🧩 1. Laboratorio: Introducción a Hadoop

> **Procesamiento distribuido de texto con Hadoop MapReduce y Docker (WordCount)**

🎯 2. Objetivos de Aprendizaje

Al finalizar el laboratorio, serás capaz de:

1. Desplegar un mini-clúster de Hadoop usando Docker y `docker compose`.
2. Diferenciar entre el sistema de ficheros local del contenedor y HDFS.
3. Cargar datos de texto en HDFS desde un contenedor de Docker.
4. Ejecutar un job MapReduce de ejemplo (WordCount) sobre datos almacenados en HDFS.
5. Interpretar la salida de un job MapReduce y verificar su correcto funcionamiento.
6. Diagnosticar y resolver errores típicos de despliegue y formateo de HDFS.
7. Apagar y limpiar correctamente el entorno Docker utilizado en la práctica.

📦 3. Requisitos Previos

3.1 Conocimientos recomendados

- Conceptos básicos de:
  - Sistemas de ficheros.
  - Línea de comandos en Linux/Unix.
  - Contenedores (nociones básicas de Docker).
- Conocer, al menos a alto nivel, qué es Hadoop y qué es MapReduce.

3.2 Software y entorno

- Sistema operativo:
  - Windows, macOS o Linux (64 bits).
- Software instalado:
  - **Docker Desktop** (con soporte para `docker compose`).
- Espacio en disco:
  - Al menos 5–10 GB libres para imágenes y contenedores.
- Conexión a Internet para descargar:
  - Imágenes Docker.
  - Paquete `docker-hadoop-master.zip`.
  - Fichero `hadoop-mapreduce-examples-2.7.1-sources.jar` (si no se proporciona en clase).

3.3 Comprobar entorno

En una terminal/powershell comprueba:

```
docker --version
docker compose version
```

- Si ambos comandos devuelven versión sin error, el entorno Docker está listo.
- Si alguno falla:
    - Revisa que Docker Desktop esté instalado y en ejecución.
    - En Windows, asegúrate de que WSL2 está correctamente configurado (si aplica).

🧱 4. Arquitectura del Laboratorio

En este laboratorio se despliega un pequeño clúster de Hadoop dentro de contenedores Docker. La arquitectura lógica es:

- **Máquina anfitriona (host)**
    - Sistema operativo del estudiante.
    - Ejecuta Docker Desktop.
- **Docker**
    - Orquesta contenedores a través de `docker compose`.
- **Contenedores principales del stack Hadoop**
    - `namenode`: nodo maestro de HDFS (gestiona el sistema de ficheros distribuido).
    - `datanode`: nodo de datos de HDFS (almacena físicamente los bloques).
    - (Pueden existir otros servicios adicionales en el stack, según la plantilla, como `resourcemanager`, `historyserver`, etc., pero en esta práctica trabajaremos explícitamente con el contenedor `namenode`)
- **Almacenamiento**
    - Sistema de ficheros local del contenedor (Linux).
    - HDFS (sistema de ficheros distribuido de Hadoop) accesible mediante `hdfs dfs`.

4.1 Estructura stack de Docker

```
[Tu PC]
   └── Docker Desktop
         └── docker-hadoop stack
               ├── contenedor: namenode
               │      ├── HDFS (metadatos)
               │      └── Cliente Hadoop (hdfs, hadoop jar, etc.)
               └── contenedor: datanode
                      └── HDFS (bloques de datos)
```

4.2 Flujo de datos

1. Creas archivos de texto dentro del contenedor `namenode` (sistema de ficheros local del contenedor).
2. Copias esos archivos a HDFS mediante `hdfs dfs -put`.
3. Ejecutas el job MapReduce WordCount que:
    - Lee desde HDFS (directorio `input`).
    - Procesa y cuenta apariciones de cada palabra.
    - Escribe la salida en un directorio `output` en HDFS.
4. Visualizas los resultados con `hdfs dfs -cat`.

📝 5. Material y conceptos básicos

5.1 Hadoop y MapReduce

- **Hadoop** es un framework para procesar grandes volúmenes de datos de forma distribuida.
- **MapReduce** es el modelo de programación de Hadoop:
    - **Map**: transforma entradas en pares clave–valor.
    - **Reduce**: agrupa por clave y combina valores asociados.

5.2 WordCount

WordCount es el "hola mundo" de MapReduce. Toma uno o varios archivos de texto y:

1. Separa el texto en palabras.
2. Cuenta cuántas veces aparece cada una.
3. Devuelve una lista ordenada (o no) de `palabra frecuencia`.

Ejemplo intuitivo:

```
"El Big Data es un medio"
"El Big Data siempre en medio"
```

Se transformará a algo similar a:

```
Big      2
Data     2
El       2
medio    2
es       1
siempre  1
un       1
...
```

(La salida exacta puede variar en orden según la implementación.)

5.3 HDFS vs sistema de ficheros local

- **Sistema local del contenedor**:
  - Directorios como `/home`, `/tmp`, etc.
  - Se accede con comandos típicos: `ls`, `cat`, `mkdir`...
- **HDFS**:
  - Sistema de ficheros distribuido gestionado por Hadoop.
  - Se accede con comandos `hdfs dfs`:
    - `hdfs dfs -mkdir`
    - `hdfs dfs -ls`
    - `hdfs dfs -put`
    - `hdfs dfs -cat`

Es fundamental no confundir ambos: `hdfs dfs` trabaja sobre HDFS, no sobre el sistema de ficheros local del contenedor.

5.4 Docker y el stack Docker Hadoop

- Docker permite empaquetar Hadoop y sus dependencias en contenedores.
- `docker compose` automatiza el despliegue de varios contenedores relacionados (namenode, datanode, etc.).
- El archivo `docker-compose.yml` define:
  - Qué imágenes se usan.
  - Qué servicios se levantan.
  - Cómo se conectan entre sí.

🩹 6. Práctica guiada paso a paso

Paso 0 — Preparación del entorno

Objetivo: Verificar que Docker está correctamente instalado y operativo.

1. Abre una terminal (Linux/macOS) o PowerShell/Terminal (Windows).
2. Ejecuta:

```
docker --version
docker compose version
```

```
Docker version 20.xx.yy, build ...
Docker Compose version v2.xx.yy
```

Verificación del paso:

- Ambos comandos devuelven una versión válida sin error.

Paso 1 — Descarga y despliegue del clúster Hadoop con Docker

Objetivo: Levantar el stack docker-hadoop en segundo plano.

1. Clona el repositorio:

```
git clone https://github.com/therobotacademy/docker-hadoop-bde.git
```

2. Entra en la carpeta descomprimida:

```
cd docker-hadoop-bde
```

3. Lanza el stack con Docker Compose:

```
docker compose up -d
```

```
[+] Running 4/4
 ✔ Container namenode-bde  Started
 ✔ Container datanode-bde  Started
 ...
```

5. Comprueba los contenedores en ejecución:

```
docker ps
```

```
CONTAINER ID   IMAGE          NAMES                      STATUS
abcd1234...    ...namenode    docker-hadoop-namenode-1   Up ...
efgh5678...    ...datanode    docker-hadoop-datanode-1   Up ...
...
```

Verificación del paso:

- En `docker ps` aparecen al menos `namenode` y `datanode` con estado `Up`.

Paso 2 — Preparación de datos en el contenedor namenode

Objetivo: Crear archivos de texto de prueba en el sistema local del contenedor `namenode`.

1. Entra en el contenedor `namenode`:

```
docker compose exec namenode bash
```

2. Dentro del contenedor, crea un directorio `input`:

```
mkdir input
```

3. Crea los archivos de texto de ejemplo:

```
echo "El Big Data es un medio, nunca un fin" > input/textoA.txt
echo "El Big Data siempre en medio, no al fin" > input/textoB.txt
```

4. Comprueba su contenido:

```
ls input
cat input/textoA.txt
cat input/textoB.txt
```

Salida esperada (ejemplo):

```
textoA.txt  textoB.txt

El Big Data es un medio, nunca un fin
El Big Data siempre en medio, no al fin
```

Errores frecuentes:

- Escribir mal la ruta (`input` vs `imput`, etc.):
    - Revisa cuidadosamente el nombre del directorio.

Verificación del paso:

- El comando `ls input` muestra `textoA.txt` y `textoB.txt`.
- `cat` muestra las frases esperadas.

Paso 3 — Carga de datos en HDFS

Objetivo: Crear un directorio en HDFS y copiar los archivos de texto desde el sistema local del contenedor.

1. Asegúrate de seguir dentro del contenedor `namenode` (el prompt suele incluir algo como `root@namenode`).

2. Crea el directorio `input` en HDFS:

```
hdfs dfs -mkdir -p /user/root/input
```

3. Copia los archivos desde el directorio local `input` a HDFS:

```
hdfs dfs -put input/* input
```

4. Verifica el contenido del directorio `input` en HDFS:

```
hdfs dfs -ls input
```

Salida:

```
Found 2 items
-rw-r--r--   1 root supergroup         48 ... input/textoA.txt
-rw-r--r--   1 root supergroup         51 ... input/textoB.txt
```

Verificación del paso:

- `hdfs dfs -ls input` muestra los dos archivos.

Paso 4 — Ejecución del proceso WordCount en Hadoop

Objetivo: Ejecutar un job MapReduce de ejemplo sobre los archivos almacenados en HDFS.

Copiar el JAR de ejemplos al contenedor namenode

1. Sal del contenedor `namenode` si aún estás dentro:

```
exit
```

2. Verifica los contenedores en ejecución:

```
docker ps
```

   Localiza el contenedor del `namenode` (nombre similar a `docker-hadoop-namenode-1`).

3. Copia el JAR de ejemplos al contenedor `namenode`:

```
docker cp hadoop-mapreduce-examples-2.7.1-sources.jar namenode:/hadoop-
mapreduce-examples-2.7.1-sources.jar
```

> Nota: en algunos entornos, el nombre real del contenedor puede no ser exactamente `namenode`. Si ves otro nombre (por ejemplo, `docker-hadoop-master-namenode-1`), usa ese en lugar de `namenode` en el comando `docker cp`.

Volver a entrar en namenode y ejecutar WordCount

4. Entra de nuevo en el contenedor:

```
docker compose exec namenode bash
```

5. Ejecuta el job WordCount:

```
hadoop jar hadoop-mapreduce-examples-2.7.1-sources.jar
org.apache.hadoop.examples.WordCount input output
```

- Parámetros:
  - `hadoop jar <archivo.jar>`: ejecuta un JAR de Hadoop.
  - `org.apache.hadoop.examples.WordCount`: clase principal a ejecutar.
  - `input`: directorio de entrada en HDFS.
  - `output`: directorio de salida en HDFS (no debe existir previamente).

Salida:

```
...
INFO mapreduce.Job:  map 100% reduce 100%
INFO mapreduce.Job: Job job_... completed successfully
...
```

Si aparece un error indicando que el directorio `output` ya existe:

```
File output/output already exists
```

entonces bórralo antes de volver a ejecutar:

```
hdfs dfs -rm -r output
```

y lanza de nuevo el comando `hadoop jar`.

Ver el resultado del WordCount

6. Lista el directorio `output` en HDFS:

   ```
   hdfs dfs -ls output
   ```

   Deberías ver un archivo tipo `part-r-00000`.

7. Muestra el contenido:

   ```
   hdfs dfs -cat output/*
   ```

Salida esperada (ejemplo):

```
Big     2
Data    2
El      2
medio,  1
medio   1
...
```

(Los tokens exactos pueden variar por signos de puntuación y normalización. Lo importante es que veas cada palabra con su número de ocurrencias.)

Verificación del paso:

- Se completan las fases `map 100%` y `reduce 100%` sin errores.
- `hdfs dfs -cat output/*` muestra líneas de `palabra frecuencia`.

Paso 5 — Cierre y limpieza del entorno

Objetivo: Detener y eliminar los contenedores del clúster Hadoop.

1. Sal del contenedor `namenode` si sigues dentro:

   ```
   exit
   ```

2. Desde la carpeta `docker-hadoop-master`, apaga el stack:

   ```
   docker compose down
   ```

Salida esperada (ejemplo):

```
[+] Stopping containers ...
[+] Removing containers ...
[+] Removing network ...
```

3. Comprueba que no quedan contenedores activos del stack:

   ```
   docker ps
   ```

Verificación del paso:

- No aparece ninguno de los contenedores del clúster Hadoop en `docker ps`.

Paso 6 — Extensión del ejercicio

Si tienes tiempo, prueba:

1. Modificar los archivos `textoA.txt` y `textoB.txt` (añadir más frases).
2. Añadir un `textoC.txt` con otras frases sobre Big Data.
3. Volver a cargar los datos en HDFS (borrando antes `input` y `output` si es necesario):

   ```
   hdfs dfs -rm -r input output
   hdfs dfs -mkdir -p /user/root/input
   hdfs dfs -put input/* input
   ```

4. Ejecutar de nuevo el WordCount y observar cómo cambian las frecuencias.

🔍 7. Preguntas de Autoevaluación

Responde brevemente a las siguientes cuestiones:

1. ¿Qué papel juega Docker en este laboratorio y qué ventaja ofrece frente a instalar Hadoop directamente en tu sistema operativo?
2. Explica la diferencia entre el sistema de ficheros local del contenedor y HDFS. ¿Por qué necesitamos usar `hdfs dfs`?
3. ¿Por qué es necesario que el directorio de salida (`output`) no exista antes de ejecutar un job MapReduce?
4. ¿Qué información principal obtienes de la salida de WordCount? ¿Cómo podrías usarla en un caso real?
5. ¿En qué situación se hace necesario formatear el namenode con `hdfs namenode -format`? ¿Qué implicaciones tiene?
6. ¿Qué podría ocurrir si el contenedor `datanode` no estuviera en ejecución cuando lanzas el job MapReduce?

7. En arquitecturas ARM, ¿por qué a veces es necesario forzar `linux/amd64` al construir/levantar las imágenes?
8. ¿Qué pasos concretos seguirías para volver a ejecutar el laboratorio desde cero (desde que no hay contenedores) hasta obtener de nuevo la salida de WordCount?