

# **Resumen global: Temas 1 al 6**

## **Tema 1: Propuesta de Arquitectura de Datos**

### **Capitalizar un Volumen de Datos Creciente y Diverso**

Para que una empresa integre un ecosistema de datos complejo para impulsar su crecimiento tiene que seguir los siguientes objetivos:

- Integración de Fuentes Múltiples: Consolidar datos de fuentes externas e internas.
- Gestión de Volumen Exponencial: Administrar una BBDD de 500TB con crecimiento de 100TB garantizando accesibilidad y sin pérdida de información.
- Inteligencia de Negocio Avanzada: Utilizar analíticas descriptivas interactivas para la toma de decisiones.
- Personalización y Crecimiento: Crear ofertas personalizadas y predictivas.

### **Nuestra visión: Un Ecosistema de Datos Unificados y Orientado al Futuro**

Arquitectura híbrida que gestiona los datos en reposo y en movimiento, que permite crecer y evolucionar.

### **Cimientos del Sistema: Ingesta y Almacenamiento Escalable**

Para gestionar los datos la base del sistema debe ser elástica y robusta

- Capas Esenciales: Dentro de esta encontramos una Capa de Ingesta para capturar datos y una Capa de Almacenamiento para almacenar la información
- Escalabilidad Horizontal: El clúster se diseña para la escalabilidad horizontal. Se añadirán nuevos servidores de forma progresiva para aumentar el almacenamiento y procesamiento de forma lineal sin interrumpir el servicio, este es el más costo-eficiente.

## **Garantizando la Integridad: Gestión de Transacciones Diarias**

Las interacciones con clientes requieren un tratamiento para garantizar la consistencia y fiabilidad de los datos

- Necesidad de un Subsistema de Procesamiento de Transacciones en Línea (OLTP): Se encarga de gestionar las operaciones de alta frecuencia del día a día de forma rápida y eficiente
- Cumplimiento ACID: Para proteger las transacciones se tienen que cumplir las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad)

## **Potenciando la Decisión: Analítica para la Junta Directiva**

Para satisfacer la demanda de análisis interactivos mensuales implementaremos:

- Estrategia de Procesamiento por Lotes (Batch): Utiliza un proceso Batch para procesar los datos históricos. Se programa una ejecución semanal que consolidará todos los datos disponibles, para la consulta una semana antes de la reunión de la junta.
- Necesidad de un Subsistema OLAP: Este sistema pre-calculará y agregará los datos procesados, permitiendo a los directivos realizar consultas complejas, filtradas y visualizaciones en el cuadro de mandos de forma casi instantánea.

## **El Repositorio Estratégico: El Papel del Almacén de Datos**

Para alimentar el subsistema OLAP y garantizar consultas de alto rendimiento contamos con un repositorio de datos estructurado y optimizado

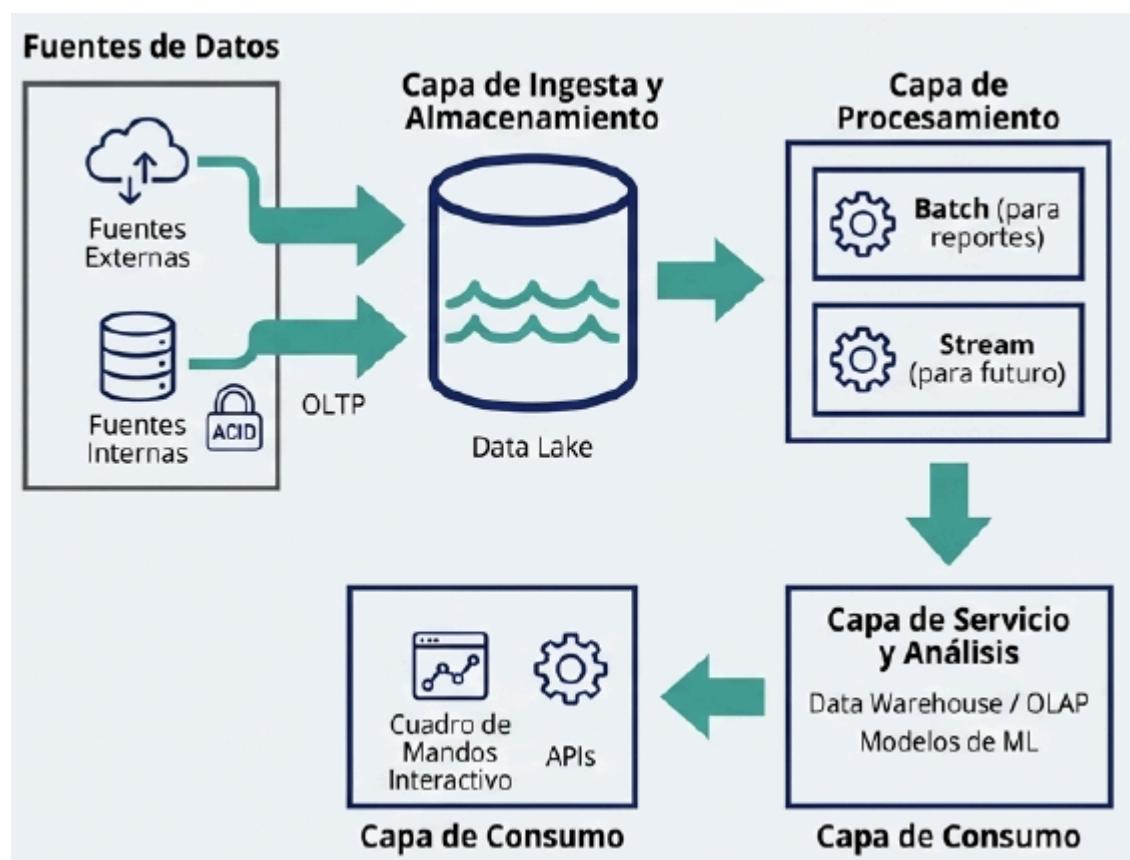
- Implementación de un Almacén de Datos (Data Warehouse): El sistema extraerá datos limpios, transformados y relevantes del Data Lake. Su diseño basado en esquemas está diseñado para acelerar las consultas analíticas que potenciarán el cuadro de mandos de la junta directiva.

## **La Arquitectura Completa: Un Sistema Integrado de Capas**

La solución propuesta integra un conjunto de capas para gestionar el ciclo de vida completo del dato, desde su origen hasta su conversión en valor.

Capas mínimas requeridas:

1. Capa de Fuentes de Datos: Origen de la información
2. Capa de Ingesta y Almacenamiento: Captura y persistencia segura en un Data Lake.
3. Capa de Procesamiento: Lógica de negocio con motores Batch y Streaming.
4. Capa de Servicio y Análisis: Exposición de los datos a través de un Data Warehouse, sistemas OLAP y APIs para modelos predictivos.



## Desbloqueando el futuro: De la Analítica Descriptiva a la Predictiva

El poder de este ecosistema consiste en predecir el futuro.

Necesidad de Modelos Predictivos: Estos utilizarán los datos históricos para encontrar patrones y hacer predicciones.

# Una Solución Integral para un Liderazgo Basado en Datos

El prediseño propuesto ofrece una solución completa y robusta que responde a cada una de las necesidades de empresa:

- Escalabilidad garantizada: Crece con su negocio sin limitaciones técnicas
- Decisiones inteligentes: Proporciona herramientas para la toma de decisiones
- Integridad y Confianza: Aseguro los datos
- Crecimiento Predictivo: Podemos personalizar y optimizar la relación con el cliente.

## Tema 2: Dominando Big Data: Una Introducción Práctica a Hadoop

### El Desafío: ¿Qué es Big Data?

Es un conjunto de grandes volúmenes de datos cuyo objetivo es extraer valor a partir de información dispersa y compleja

- Volumen: Datos masivos que superan la capacidad de las herramientas tradicionales
- Variedad: Datos estructurados BBDD, semiestructurados (logs) y no estructurados (imágenes, tweets).
- Velocidad: La necesidad de procesar los datos de forma continua y casi en tiempo real

### La Solución: ¿Qué es Hadoop?

Un framework open source de Apache diseñado para almacenar y procesar Big Data de forma distribuida y paralela

#### ¿Cómo lo hace?

- Almacenamiento: Guarda los datos de forma distribuida en múltiples máquinas (HDFS)

- Procesamiento: Procesa los datos en paralelo aprovechando todos los nodos del clúster (MapReduce, YARN)

### Ventajas Clave

- Escalabilidad horizontal: Se pueden añadir más máquinas para aumentar la capacidad
- Tolerancia a fallos: La replicación de datos evita pérdidas si un nodo falla
- Bajo coste: Utiliza hardware común y accesible

## El núcleo de Hadoop

- HDFS: Es un sistema de archivos que divide los datos en bloques, los reparte y replica entre los nodos del clúster
- YARN: Asigna CPU, memoria y red a los trabajos, permitiendo que varias aplicaciones compartan el clúster de forma ordenada.
- MapReduce: Modelo de programación que divide la tarea en dos fases Map para procesar fragmentos y Reduce para combinar resultados

## El Ecosistema Hadoop

- Hive: Consultas tipo SQL en Big Data
- Pig: Lenguaje de scripting para análisis de datos
- Sqoop: Importa/exporta datos entre BBDD y Hadoop
- Spark: Motor de procesamiento rápido en memoria
- HBase: Base de datos NoSQL distribuida para acceso rápido
- Zookeeper: Coordina y sincroniza los servicios del clúster
- Flume: Ingesta de datos masivos en streaming

## ¿Cómo se usa en el mundo real? Distribuciones y Cloud

¿Qué es una distribución?

Una implementación de Hadoop y su ecosistema, lista para usar

## Laboratorio: Objetivo y Entorno

Directorio Hadoop:

/home/manuu/Desktop/hadoop

Jar de ejemplos:

/home/manuu/Desktop/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar

Directarios de Datos (locales):

- Entrada:
  - /tmp/entrada (debe contener archivos XML)
- Salida:
  - /tmp/salida

## Paso 1: Preparación del entorno

Acción 1: Navegar al directorio de trabajo

cd "ruta\_a\_la\_que\_queremos\_ir"

Acción 2: Asegurar los datos de entrada

El directorio "/tmp/entrada" debe contener los XML fuentes de datos

## Paso 2: Ejecución 1. Busqueda de "dfs"

Ejecutar el job "grep" de ejemplo para encontrar todas las líneas que contengan el patrón "dfs" seguido de letras minúsculas

hadoop jar hadoop-mapreduce-examples-3.3.6.jar grep \ /tmp/entrada \ /tmp/salida \ 'dfs[a-z.]+'

## Paso 3: verificación #1 - El Resultado

Comprobar que el job se ejecutó correctamente y generó la salida esperada

Listar el contenido del directorio de salida

ls /tmp/salida

Ver el contenido del fichero

cat /tmp/salida/part-r-00000

## Paso 4: Ejecución #2 - Conteo de la letra "a"

Borramos la salida anterior

```
rm -rf /tmp/salida
```

Ejecutar el segundo job: Contar las ocurrencias de la letra "a"

```
hadoop jar hadoop-mapreduce-examples-3.3.6.jar grep \ /tmp/entrada \ /tmp/salida \ 'a'
```

## Paso 5: Verificación #2 e Interpretación

Comprobar el resultado del segundo job

```
cat /tmp/salida/part-r-00000
```

# Tema 3: SSH La Clave de tu Clúster Hadoop

## ¿Por qué hadoop necesita SSH?

Hadoop es un sistema distribuido, sus componentes que están repartidos necesitan comunicarse para coordinar tareas.

Esta comunicación debe ser segura y automatizada.

La solución es el acceso SSH mediante autenticación por clave que permite conectar los nodos de forma segura.

## El Principio #1: La magia de la criptografía asimétrica

Encontramos un par de claves digitales:

- Clave Privada: Se guarda en el equipo desde que te conectas.
- Clave Pública: Es tu identidad compartida. Puedes distribuida libremente

## La Práctica: Generando Tu Identidad con “ssh-keygen”

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa): [ENTER]
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase): [ENTER]
Enter same passphrase again: [ENTER]

Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
```

Se crea el directorio .ssh para alojar tus claves.

Se generan los dos ficheros cruciales: la clave privada (id\_rsa) y la pública (id\_rsa.pub).

## El principio #2: Otorgando Confianza al Servidor

Para que un servidor te permita el acceso sin contraseña debes decirle que confíe en tu clave pública. Para ello añades tu clave pública a un fichero llamado “~/.ssh/authorized\_keys”

### La práctica: Instalando la Clave en “authorized\_keys”

Autorizamos nuestra propia clave pública en nuestro propio sistema

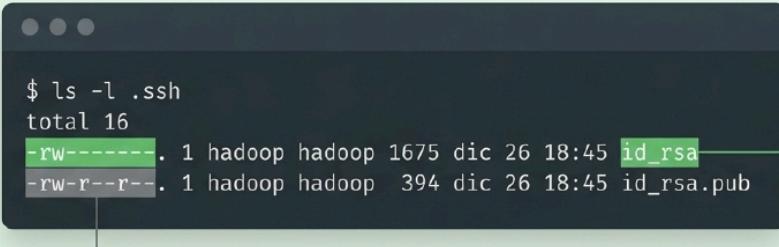
```
# Navegamos al directorio de configuración de SSH
$ cd .ssh

# Copiamos la clave pública para crear el fichero de autorización
$ cp id_rsa.pub authorized_keys
```

## Principio #3: La seguridad reside en los permisos

La seguridad desaparece si se conoce tu clave privada es conocida por otros usuarios. El protocolo SSH es estricto.

### La práctica: Inspeccionando Permisos con “ls -l”



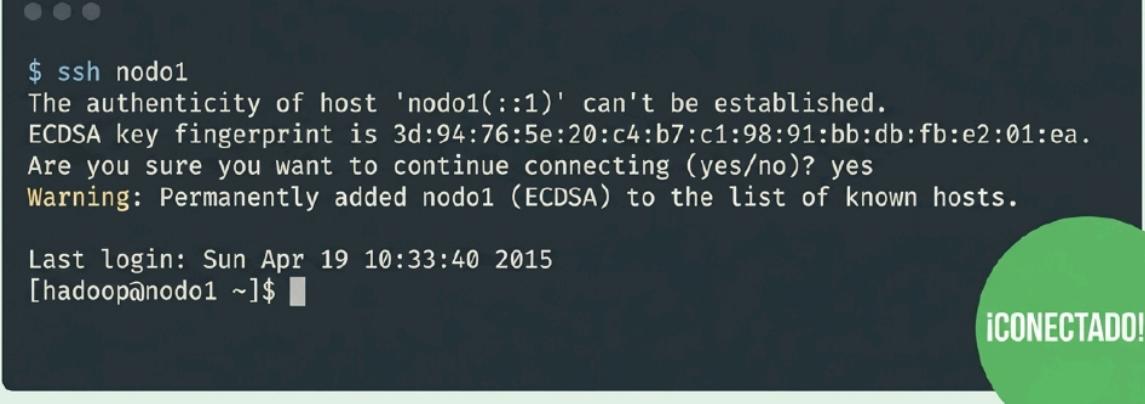
\$ ls -l .ssh  
total 16  
-rw----- 1 hadoop hadoop 1675 dic 26 18:45 id\_rsa  
-rw-r--r-- 1 hadoop hadoop 394 dic 26 18:45 id\_rsa.pub

**Correcto.** Solo el propietario ('hadoop') tiene permisos de lectura ('r') y escritura ('w'). El acceso para el grupo y otros está denegado ('---').

**Seguro.** La clave pública está diseñada para ser compartida, por lo que permisos más abiertos son aceptables y normales.

## Conexión exitosa

Con las claves generadas la autorización configurada y los permisos verificados probamos la conexión.



\$ ssh nodo1  
The authenticity of host 'nodo1(::1)' can't be established.  
ECDSA key fingerprint is 3d:94:76:5e:20:c4:b7:c1:98:91:bb:db:fb:e2:01:ea.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanent added nodo1 (ECDSA) to the list of known hosts.  
  
Last login: Sun Apr 19 10:33:40 2015  
[hadoop@nodo1 ~]\$

**iCONECTADO!** ✓

## Tema 4: Construyendo tu primer Pseudoclúster Hadoop

### El objetivo: Un laboratorio Hadoop en tu propia máquina

Un pseudoclúster ejecuta todos los componentes de Hadoop en una única máquina.

- Ideal para aprender: Permite familiarizarse con la arquitectura y comandos
- Perfecto para pruebas: Facilita el desarrollo y la depuración de aplicaciones Big Data en un entorno controlado

### Etapa 1: Preparación del entorno

Paso 1: Descargar y ubicar hadoop

Descargar hadoop de la pg oficial y extraer en un directorio del sistema

Paso 2: Configurar variables de entorno

Define rutas para que tu sistema pueda ejecutar hadoop

### Ejemplo para `~/.bashrc` o similar

```
export HADOOP_HOME=/opt/hadoop
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/
hadoop
export PATH=$PATH:$HADOOP_HOME/bin:
$HADOOP_HOME/sbin
```

## Etapa 2: Configurando el cerebro del clúster

### Archivo Clave: `core-site.xml`

#### Propósito

Este archivo contiene la configuración básica del sistema de archivos distribuido (HDFS). Su parámetro más importante, `fs.defaultFS`, indica al clúster la dirección del NameNode.

#### Ubicación

/opt/hadoop/etc/hadoop/core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

## Etapa 2: Definiendo donde viven los datos

## Archivo Clave: hdfs-site.xml

### Propósito

Aquí se configuran los directorios locales donde el NameNode almacenará sus metadatos y el DataNode guardará los bloques de datos. También se define el factor de replicación.

### Ubicación

/opt/hadoop/etc/hadoop/hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/datos/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/datos/datanode</value>
  </property>
</configuration>
```

#### Nota

Un factor de replicación de `1` es adecuado para un pseudoclúster.

## Etapa 2: Creando el espacio físico

Los directorios especificados en hdfs-site.xml deben existir en tu sistema.

```
# Crea una carpeta central para los datos de
# Hadoop
sudo mkdir -p /datos

# Crea los directorios para el NameNode y
# DataNode
sudo mkdir -p /datos/namenode
sudo mkdir -p /datos/datanode

# Asegura los permisos para el usuario que
# ejecutará Hadoop
sudo chown -R tu_usuario:tu_grupo /datos
```

## Etapa 3: Inicialización del Sistema

Antes de iniciar el clúster por primera vez, debes formatear el NameNode. Este comando solo se ejecuta la primera vez

```
hdfs namenode -format
```

## Etapa 3: A encender los motores

Iniciamos los servicios de HDFS

```
start-dfs.sh
```

## Etapa 4: Verificación por línea de comandos

Herramienta

jps (Java Virtual Machine Process Status Tool)

Propósito: Ver todos los procesos de Java que se están ejecutando

## Etapa 4: Verificación a través de la interfaz web

En el navegador escribimos la dirección: "<http://localhost:9870>" aquí encontraremos el resumen del estado del cluster HDFS

## Etapa 4: Cargar un archivo

Paso 1: Crear un archivo de prueba

```
echo "Hola Hadoop" > prueba.txt
```

Paso 2: Subir el archivo a HDFS

```
hdfs dfs -put prueba.txt /
```

Paso 3: Verificar que el archivo está en HDFS

```
hdfs dfs -ls /
```

### Resultado Esperado

Deberías ver `prueba.txt` en la lista de archivos.

```
Found 1 items
```

```
-rw-r--r-- 1 user supergroup
```

```
12 2023-10-27 10:30 /prueba.txt
```

## Tema 5: Quijote en el Data Center

### La solución conceptual: El paradigma MapReduce

MapReduce es un modelo de programación para procesar grandes conjuntos de datos. Se divide en 3 fases Map, Shuffle y Reduce

## Fase 1: Map

Cada Mapper recibe un trozo de texto, lo divide en palabras y emite una clave-valor por cada palabra.

## Fase 2 y 3: Shuffle y Reduce

- Shuffle (Agrupar): Agrupa y ordena los pares emitidos por los mappers
- Reduce: Procesa una clave y su lista de valores para generar un resultado final. WordCount suma los 1 para obtener el conteo final

## Paso 1: Preparando el Manuscrito en HDFS

Subimos el fichero `quijote.txt` a nuestro clúster Hadoop.

```
1. `hdfs dfs -mkdir -p /practicas`  
2. `hdfs dfs -put ~/Descargas/quijote.txt /practicas/`
```

## Paso 2: Lanzando el Job de MapReduce

Ejecutamos el programa WordCount indicando el fichero de entrada y salida

El programa específico a ejecutar.

```
hadoop jar <path_to_examples>.jar wordcount \  
/practicas/quijote.txt \  
/practicas/resultado
```

Fichero de entrada en HDFS.      Directorio de salida en HDFS (no debe existir previamente).

## El Resultado en HDFS: La evidencia del éxito

Una vez finalizado el job, verificamos el contenido del directorio de salida.

```
hdfs dfs -ls /practicas/resultado
```

Este fichero vacío confirma que el job terminó sin errores.

```
Found 2 items
-rw-r--r-- 3 cloudera cloudera      0 2023-10-27 10:35 /practicas/resultado/_SUCCESS
-rw-r--r-- 3 cloudera cloudera 691498 2023-10-27 10:35 /practicas/resultado/part-r-00000
```

Este es el fichero que contiene nuestro resultado. El 'r' indica que es la salida de un Reducer.

## Un vistazo al conteo: Las primeras 10 líneas

Inspeccionamos el contenido del fichero de resultados para ver las palabras más frecuentes (el resultado no está ordenado por frecuencia, sino alfabéticamente).

```
hdfs dfs -cat /practicas/resultado/part-r-00000 | head
```

"a"	21583
"abad"	14
"abadejo"	2
"abajo"	130
"abalanzó"	25
"abalorios"	3
"abarcaba"	2
"abarcar"	5
"abiertas"	55
"abierto"	72
...	

## ¿Que fase tardó más y por qué?

La fase Shuffle es la mas costosa ya que mueve grandes volúmenes de datos a través de la red desde el Mapper hasta los Reducer

# Tema 6: Dominando MapReduce

## Fase 1: Preparando el terreno



### 1. Configurar CLASSPATH

Define la variable de entorno para que Hadoop encuentre las librerías de compilación de Java.

```
export  
HADOOP_CLASSPATH=$JAVA_  
HOME/lib/tools.jar
```



### 2. Crear Workspace y Datos

Organiza el código fuente (.java) y asegura que los datos de entrada (quijote.txt, log1.log) residen en HDFS.

**Ubicación Sugerida:**  
`/practicas` en HDFS.



### 3. Activar Monitorización

Inicia el Job History Server para visualizar el progreso y los detalles de los jobs en la UI web.

**Verificación:** La interfaz web debe estar disponible en `nodo1:8088`.

## Anatomía del código: ContarPalabras.java

### El Mapper ('TokenizerMapper')

- ⚙️ **Función:** Recibe una línea de texto.
- 💡 **Lógica:** La tokeniza en palabras.
- 📦 **Salida:** Emite un par clave-valor '(palabra, 1)' por cada palabra encontrada.

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

### El Reducer ('IntSumReducer')

- ⚙️ **Función:** Recibe una palabra y una lista de '1's.
- 💡 **Lógica:** Suma todos los valores para obtener el conteo total.
- 📦 **Salida:** Emite el par final '(palabra, suma\_total)'.

```
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);  
}
```

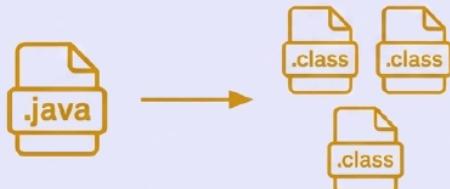
### El Driver (`main`)

- ⚙️ **Función:** Orquesta el job completo.
- 💡 **Lógica:** Configura las clases Mapper y Reducer, y define las rutas de entrada y salida en HDFS.

## Del código al paquete: Creando el archivo JAR

### Paso 1: Compilar el Código Fuente

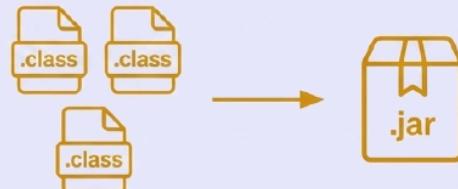
```
hadoop com.sun.tools.javac.Main  
ContarPalabras.java
```



Se generan los ficheros '.class' (uno para cada clase: 'ContarPalabras', 'TokenizerMapper', 'IntSumReducer').

### Paso 2: Empaquetar las Clases

```
jar cf mi_libreria.jar Contar*.class
```



Se crea un único archivo 'mi\_libreria.jar' que contiene todas las clases compiladas, listo para ser distribuido en el clúster.

## Lanzamiento, Monitorización y Verificación



## 1. Ejecutar el Job

```
hadoop jar mi_libreria.jar
ContarPalabras
/practicas/quijote.txt
/resultado3
```



## 2. Monitorizar el Progreso

- **Consola:** Observar el avance de las fases map.
- **Interfaz Web (nodo1:8088):** Navegar a la vista de "History" para ver detalles de los Mappers y Reducers lanzados y los nodos donde se ejecutaron.



## 3. Verificar la Salida en HDFS

```
hdfs dfs -cat
/resultado3/part-r-00000
```

Resultado Esperado:

amigo	150
batalla	87
caballero	250
...	

## Fase 3: Analizando Logs con AnalizarLog.java

**Objetivo:** Calcular el tamaño máximo, mínimo y la media de los ficheros solicitados en un log de acceso web.

### Mapper ('AMapper')

**Clave:** Utiliza una expresión regular ('httplogPattern') para parsear cada línea del log.

**Lógica Destacada:** Extrae el tamaño del fichero (grupo 5 del regex) y lo emite con una clave fija: ('"msgSize"', tamaño).

```
public static class AMapper extends Mapper<Object, Text, Text,
IntWritable> {
    private static final Pattern httplogPattern = Pattern.compile(
        "^(\\S+) - - [\\[(\\d{2})/[A-Z-a-z]{3}/\\d{4}:\\d{2}:\\d{2}"+
        "[+\\/-]\\d{4}\\]\\]) \\] \"(GET|POST) .+ HTTP/.+\" (\\d{3}) (\\d{+})"+
        "(\\d+)");
    private final IntWritable tamaño = new IntWritable();
    private final Text clave = new Text("msgSize");

    public void map(Object key, Text value, Context context) throws
        IOException, InterruptedException {
        Matcher matcher = httplogPattern.matcher(value.toString());
        if (matcher.matches()) {
            tamaño.set(Integer.parseInt(matcher.group(5)));
            context.write(clave, tamaño);
        }
    }
}
```

### Reducer ('AReducer')

**Clave:** Recibe todos los tamaños bajo la clave "msgSize".

**Lógica Destacada:** Itera sobre los valores para calcular 'min', 'max', 'sum', y 'count', y finalmente emite la media.

```
public static class AReducer extends Reducer<Text, IntWritable, Text,
DoubleWritable> {
    private final DoubleWritable media = new DoubleWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context
        Context context) throws IOException, InterruptedException {
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        long sum = 0;
        int count = 0;
        for (IntWritable val : values) {
            int v = val.get();
            if (v < min) min = v;
            if (v > max) max = v;
            sum += v;
            count++;
        }
        media.set((double) sum / count);
        context.write(key, media);
    }
}
```

## Actualización del Paquete y ejecución del análisis

## 1. Compilar el Nuevo Código

```
hadoop com.sun.tools.javac.Main AnalizarLog.java
```

## 2. Actualizar el JAR (Paso Clave)

Se utiliza el flag `uf` (update file) para añadir las nuevas clases al JAR existente sin reconstruirlo.

```
jar uf mi_libreria.jar Analizar*.class ← update file
```

## 3. Ejecutar el Nuevo Job

```
hadoop jar mi_libreria.jar AnalizarLog /practicas/log1.log /resultado_log
```

## 4. Verificar el Resultado

Se muestra la salida esperada del fichero `part-r-00000`.

Mean	1150
Max	6823936
Min	0

# Fase 4: Python y Hadoop Streaming, una Ruta Alternativa

**Concepto:** Hadoop Streaming permite usar cualquier ejecutable (como un script de Python) para las fases de Map y Reduce. La comunicación se realiza a través de la entrada/salida estándar.

### El Mapper: `pymap.py`

Lee líneas de `sys.stdin`, las divide en palabras y escribe `palabra\t1` en `sys.stdout`.

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print '%s\t1' % word
```

### El Reducer: `pyreduce.py`

Lee la salida ordenada del mapper, agrega los conteos para la misma palabra y escribe `palabra\tconteo\_total` en `sys.stdout`.

```
#!/usr/bin/env python
from operator import itemgetter
import sys
last_word = None
last_count = 0
cur_word = None
for line in sys.stdin:
    line = line.strip()
    cur_word, count = line.split('\t', 1)
    count = int(count)
    if last_word == cur_word:
        last_count += count
    else:
        if last_word:
            print '%s\t%s' % (last_word, last_count)
        last_count = count
        last_word = cur_word
    if last_word == cur_word:
        print '%s\t%s' % (last_word, last_count)
```

# Descomprimiendo el Comando de Streaming

