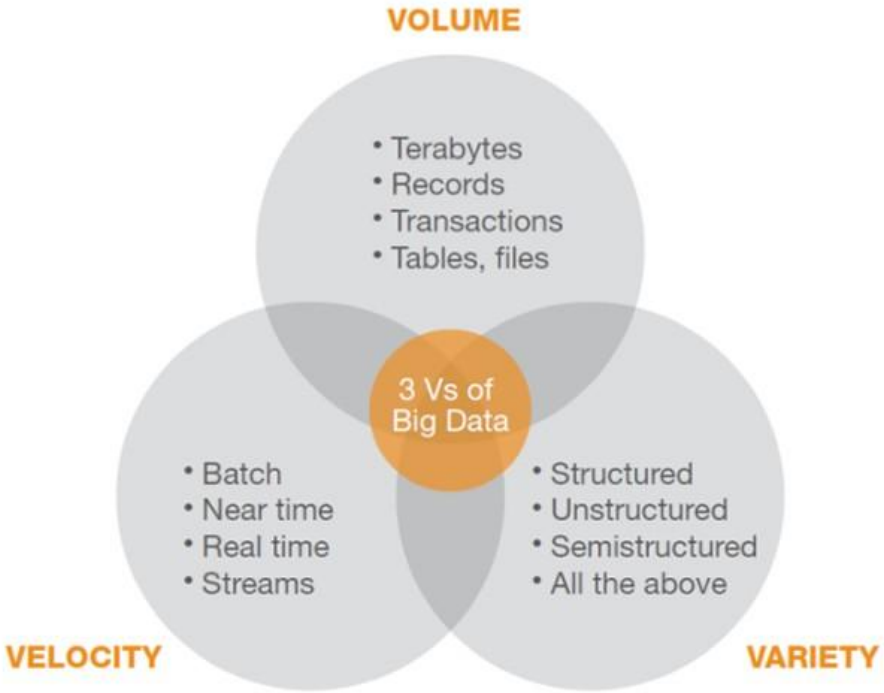
Grupo Aluman

Yump

OTOGAMI™

ets ASSET MANAGEMENT FACTORY

¿QUÉ ES BIG DATA?



Estrategia

1. Clasifica los datos
2. Diseña tu arquitectura
    - Persistencia
    - Comunicación
    - Procesamiento

1.Clasifica los datos

- Formato
- Tipo de datos
  - Operacionales,
  - Históricos,
  - Maestros
- Frecuencia
- Intención
- Procesamiento
  - Tiempo real, casi- tiempo real, por lotes, ...

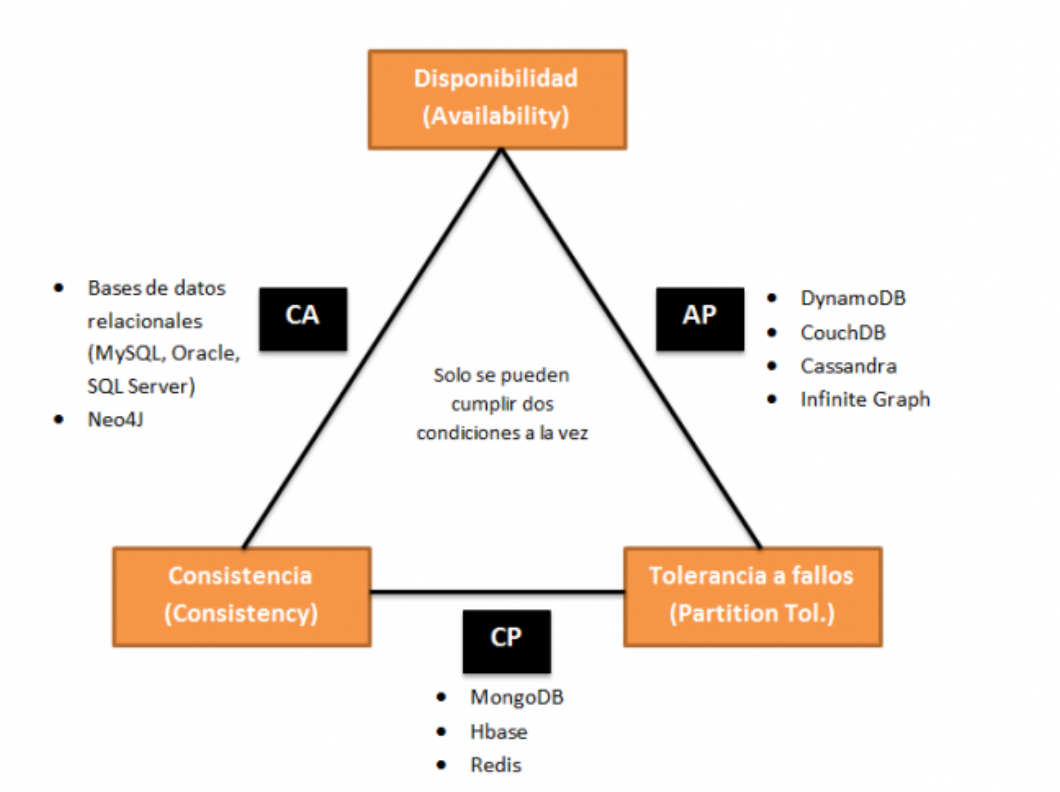
2.Diseña tu arquitectura

- Escalabilidad lineal
- Tolerancia a fallos
- Procesamiento distribuido y localidad de los datos
- Despliegue sobre hardware de propósito general

Componentes

- PERSISTENCIA
- COMUNICACIÓN
- PROCESAMIENTO

Persistencia: Teorema CAP



Persistencia: Resumen

Tipo de BD	Cuando	Ejemplos
Relacionales	Alta integridad y consistencia	Postgresql, mysql, oracle
De clave valor	Cachés	Redis, DynamoDB
Orientadas a columnas	Consultas y agregaciones sobre grandes cantidades de datos	Cassandra, HBase, Bigquery
Orientadas a documentos	Datos como documentos	MongoDB, CouchDB
En grafo	Modelos con muchas relaciones	Neo4j

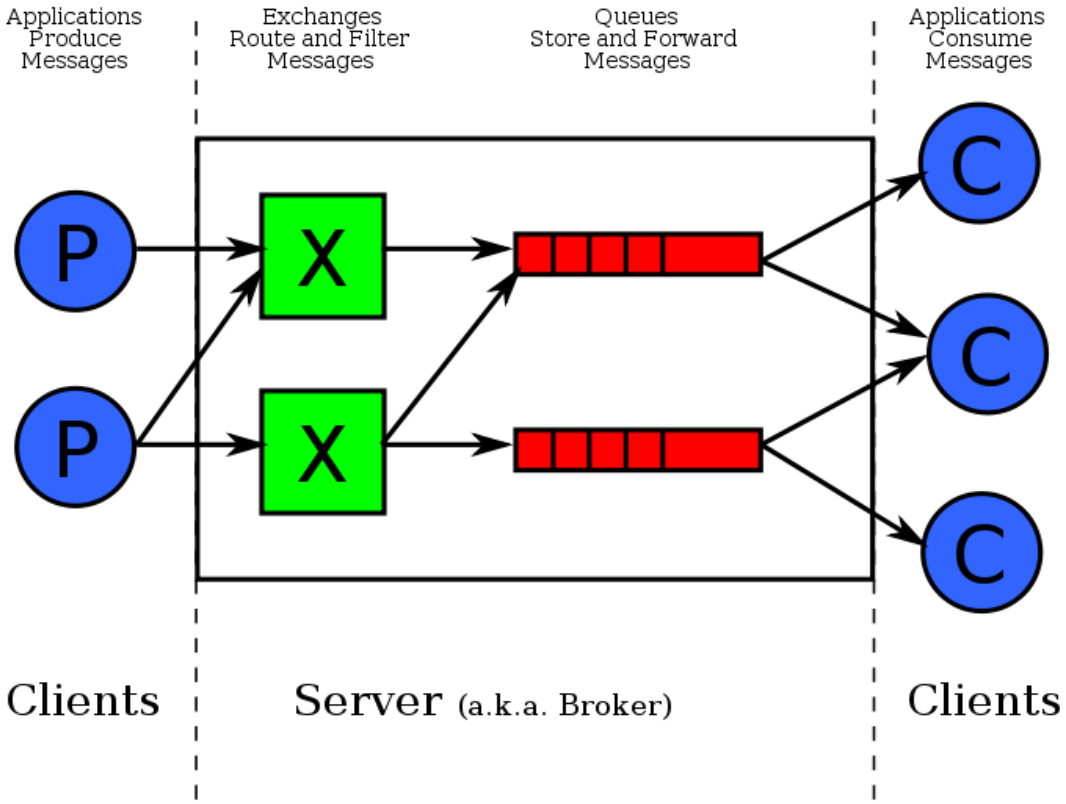
Comunicación

- Desacoplar las piezas del sistema
- Evitar estructuras compartidas
- 2 opciones:
  - Broker de mensajería
    - RabbitMQ
  - Modelo basado en actores
    - Akka

Comunicación: RabbitMQ

- Broker de mensajería basado en colas
- Open source
- Implementa el protocolo AMQP (entre otros)
- Clusterizable
- Persistente (opcional)
- Interfaz gráfica de administración (plugin)
- Disponible en casi todas las plataformas:
  - Python, Java, Ruby, Php, C#, Javascript, Go, ...
  - Incluso Cobol y Ocaml !!

Comunicación: RabbitMQ (II)



Comunicación: RabbitMQ (III)

- Emisor

```
 ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

String message = ”Hello CorunaDev”

channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
```

Comunicación: RabbitMQ (IV)

- Receptor

```
 channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
String queueName = channel.queueDeclare().getQueue();
channel.queueBind(queueName, EXCHANGE_NAME, "");

Consumer consumer = new DefaultConsumer(channel) {
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {
                String message = new String(body, "UTF-8");
                System.out.println(" [x] Received '" + message + "'");
    }
};
channel.basicConsume(queueName, true, consumer);
```

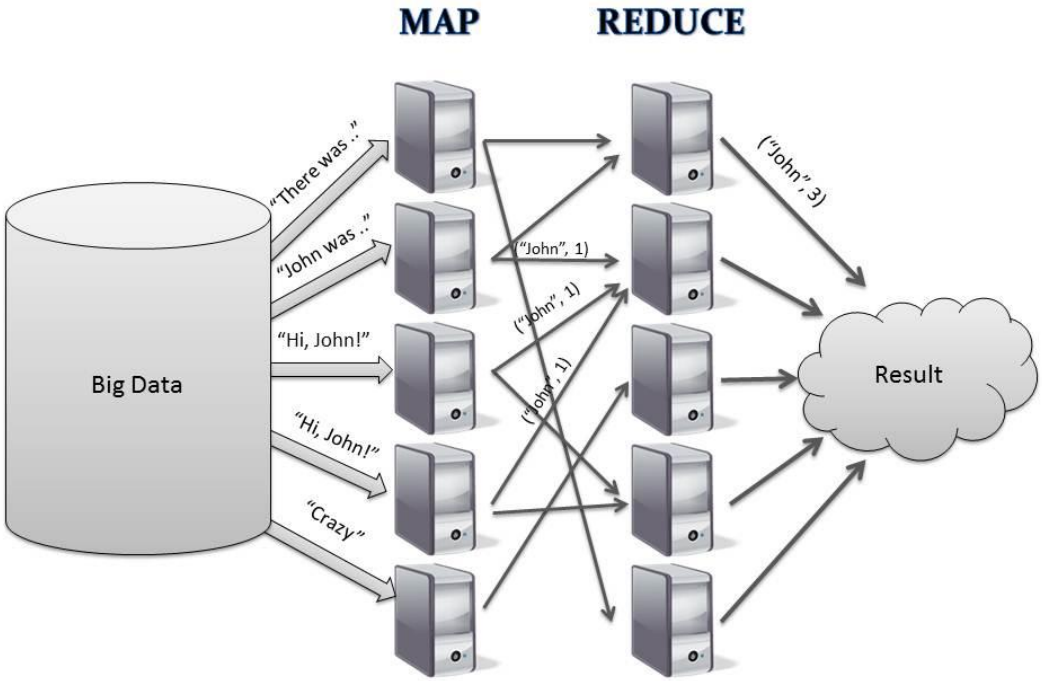
Procesamiento

- Procesamiento tradicional
  - Optimizaciones tradicionales
- Procesamiento paralelo
  - Hilos de aplicación vs aplicaciones paralelas
  - Computación distribuída

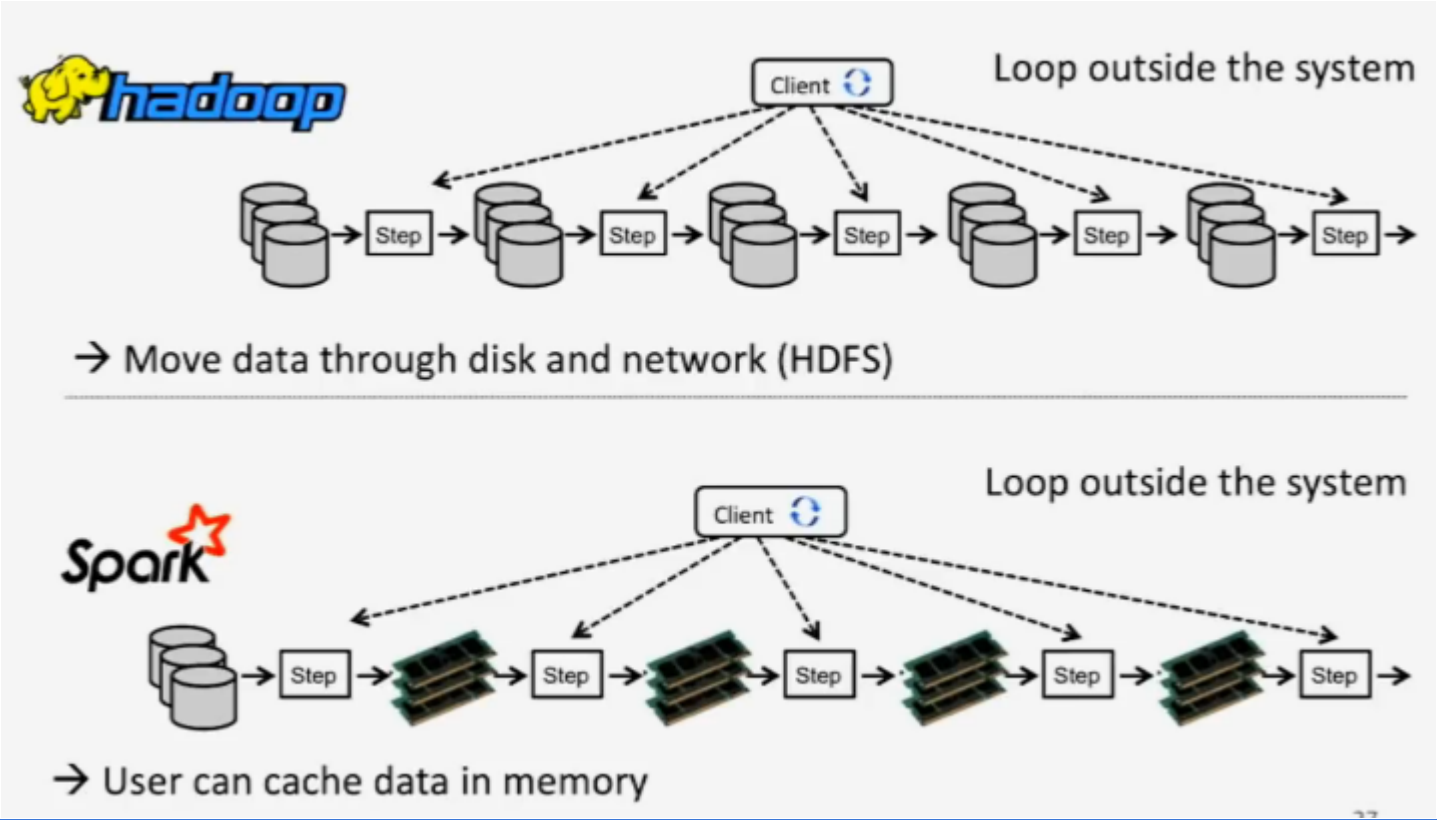
Procesamiento: Computación distribuída

- Basado modelos MapReduce
- Operaciones sobre conjuntos de datos
  - filter, join, distinct, sortByKey, ...
- Aprovechan BD distribuídas
- Ejemplo:
  - Hadoop, Spark, Hive, Pig,

Procesamiento: MapReduce



Procesamiento: Hadoop vs Spark



Procesamiento: Ejemplo Spark

- SparkConf

```
val conf = new SparkConf()
             .setMaster("local[4]")
             .setAppName("CountingWords")
val sc = new SparkContext(conf)
```

--

The master URL to connect to, such as "local" to run locally with one thread, "local[4]" to run locally with 4 cores, or "spark://master:7077" to run on a Spark standalone cluster.

Procesamiento: Ejemplo Spark (II)

- Parallelized collections

```
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)
```

- Spark word count

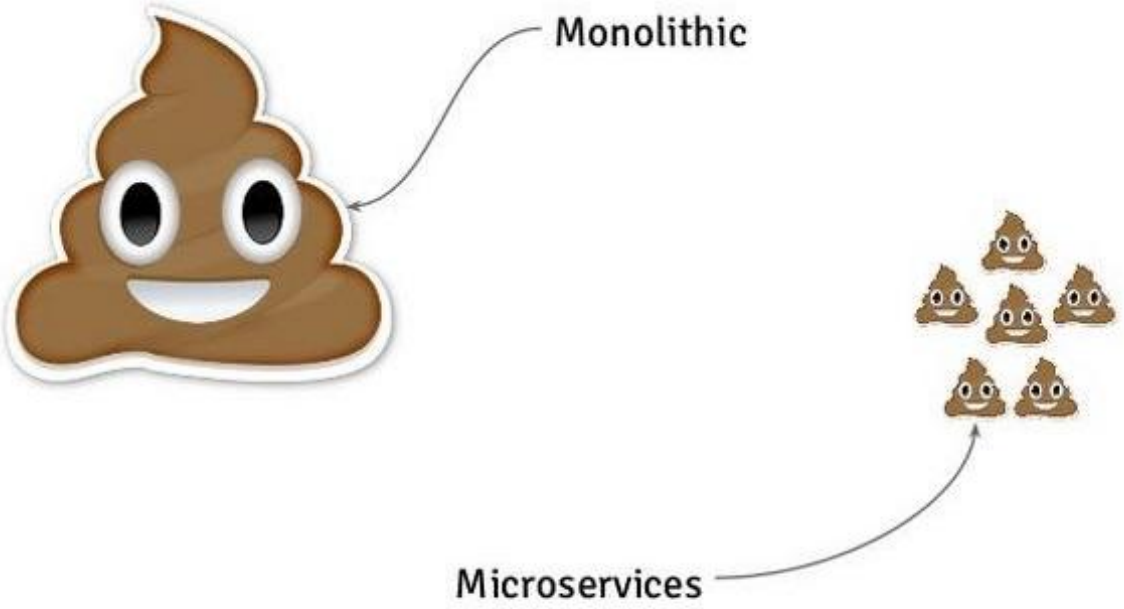
```
file = spark.textFile("hdfs://...")

file.flatMap(line => line.split(" "))
.map(word => (word, 1))
.reduceByKey(_ + _)
.reduceByKey(_ + _)
```

Arquitectura

- Orientada a microservicios
- Lambda

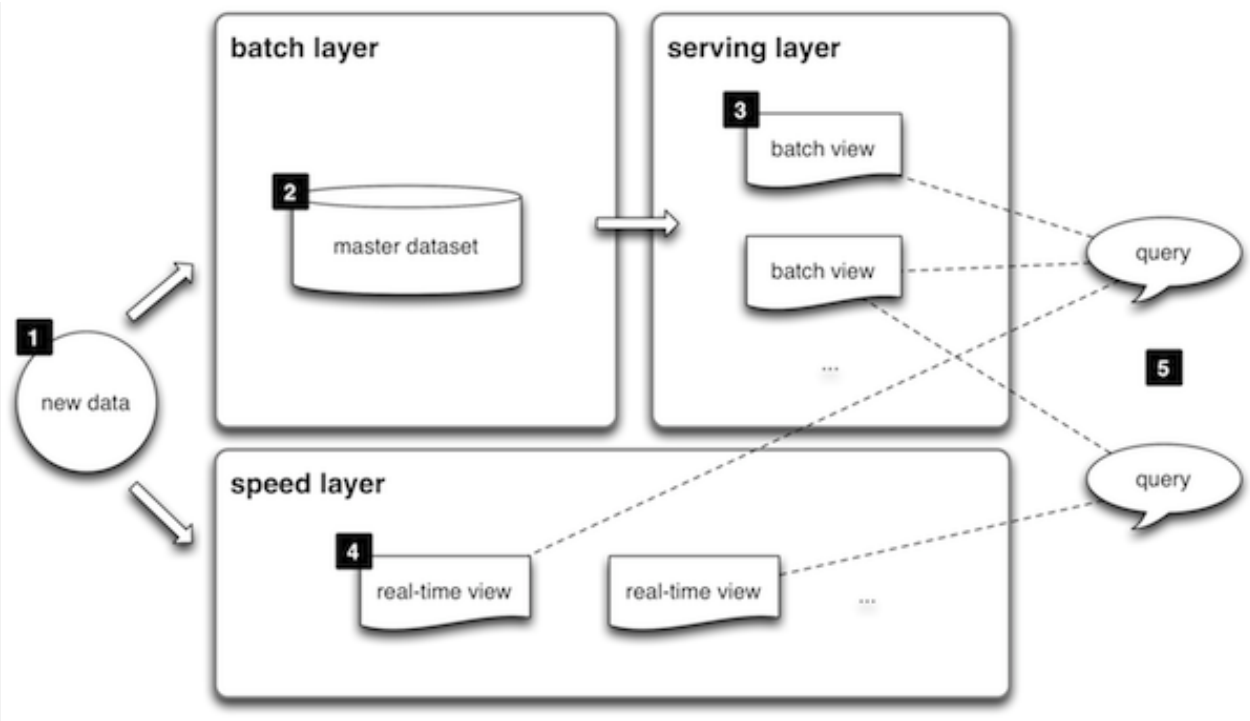
Arquitectura orientada a microservicios



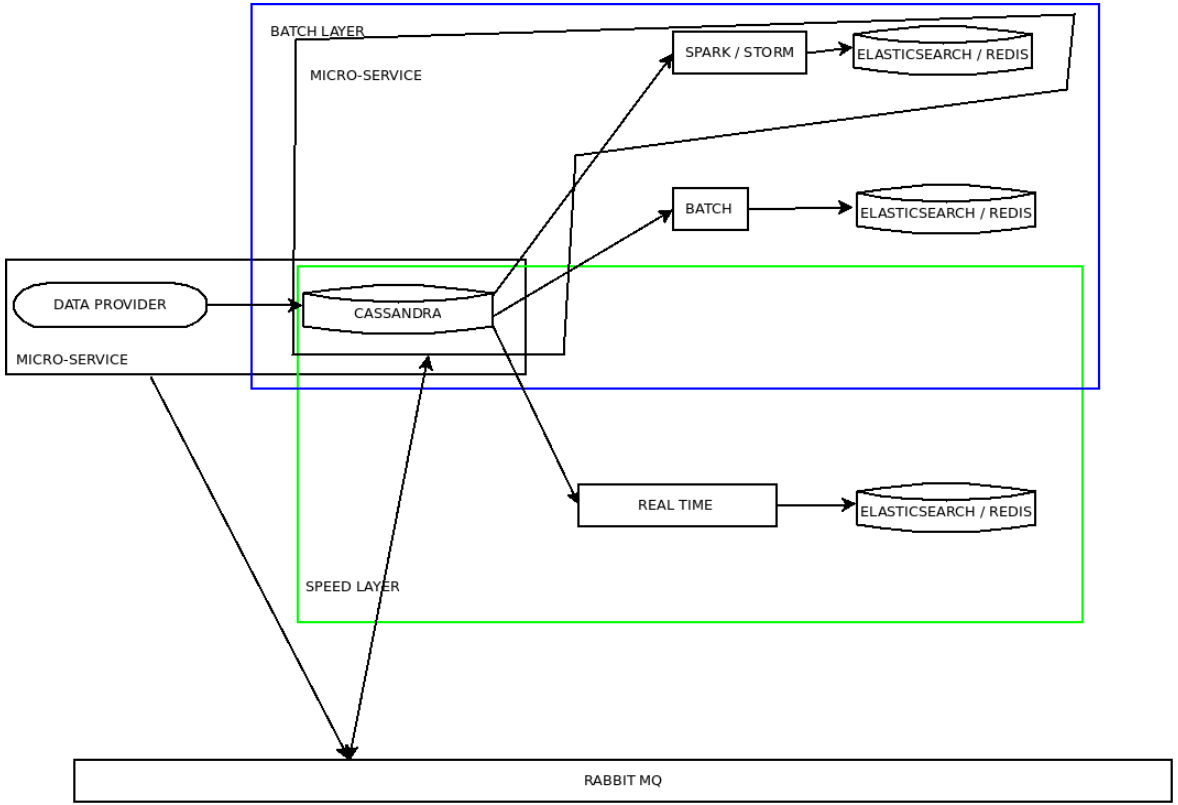
Arquitectura orientada a microservicios (II)

- No son objetos distribuídos
- Bajo acoplamiento
- Alta cohesión
- Objetivo a perseguir:
  - Eliminar la necesidad de coordinación
  - Evitar Dependencias de otros microservicios
  - Transacciones distribuídas
  - ¡Eureka! Es escalable

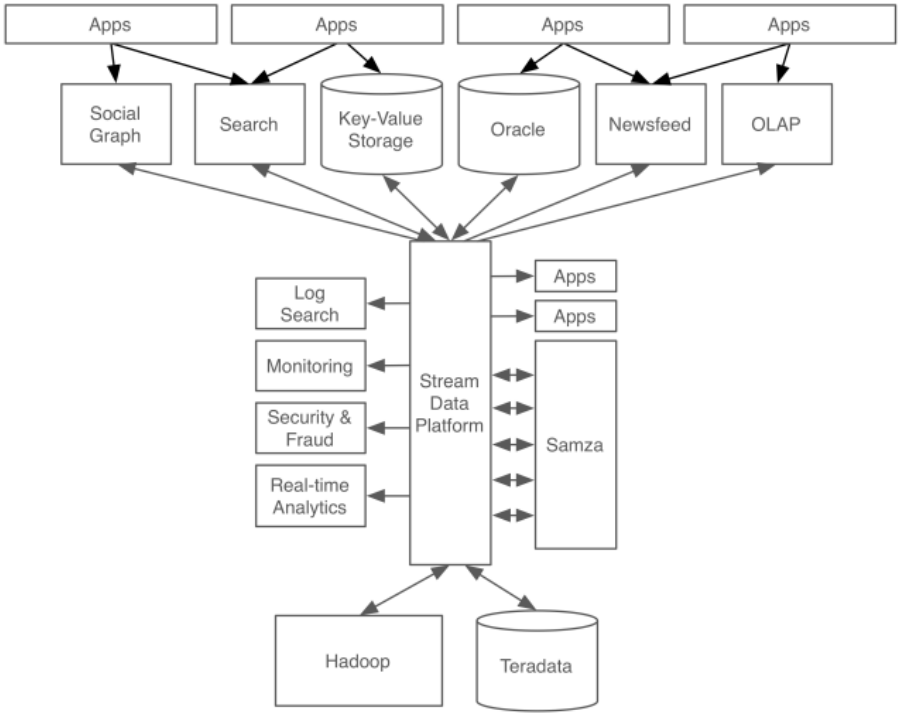
Arquitectura Lambda



Arquitectura Lambda (II)



Bonus: Arquitectura de pipeline central



Despliegue

- Docker
  - Máquinas virtuales ligeras
- Orquestación: Netflix oss
  - Eureka (registro y autoreconocimiento de servicios)
  - Hystrix & Turbine (control de ruptura de comunicación con los servicios)
  - Ribbon (balance de carga)
  - Feign (llamadas servicio a servicio)
  - Zuul (enrutado)
  - Archaius (configuración distribuida)
  - Curator (clustering)
  - Asgaard (deployments y gestión del cloud)
  - rabbitmq (mensajería distribuida)

¡Gracias! ¿ Opiniones y Preguntas?

Piensa en grande: Big data para programadores

1. Clasifica los datos
2. Diseña tu arquitectura
    - Persistencia
    - Comunicación
    - Procesamiento