

---

# BIG DATA

## HBASE



EDUARD LARA

# 1. INTRODUCCION A HBASE

---

- ❑ Hbase es una de las herramientas más interesantes dentro del ecosistema de Hadoop
- ❑ Es una base de datos tipo NoSQL que se ejecuta sobre HDFS. Está muy integrada con la infraestructura y la arquitectura de HDFS Hadoop, Zookeeper, etc..
- ❑ Es enormemente potente para trabajar con tablas y con datos muy grandes de manera masiva de tipo petabytes.

# 1. CARACTERÍSTICAS HBASE

---

- ❑ Es una base datos muy fault tolerant
- ❑ Tiene una alta disponibilidad, conseguida gracias que está sobre Hadoop y que utiliza todas sus técnicas
- ❑ Es muy escalable y tiene un gran rendimiento cuando trabaja con grandes cantidades de información.
- ❑ Es capaz de manejar tablas muy grandes y masivas de forma sencilla
- ❑ Está adaptado a filas dispersas (sparses rows) donde puede variar el número de columnas (impensable en una base de datos típica relacional).
- ❑ Se trata de un proyecto "top" de Apache: ha pasado de ser un proyecto secundario a un proyecto principal dentro de Apache

# 1. CARACTERÍSTICAS HBASE

---

## NoSQL

- ❑ NoSQL, que significa "not only SQL". No quiere decir que estas bases de datos no contengan SQL sino que no están basadas principalmente en SQL.
- ❑ Cuando contienen SQL, suelen tener muchas diferencias con SQL tradicional
- ❑ Las bases de datos NoSQL permiten representar datos distintos a los formatos tabulares, que son los utilizados por bases de datos relacionales
- ❑ La mayoría de las bases de datos NoSQL priman la disponibilidad y la velocidad sobre la consistencia.

# 1. CARACTERÍSTICAS HBASE

---

- ❑ Hay diferentes tipos de bases de datos NoSQL basadas en su formato de representación.
  - ❑ Clave-valor
  - ❑ Orientadas a documentos
  - ❑ Orientadas a columnas
  - ❑ Orientadas a grafos
- ❑ Bases de datos NoSQL: Hbase, Cassandra, MongoDB, Elastic Search, etc. Compiten con bases de datos tradicionales como Oracle, DB2, PostgreSQL, etc..
- ❑ Ninguna es mejor que otra. Cada una tiene una funcionalidad concreta y unas determinadas características que la hacen útil para determinados productos o componentes

# 1. CARACTERÍSTICAS HBASE

---

- ❑ Hbase está basada en BigTable de Google, concepto creado hace ya tiempo para poder realizar grandes búsquedas en Internet.
- ❑ Dispone de una gran capacidad de almacenamiento (terabytes, petabytes), ideal para mover todos los datos que vienen de Internet
- ❑ Permite una alta velocidad en lectura y escritura, sobretudo de tipo random
- ❑ Dispone de una gran escalabilidad en memoria caché, utiliza mucho la memoria cache. Almacena los datos más leídos en memoria

# 1. CARACTERÍSTICAS HBASE

---

- ❑ Si se necesita acceder de manera más o menos random en modo lectura y escritura a grandes cantidades de información, y que esta no este muy estructurada, HBASE es una elección excelente
- ❑ Si lo que queremos es modificar datos o tenemos muchos datos transaccionales con muchas lecturas secuenciales, HBASE no es la más indicada
- ❑ No dispone de determinadas características de una RDBMS, como índices normales, transaccionalidad, lenguaje SQL, etc...
- ❑ HBASE está enfocada a tratar grandes cantidades de información distribuida, en modo lectura y escritura pero no transaccional.

# 1. CARACTERÍSTICAS HBASE

---

- ❑ Hay una gran cantidad de empresas que usan hoy en día Hbase
  - ❑ eBay
  - ❑ Facebook
  - ❑ Twitter
  - ❑ Etc....



# 1. TIPOS INSTALACION HBASE

---

Hay 4 posibles maneras de instalar HBASE:

- La forma standalone es cuando se instala individualmente en una computadora, sin cluster HBASE. Tiene dos posibilidades

	Características
Standalone	Se utiliza para desarrollo. Todos los procesos de Hbase se ejecutan en local, dentro de la misma JVM. Se usa el sistema de ficheros local
Standalone sobre HDFS	Igual que el anterior, pero para los datos se utiliza HDFS de Hadoop

# 1. TIPOS INSTALACION HBASE

---

- La forma de instalación distribuida es cuando queremos trabajar con un clúster de verdad pero también tiene dos posibilidades:

	Características
Pseudo-distribuido	Los procesos de HBASE son independientes entre sí, pero se ejecutan todos en el mismo nodo. Creamos un clúster de HBASE en un solo nodo, igual que con el clúster de Hadoop de un solo nodo
Fully-Distributed	Los procesos de HBASE se ejecutan en distintos nodos de un clúster normal (con maestros y esclavos). Aquí es obligatorio trabajar con HDFS. En los 3 anteriores se puede trabajar en modo sistema de ficheros local

## 2. CONCEPTOS DE HBASE

---

Veremos con más detalle cómo funciona HBASE de forma interna y sus conceptos principales.

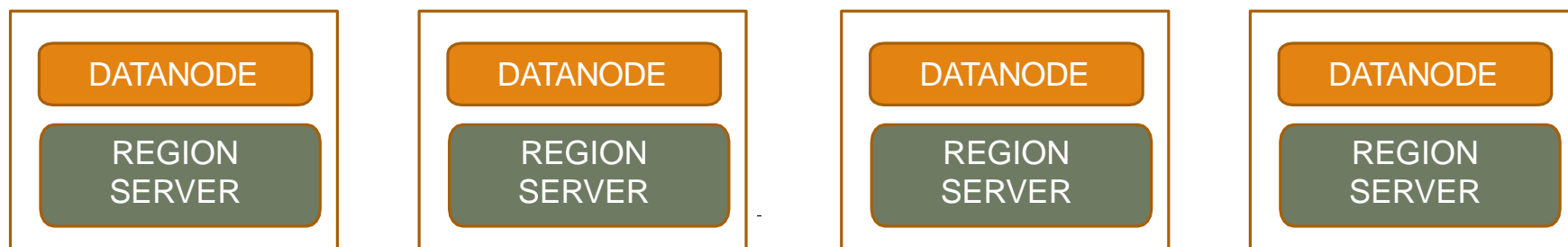
CONCEPTOS	DEFINICION
Nodo	Un servidor dentro del cluster de HBASE, similar al nodo de Hadoop
Clúster	Un grupo de servidores trabajando de forma conjunta y coordinados por ciertos nodos. Como está tan integrado con Hadoop en nuestros servidores además de los procesos de Hbase tendremos los de Hadoop (datanode, namenode, zookeeper, etc..)
Nodo Maestro	Nodo que se encarga de la coordinación de las tareas
Nodo Esclavo	Servidores que ejecutan las tareas

# Arquitectura HBASE

## SERVIDORES MAESTROS



## SERVIDORES ESCLAVOS



## 2. ARQUITECTURA DE HBASE

---

### Servidores maestros

- ❑ Son servidores que se conectan a Zookeeper, que se utiliza conjuntamente en muchas herramientas Hadoop
- ❑ Un proceso denominado master va a conectarse a zookeeper para convertirse en el coordinador de todo el clúster. En realidad pueden haber distintos maestros de tipo HBASE, siendo uno de ellos será el que tomará el control.
- ❑ Normalmente el primero que se pone en contacto con zookeeper se le nombrará coordinador del clúster.
- ❑ Este proceso coordinará todos los trabajos de esta base de datos

## 2. ARQUITECTURA DE HBASE

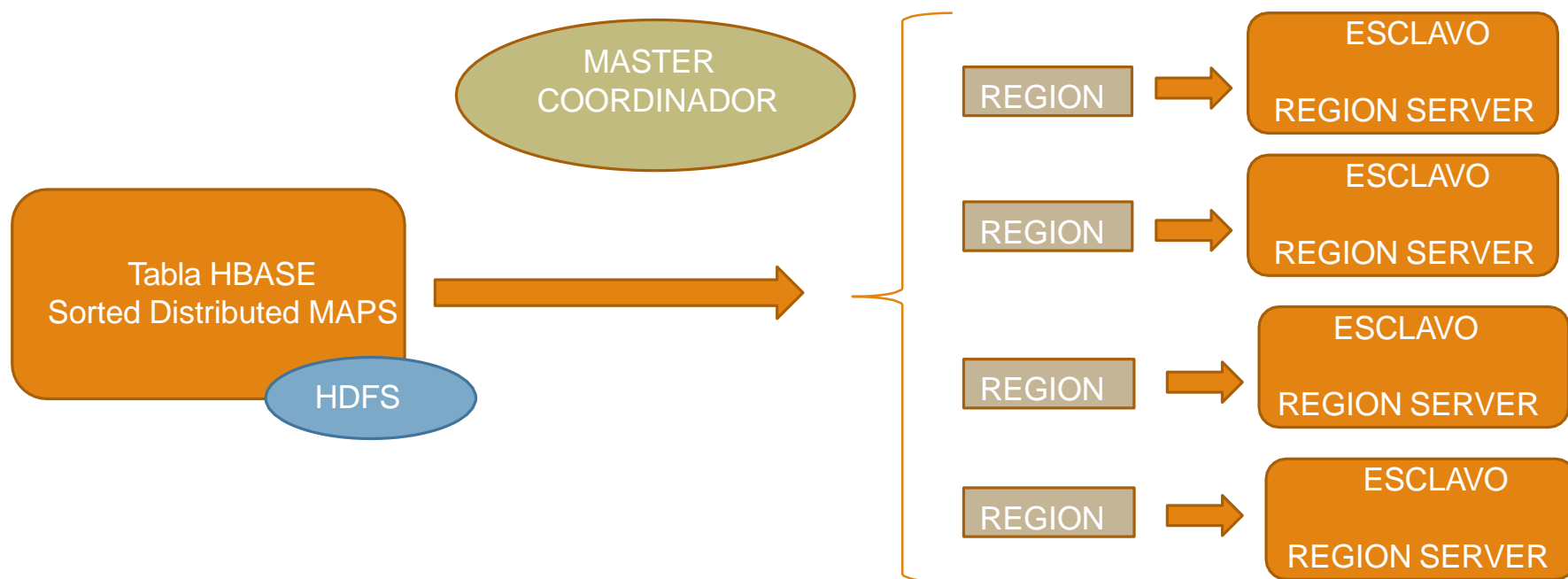
---

### Servidores esclavos

- ❑ Están compuestos de distintos recursos:
  - ❑ Datanode. Tendremos nuestros datos de HBASE repartidas por los nodos de Hadoop.
  - ❑ Region Server (servidor de regiones). Componente muy especial que es realmente el encargado de distribuir y de gestionar la información, los datos que tenemos dentro de nuestras tablas HBASE.

## 2. ARQUITECTURA DE HBASE

### Funcionamiento Arquitectura HBASE



## 2. ARQUITECTURA DE HBASE

---

### Funcionamiento Arquitectura HBASE

- ❑ El master-coordinador, es el proceso que va a coordinar todo el clúster de HBASE.
- ❑ Las tablas HBASE son de tipo Sorted Distributed Maps (mapas distribuidos y ordenados) parecidos a las Hashes o Maps de otros lenguajes. Estarán guardadas en HDFS si se ha realizado una instalación completamente distribuida y no standalone.
- ❑ Las tablas HBASE se particionan por la clave y se van repartiendo en lo que se denominan regiones.
- ❑ Estas regiones se va guardando en los nodos esclavos de Hadoop



## 2. ARQUITECTURA DE HBASE

---

- ❑ En los nodos esclavos de Hadoop, además de lo que ya tenga (datanode, etc) tendrá el proceso Region Server que se coordina con el master para gestionar todos los datos que guarda dentro de este esclavo.
- ❑ El Region Server va a servir a varias regiones, y esas regiones pertenecerán a una tabla o varias tablas.
- ❑ La arquitectura completa de HBASE queda definida con los Region servers y el master-coordinador

La característica de HBASE, de dividir la información entre los distintos esclavos, con los diferentes Region Server que la tratan, hace que sea tan rápida y que sea capaz de manejar la enorme cantidad de datos para la que está preparada

### 3. INSTALACION STANDALONE

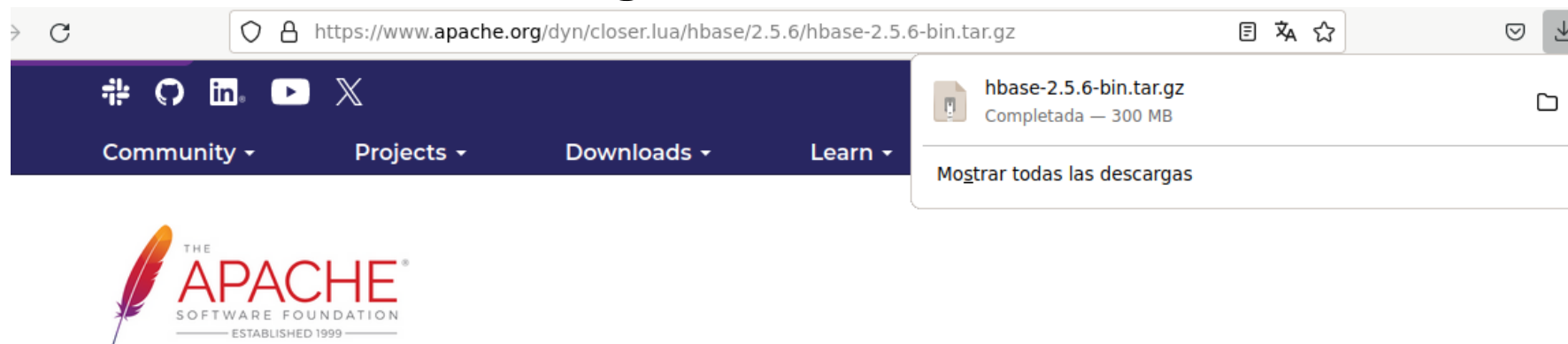
Instalaremos la base datos HBASE en modo standAlone, de modo independiente, sin depender de Hadoop ni HDFS

**Paso 1.** Vamos a la pagina de descarga de HBASE  
<https://hbase.apache.org/downloads.html>

Downloads					
The below table lists mirrored release artifacts and their associated hashes and signatures available ONLY at apache.org. The keys used to sign releases can be found in our published <a href="#">PGP keys</a> for your mirrored downloads.					
Releases					
Version	Release Date	Compatibility Report	Changes	Release Notes	Download
3.0.0-alpha-4	2023/06/07	<a href="#">3.0.0-alpha-4 vs 2.0.0</a>	<a href="#">Changes</a>	<a href="#">Release Notes</a>	<a href="#">src</a> (sha512 asc) <a href="#">bin</a> (sha512 asc) <a href="#">client-bin</a> (sha512 asc)
2.5.6	2023/10/20	<a href="#">2.5.6 vs 2.5.5</a>	<a href="#">Changes</a>	<a href="#">Release Notes</a>	<a href="#">src</a> (sha512 asc) <a href="#">bin</a> (sha512 asc) <a href="#">client-bin</a> (sha512 asc) <a href="#">hadoop3-bin</a> (sha512 asc) <a href="#">hadoop3-client-bin</a> (sha512 asc)
2.4.17	2023/04/06	<a href="#">2.4.17 vs 2.4.16</a>	<a href="#">Changes</a>	<a href="#">Release Notes</a>	<a href="#">src</a> (sha512 asc) <a href="#">bin</a> (sha512 asc) <a href="#">client-bin</a> (sha512 asc)

## 3. INSTALACION STANDALONE

**Paso 2.** Descargamos la ultima versión estable:  
`hbase-2.5.6-bin.tar.gz`



We suggest the following location for your download:

<https://dlcdn.apache.org/hbase/2.5.6/hbase-2.5.6-bin.tar.gz>

Alternate download locations are suggested below.

It is essential that you [verify the integrity](#) of the downloaded file using the PGP signature ( `.asc` file) or a hash ( `.md5` or `.sha*` file).

### HTTP

<https://dlcdn.apache.org/hbase/2.5.6/hbase-2.5.6-bin.tar.gz>

### BACKUP SITES

<https://dlcdn.apache.org/hbase/2.5.6/hbase-2.5.6-bin.tar.gz>

### 3. INSTALACION STANDALONE

---

**Paso 3.** En el directorio Downloads descomprimos el fichero tar.gz de binarios de HBASE, lo renombramos a hbase y lo llevamos a /opt/hadoop, como hemos hecho con cada uno de los productos que hemos instalado

```
hbase-2.5.6/lib/jdk11/commonj.sdo-2.1.1.jar
hbase-2.5.6/lib/jdk11/release-documentation-2.3.2-docbook.zip
hbase-2.5.6/lib/jdk11/samples-2.3.2.zip
hbase-2.5.6/lib/jdk11/jakarta.xml.ws-api-2.3.2.jar
hbase-2.5.6/lib/jdk11/jakarta.xml.bind-api-2.3.2.jar
hbase-2.5.6/lib/jdk11/jakarta.xml.soap-api-1.4.1.jar
hbase-2.5.6/lib/jdk11/jakarta.jws-api-1.1.1.jar
hadoop@nodol1:~/Downloads$ tar xvf hbase-2.5.6-bin.tar.gz hbase-2.5.6/
```

```
hadoop@nodol1:~/Downloads$ mv hbase-2.5.6 hbase
hadoop@nodol1:~/Downloads$ ls
access_log                                hadoop-3.2.4.tar.gz  hbase-2.5.6-bin.tar.gz
apache-hive-3.1.3-bin.tar.gz              hadoop-3.3.6.tar.gz  hue-4.10.0
apache-zookeeper-3.8.3-bin.tar.gz         hbase
hadoop@nodol1:~/Downloads$ mv hbase /opt/hadoop
hadoop@nodol1:~/Downloads$ █
```

### 3. INSTALACION STANDALONE

**Paso 4.** Si hacemos un ls dentro de hbase observamos los típicos directorios:

```
hadoop@nodol:/opt/hadoop$ cd hbase/
hadoop@nodol:/opt/hadoop/hbase$ ls
bin          conf  hbase-webapps  lib          NOTICE.txt
CHANGES.md docs  LEGAL          LICENSE.txt  RELEASNOTES.md
hadoop@nodol:/opt/hadoop/hbase$ ls -al
total 2448
drwxrwxr-x  7 hadoop hadoop    4096 nov 26 10:21 .
drwxr-xr-x 15 hadoop hadoop    4096 nov 26 10:24 ..
drwxr-xr-x  4 hadoop hadoop    4096 ene 22  2020 bin
-rw-r--r--  1 hadoop hadoop 1090123 ene 22  2020 CHANGES.md
drwxr-xr-x  2 hadoop hadoop    4096 ene 22  2020 conf
drwxr-xr-x 10 hadoop hadoop    4096 ene 22  2020 docs
drwxr-xr-x  8 hadoop hadoop    4096 ene 22  2020 hbase-webapps
-rw-r--r--  1 hadoop hadoop    262 ene 22  2020 LEGAL
drwxrwxr-x  8 hadoop hadoop   12288 nov 26 10:21 lib
-rw-r--r--  1 hadoop hadoop  139942 ene 22  2020 LICENSE.txt
-rw-r--r--  1 hadoop hadoop  602236 ene 22  2020 NOTICE.txt
-rw-r--r--  1 hadoop hadoop  611188 ene 22  2020 RELEASNOTES.md
hadoop@nodol:/opt/hadoop/hbase$
```

- bin : tenemos los binarios
- conf: tenemos los ficheros de configuración
- hbase-webapps: la parte grafica de hbase que nos permite acceder en modo web (Web UI)
- lib: librerias hbase
- docs: documentacion hbase

### 3. INSTALACION STANDALONE

---

**Paso 5.** En el directorio bin tenemos comandos que nos permite hacer determinadas operaciones sobre Hbase: arrancarlo, pararlo, etc.

```
hadoop@nodol:/opt/hadoop/hbase$ cd bin/  
hadoop@nodol:/opt/hadoop/hbase/bin$ ls  
chaos-daemon.sh      hbase-daemon.sh      replication  
considerAsDead.sh    hbase-daemons.sh    rolling-restart.sh  
draining_servers.rb  hbase-jruby           shutdown_regionserver.rb  
get-active-master.rb hbase_startup.jsh     start-hbase.cmd  
graceful_stop.sh     hirb.rb               start-hbase.sh  
hbase                local-master-backup.sh stop-hbase.cmd  
hbase-cleanup.sh     local-regionervers.sh stop-hbase.sh  
hbase.cmd            master-backup.sh      test  
hbase-common.sh      region_mover.rb       zookeepers.sh  
hbase-config.cmd     regionervers.sh  
hbase-config.sh      region_status.rb  
hadoop@nodol:/opt/hadoop/hbase/bin$
```

### 3. INSTALACION STANDALONE

---

**Paso 6.** Para empezar a trabajar con Hbase en modo standalone (donde todos los procesos Region Server y master, se van a ejecutar en una sola máquina), lo primero que tenemos que hacer es configurar el fichero hbase-site.xml.

```
See also https://hbase.apache.org/book.html#standalone\_dist
-->
<property>
  <name>hbase.cluster.distributed</name>
  <value>false</value>
</property>
<property>
  <name>hbase.tmp.dir</name>
  <value>./tmp</value>
</property>
<property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>false</value>
</property>
</configuration>
```

**NOTA:** Guardaremos los datos en un fichero local en un directorio local, todavía no utilizaremos HDFS

### 3. INSTALACION STANDALONE

---

**Paso 7.** Para poder empezar a trabajar con Hbase y usarlo en modo standalone tenemos que crear tres propiedades:

- 1) hbase.rootdir → donde HBASE va a dejar los datos. Indicamos /opt/hadoop/hbase/data/hbase, el cual HBASE creará de forma automática.

```
<property>  
  <name>hbase.rootdir</name>  
  <value>file:///opt/hadoop/hbase/data/hbase</value>  
</property>
```

- 2) hbase.zookeeper.property.dataDir → dónde deja los datos del zookeeper. Indicaremos /opt/hadoop/hbase/data/zookeeper

```
<property>  
  <name>hbase.zookeeper.property.dataDir</name>  
  <value>/opt/hadoop/hbase/data/zookeeper</value>  
</property>
```



### 3. INSTALACION STANDALONE

---

- 3) `hbase.unsafe.stream.capability.enforce` → una propiedad muy importante que habilita para poder trabajar en modo sistema de ficheros local. Si no se indica comienza a dar errores

```
<property>  
  <name>hbase.unsafe.stream.capability.enforce</name>  
  <value>false</value>  
</property>
```

## 4. ARRANQUE STANDALONE

---

**Paso 1.** Arrancaremos Hbase en modo standalone para empezar a hacer nuestras primeras pruebas. Vamos al directorio `/opt/hadoop/hbase/bin` y ejecutamos el comando `start-hbase.sh`

```
hadoop@nodol1:/opt/hadoop/hbase/bin$ ./start-hbase.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.2.jar:]
SLF4J: Found binding in [jar:file:/opt/hadoop/hbase/lib/client-facing-thirdparty/log4j-slf4j-impl-1.7.12.jar:]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
running master, logging to /opt/hadoop/hbase/bin/../logs/hbase-hadoop-master-nodol1.out
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.2.jar:]
SLF4J: Found binding in [jar:file:/opt/hadoop/hbase/lib/client-facing-thirdparty/log4j-slf4j-impl-1.7.12.jar:]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
2023-11-26 16:09:24,443 INFO [main] master.HMaster (HMaster.java:main(3302)) - STARTING
2023-11-26 16:09:24,447 INFO [main] util.VersionInfo (VersionInfo.java:logVersion(112))
2023-11-26 16:09:24,447 INFO [main] util.VersionInfo (VersionInfo.java:logVersion(112))
842797dc26bedb7adc0759358e4c8fd5a992
2023-11-26 16:09:24,447 INFO [main] util.VersionInfo (VersionInfo.java:logVersion(112))
2023-11-26 16:09:24,447 INFO [main] util.VersionInfo (VersionInfo.java:logVersion(112))
64852d9e883beaae87c852def62eba7910d56eb9cc5f6cf56dd46b242ef63
hadoop@nodol1:/opt/hadoop/hbase/bin$
```

## 4. ARRANQUE STANDALONE

---

**NOTA:** Sale el warning "**Class Path contains multiple SLF4J bindings**". Nos indica que ha encontrado 2 librerías de Logger dentro de su Path, una la que corresponde a Hadoop y otra la que corresponde al propio producto al propio Hbase. En la vida real posiblemente se tendría que tocar el classpath para que no surgiera.

**Paso 2.** Si hacemos JPS vemos que tenemos el proceso HMaster que se está ejecutando dentro del servidor.

```
hadoop@nodo1:/opt/hadoop/hbase/bin$ jps
380403 HMaster
393486 Jps
hadoop@nodo1:/opt/hadoop/hbase/bin$ █
```

Como estamos en modo standAlone, HBASE mete todo en el mismo proceso: el master, el region Server, zookeeper

## 4. ARRANQUE STANDALONE

---

**Paso 3.** Tenemos dos formas de ver que Hbase funciona:

- Una desde el modo comando con hbase Shell
- Otra desde un entorno web

La manera más rápida de comprobar que Hbase funciona es ejecutando `/bin/hbase shell`. Aparece una especie de línea de comandos o shell.

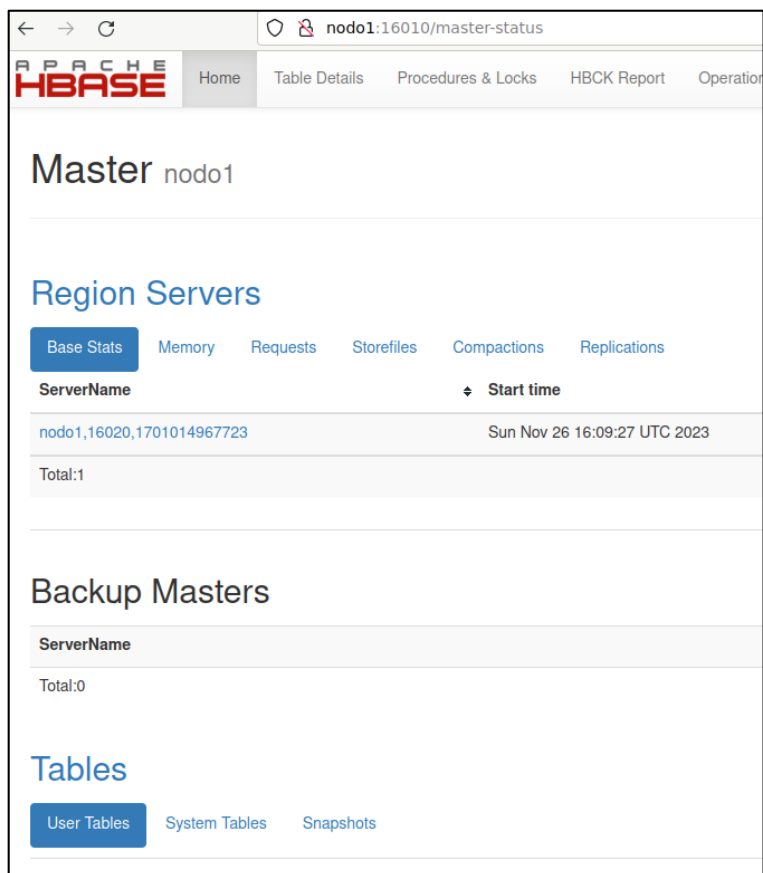
```
hadoop@nodo1:/opt/hadoop/hbase/bin$ ./hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop
SLF4J: Found binding in [jar:file:/opt/hadoop/hbase/lib/c
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindi
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4j
2023-11-26 16:32:41,623 INFO [main] Configuration.deprec
```

Se conecta  
correctamente porque ha  
arrancado con `hbase:001`

```
t complete on server localhost/127.0.0.1:2181, sessionId = 0x10001608d690002, negotiated
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.5.6, r6bac842797dc26bedb7adc0759358e4c8fd5a992, Sat Oct 14 23:36:46 PDT 2023
Took 0.0010 seconds
hbase:001:0> █
```

## 4. ARRANQUE STANDALONE

**Paso 4.** Otra opción para comprobar que Hbase funciona es ir a un navegador y poner localhost:16010. El puerto 16010 es el puerto de la interfaz gráfica de Hbase



The screenshot shows the Apache HBase web interface. The browser address bar displays 'nodo1:16010/master-status'. The interface includes a navigation bar with 'Home', 'Table Details', 'Procedures & Locks', 'HBCK Report', and 'Operation'. The main content area is titled 'Master nodo1'. Below this, there is a section for 'Region Servers' with tabs for 'Base Stats', 'Memory', 'Requests', 'Storefiles', 'Compactions', and 'Replications'. The 'Base Stats' tab is active, showing a table with columns 'ServerName' and 'Start time'. One server is listed: 'nodo1,16020,1701014967723' with a start time of 'Sun Nov 26 16:09:27 UTC 2023'. The total count is 'Total:1'. Below this is a section for 'Backup Masters' with a table showing 'ServerName' and 'Total:0'. At the bottom, there is a section for 'Tables' with tabs for 'User Tables', 'System Tables', and 'Snapshots'.

Es un entorno web donde podemos ver un resumen de todo lo que tenemos dentro de nuestra instalación: los Region Server, los master, las tablas, atributos y propiedades de Hbase, etc

## 4. ARRANQUE STANDALONE

**Paso 5.** Podemos comprobar que tenemos ahora un nodo funcionando, el nodo1, donde hemos preparado y arrancado HBase y la información de la última vez que se contactó, versión, numero de regiones, etc..

### Region Servers

Base Stats

Memory

Requests

Storefiles

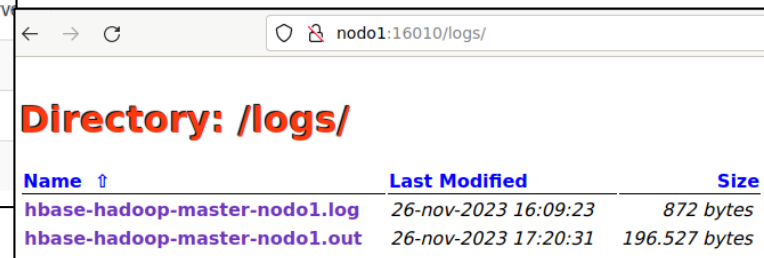
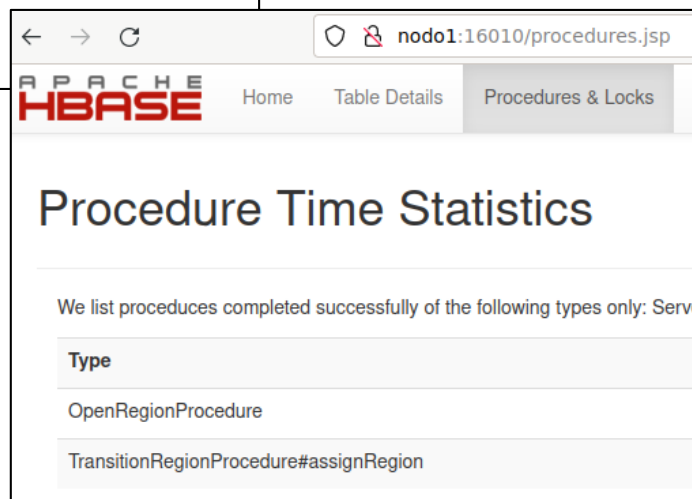
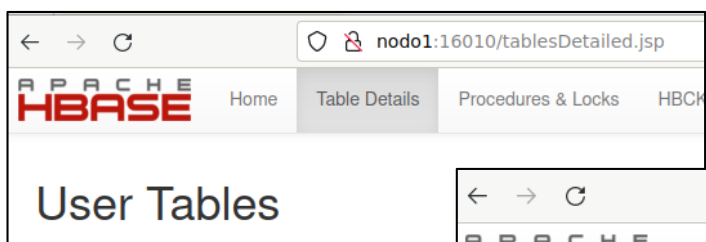
Compactions

Replications

ServerName	Start time	Last contact	Version	Requests Per Second	Num. Regions
nodo1,16020,1701014967723	Sun Nov 26 16:09:27 UTC 2023	1 s	2.5.6	0	2
Total:1				0	2

## 4. ARRANQUE STANDALONE

**Paso 6.** En el menú superior tenemos información de tablas (donde vemos que no tenemos ninguna), métricas, procedimientos, logs, etc. Desde aquí tenemos acceso a todos los datos de Hbase, pero no podemos interactuar como si lo podemos hacer desde el modo comando



## 4. ARRANQUE STANDALONE

---

**Paso 7.** Para ver que todo funciona, desde línea de comandos creamos una tabla con **create 'prueba', 'cf'**  
Con este comando creamos una tabla llamada Prueba.  
Cf significa ColumnFamily ó Familia de columnas

```
hbase:002:0> create 'prueba','cf'
2023-11-26 17:44:50,920 INFO [ReadOnlyZKClient-127.0.0.1:2181@
181 sessionTimeout=90000 watcher=org.apache.hadoop.hbase.zookeeper
2023-11-26 17:44:50,920 INFO [ReadOnlyZKClient-127.0.0.1:2181@
ytes
2023-11-26 17:44:50,920 INFO [ReadOnlyZKClient-127.0.0.1:2181@
ature enabled=
2023-11-26 17:44:50,923 INFO [ReadOnlyZKClient-127.0.0.1:2181@
onnection to server localhost/127.0.0.1:2181. Will not attempt
2023-11-26 17:44:50,925 INFO [ReadOnlyZKClient-127.0.0.1:2181@
established, initiating session, client: /127.0.0.1:47832, ser
2023-11-26 17:44:50,928 INFO [ReadOnlyZKClient-127.0.0.1:2181@
t complete on server localhost/127.0.0.1:2181, sessionid = 0x10
2023-11-26 17:44:51,925 INFO [main] client.HBaseAdmin (HBaseAd
Created table prueba
Took 1.0279 seconds
=> Hbase::Table - prueba
hbase:003:0>
```

No se pone table en  
create porque no es una  
base de datos relacional  
En Hbase no hay índices  
ni vistas ni otras cosas  
que tenemos dentro de  
una base de datos  
tradicional.

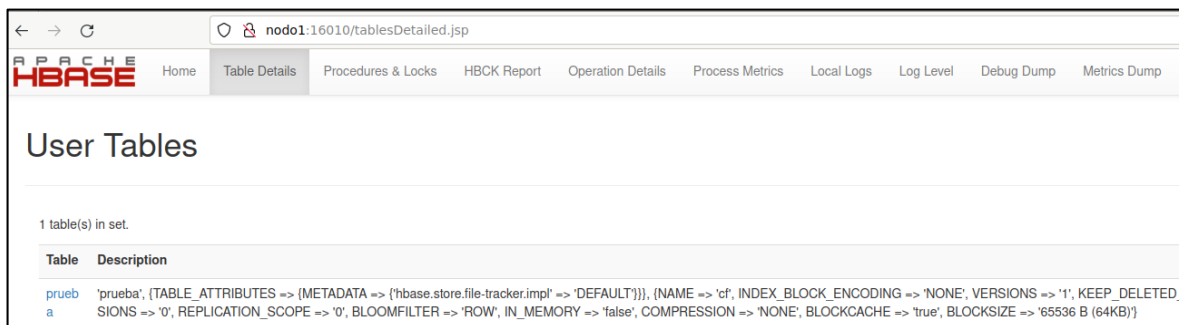


## 4. ARRANQUE STANDALONE

**Paso 8.** Con el comando List, vemos que tenemos una tabla llamada prueba

```
hbase:004:0> list
TABLE
prueba
1 row(s)
Took 0.0060 seconds
=> ["prueba"]
hbase:005:0> █
```

**Paso 9.** Si vamos al modo web de Hbase, en la opción table details podemos comprobar que la tabla sale aquí. Hbase pone ya una serie de propiedades por defecto a la tabla.



The screenshot shows the Apache HBase web interface. The browser address bar displays 'nodo1:16010/tablesDetailed.jsp'. The navigation menu includes 'Home', 'Table Details' (selected), 'Procedures & Locks', 'HBase Report', 'Operation Details', 'Process Metrics', 'Local Logs', 'Log Level', 'Debug Dump', and 'Metrics Dump'. The main content area is titled 'User Tables' and indicates '1 table(s) in set.' Below this, a table lists the details for the 'prueba' table.

Table	Description
prueba	'prueba', {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}, {NAME => 'cf', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_C
a	SIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536 B (64KB)'

## 4. ARRANQUE STANDALONE

**Paso 10.** Ahora eliminamos la tabla prueba con  
**disable 'prueba'** → desactivamos la tabla  
**drop 'prueba'** → borramos la tabla

```
hbase:005:0> disable 'prueba'
2023-11-26 18:11:24,159 INFO [ReadOnlyZKClient
181 sessionTimeout=90000 watcher=org.apache.had
2023-11-26 18:11:24,160 INFO [ReadOnlyZKClient
ytes
2023-11-26 18:11:24,160 INFO [ReadOnlyZKClient
ature enabled=
2023-11-26 18:11:24,161 INFO [ReadOnlyZKClient
onnection to server localhost/127.0.0.1:2181. V
2023-11-26 18:11:24,162 INFO [ReadOnlyZKClient
established, initiating session, client: /127.
2023-11-26 18:11:24,170 INFO [ReadOnlyZKClient
t complete on server localhost/127.0.0.1:2181,
2023-11-26 18:11:24,235 INFO [main] client.HBa
2023-11-26 18:11:24,584 INFO [main] client.HBa
Took 0.4490 seconds
hbase:006:0> drop 'prueba'
2023-11-26 18:11:32,248 INFO [main] client.HBa
Took 0.1488 seconds
hbase:007:0>
```

Siempre hay que poner los nombres de los objetos entre comillas simples, porque si no piensa que son variables.

## 5. CONCEPTOS DE HBASE

---

- ❑ Vamos a ver cómo funciona Hbase a nivel de objetos de tablas, filas, columnas etc.
- ❑ Dentro de Hbase, una tabla es un concepto muy parecido al que conocemos de otras bases tradicionales, como Mysql, Oracle. Pero desde un punto de vista físico su implementación y almacenamiento es diferente.
- ❑ Cada tabla de Hbase debe tener una clave "primaria" o "row key". Una clave de fila que identifique de manera inequívoca a un elemento dentro de Hbase.
- ❑ Permite ordenar las filas según ese Row key, de forma que las búsquedas van a ser muy rápidas.

## 5. CONCEPTOS DE HBASE

---

- ❑ Una cosa muy peculiar dentro de este tipo de bases de datos orientadas a columnas, como Hbase es que cuando creamos las columnas dentro de una tabla, no se crean de manera independiente, sino que deben de pertenecer a una columna de familias (column family)
- ❑ No podemos crear la columna **nombre** dentro de una tabla, tenemos que poner **familia nombre**, tenemos que crear esa columna dentro de alguna familia.
- ❑ Las columnas no son estáticas, se puede añadir una nueva columna a una nueva a una familia existente en cualquier momento, por ejemplo al insertar datos

## 5. CONCEPTOS DE HBASE

---

### Column family

- ❑ Una Column Family es una colección de columnas variable que podemos cambiar a lo largo del tiempo de la tabla.
- ❑ Las Column Family se crean cuando se define la tabla. Se indica que vamos a tener tantas Family columns
- ❑ Cada columna de una column family se identifica con el nombre\_familia + ":" + nombre\_columna
  - ❑ factura:codigo, factura:precio
  - ❑ La familia es la factura y el codigo y el precio son las columnas pertenecientes a esa familia.
- ❑ Una familia puede tener un número indeterminado de columnas y se pueden ir añadiendo en cualquier momento

## 5. CONCEPTOS DE HBASE

---

### Column family

- ❑ El concepto de familia es importante porque lo utiliza Hbase por debajo para almacenar la información
- ❑ Todas las columnas de una familia se ordenan y se almacenan juntas
- ❑ Una celda es el cruce entre una familia y una columna
- ❑ Dentro de una fila, una celda vacía no ocupa espacio, ya que no se llega a crear nada. Por eso se le denomina "sparse" (dispersa).

## 5. CONCEPTOS DE HBASE

### Ejemplo

- ❑ Tabla persona que tienen dos familias:
  - ❑ Familia personal que tiene dos columnas: nombre y apellidos
  - ❑ Familia trabajo tiene tres columnas: cargos responsabilidad y salario

Row Key o clave primaria	Familia personal	Familia trabajo
Juan	Personal:nombre="juan"	
Juan	Personal:apellidos="rodriguez"	
Juan		Trabajo:cargo="jefe"
Juan		Trabajo:responsabilidad="mucha"
Juan		Trabajo:salario=1000

## 5. CONCEPTOS DE HBASE

---

### Visión lógica

- ❑ A nivel lógico tendríamos la fila Juan (row key), y las columnas agrupadas por familias.
- ❑ Cada contenido se trata de una celda. Tenemos Juan Rodríguez, jefe, responsabilidad mucha y salario 1000.
- ❑ Dentro de una base datos relacional la fila la veríamos con todos los campos seguidos, mientras que en una base de datos como Hbase la vemos un poquito más distinta, podemos decir en modo mapa o modo array.
- ❑ Podemos ver que hay celdas que están vacías, no existen, no ocupan sitio porque realmente no llegan a crearse.
- ❑ Vemos que tenemos 5 datos repartidos en dos Column Family, o en dos familias de columna.



## 5. CONCEPTOS DE HBASE

### Vision física

- ❑ En realidad a nivel físico dentro de Hbase, cada Column Family o familia se guardan de manera independiente
- ❑ Estas particiones a nivel de Column Family junto con el row key nos permite crear las regiones, en las que se divide una tabla. Esas regiones se llevan a los esclavos y luego el region server se encarga de su gestion.

Row Key o clave primaria	Timestamp	Familia personal
Juan	T2	Personal:nombre="juan"
Juan	t3	Personal:apellidos="rodriguez"

Row Key o clave primaria	Timestamp	Familia trabajo
Juan	t5	Trabajo:cargo="jefe"
Juan	t6	Trabajo:responsabilidad="mucha"
Juan	t7	Trabajo:salario=1000

La columna denominada TimeStamp nos permite tener versionados y determinar el momento en el que una fila se ha agregado o modificado

## 5. CONCEPTOS DE HBASE

---

### Operaciones

- ❑ Operaciones se pueden hacer sobre las filas de Hbase

Row Key o clave primaria	Descripción
Get	Recuperar una fila
Scan	Recuperar varias filas
Put	Insertar datos
Delete	Borrar datos.

- ❑ Son las operaciones más habituales con las que vamos a trabajar normalmente.
- ❑ Dentro de Hbase cuando vamos añadiendo valores a una celda se versionan automáticamente. Por defecto, si no se indica lo contrario en la configuración, se guardan tres versiones ordenadas.

## 6. TRABAJAR CON HBASE SHELL

---

**Paso 1.** Para trabajar con la consola en modo comando con hbase shell, insertaremos en el path su directorio origen, para así poder ejecutar el arranque, la parada o el acceso a la shell de manera transparente sin tener que ir poniendo el directorio. Vamos al directorio del usuario hadoop y editaremos el fichero .bashrc, para crear la variable HBASE\_HOME y añadirla al path, añadiéndole /bin.

```
export HADOOP_HOME=/opt/hadoop
export HIVE_HOME=/opt/hadoop/hive
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_CONF_DIR=/opt/hadoop/etc/hadoop
export ZOOKEEPER_HOME=/opt/hadoop/zoo
export HBASE_HOME=/opt/hadoop/hbase
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$HIVE_HOME/bin:$HIVE_HOME/conf:$HADOOP_CONF_DIR:$ZOOKEEPER_HOME/bin:$HBASE_HOME/bin
```

```
hadoop@nodo1: /opt/hadoop/hbase/bin$ cd
hadoop@nodo1: ~$ sudo nano .bashrc
[sudo] password for hadoop:
hadoop@nodo1: ~$ sudo nano .bashrc
hadoop@nodo1: ~$ source .bashrc
hadoop@nodo1: ~$ echo $HBASE_HOME
/opt/hadoop/hbase
hadoop@nodo1: ~$
```

## 6. TRABAJAR CON HBASE SHELL

---

**Paso 2.** Lanzamos Hbase. Los warnings obtenidos de las librerías los solucionaremos cuando trabajemos en modo Clúster. Comprobamos con jps que el proceso Hbase esta trabajando. Por ultimo lanzamos hbase Shell:

```
hadoop@nodo1:~$ start-hbase.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12.jar:/opt/hadoop/share/hadoop/common/lib/slf4j-api-1.7.25.jar]
SLF4J: Found binding in [jar:file:/opt/hadoop/hbase/lib/client-api-1.2.0.jar:/opt/hadoop/hbase/lib/client-api-1.2.0.jar]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
HMaster running as process 380403. Stop it first.
hadoop@nodo1:~$ jps
380403 HMaster
676359 Jps
hadoop@nodo1:~$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
```

```
- Session: 0x10001608d690005 closed
2023-11-26 21:32:57,473 INFO [ReadOnlyZKClient-127.0.0.1:2181]
EventThread shut down for session: 0x10001608d690005
hbase:002:0> █
```

## 6. TRABAJAR CON HBASE SHELL

---

### Paso 3. Con list obtenemos las tablas que tenemos

```
hbase:002:0> list
TABLE
2023-11-26 21:44:55,137 INFO [ReadOnlyZKClient-127.0.0.1
client connection, connectionString=127.0.0.1:2181 sessionTi
27/1889235700@195a9bf7
2023-11-26 21:44:55,138 INFO [ReadOnlyZKClient-127.0.0.1
es(237)) - jute.maxbuffer value is 4194304 Bytes
2023-11-26 21:44:55,138 INFO [ReadOnlyZKClient-127.0.0.1
) - zookeeper.request.timeout value is 0. feature enabled
2023-11-26 21:44:55,139 INFO [ReadOnlyZKClient-127.0.0.1
va:logStartConnect(1112)) - Opening socket connection to
known error)
2023-11-26 21:44:55,140 INFO [ReadOnlyZKClient-127.0.0.1
va:primeConnection(959)) - Socket connection established,
2023-11-26 21:44:55,144 INFO [ReadOnlyZKClient-127.0.0.1
va:onConnected(1394)) - Session establishment complete on
out = 40000
0 row(s)
Took 0.3527 seconds
=> []
hbase:003:0> █
```

## 6. TRABAJAR CON HBASE SHELL

---

**Paso 4.** Con help obtenemos las opciones de todos los comandos que son miles

```
{'key1' => 'value1', 'key2' => 'value2', ...}
```

and are opened and closed with curly-braces. Key/values are delimited by the '=' character combination. Usually keys are predefined constants such as NAME, VERSIONS, COMPRESSION, etc. Constants do not need to be quoted. Type 'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use double-quoted hexadecimal representation. For example:

```
hbase> get 't1', "key\x03\x3f\xcd"  
hbase> get 't1', "key\003\023\011"  
hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"
```

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added. For more on the HBase Shell, see <http://hbase.apache.org/book.html>

```
hbase:005:0>
```

## 6. TRABAJAR CON HBASE SHELL

---

**Paso 5.** El comando `status` nos permite ver cuántos masters tenemos activos, si tenemos algún master de backup, etc

```
hbase:005:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
Took 0.0873 seconds
hbase:006:0> █
```

**Paso 6.** El comando `version` nos va a indicar la versión de Hbase con la que estamos trabajando

```
hbase:010:0> version
2.5.6, r6bac842797dc26bedb7adc0759358e4c8fd5a992, Sat Oct 14 23:36:46 PDT 2023
Took 0.0003 seconds
hbase:011:0> █
```

## 6. TRABAJAR CON HBASE SHELL

---

**Paso 7.** Si ponemos un comando con argumentos, como estamos trabajando con Ruby, estos tienen que ir entre comillas simples. Por ejemplo:

**create 't1','cf1'** → crea la tabla t1, junto con la column family cf1

**Paso 8.** Con el comando list, vemos la tabla t1 recién creada

```
hbase:011:0> create 't1','cf1'
2023-11-26 22:10:14,233 INFO [main] client.HBaseAdmin
lt:t1, procId: 16 completed
Created table t1
Took 0.6745 seconds
=> Hbase::Table - t1
hbase:012:0> list
TABLE
t1
1 row(s)
Took 0.0109 seconds
=> ["t1"]
hbase:013:0>
```



## 6. TRABAJAR CON HBASE SHELL

**Paso 9.** Con el comando describe 't1' (comando tipo Ruby entre comillas simples) indica que la tabla t1 esta activa, tiene una Family Column llamada cf1,etc.

```
hbase:013:0> describe 't1'
2023-11-26 22:12:11,371 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb]
client connection, connectString=127.0.0.1:2181 sessionTimeout=90000 watch
27/1889235700@195a9bf7
2023-11-26 22:12:11,371 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb]
es(237)) - jute.maxbuffer value is 4194304 Bytes
2023-11-26 22:12:11,372 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb]
) - zookeeper.request.timeout value is 0. feature enabled=
2023-11-26 22:12:11,375 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb]
va:logStartConnect(1112)) - Opening socket connection to server localhost/
known error)
2023-11-26 22:12:11,375 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb]
va:primeConnection(959)) - Socket connection established, initiating sessi
2023-11-26 22:12:11,379 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb]
va:onConnected(1394)) - Session establishment complete on server localhost
out = 40000
Table t1 is ENABLED
t1, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' =>
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELE
REVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW
true', BLOCKSIZE => '65536 B (64KB)'}

1 row(s)
Quota is disabled
Took 0.1301 seconds
hbase:014:0>
```

Aunque en el create table no le hemos dado nada más información que el nombre de la Family CF1, Hbase asigna una serie de propiedades por defecto a las tablas que crea

## 7. CREAR TABLAS E INSERTAR DATOS

---

**Paso 1.** Crearemos una tabla llamada empleados con dos column family: la familia personal y la familia de trabajo.

**create 'empleados', 'personal', 'trabajo'**

Hacemos **list** para ver las tablas creadas

```
hbase:015:0> create 'empleados','personal','trabajo'
2023-11-28 17:55:53,364 INFO [main] client.HBaseAdmin
lt:empleados, procId: 19 completed
Created table empleados
Took 0.6412 seconds
=> Hbase::Table - empleados
hbase:016:0> list
TABLE
empleados
t1
2 row(s)
Took 0.0081 seconds
=> ["empleados", "t1"]
hbase:017:0>
```

## 7. CREAR TABLAS E INSERTAR DATOS

---

Paso 2. Ejecutamos **describe 'empleados'**. Comprobamos que tenemos dos Columns family: personal y trabajo, cada una de ellas con las propiedades por defecto.

```
hbase:017:0> describe 'empleados'
Table empleados is ENABLED
empleados, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA
> 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', CO
=> 'true', BLOCKSIZE => '65536 B (64KB)'}
{NAME => 'trabajo', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA
'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', CO
=> 'true', BLOCKSIZE => '65536 B (64KB)'}

2 row(s)
Quota is disabled
Took 0.0360 seconds
hbase:018:0>
```

## 7. CREAR TABLAS E INSERTAR DATOS

---

**Paso 3.** Insertaremos una fila en la tabla mediante el comando put. Este comando sería el insert de las tablas de las bases relacionales.

**put 'empleados','1','personal:nombre','Sergio'**

En la tabla empleados añadimos en la column family personal la columna nombre con el valor Sergio

```
hbase:018:0> put 'empleados','1','personal:nombre','Sergio'  
Took 0.0760 seconds  
hbase:019:0> █
```

- Las columnas pueden ser añadidas de manera dinámica sin necesidad de haberlas definido previamente
- Toda fila dentro de HBase tiene que tener un row key, una clave primaria que puede ser de tipo numérico o cadena de caracteres.

## 7. CREAR TABLAS E INSERTAR DATOS

---

**Paso 4.** El comando más simple para poder ver la información que hay en la tabla es el comando Scan  
**scan 'empleados'**

```
hbase:019:0> scan 'empleados'
ROW                                COLUMN+CELL
 1                                column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio
1 row(s)
Took 0.0326 seconds
hbase:020:0>
```

Vemos: el row 1, la columna personal:nombre, el timestamp

**Paso 5.** Insertamos los apellidos 'Pérez Rodríguez' en el mismo empleado o en la misma fila

**put 'empleados','1','personal:apellidos','Perez Rodriguez'**

```
hbase:020:0> put 'empleados','1','personal:apellidos','Perez Rodriguez'
Took 0.0068 seconds
hbase:021:0>
```

## 7. CREAR TABLAS E INSERTAR DATOS

---

Paso 6. Hacemos un scan de empleados  
**scan 'empleados'**

```
hbase:021:0> scan 'empleados'
ROW
1
1
1 row(s)
Took 0.0091 seconds
hbase:022:0>
```

COLUMN+CELL
column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez
column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio

Vemos que es la row 1, es decir en la misma fila. Hbase va guardando los datos de esta manera, el nombre Sergio y los apellidos Perez Rodriguez

Paso 7. Pondremos datos pero ahora en la column family trabajo y la columna cargo

**put 'empleados','1','trabajo:cargo','jefe'**

```
hbase:022:0> put 'empleados','1','trabajo:cargo','jefe'
Took 0.0085 seconds
hbase:023:0>
```

## 7. CREAR TABLAS E INSERTAR DATOS

---

**Paso 8.** Si hacemos un scan, ahora nos van a salir esos datos.

```
hbase:037:0> scan 'empleados'
ROW                                COLUMN+CELL
1                                  column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez
1                                  column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio
1                                  column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe
1 row(s)
Took 0.0049 seconds
hbase:038:0>
```

**Nota:** Vemos que esta información se va guardando de una manera un poco peculiar. Los que venimos de bases de datos relacionales, con el scan parece que la fila está como separada pero en realidad es la forma que él tiene de almacenarlo.

## 7. CREAR TABLAS E INSERTAR DATOS

---

Paso 9. Añadimos a la tabla empleados una segunda fila, con row key 2:

```
put 'empleados','2','trabajo:salario','5000'  
scan 'empleados'
```

```
hbase:038:0> put 'empleados','2','trabajo:salario','5000'  
Took 0.0109 seconds  
hbase:039:0> scan 'empleados'  
ROW                                COLUMN+CELL  
1                                  column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez  
1                                  column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio  
1                                  column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe  
2                                  column=trabajo:salario, timestamp=2023-11-28T19:07:58.188, value=5000  
2 row(s)  
Took 0.0138 seconds  
hbase:040:0>
```

Ponemos una columna nueva en una fila 2, que también es nueva. Entre la fila 1 y la fila 2 no coincide ninguna columna. Hbase permite mantener mucha heterogeneidad, pero en la vida real va a ser raro que tengamos valores dispares y que no tengan nada que ver entre sí porque entonces la tabla Empleado no tendría mucho sentido



## 7. CREAR TABLAS E INSERTAR DATOS

Paso 10. Si en lugar de la column family **trabajo** ponemos otra, va a indicar que la familia de columna no existe.

**put 'empleados','2','trab:salario','5000'**

```
hbase:040:0> put 'empleados','2','trab:salario','5000'

ERROR: org.apache.hadoop.hbase.regionserver.NoSuchColumnFamilyException: Column family trab does not exist in region emplead
bbdec5. in table 'empleados', {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}, {NAME => '
ERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLIC
MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536 B (64KB)'}, {NAME => 'trabajo', INDEX_B
P_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0',
COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536 B (64KB)'}
    at org.apache.hadoop.hbase.regionserver.HRegion.mutate(HRegion.java:4862)
    at org.apache.hadoop.hbase.regionserver.HRegion.mutate(HRegion.java:4852)
    at org.apache.hadoop.hbase.regionserver.HRegion.mutate(HRegion.java:4848)
    at org.apache.hadoop.hbase.regionserver.HRegion.lambda$put$7(HRegion.java:3143)
    at org.apache.hadoop.hbase.trace.TraceUtil.trace(TraceUtil.java:216)
    at org.apache.hadoop.hbase.regionserver.HRegion.put(HRegion.java:3132)
    at org.apache.hadoop.hbase.regionserver.RSRpcServices.put(RSRpcServices.java:3043)
    at org.apache.hadoop.hbase.regionserver.RSRpcServices.mutate(RSRpcServices.java:3006)
    at org.apache.hadoop.hbase.shaded.protobuf.generated.ClientProtos$ClientService$2.callBlockingMethod(ClientProtos.ja
    at org.apache.hadoop.hbase.ipc.RpcServer.call(RpcServer.java:415)
    at org.apache.hadoop.hbase.ipc.CallRunner.run(CallRunner.java:124)
    at org.apache.hadoop.hbase.ipc.RpcHandler.run(RpcHandler.java:102)
    at org.apache.hadoop.hbase.ipc.RpcHandler.run(RpcHandler.java:82)

For usage try 'help "put"'

Took 0.1979 seconds
hbase:041:0>
```

Podemos crear todas las columnas que queramos, lo que no podemos hacer alegremente es hacer mención a una Column Family que no está, dando un error tipo java

## 8. SCAN Y GET

### Paso 1. Observamos la estructura de la tabla empleados

```
hbase:041:0> describe 'empleados'
Table empleados is ENABLED
empleados, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true'}
{NAME => 'trabajo', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true'}
2 row(s)
Quota is disabled
Took 0.0556 seconds
hbase:042:0>
```

Contiene dos familias de columnas: personal y trabajo

### Paso 2. Hacemos un scan de empleados, para ver las filas existentes. Teníamos la row key y luego valores para cada una de las family Columns y columnas

```
hbase:042:0> scan 'empleados'
ROW                                COLUMN+CELL
1      column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez
1      column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio
1      column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe
2      column=trabajo:salario, timestamp=2023-11-28T19:07:58.188, value=5000
2 row(s)
Took 0.0138 seconds
hbase:043:0>
```

## 8. SCAN Y GET

Paso 3. Insertamos el nombre Raul a la fila 2

**put 'empleados', 2, 'personal:nombre', 'Raul'**

```
hbase:043:0> put 'empleados','2','personal:nombre','Raul'  
Took 0.0056 seconds  
hbase:044:0> █
```

Paso 4. Volvemos a hacer de nuevo **scan 'empleados'** y vemos que tenemos 3 valores para la row key 1 y 2 valores para la row key 2.

```
hbase:044:0> scan 'empleados'  
ROW                                COLUMN+CELL  
  1                                column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodri  
  1                                column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio  
  1                                column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe  
  2                                column=personal:nombre, timestamp=2023-11-28T19:51:23.132, value=Raul  
  2                                column=trabajo:salario, timestamp=2023-11-28T19:07:58.188, value=5000  
2 row(s)  
Took 0.0130 seconds  
hbase:045:0> █
```

La forma en la que aparece la información y como se trata es distinta respecto a la de una base de datos relacional<sup>59</sup>

## 8. SCAN Y GET

---

**Paso 5.** El comando scan tiene distintas opciones. Si sólo quisiéramos ver una determinada columna, por ejemplo la columna nombre:

**scan 'empleados', {COLUMNS=>'personal:nombre'}**

```
hbase:047:0> scan 'empleados',{COLUMNS=>'personal:nombre'}
ROW                               COLUMN+CELL
1                                 column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio
2                                 column=personal:nombre, timestamp=2023-11-28T19:51:23.132, value=Raul
2 row(s)
Took 0.0059 seconds
hbase:048:0> █
```

En este caso usamos una propiedad de la tabla denominada **COLUMNS**, que va a devolver todos aquellos valores que tienen la Column Family personal y la columna Nombre.

## 8. SCAN Y GET

**Paso 6.** En la creación de una tabla podríamos haber utilizado algunas de las opciones que salen en **describe 'empleados'**.

Podríamos haber puesto por ejemplo el número de versiones, si va a estar o no en memoria, su ámbito de replicación, etc

```
hbase:048:0> desc 'empleados'
Table empleados is ENABLED
empleados, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
, REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
{NAME => 'trabajo', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
, REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
2 row(s)
Quota is disabled
Took 0.0309 seconds
hbase:049:0>
```

## 8. SCAN Y GET

**Paso 7.** Crearemos la tabla `empleados1`, usando algunas opciones, pero siempre encerrándolas entre llaves.

**`create 'empleados1',{NAME=>'familia1', VERSIONS=>3}`**

```
hbase:049:0> create 'empleados1',{NAME=>'familia1',VERSIONS=>3}
2023-11-28 20:43:41,564 INFO [main] client.HBaseAdmin (HBaseAdmin.java:postOperationResult(3591)) -
eted
Created table empleados1
Took 0.6462 seconds
=> Hbase::Table - empleados1
hbase:050:0>
```

Creamos la tabla `empleados1` con una column family llamada `familia1`, y versiones igual a 3, para que versione hasta 3 posibles valores de cada fila que haya dentro de esa familia. Por defecto el número de versiones es 1.

En vez de dejar que HBASE ponga las opciones por defecto en la creación de una tabla, las podemos poner nosotros mismos

## 8. SCAN Y GET

**Paso 8.** Si hacemos **describe empleado1** vemos que el Column Family familia1 propone una versión 3, es decir que va a versionar hasta 3 elementos.

```
hbase:050:0> desc 'empleados1'
Table empleados1 is ENABLED
empleados1, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'familia1', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '3', KEEP_DELETED_CELLS => 'FALSE',
, REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE',
1 row(s)
Quota is disabled
Took 0.0357 seconds
hbase:051:0>
```

**NOTA:** En el scan además de poner la tabla también podemos poner alguna propiedad para hacer algún filtro (con COLUMNS por ej indicamos las columnas de la tabla)

**scan 'empleados', {COLUMNS=>'personal:nombre'}**

## 8. SCAN Y GET

**Paso 9.** El comando scan permite recuperar datos de manera masiva. Pero también existe el comando get que permite recuperar una determinada fila o row key.

**get 'empleados', '2'** → Recupera la información de las filas que tengan el row key 2

```
hbase:051:0> get 'empleados','2'
COLUMN                                CELL
personal:nombre                       timestamp=2023-11-28T19:51:23.132, value=Raul
trabajo:salario                       timestamp=2023-11-28T19:07:58.188, value=5000
1 row(s)
Took 0.0177 seconds
hbase:052:0>
```

**get 'empleados', '1'** → Recupera las filas con row key 1

```
hbase:052:0> get 'empleados','1'
COLUMN                                CELL
personal:apellidos                    timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez
personal:nombre                       timestamp=2023-11-28T18:19:34.503, value=Sergio
trabajo:cargo                         timestamp=2023-11-28T18:48:37.381, value=jefe
1 row(s)
Took 0.0091 seconds
hbase:053:0> █
```



## 8. SCAN Y GET

**Paso 10.** Si queremos recuperar los datos del empleado 1 pero sólo me interesa saber su nombre

**get 'empleados','1',{COLUMN=>'personal.nombre'}**

```
hbase:054:0> get 'empleados','1',{COLUMN=>'personal:nombre'}  
COLUMN                                CELL  
personal:nombre                       timestamp=2023-11-28T18:19:34.503, value=Sergio  
1 row(s)  
Took 0.0088 seconds  
hbase:055:0>
```

Es bastante sencillo poder hacer Select con get y con scan contra una determinada tabla



## 9. DELETE, UPDATE Y VERSIONADO

**Paso 1.** Vamos a borrar algún elemento de la tabla, por ejemplo el salario de los row key 2. Utilizaremos el comando delete:

**delete 'empleados', '2', 'trabajo:salario'**

```
hbase:055:0> delete 'empleados','2','trabajo:salario'  
Took 0.0229 seconds  
hbase:056:0>
```

**Paso 2.** Si hacemos un scan podemos comprobar que dentro del row key 2 ya sólo tenemos la fila con el nombre, ya que la fila con el salario la hemos eliminado. No hemos borrado la row key completamente, sólo la columna indicada

```
hbase:058:0> scan 'empleados'  
ROW                                COLUMN+CELL  
1      column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez  
1      column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio  
1      column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe  
2      column=personal:nombre, timestamp=2023-11-28T19:51:23.132, value=Raul  
2 row(s)  
Took 0.0046 seconds  
hbase:059:0>
```



## 9. DELETE, UPDATE Y VERSIONADO

**Paso 3.** También podemos modificar un valor con put. Modificaremos el nombre del row key 2, cuyo valor es Raul por el valor Pedro.

**put 'empleados', '2', 'personal:nombre', 'Pedro'**

```
hbase:060:0> put 'empleados','2','personal:nombre','Pedro'  
Took 0.0076 seconds  
hbase:061:0> █
```

**Paso 4.** Si hacemos un scan, vemos que put ha actualizado el valor del nombre de row key 2. Put inserta un row Column Family valor si no existe, y si existe lo modifica

```
hbase:061:0> scan 'empleados'  
ROW                                COLUMN+CELL  
1                                  column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez  
1                                  column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio  
1                                  column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe  
2                                  column=personal:nombre, timestamp=2023-11-29T21:25:39.194, value=Pedro  
2 row(s)  
Took 0.0225 seconds  
hbase:062:0> █
```



## 9. DELETE, UPDATE Y VERSIONADO

Paso 5. Siempre podemos crear una nueva columna

**put 'empleados','2','personal:comision',1000**

```
hbase:062:0> put 'empleados','2','personal:comision',10000
Took 0.0083 seconds
hbase:063:0> █
```

Paso 6. Si hacemos un scan de empleados podemos ver que en la row key 2 tenemos un column Family personal que tiene dos columnas, nombre y comision

```
hbase:063:0> scan 'empleados'
ROW                                COLUMN+CELL
1      column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez
1      column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio
1      column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe
2      column=personal:comision, timestamp=2023-11-29T21:46:53.804, value=10000
2      column=personal:nombre, timestamp=2023-11-29T21:25:39.194, value=Pedro
2 row(s)
Took 0.0108 seconds
hbase:064:0> █
```

## 9. DELETE, UPDATE Y VERSIONADO

---

**Paso 7.** También podemos modificar la tabla de manera dinámica, añadiéndole un nuevo tipo de Column Family llamado clientes.

**alter 'empleados','clientes'**

```
hbase:064:0> alter 'empleados','clientes'  
Updating all regions with the new schema...  
1/1 regions updated.  
Done.  
Took 1.6808 seconds  
hbase:065:0> █
```

Ha hecho un update en todas las regiones. Las regiones son las particiones en las que se dividía la tabla



## 9. DELETE, UPDATE Y VERSIONADO

---

**Paso 8.** Hacemos un describe empleados vemos que ahora tenemos tres Column Family: clientes, personal y trabajo.

```
hbase:065:0> describe 'empleados'
Table empleados is ENABLED
empleados, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'clientes', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE =>
}
{NAME => 'personal', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE =>
}
{NAME => 'trabajo', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE =>
}
3 row(s)
Quota is disabled
Took 0.0376 seconds
hbase:066:0> █
```

## 9. DELETE, UPDATE Y VERSIONADO

**Paso 9.** Vamos a modificar el comportamiento de la tabla y en concreto vamos a ver el versionado. En describe vemos una propiedad que se llama VERSIONS que representa las versiones que se van guardando de una determinada Column Family valor cuando lo modifico.

Hacemos un scan de empleados.

```
hbase:066:0> scan 'empleados'
```

```
ROW
```

```
1
```

```
COLUMN+CELL
```

```
column=personal:apellidos, timestamp=2023-11-28T18:35:13.472, value=Perez Rodriguez
```

```
1
```

```
column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio
```

```
1
```

```
column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe
```

```
2
```

```
column=personal:comision, timestamp=2023-11-29T21:46:53.804, value=10000
```

```
2
```

```
column=personal:nombre, timestamp=2023-11-29T21:25:39.194, value=Pedro
```

```
2 row(s)
```

```
Took 0.0260 seconds
```

```
hbase:067:0>
```

## 9. DELETE, UPDATE Y VERSIONADO

**Paso 10.** Modificaremos los apellidos del row key 1 y en vez de Perez Rodriguez pondremos apellido2

**put 'empleados','1','personal:apellidos','apellido2'**

```
hbase:005:0> put 'empleados','1','personal:apellidos','apellido2'  
Took 0.0054 seconds  
hbase:006:0>
```

**Paso 11.** Con scan 'empleados' comprobamos que ha modificado Perez Rodríguez por apellido2

```
hbase:006:0> scan 'empleados'
```

ROW

1

1

1

2

2

2 row(s)

Took 0.0142 seconds

```
hbase:007:0>
```

COLUMN+CELL

column=personal:apellidos, timestamp=2023-11-29T22:45:36.921, value=apellido2

column=personal:nombre, timestamp=2023-11-28T18:19:34.503, value=Sergio

column=trabajo:cargo, timestamp=2023-11-28T18:48:37.381, value=jefe

column=personal:comision, timestamp=2023-11-29T21:46:53.804, value=10000

column=personal:nombre, timestamp=2023-11-29T21:25:39.194, value=Pedro



## 9. DELETE, UPDATE Y VERSIONADO

---

**Paso 12.** Dentro de las opciones del get le podemos indicar que me diga la versión de una determinada fila.

**get 'empleados','1',{COLUMN=>'personal:apellidos',VERSION=>2}**

```
hbase:007:0> get 'empleados','1',{COLUMN=>'personal:apellidos',VERSIONS=>2}
COLUMN                                CELL
personal:apellidos                    timestamp=2023-11-29T22:45:36.921, value=apellido2
1 row(s)
Took 0.0372 seconds
hbase:008:0>
```

Queremos recuperar de la tabla empleados del row key 1, la columna personal apellido pero la versión 2. Obtenemos apellido2

## 9. DELETE, UPDATE Y VERSIONADO

**Paso 13.** ¿Que pasa si queremos ir mirando distintas versiones? Pues puede que no aparezcan, ya que por defecto siempre salen los mismos valores

```
hbase:008:0> get 'empleados','1',{COLUMN=>'personal:apellidos',VERSIONS=>1}
COLUMN                                CELL
personal:apellidos                    timestamp=2023-11-29T22:45:36.921, value=apellido2
1 row(s)
Took 0.0152 seconds
hbase:009:0> get 'empleados','1',{COLUMN=>'personal:apellidos',VERSIONS=>3}
COLUMN                                CELL
personal:apellidos                    timestamp=2023-11-29T22:45:36.921, value=apellido2
1 row(s)
Took 0.0114 seconds
```

## 9. DELETE, UPDATE Y VERSIONADO

**Paso 14.** Esto es porque si hacemos un describe de la tabla empleados, vemos que el número de versionados que va a guardar es 1, sólo hay una versión.

```
hbase:010:0> desc 'empleados'
2023-11-29 23:06:57,711 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb] zookeeper.ZooKeeper (ZooKeeper.java:127.0.0.1:2181 sessionTimeout=90000 watcher=org.apache.hadoop.hbase.zookeeper.ReadOnlyZKClient$$Lambda$227/2023-11-29 23:06:57,711 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb] zookeeper.ClientCnxnSocket (ClientCnxnSocket.java:4194304 Bytes)
2023-11-29 23:06:57,711 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb] zookeeper.ClientCnxn (ClientCnxn.java:127.0.0.1:2181 value is 0. feature enabled=)
2023-11-29 23:06:57,712 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb-SendThread(127.0.0.1:2181)] zookeeper.ClientCnxnSocket (ClientCnxnSocket.java:127.0.0.1:2181) Will not attempt to authenticate using SASL (unknown keytab)
2023-11-29 23:06:57,713 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb-SendThread(127.0.0.1:2181)] zookeeper.ClientCnxnSocket (ClientCnxnSocket.java:127.0.0.1:2181) zookeeper connection established, initiating session, client: /127.0.0.1:33620, server: localhost/127.0.0.1:2181
2023-11-29 23:06:57,722 INFO [ReadOnlyZKClient-127.0.0.1:2181@0x3a88f6fb-SendThread(127.0.0.1:2181)] zookeeper.ClientCnxnSocket (ClientCnxnSocket.java:127.0.0.1:2181) zookeeper establishment complete on server localhost/127.0.0.1:2181, sessionId = 0x10001608d69000a, negotiated timeout=30000
Table empleados is ENABLED
empleados, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'clientes', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'FALSE'}
{NAME => 'personal', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'FALSE'}
{NAME => 'trabajo', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'FALSE'}

3 row(s)
Quota is disabled
Took 0.1315 seconds
hbase:011:0>
```

## 9. DELETE, UPDATE Y VERSIONADO

Paso 15. Para conseguir que nos guarde más versiones:

**alter 'empleados',{NAME=>'personal',VERSIONS=>3}**

Modificamos la tabla empleados, y hacemos que la Column Family personal tenga 3 versiones

```
hbase:012:0> alter 'empleados',{NAME=>'personal',VERSIONS=>3}
Updating all regions with the new schema...
1/1 regions updated.
Done.
Took 1.6753 seconds
hbase:013:0>
```

## 9. DELETE, UPDATE Y VERSIONADO

**Paso 16.** Si hacemos un describe de empleados, vemos que la column family personal tiene 3 versiones

```
hbase:013:0> desc 'empleados'
Table empleados is ENABLED
empleados, {TABLE_ATTRIBUTES => {METADATA => {'hbase.store.file-tracker.impl' => 'DEFAULT'}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'clientes', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE =>
{NAME => 'personal', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '3', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE =>
{NAME => 'trabajo', INDEX_BLOCK_ENCODING => 'NONE', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '
3 row(s)
Quota is disabled
Took 0.0304 seconds
hbase:014:0> █
```

## 9. DELETE, UPDATE Y VERSIONADO

---

**Paso 17.** Modificamos el apellido, poniendo apellido33 y apellido44

```
put 'empleados','1','personal:apellidos','apellido33'  
put 'empleados','1','personal:apellidos','apellido44'
```

```
hbase:014:0> put 'empleados','1','personal:apellidos','apellido33'  
Took 0.0137 seconds  
hbase:015:0>
```

```
hbase:016:0> put 'empleados','1','personal:apellidos','apellido44'  
Took 0.0088 seconds  
hbase:017:0>
```

## 9. DELETE, UPDATE Y VERSIONADO

**Paso 18.** Miramos las 3 versiones de la columna apellidos:

**get 'empleados','1',{COLUMN=>'personal:apellidos',VERSION=>1}**

```
hbase:017:0> get 'empleados','1',{COLUMN=>'personal:apellidos',VERSIONS=>1}
COLUMN                                CELL
personal:apellidos                    timestamp=2023-11-29T23:46:11.549, value=apellido44
1 row(s)
Took 0.0171 seconds
hbase:018:0> █
```

En la versión 1 el apellido tiene un valor de apellido44

**get 'empleados','1',{COLUMN=>'personal:apellidos',VERSION=>2}**

```
hbase:018:0> get 'empleados','1',{COLUMN=>'personal:apellidos',VERSIONS=>2}
COLUMN                                CELL
personal:apellidos                    timestamp=2023-11-29T23:46:11.549, value=apellido44
personal:apellidos                    timestamp=2023-11-29T23:36:33.824, value=apellido33
1 row(s)
Took 0.0150 seconds
hbase:019:0> █
```

En la versión 2 el apellido tiene un valor de apellido33.

## 9. DELETE, UPDATE Y VERSIONADO

`get 'empleados','1',{COLUMN=>'personal:apellidos',VERSION=>3}`

```
hbase:019:0> get 'empleados','1',{COLUMN=>'personal:apellidos',VERSIONS=>3}
COLUMN                                CELL
personal:apellidos                    timestamp=2023-11-29T23:46:11.549, value=apellido44
personal:apellidos                    timestamp=2023-11-29T23:36:33.824, value=apellido33
personal:apellidos                    timestamp=2023-11-29T22:45:36.921, value=apellido2
1 row(s)
Took 0.0073 seconds
hbase:020:0> █
```

En la versión 3 el apellido tiene un valor de apellido2

**NOTA:** Esto es muy útil porque así puedo recuperar valores o información de sitios donde he estado



## 9. DELETE, UPDATE Y VERSIONADO

---

**Paso 19.** Podemos desactivar la tabla mediante `disable table`, para posteriormente hacer un `drop` y para eliminarla definitivamente

O hacer un `enable empleados` para volverla a poner en activo.

```
hbase:020:0> disable 'empleados'
2023-11-30 00:06:55,937 INFO [main] client.HBaseAdmin (HBaseAdmin.java:rpcCall(926)) - Started disable of empleados
2023-11-30 00:06:56,277 INFO [main] client.HBaseAdmin (HBaseAdmin.java:postOperationResult(3591)) - Operation: DISABLE, Table Name: empleados
Took 0.3649 seconds
hbase:021:0> enable 'empleados'
2023-11-30 00:10:12,442 INFO [main] client.HBaseAdmin (HBaseAdmin.java:rpcCall(866)) - Started enable of empleados
2023-11-30 00:10:13,126 INFO [main] client.HBaseAdmin (HBaseAdmin.java:postOperationResult(3591)) - Operation: ENABLE, Table Name: empleados
Took 0.7075 seconds
hbase:022:0>
```

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO

---



## De standalone a pseudo-distribuido

Hemos trabajado con Hbase en modo standAlone, es decir en modo local. Hemos visto sus comandos, creado tablas, insertado filas, borrado, actualizaciones, etc

Empezaremos a integrarlo con Hadoop paso a paso:

- Primero arrancaremos HBASE en modo Pseudo-distribuido donde integraremos HBASE con Hadoop y lo conectaremos con HDFS, pero todo en la misma maquina, es decir en local.
- Después instalaremos Hbase en modo completo, es decir lo desplegaremos entre los nodos del cluster Hadoop

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO

---

**Paso 1.** Lo primero que haremos es modificar el fichero hbase-site.xml que está en el directorio de configuración de hbase. Quitaremos la siguiente propiedad: hbase.unsafe.stream.capability.enforce → porque esto era para instalar o para trabajar en modo local.

```
<property>
  <name>hbase.cluster.distributed</name>
  <value>false</value>
</property>

<property>
  <name>hbase.tmp.dir</name>
  <value>./tmp</value>
</property>

<!-- <property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>false</value>
</property> -->

</configuration>
```

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO

---

**Paso 2.** Ponemos 3 propiedades que nos permiten instalar Hbase dentro de un entorno pseudo-distribuido

La primera propiedad es **hbase.cluster.distributed**. Si la ponemos a true le indicamos que el Hbase se va a comportar como si estuviera dentro de un cluster. En vez de ejecutar todo dentro de una maquina virtual de Java, vamos a tener básicamente dos procesos:

- el region Server → los procesos esclavo que gestionan las regiones donde se iban dividiendo los datos de Hbase
- el master → el proceso maestro

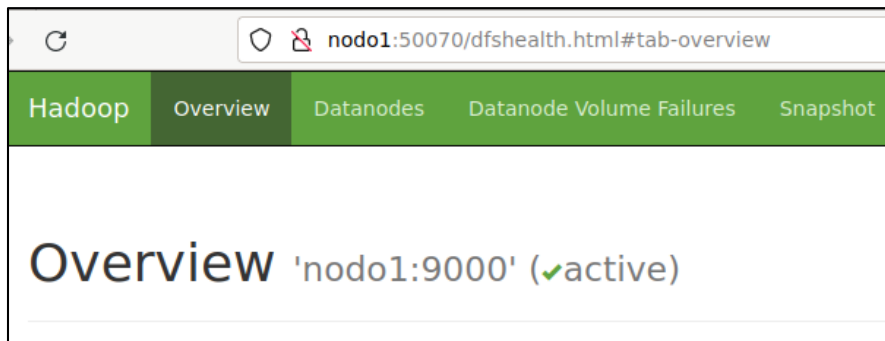
```
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>

<!-- <property>
```

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO



**Paso 3.** La segunda propiedad **hbase.rootdir**, debe apuntar a un directorio HDFS del nodo que este activo para zookeeper. Esto se ve en la pagina web de HDFS. En este directorio HDFS, Hbase guardará sus cosas. En lugar de dejar los datos en el directorio local, como hemos hecho antes, lo va a dejar en el directorio /hbase de HDFS, el cual se creará automáticamente



```
<!-- <property>
  <name>hbase.rootdir</name>
  <value>file:///opt/hadoop/hbase/data/hbase</value>
</property> -->

<property>
  <name>hbase.rootdir</name>
  <value>hdfs://nodo1:9000/hbase</value>
</property>
```

Para obtener la pagina web de gestión HDFS, debemos activar start-dfs.sh y ejecutar zookeeper en cada nodo.<sup>85</sup>

## 10. HBASE EN MODO PSEUDO-DISTRIBUIDO

---

**Paso 4.** Por ultimo tenemos que indicarle con este parámetro **hbase.zookeeper.quorum**, en que nodos están presentes los procesos de zookeeper: nodo1, nodo2 y nodo3

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>nodo1,nodo2,nodo3</value>
</property>

<!-- <property>
```

**NOTA:** Con estos 3 parámetros ya hemos configurado el fichero hbase-site.xml, para que Hbase trabaje en modo distribuido

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO



**Paso 5.** Hay que modificar el fichero `hbase-env.sh`, el cual contiene las variables de entorno de Hbase. Hay que poner dos variables muy importantes:

**1) `HBASE_MANAGES_ZK=false`** → Impide que hbase arranque su propio zookeeper. Si lo dejáramos a true, su valor por defecto, intentaría levantar sus propios zookeepers. Y esto chocaría con los zookeeper de Hadoop que tenemos funcionando. Le indicamos a Hbase que el zookeeper configurado en el fichero `hbase-site.xml` lo va a hacer Hadoop.

```
121 # can be useful in large clusters, where, e.g., slave machines can
122 # otherwise arrive faster than the master can service them.
123 # export HBASE_SLAVE_SLEEP=0.1
124
125 export HBASE_MANAGES_ZK=false
126 export HBASE_DISABLE_HADOOP_CLASSPATH_LOOKUP=true
127
```

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO

---



- 2) **HBASE\_DISABLE\_HADOOP\_CLASSPATH\_LOOKUP = true**. Es para eliminar los errores de classpath (ficheros de logs SLF4J) que se producen cuando se inicializa Hbase. A true le indicamos que no busque en el classpath de Hadoop, porque las librerías que necesitamos ya las encontramos dentro de Hbase.
- En caso contrario, encontrará antes las librerías de Hadoop y como algunos de los métodos y componentes se llaman igual, nos van a dar error



# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO



**Paso 6.** Hacemos `start-hbase.sh`, para arrancar Hbase. Esto no ha cambiado de cómo arrancarlo en modo local lo único lo único que hemos cambiado es la configuración.

```
hadoop@nodol1:/opt/hadoop/hbase/bin$ ./start-hbase.sh
running master, logging to /opt/hadoop/hbase/bin/../logs/hbase-hadoop-master-nodol1.out
: running regionserver, logging to /opt/hadoop/hbase/bin/../logs/hbase-hadoop-regionserver-nodol1.out
hadoop@nodol1:/opt/hadoop/hbase/bin$
```

- Ya no indica los warnings diciendo que se ha encontrado con librerías repetidas. Indica que ha arrancado el master y el region server. No nos dice nada de zookeeper porque el no lo gestiona.
- En estos ficheros de log podemos encontrar toda la información en el caso de que nos dé algún error

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO

**Paso 7.** Si hacemos un JPS en el nodo1 debemos de tener:

- Los procesos de Hadoop (Namenode, ResourceManager),
- Otros componentes que hemos arrancado (JournalNode, QuorumPeerMain, DFSZKController),
- Los 2 procesos que gobiernan Hbase: HRegionServer y HMaster

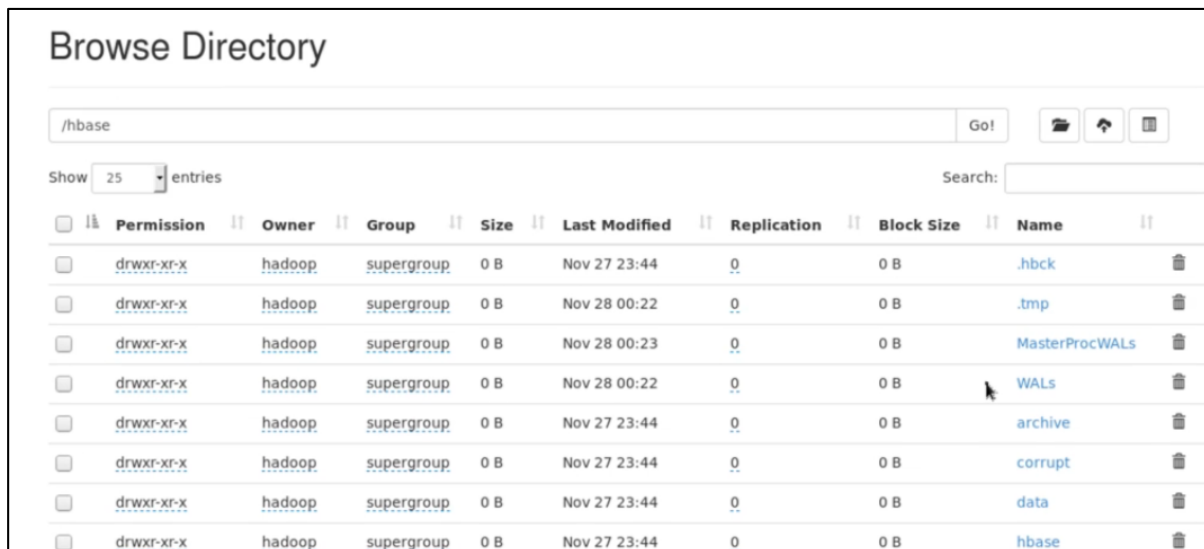
Estos procesos están en la misma maquina, aunque lo normal es que HMaster esté en un sitio y HRegionServer con los esclavos

```
[hadoop@nodo1 conf]$ jps
14466 ResourceManager
12259 DFSZKFailoverController
11491 HRegionServer
11719 NameNode
11976 JournalNode
11260 QuorumPeerMain
11821 Jps
11406 HMaster
[hadoop@nodo1 conf]$
```

# 10. HBASE EN MODO PSEUDO-DISTRIBUIDO

**Paso 8.** Para comprobar que ha funcionado, vamos a la pagina web de HDFS, opción Utilities/Browse the file System. Si todo ha ido bien tendremos el directorio /hbase, creado automáticamente por Hbase

Si entramos tenemos los mismos directorios que teníamos en el directorio local pero ahora subidos dentro de /hbase



The screenshot shows the 'Browse Directory' web interface for HDFS. The path '/hbase' is entered in the search bar. Below the search bar, there are controls for 'Show 25 entries' and a 'Search:' field. A table lists the contents of the directory, including files like .hbck, .tmp, MasterProcWALS, WALs, archive, corrupt, data, and hbase. Each row shows details like Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name, along with a delete icon.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	Nov 27 23:44	0	0 B	.hbck
drwxr-xr-x	hadoop	supergroup	0 B	Nov 28 00:22	0	0 B	.tmp
drwxr-xr-x	hadoop	supergroup	0 B	Nov 28 00:23	0	0 B	MasterProcWALS
drwxr-xr-x	hadoop	supergroup	0 B	Nov 28 00:22	0	0 B	WALs
drwxr-xr-x	hadoop	supergroup	0 B	Nov 27 23:44	0	0 B	archive
drwxr-xr-x	hadoop	supergroup	0 B	Nov 27 23:44	0	0 B	corrupt
drwxr-xr-x	hadoop	supergroup	0 B	Nov 27 23:44	0	0 B	data
drwxr-xr-x	hadoop	supergroup	0 B	Nov 27 23:44	0	0 B	hbase