Prática sesión 6 - mapreduce y HIVE

Resumen

En este documento se trata de explicar mediante ejercicios prácticos para Mapreduce y Hive. En la primera parte del documento se muestra el funcionamiento de MapReduce y cuáles son los métodos que intervienen en este tipo de procesamiento, utilizando para ello una serie de plantillas con las que ir elaborando el esquema funcional de MapReduce. En la segunda parte podemos ver una instalación base de Hive junto con algunos ejercicios básicos para entender el funcionamiento y cómo operar a través del cliente hive.

Contenido

[1. MAPREDUCE 1](#_Toc71799191)

[1.1. Ejercicio 1 - Aplicación Prices MapReduce 1](#_Toc71799192)

[1.2. Ejercicio 2 – Calcular el Momento en Twitter 7](#_Toc71799193)

[2. HIVE 14](#_Toc71799194)

[2.1. Instalación y configuración básica de Hive (**Solo para aquellos que quieran realizar la instalación completa de Hive**) 14](#_Toc71799195)

[2.2. Ejemplo de creación y consultas básicas con HIVE 18](#_Toc71799196)

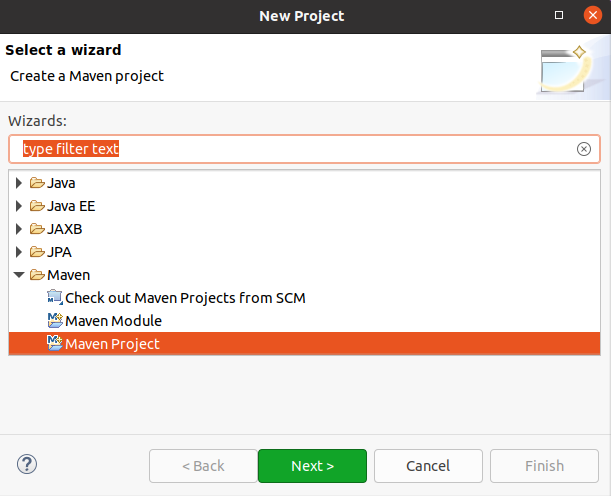
[2.3. Supuesto práctico 19](#_Toc71799197)

[2.4. Probar con Beeline, Hiveserver2 y Hue (**Solo para aquellos que hayan hecho la instalación completa**) 20](#_Toc71799198)

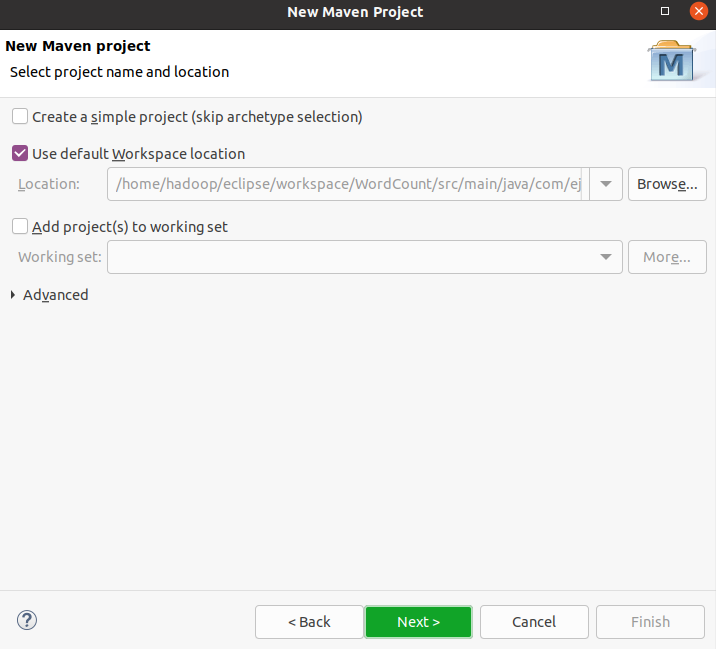
## MAPREDUCE

### Ejercicio 1 - Aplicación Prices MapReduce

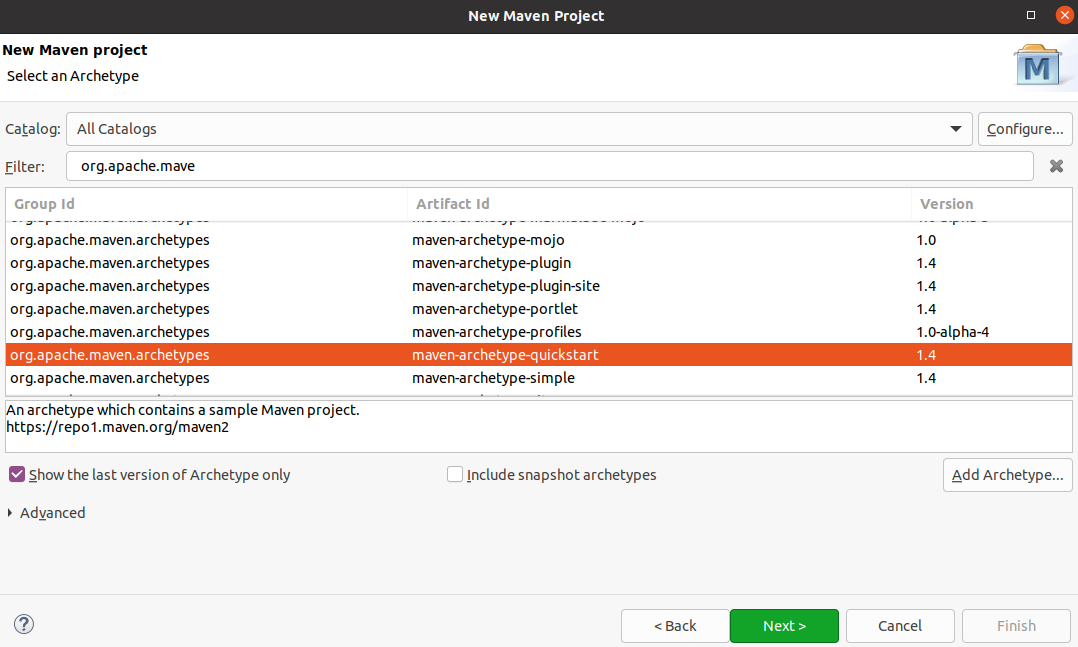
* Creamos un nuevo proyecto Maven en Eclipse



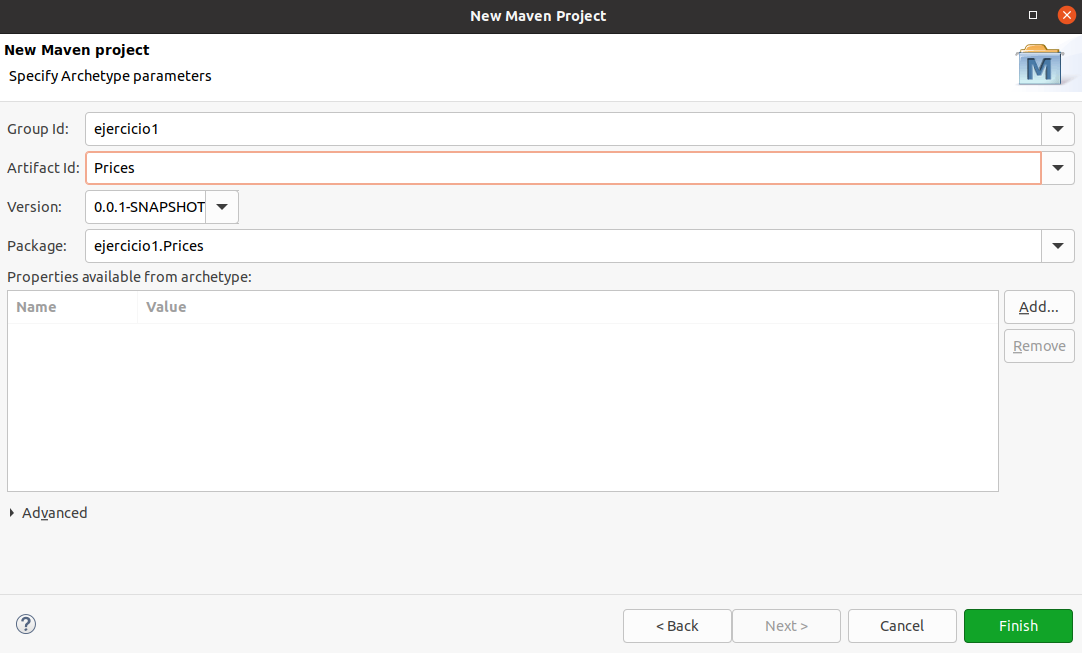
* Indicamos donde se va a ubicar el proyecto



* Seleccionamos un arquetipo para que nos monte la estructura de un proyecto java típico.



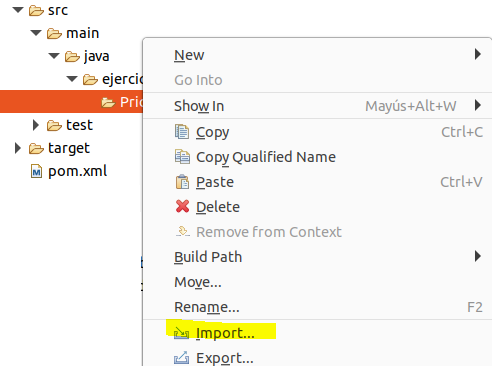
* Completamos el resto parámetros como se muestran en pantalla y damos a Finish



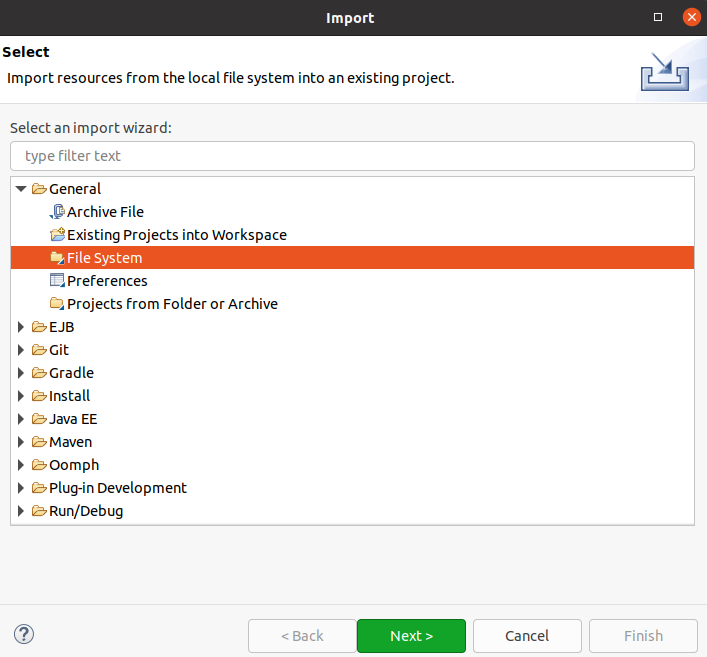
* Con el proyecto ya creado configuramos el pom.xml e indicamos la dependencia con hadoop.



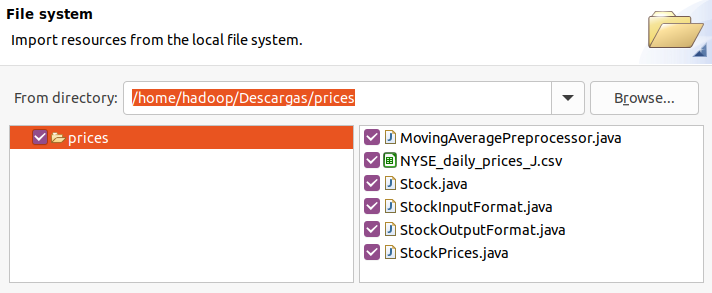
* Ahora descargamos de la sección de materiales del curso Hadoop: Recursos\_Sesión6\_Mapreduce donde tenemos dos proyectos: Prices y Twitter, en este caso vamos a trabajar con Prices. Importamos las clases java como se muestra a continuación:
  + Paso 1: Importar



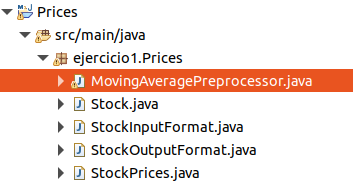
* + Paso 2: Sistema de ficheros



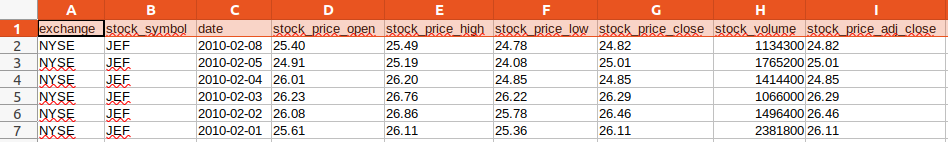
* + Paso 3: indicamos el directorio y seleccionamos todos los ficheros y le damos a finish.



* Después de seguir todos los pasos el resultado debería ser el siguiente:



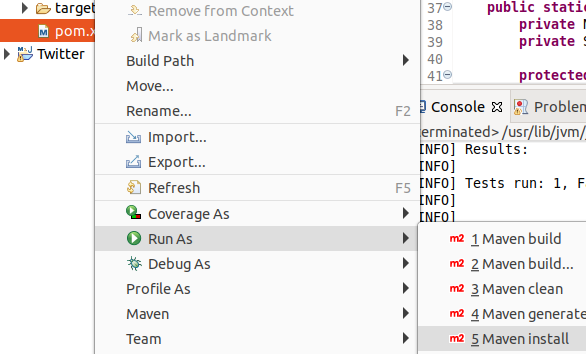
* Antes de comenzar hacemos un preview del archivo que viene en la carpeta NYSE\_daily\_prices\_J.csv y sobre el que vamos a trabajar.



* Podemos ver que se trata de una colección de valores de un mercado bursátil sobre el que deseamos establecer la diferencia entre el precio de cierre y apertura para cada uno de los registros, la clave estará formada por stock\_symbol+date y el valor será la diferencia entre stock\_price\_close y stock\_price\_open:
  + Key: concatenación de stock\_symbol+date en String
  + Value: diferencia stock\_price\_close - stock\_price\_open en Double
* Para empezar creamos el directorio de entrada y subimos el archivo con el que vamos a trabajar como se muestra a continuación:

|  |
| --- |
| hdfs dfs -mkdir /user/hadoop/inPrices  hdfs dfs -put /home/hadoop/Descargas/prices/NYSE\_daily\_prices\_J.csv /user/hadoop/inPrices/ |

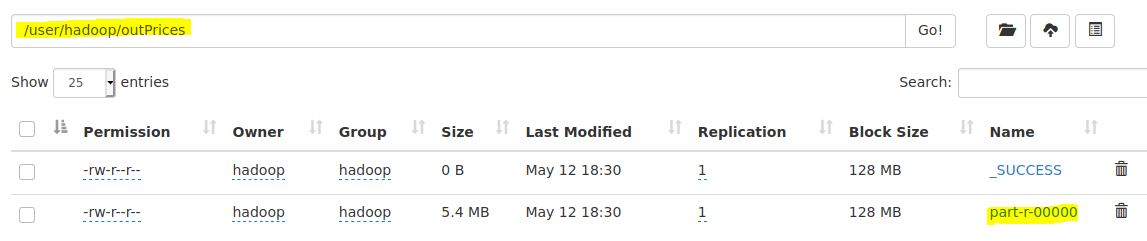
* Si ejecutamos tal y como tenemos ahora el proyecto:
  + Generar el Jar:

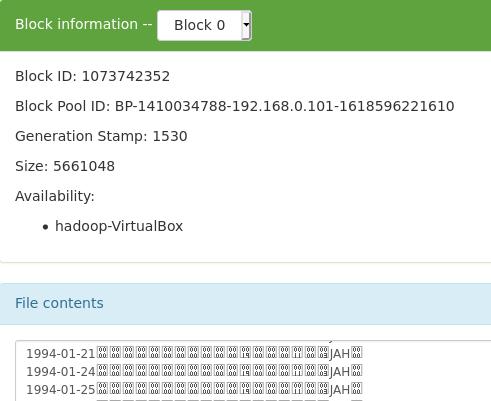


* + Ejecutar mediante hadoop:



* Vemos que no estamos obteniendo el resultado deseado.





* Para poder tener la salida deseada tenemos que realizar los siguientes pasos:

1. En StockInputFormat.java no estamos extrayendo los valores para Value que deben pasar al proceso Map, para ello añadir en la sección marcada como “//--completar” con:

|  |
| --- |
| value.setOpen(Double.parseDouble(values[3]));  value.setClose(Double.parseDouble(values[6])); |

1. Ahora ya estamos retornando el par clave-valor correcto.
2. Nos vamos a MovingAveragePreprocessor.java y modificamos el método map para devolver en la variable outputValue el valor de la diferencia entre los 2 valores que acabamos de añadir al Value:

|  |
| --- |
| outputValue.set(value.getClose()-value.getOpen()) |

1. El método map mandará ahora a la fase reduce:
   1. Key: el Objeto Stock
   2. Value: la diferencia entre valores deseada
2. Ahora nos toca completar el método reduce completando:

|  |
| --- |
| double prices = 0;  for(DoubleWritable value : values){  prices+=value.get();  }  context.write(key, new DoubleWritable(prices)); |

1. Dentro de la misma clase en el método run cambiamos la configuración del job para que tome como clase Reducer la que acabamos de crear

|  |
| --- |
| job.setReducerClass(~~Reducer.class~~)  job.setReducerClass(**StockReducer.class**) |

1. Si ejecutamos ahora vemos que no nos muestra el resultado deseado.
2. Para poder formatear los resultados de salida tenemos que indicarle al Job que utilice la clase StockOutputFormat

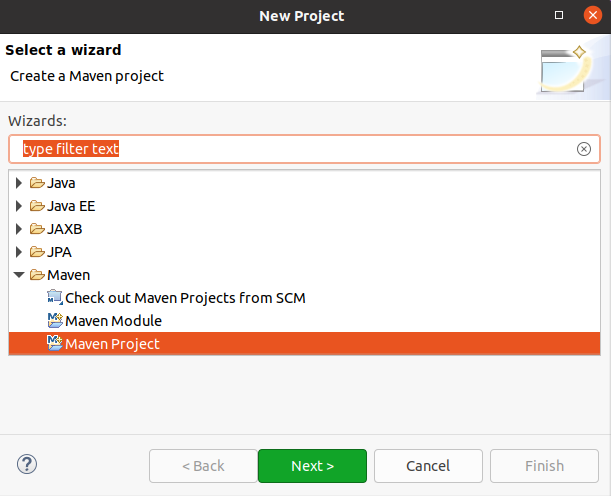
|  |
| --- |
| job.setOutputFormatClass(~~SequenceFileOutputFormat.class~~)  job.setOutputFormatClass(**StockOutputFormat.class**) |

* Si se han seguido todos los pasos, adjuntar en el siguiente recuadro una captura de la muestra de salida que puede obtenerse desde la ui de HDFS mediante la utilidad de Browse the file system: (localhost:9870)

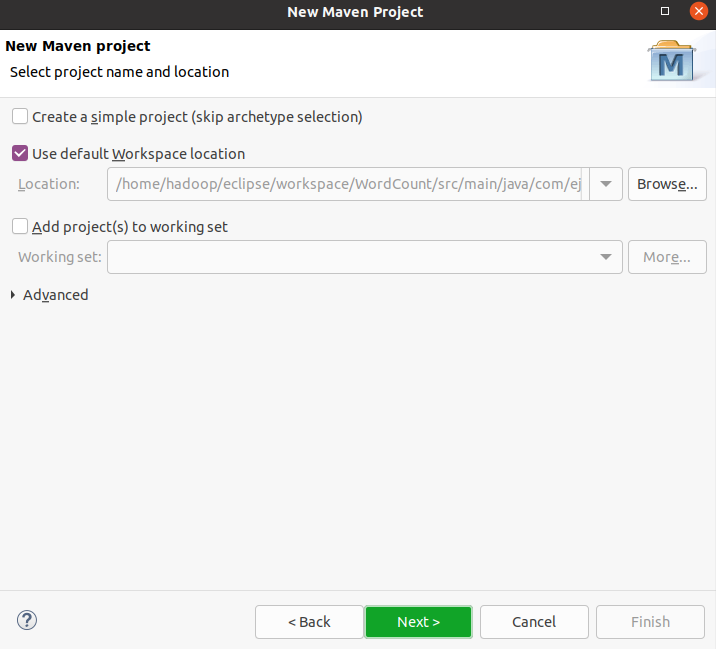
|  |
| --- |
| **EJERCICIO** |

### Ejercicio 2 – Calcular el Momento en Twitter

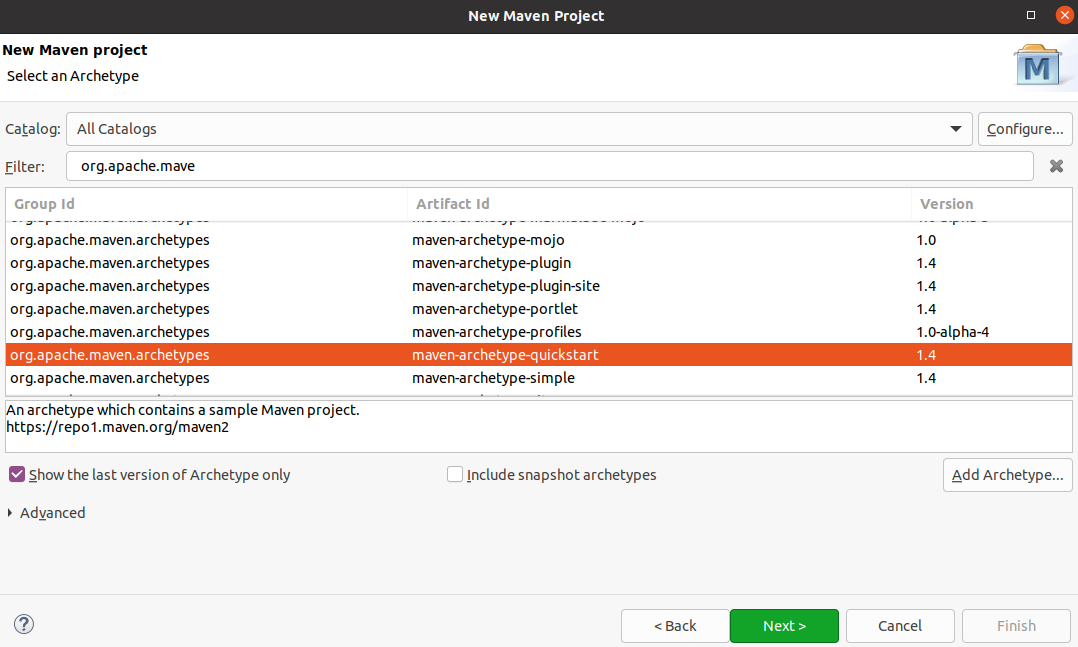
* Creamos un nuevo proyecto Maven en Eclipse



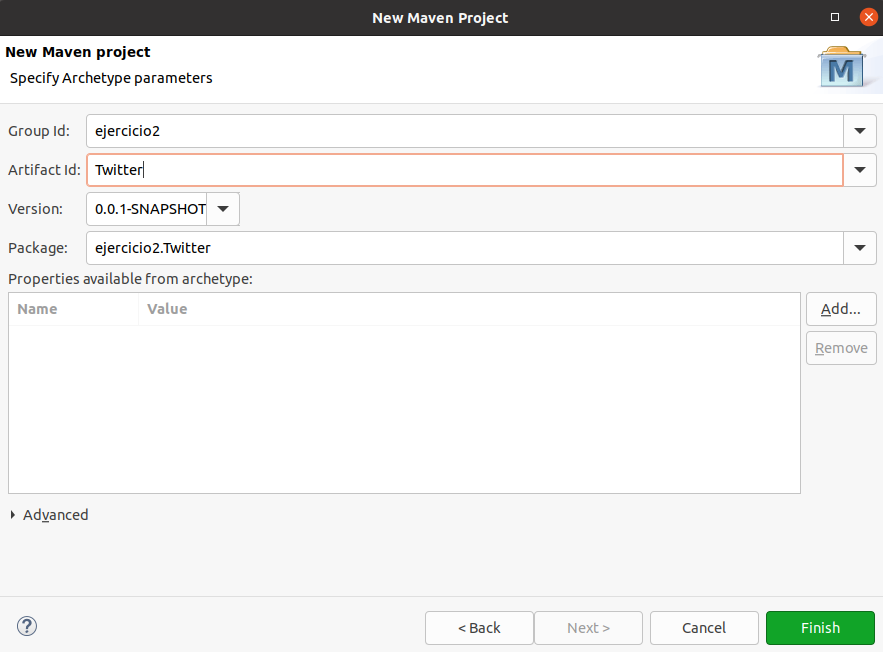
* Indicamos donde se va a ubicar el proyecto



* Seleccionamos un arquetipo para que nos monte la estructura de un proyecto java típico.



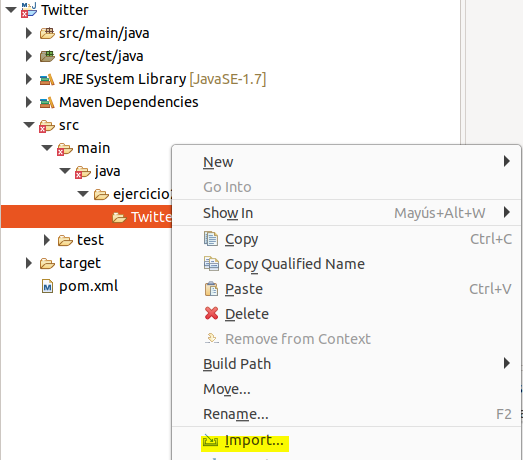
* Completamos el resto parámetros y damos a Finish



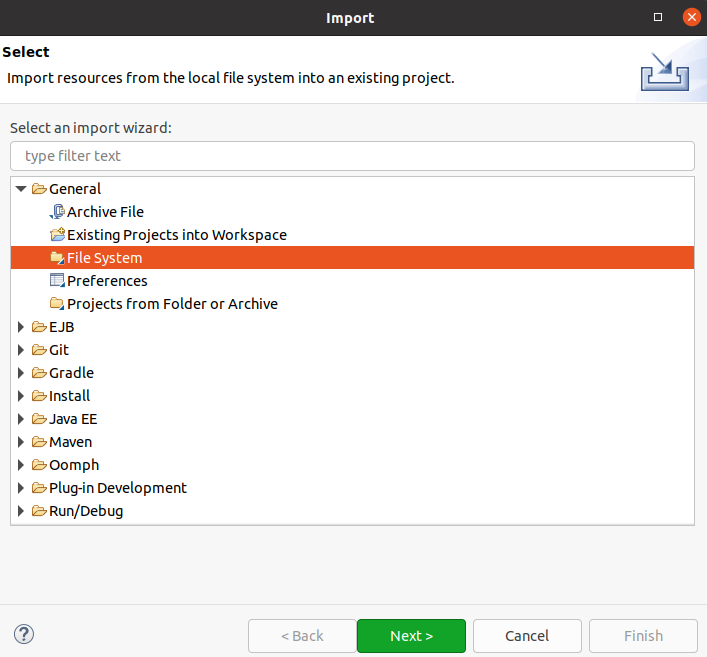
* Con el proyecto ya creado configuramos el pom.xml e indicamos la dependencia con hadoop.



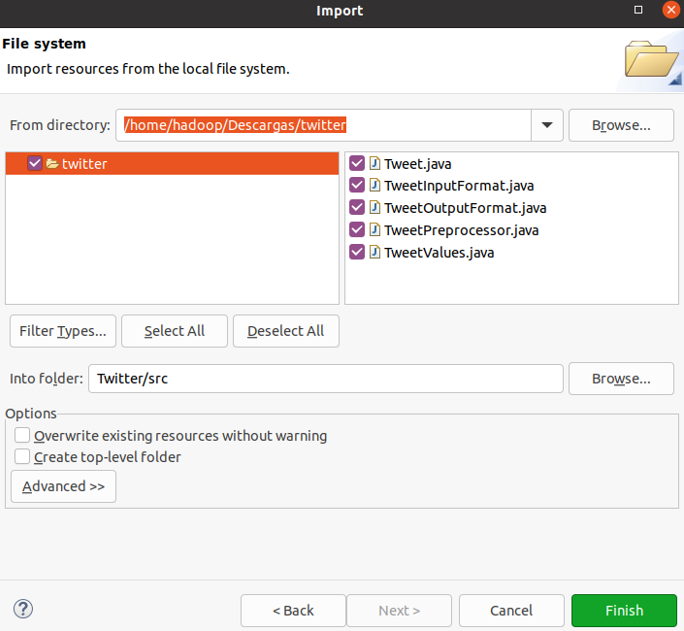
* Ahora descargamos de la sección de materiales del curso Tema 4: Recursos\_Mapreduce donde tenemos una serie de clases y un json. Importamos las clases java como se muestra a continuación:
  + Paso 1: Importar



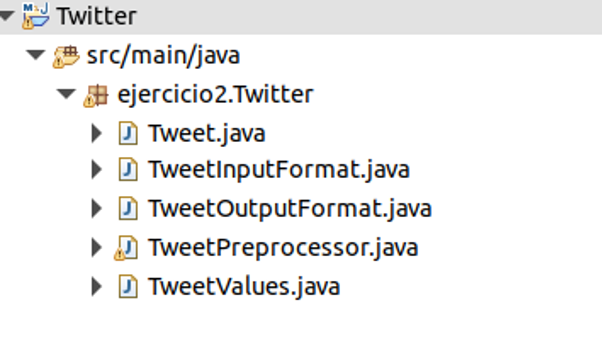
* + Paso 2: Sistema de ficheros



* + Paso 3: indicamos el directorio y seleccionamos todos los ficheros y le damos a finish.



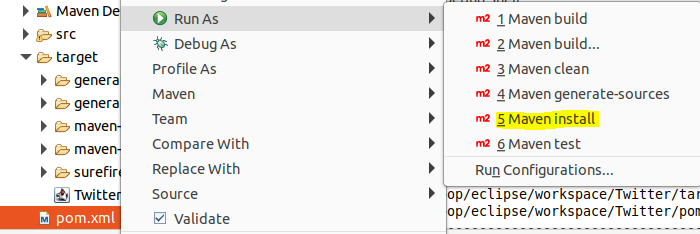
* Después de seguir todos los pasos el resultado debería ser el siguiente:



* Para empezar creamos el directorio de entrada y subimos el archivo con el que vamos a trabajar como se muestra a continuación:

|  |
| --- |
| hdfs dfs -mkdir /user/hadoop/inTwitter  hdfs dfs -put /home/hadoop/Descargas/twitter/cache-1000000.json /user/hadoop/inTwitter/ |

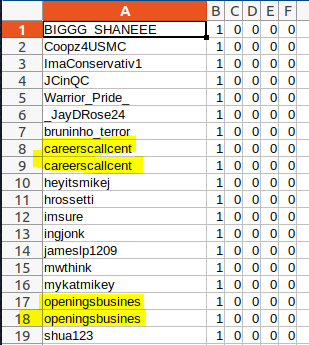
* Si creamos el target como en el ejercicio anterior:



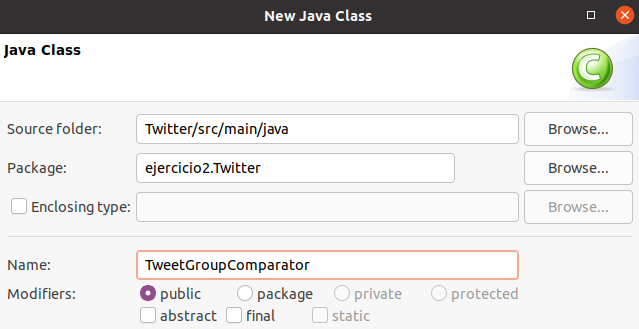
* Ejecutamos el jar con el comando hadoop:

|  |
| --- |
| hadoop jar Twitter-0.0.1-SNAPSHOT.jar ejercicio2.Twitter.TweetPreprocessor |

* Si comprobamos el archivo que ha generado, vemos que el reduce no está funcionando bien:



* Es necesario crear una nueva clase que ayude a Mapreduce a comparar las claves. Para ello creamos una nueva clase newClass



* Que tendrá el siguiente código:

|  |
| --- |
| package twitter;  import org.apache.hadoop.io.WritableComparable;  import org.apache.hadoop.io.WritableComparator;  public class TweetGroupComparator extends WritableComparator {  protected TweetGroupComparator(){  super(Tweet.class, true);  }  @SuppressWarnings("rawtypes")  @Override  public int compare(WritableComparable a, WritableComparable b){  Tweet l = (Tweet) a;  Tweet r = (Tweet) b;    return l.getSreen\_name().compareTo(r.getSreen\_name());  }  } |

* En el job, especificamos el GroupingComparator:

|  |
| --- |
| job.setGroupingComparatorClass(TweetGroupComparator.class); |

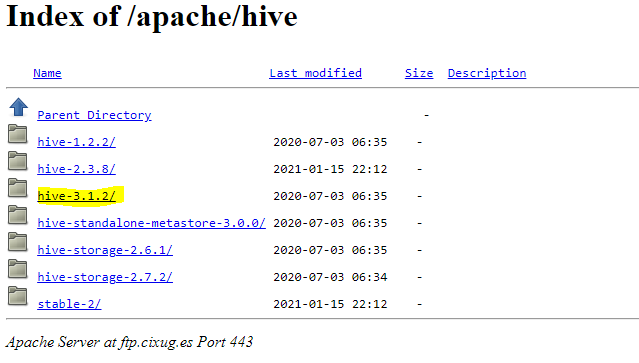
* Volvemos a generar el jar y a ejecutarlo para comprobar que es correcto.
* Ajduntad la salida del comando: hdfs dfs -cat /user/hadoop/outTwitter/Estadisticas\_Twitter\_0.csv

|  |
| --- |
| **EJERCICIO** |

## HIVE

### Instalación y configuración básica de Hive (**Solo para aquellos que quieran realizar la instalación completa de Hive**)

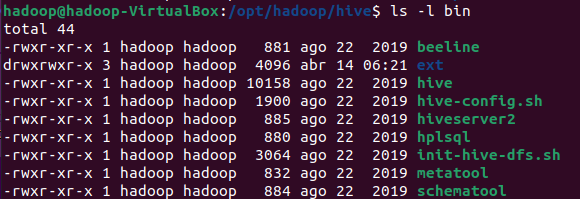
* Accedemos a la página de Apache Hive para descargárnoslo: <http://www.apache.org/dyn/closer.cgi/hive/>



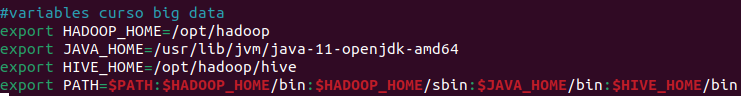
* Descomprimimos el archivo

|  |
| --- |
|  |

* Si queremos, podemos moverlo al mismo directorio donde tenemos hadoop
* Si miramos el directorio podemos ver:



* + HIVE: Herramienta cliente
  + BEELINE: Otra herramienta cliente
  + HISERVER2: Nos permite arrancar el servidor de Hive
  + SCHEMATOOL: Nos permite trabajar contra la base de datos de metadataos (Metastore)
* Configuramos las variables de entorno



* Copiamos para renombrar los ficheros de configuración de hive:
  + Fichero primario de configuración de Hive.

|  |
| --- |
| cp hive-default.xml.template hive-site.xml |

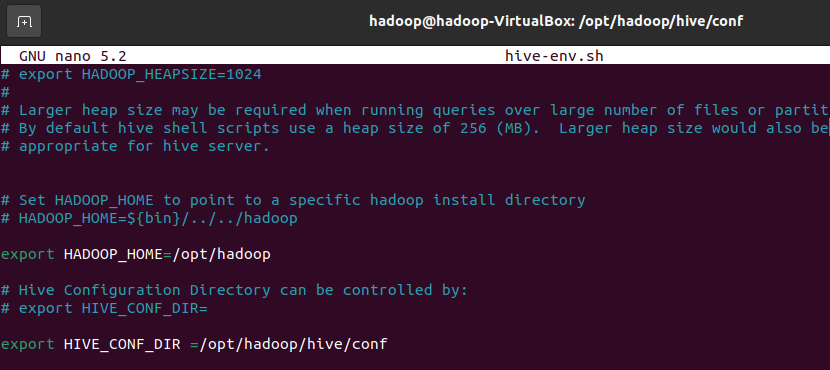
* + El fichero de entorno es el que se invoca desde los distintos ejecutables de Hive para encontrar las variables de entorno.

|  |
| --- |
| cp hive-env.sh.template hive-env.sh |

* + Por último hacemos lo mismo con los ficheros de log.

|  |
| --- |
| cp hive-exec-log4j.properties.template hive-execlog4j.properties  cp hive-log4j.properties.template hive-log4j.properties  cp beeline-log4j2.properties.template beeline-log4j2.properties |

* El siguiente paso es configurar los ficheros que acabamos de copiar
  + Modificamos hive.env y añadimos los siguientes export



* + Modificamos hive-site.xml y añadimos las siguientes propiedades por un lado para que no nos dé error y por otro para que pille nuestro usuario mientras estemos trabajando.

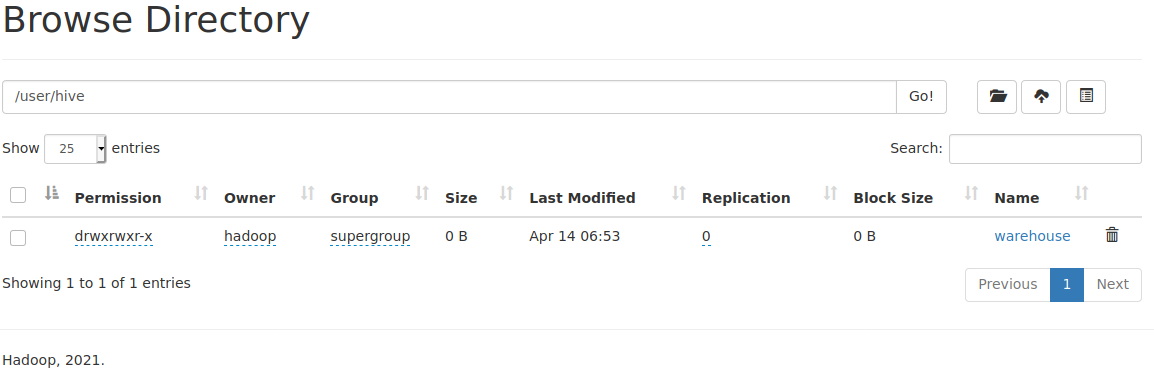
|  |
| --- |
| <property>  <name>system:java.io.tmpdir</name>  <value>/tmp/hive/java</value>  </property>  <property>  <name>system:user.name</name>  <value>${user.name}</value>  </property> |

* Dentro del mismo fichero configuramos la propiedad javax.jdo.option.ConnectionURL para indicarle que vamos a trabajar con una bbdd derby.

|  |
| --- |
| <property>  <name>javax.jdo.option.ConnectionURL</name>  <value>jdbc:derby:/opt/hadoop/hive/bbdd/metastore\_bd;databaseName=metastore\_db;creat\_true</value>  </property> |

* Con los ficheros de configuración ya listos Hive nos pide disponer de un directorio en HDFS propio para ello ejecutamos los siguientes comandos sobre HDFS (**El nombre tiene que ser el mismo que se especifica aquí no se puede poner otro nombre**).

|  |
| --- |
| hdfs dfs –mkdir /tmp  hdfs dfs –mkdir -p /user/hive/warehouse  hdfs dfs -chmod g+w /tmp  hdfs dfs -chmod g+w /user/hive/warehouse |



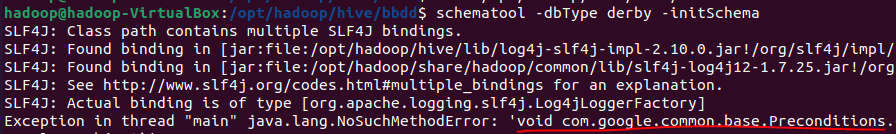
* Por último tenemos que crear un directorio donde guardar el metastore, para ello, dentro del propio directorio creamos una entidad llamada bbdd.



* Una vez creado el directorio, ejecutamos el comando schematool para crear el metastore.



* Aquí nos encontraremos con un par de problemas y es que primeramente nos dará un error por la versión de guava que tiene por defecto hive.



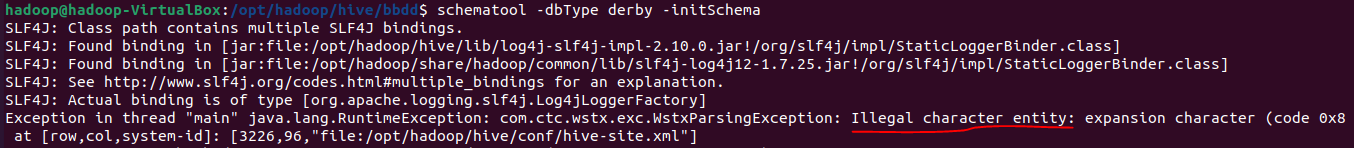
* Para solucionarlo basta con borrar el que trae y copiar el guava.jar que viene en el directorio de hadoop, en este caso podemos ver que en hive tenemos la versión 19 y hadoop la 27, esta es la que tenemos que copiar de



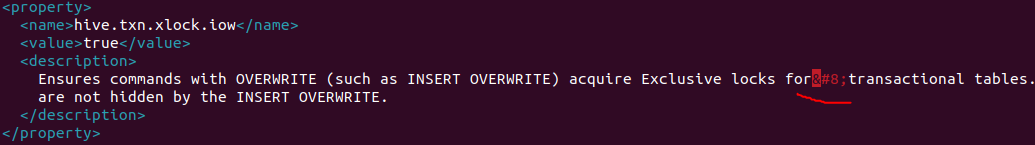




* Una vez realizados los pasos siguientes, si intentamos lanzar de nuevo el comando schematool, nos volverá a dar problemas:



* Tenemos que irnos al fichero hive-site.xml y borrar



* Una vez hecho este último paso, ya sí que podemos lanzar el comando schematool



* Si todo ha ido bien veremos nuestra mestastore ya creada:



* Para comprobar que todo ha ido podemos probar lanzando el comando: hive y comprobar que arranca.



### 2.2. Ejemplo de creación y consultas básicas con HIVE

(\*) Nota, **si no** has instalado hive en el paso anterior, es probable que al ejecutar “hive” con la instalación de la Máquina Virtual dé un error producido por el metastore. Para solucionarlo basta con dirigirnos al directorio: /opt/hadoop/hive/ y borrar el contenido del directorio bbdd como se muestra a continuación:



Luego, nos movemos al directorio y ejecutamos schematool para que genere de nuevo el metastore



Una vez hecho estos pasos, se puede ejecutar “hive” sin problemas.



* Sobre el cliente: hive, creamos una base de datos llamada ***curso***

|  |
| --- |
| create database curso; |

* Nos conectamos a la Base de Datos ***curso***

|  |
| --- |
| use curso; |

* Creamos la siguiente tabla Interna

|  |
| --- |
| CREATE TABLE IF NOT EXISTS empleados\_interna  (  name string,  work\_place ARRAY<string>,  sex\_age STRUCT<sex:string,age:int>,  skills\_score MAP<string,int>,  depart\_title MAP<STRING,ARRAY<STRING>>  )  COMMENT 'Esto es una tabla interna'  ROW FORMAT DELIMITED  FIELDS TERMINATED BY '|'  COLLECTION ITEMS TERMINATED BY ','  MAP KEYS TERMINATED BY ':' |

* Cargamos los datos del fichero empleados.txt que se encuentra en la carpeta Recursos\_Sesion6 en la carpeta hive (en este ejemplo lo descomprimido en Descargas)

|  |
| --- |
| LOAD DATA LOCAL INPATH '/home/hadoop/Descargas/empleados.txt' OVERWRITE INTO TABLE empleados\_interna; |

* Comprobamos que realmente lo tenemos en el directorio warehouse de HIVE, dentro de la base de datos ***curso***
* Ahora vamos a crear una tabla externa

|  |
| --- |
| CREATE EXTERNAL TABLE IF NOT EXISTS empleados\_externa  (  name string,  work\_place ARRAY<string>,  sex\_age STRUCT<sex:string,age:int>,  skills\_score MAP<string,int>,  depart\_title MAP<STRING,ARRAY<STRING>>  )  COMMENT 'Esto es una tabla externa'  ROW FORMAT DELIMITED  FIELDS TERMINATED BY '|'  COLLECTION ITEMS TERMINATED BY ','  MAP KEYS TERMINATED BY ':'  LOCATION '/ejemplo/empleados; |

* Hacemos la misma carga que en el caso anterior

|  |
| --- |
| LOAD DATA LOCAL INPATH '/home/hadoop/Descargas/empleados.txt' OVERWRITE INTO TABLE empleados\_externa; |

* Hacemos una select \* sobre la tabla para ver que están todos los campos

|  |
| --- |
| SELECT \* FROM EMPLEADOS\_EXTERNA; |

* Hacemos una select para buscar al empleado “Will”

|  |
| --- |
| SELECT \* FROM EMPLEADOS\_EXTERNA WHERE NAME = ‘Will’ |

### Supuesto práctico

Ahora vamos a trabajar con las tablas que dos de las entidades que creamos en la sesión 4 para el modelo dimensional:

* DIMCENTRO con la información de centros del dataset de la Generalitat de Catalunya
* HECINCIDENCIACOVID con la información de la aplicación Traçacovid

Dentro de Recursos\_Sesion6 en la carpeta hive podréis encontrar ambos .csv relacionados con cada una de estas tablas.

Una vez comprobados los campos que contienen, se pide realizar dentro de la base de datos **curso** las siguientes acciones:

1. Crear la tabla DIMCENTRO

|  |
| --- |
|  |

1. Cargar la tabla DIMCENTRO desde el fichero DIMCENTRO.csv

|  |
| --- |
|  |

1. Crear la tabla HECINCIDENCIACOVID

|  |
| --- |
|  |

1. Cargar la tabla HECINCIDENCIACOVID desde el fichero HECINCIDENCIACOVID.csv

|  |
| --- |
|  |

1. Sacar el nombre de los 3 centros que estuvieron abiertos el día ‘18/10/2020’ con el mayor número de confinados ordenados de mayor a menor

|  |
| --- |
| [Adjuntar captura de pantalla de query y resultados] |

Para hacer el seguimiento del curso enviar esta memoria a [clopez@teralco.com](mailto:clopez@teralco.com)

### Probar con Beeline, Hiveserver2 y Hue (**Solo para aquellos que hayan hecho la instalación completa**)

Lanzamos el servidor de hive

|  |
| --- |
| hiveserver2 |

Lanzamos beeline

|  |
| --- |
| beeline |

Dentro de beeline nos conectamos al servidor de hive

|  |
| --- |
| !connect jdbc:hive2://localhost:10000; |

Especificamos usar la base de datos **curso**

|  |
| --- |
| use curso; |

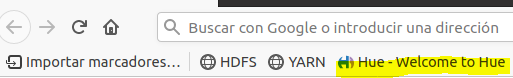
Lanzamos la misma consulta del ejercicio anterior

|  |
| --- |
|  |

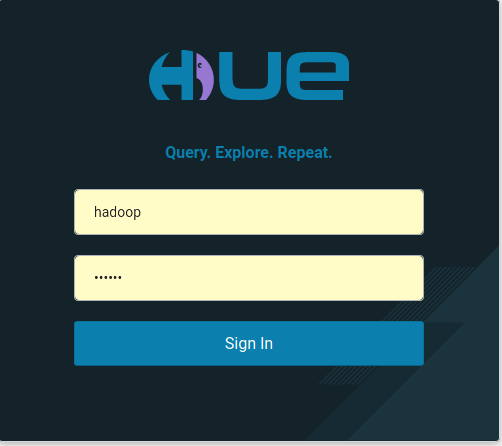
Ahora arrancamos hue

|  |
| --- |
|  |

Y nos vamos al enlace que tenéis en Firefox: **localhost:8000**



El usuario y la contraseña es hadoop/hadoop



Podemos investigar nuestra base de datos **curso y** lanzar queries sencillas, no obstante es al utilizar derby como bbdd para el metastore es bastante inestable y provoca fallos, ya que su uso está recomendado para que el metastore esté ubicado en una base de datos tipo mysql o postgres.