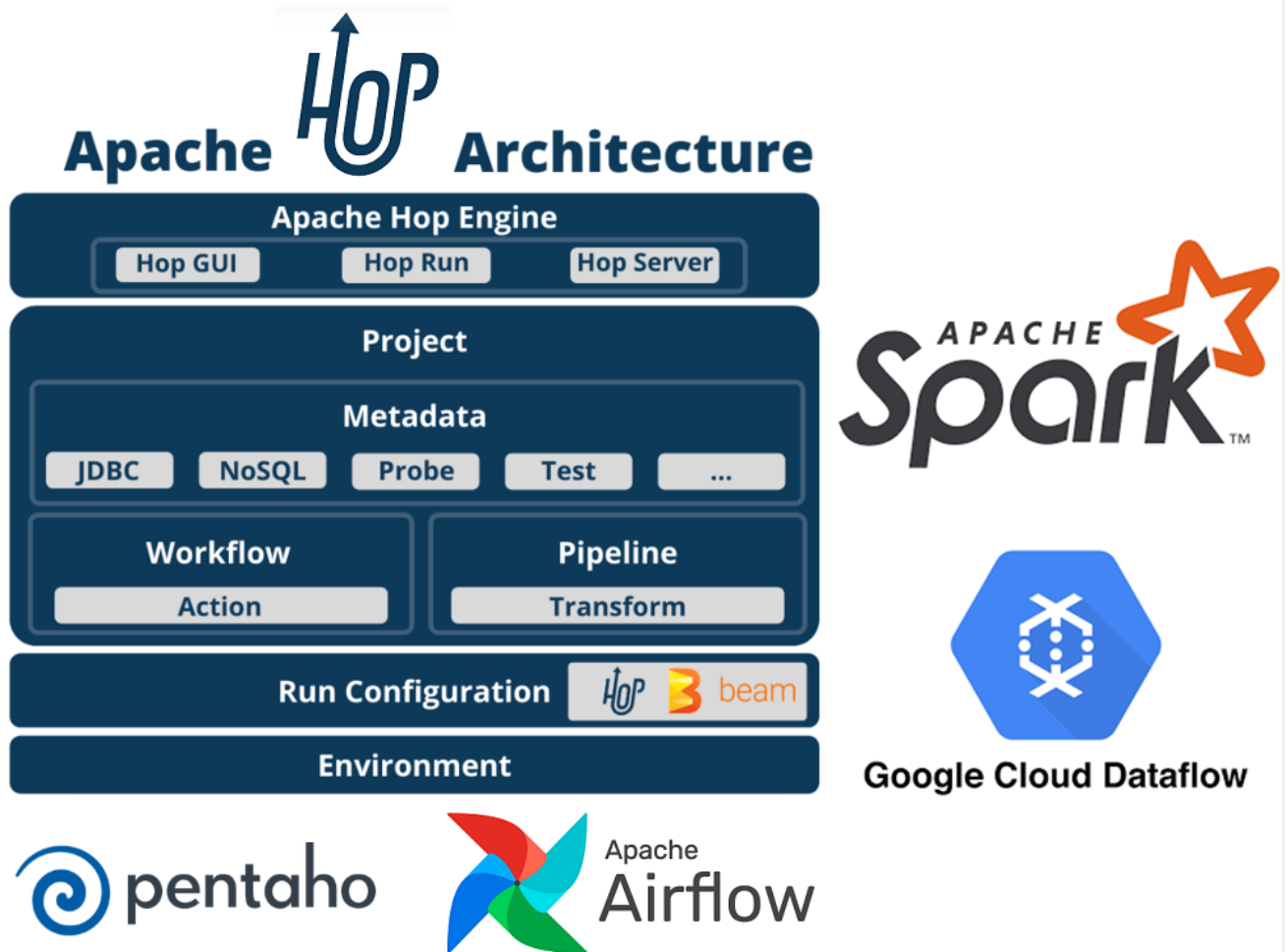


# Apache Hop: Ejecución de pipelines con Apache Spark y Google Dataflow



# CONTENIDO

<b>1. INTRODUCCIÓN</b> .....	<b>4</b>
REFERENCIAS .....	7
DESCARGA .....	7
<b>2. OVERVIEW DE LA HERRAMIENTA</b> .....	<b>8</b>
TRANSFORMACIONES DISPONIBLES .....	8
PROYECTOS Y ENVIRONMENTS .....	9
JDBC DRIVERS Y OTROS PLUGINS .....	10
CONTROL DE VERSIONES .....	11
KETTLE IMPORT .....	11
<b>3. MIGRACIÓN Y COMPARACIÓN CON PENTAHO DATA INTEGRATION</b> .....	<b>12</b>
<b>4. EJECUCIÓN DE PIPELINES CON APACHE SPARK</b>	<b>18</b>
ARRANCAR SPARK .....	19
CONFIGURACIÓN DEL PIPELINE RUNTIME ENGINE DE SPARK EN HOP .....	21
EJECUCIÓN DEL PIPELINE .....	22

<b>5. EJECUCIÓN DE PIPELINES CON GOOGLE DATAFLOW .....</b>	<b>25</b>
<b>6. PLANIFICACIÓN DE LA ETL .....</b>	<b>29</b>
SCHEDULING CON GOOGLE DATAFLOW .....	29
SCHEDULING CON APACHE AIRFLOW .....	29
<b>7. LOGS Y HOP-RUN .....</b>	<b>34</b>
<b>8. BUENAS PRÁCTICAS .....</b>	<b>35</b>
NAMING .....	35
VARIABLES .....	35
GOBERNANZA.....	35
<b>9. CONCLUSIONES.....</b>	<b>37</b>

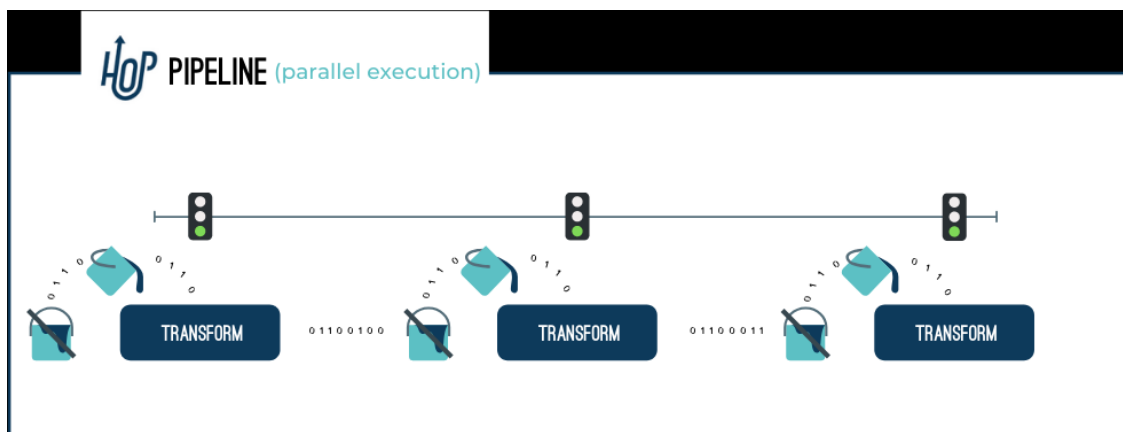
## 1. INTRODUCCIÓN

**Apache Hop es un fork del PDI de Pentaho (Pentaho Data Integration)** por lo que cualquiera que esté familiarizado con este último, no tendrá problema alguno en comenzar con Apache Hop. Sin embargo, Hop trae algunas mejoras notables en cuanto a la integración con otras tecnologías Big Data, así como en el workspace de la aplicación.

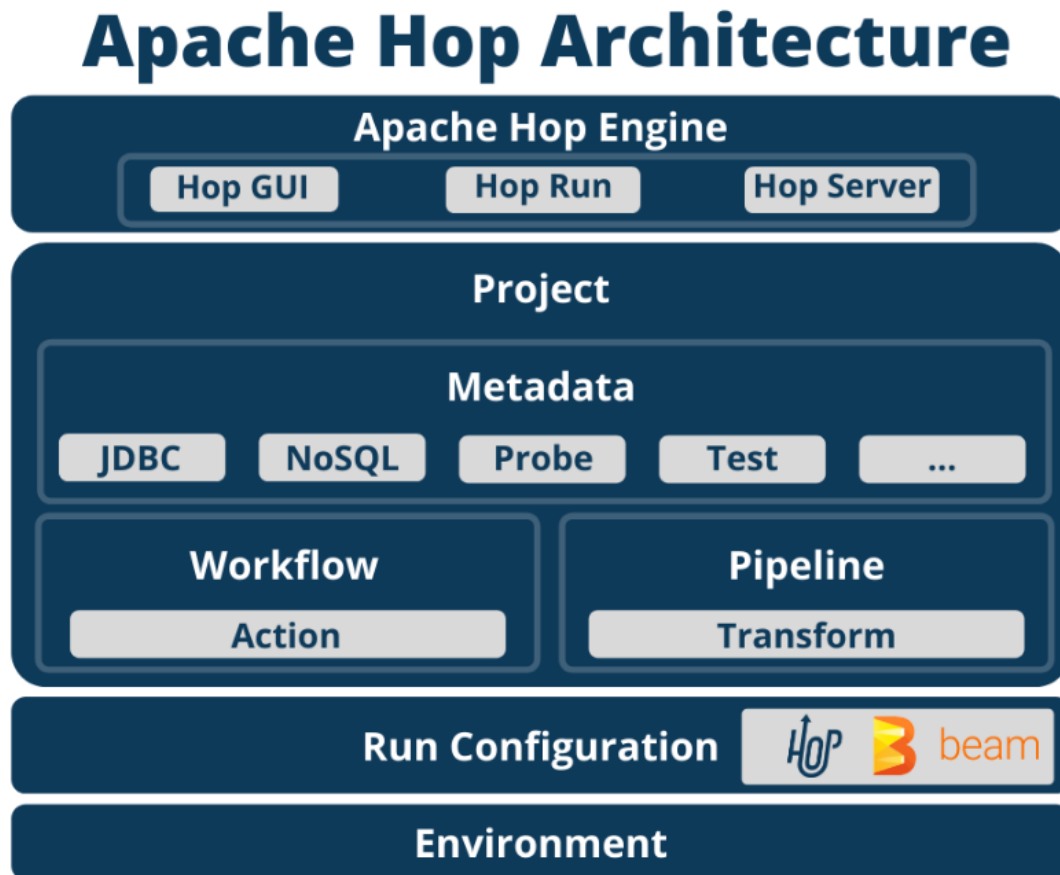


De la misma manera que Pentaho Data Integration, Hop permite que los profesionales en el mundo de los datos trabajen de manera visual empleando metadatos para describir cómo se han de procesar una serie de datos.

**La arquitectura de Hop está diseñada lo más flexiblemente posible y puede ser utilizado en cualquier escenario**, tanto on premise como en la nube, desde el IoT a grandes volúmenes de datos incluso desde nuestro sistema operativo o en containers y kubernetes.



Además de esto, **Hop dispone de varios engines sobre los cuales se pueden ejecutar nuestros pipelines y workflows mediante configuraciones del runtime de Apache Beam** (Apache Beam runtime configurations) entre las cuales podemos encontrar: native Hop engine tanto en local como en remoto, Apache Spark, Apache Flink o Google Dataflow.



**Ilustración 1: Arquitectura de Apache Hop**

## REFERENCIAS

### Documentación

[Overview de Apache Hop](#)

[Descarga e instalación](#)

[Pipelines con el motor de Apache Spark](#)

[Pipelines con el motor Google dataflow](#)

[Plugin para Apache Airflow](#)

### Buenas prácticas

[Buenas prácticas](#)

## DESCARGA

Para descargar una versión de Apache Hop simplemente tendremos que entrar en el [enlace](#) y elegir la versión que queramos.

De la misma manera que el pdi de Pentaho, podremos probar las herramientas de hop ejecutando los .sh o .bat dependiendo del sistema operativo en el que nos encontremos. Eso sí, como prerequisite hemos de tener instalada la versión 11 de java.

## 2. OVERVIEW DE LA HERRAMIENTA

Comenzaremos en primer lugar con una overview de la herramienta en la cual veremos sus características y elementos diferenciadores con respecto al pdi de Pentaho.

**Para iniciar Apache Hop ejecutaremos el fichero `hop-gui.bat` o `hop-gui.sh` dependiendo del sistema operativo.**

### Transformaciones disponibles

Cuando creamos un pipeline, al hacer click sobre cualquier parte del lienzo sobre el cual pondremos las transformaciones a realizar, se nos abrirá un cuadro donde podremos elegir el tipo de transformación a emplear.

**Muchas de las transformaciones de Pentaho están disponibles en Hop** a diferencia de alguna que otra en los apartados de Big Data de cada una que puedan diferir. Pero en general, disponemos de lo mismo.



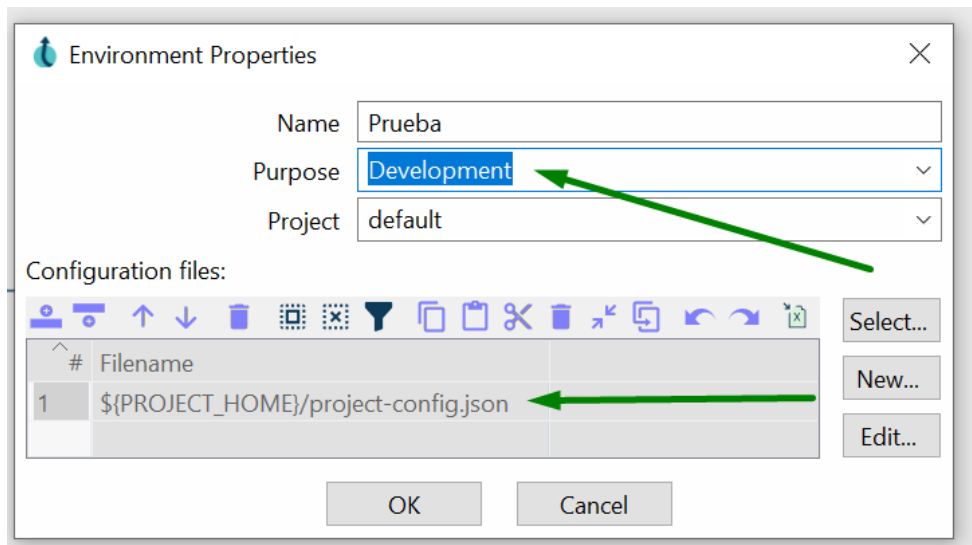
## Proyectos y environments

**Los proyectos Hop son una agrupación conceptual de configuraciones, variables, objetos de metadatos, pipelines y workflows.** Los proyectos pueden heredar metadatos de proyectos parent. Un proyecto contiene uno o más entornos donde se define la configuración real. Por ejemplo:

Un proyecto de 'Ventas' contiene una conexión de base de datos de 'clientes' y una serie de workflows y pipelines. Las configuraciones ejecución, las propiedades de conexión de la base de datos, etc., se definen en los environments 'dev', 'uat' y 'prd'.

**Los environments son instancias de proyectos que contienen las configuraciones de ejecución reales y otros objetos de metadatos para un proyecto.**

Ejemplo: El entorno 'dev' para el proyecto 'Ventas' especifica leer desde el host '10.0.0.1' para la conexión de base de datos de 'clientes'.



**Ilustración 2: Configuración de un environment**

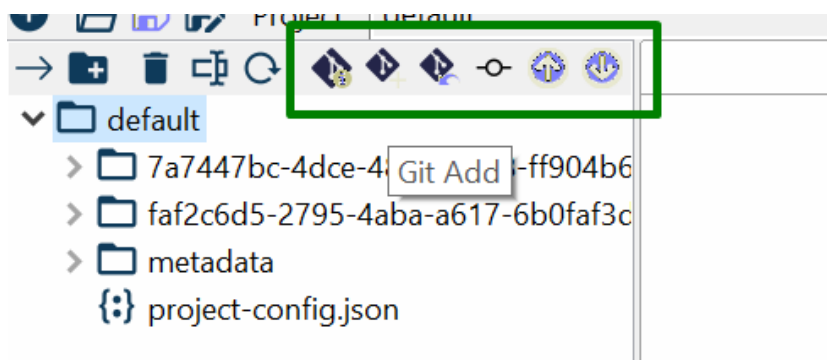
## JDBC drivers y otros plugins

Apache Hop viene con soporte para varios tipos de bases de datos y otras tecnologías, En caso de que se necesiten otras librerías y plugins se añadirán al directorio /lib dentro de la instalación de Hop.

En caso de añadir JDBC drivers para mysql por ejemplo, situaremos el .jar en `<PATH>/hop/plugins/databases/mysql`

## Control de versiones

El control de versiones es algo importante para cualquier tipo de proyecto, en este caso, Hop viene con opciones integradas de git en el propio file explorer.



**Ilustración 3: Git para el control de versiones**

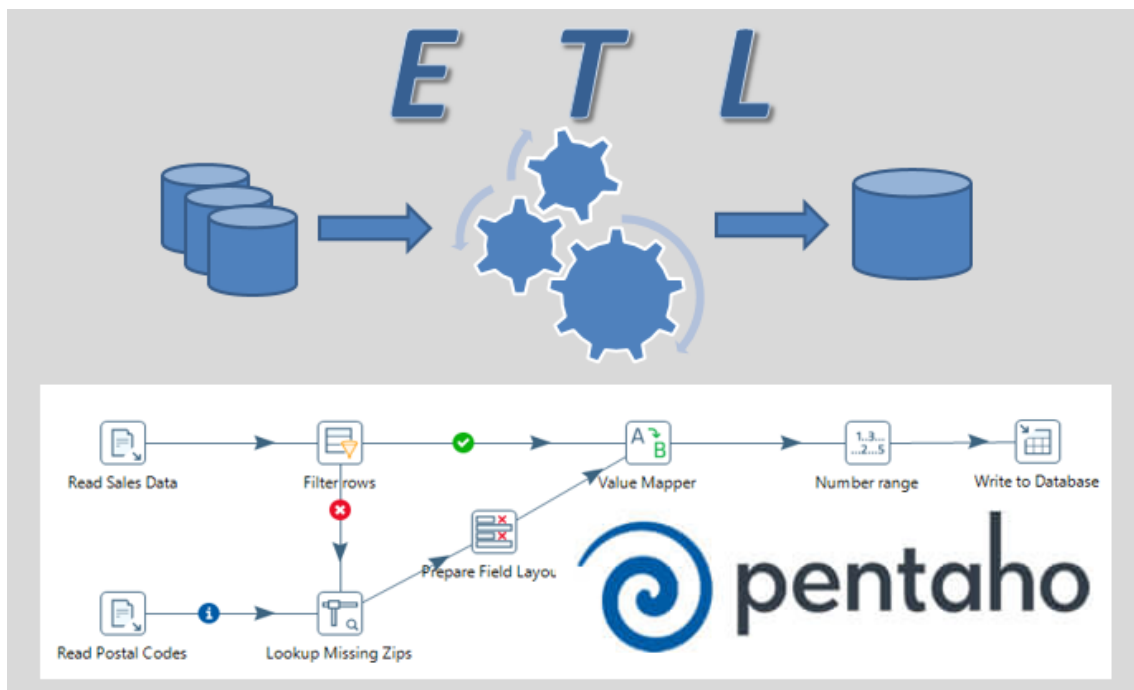
## Kettle import

Al tratarse de un fork de Pentaho Data Integration, **podemos emplear este plugin para importar transformaciones y Jobs.** Podemos realizarlo mediante Files → Import from Kettle/PDI en el menú de Hop o usando el script hop-import con la opción -type kettle.

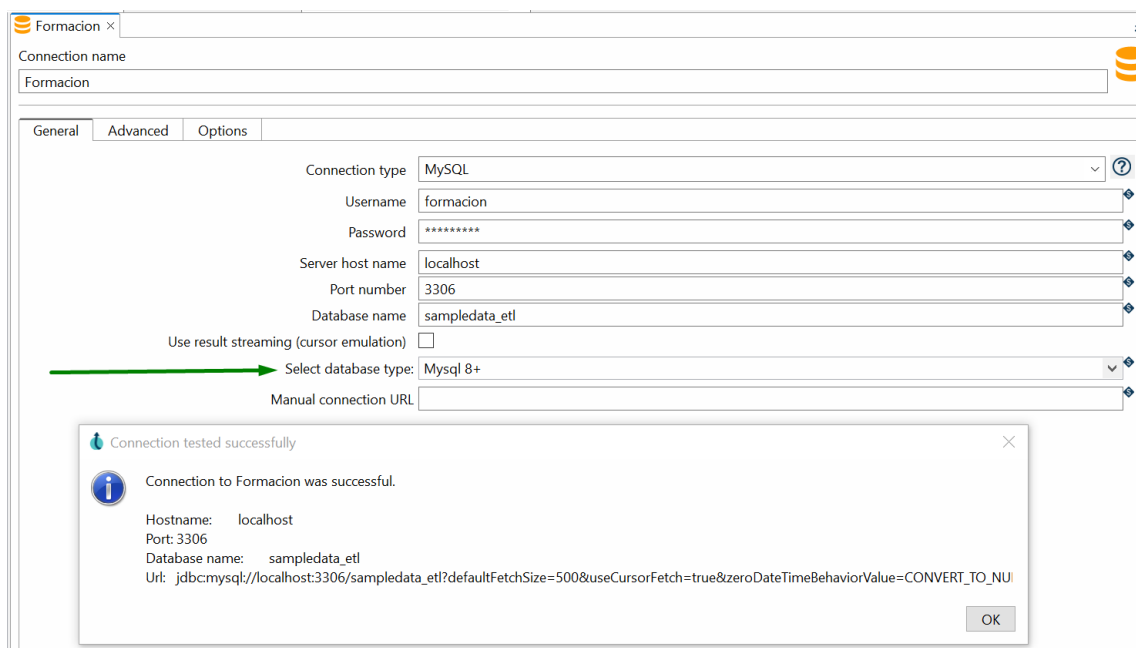
Esto es algo importante puesto que, si se considera una migración a Apache Hop desde Kettle, vemos que ambas herramientas son perfectamente compatibles.

### 3. MIGRACIÓN Y COMPARACIÓN CON PENTAHO DATA INTEGRATION

Veremos un pequeño caso práctico en el cual comprobaremos el funcionamiento de las distintas transformaciones que Hop nos ofrece, así como su rendimiento y peculiaridades de la herramienta. Lo que haremos será replicar el proceso ETL de la formación de Kettle/PDI que realizamos en Stratebi.



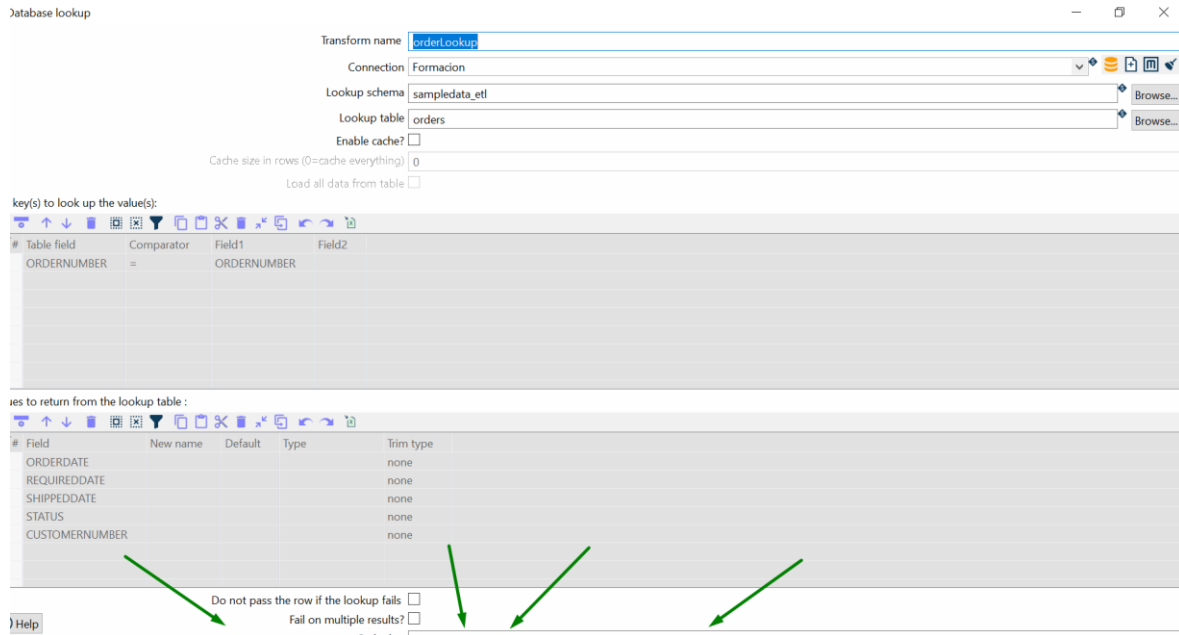
En primer lugar, estableceremos una nueva conexión del tipo mysql a la base de datos formación\_etl que se nos proporciona. Importante comprobar los jdbc drivers.



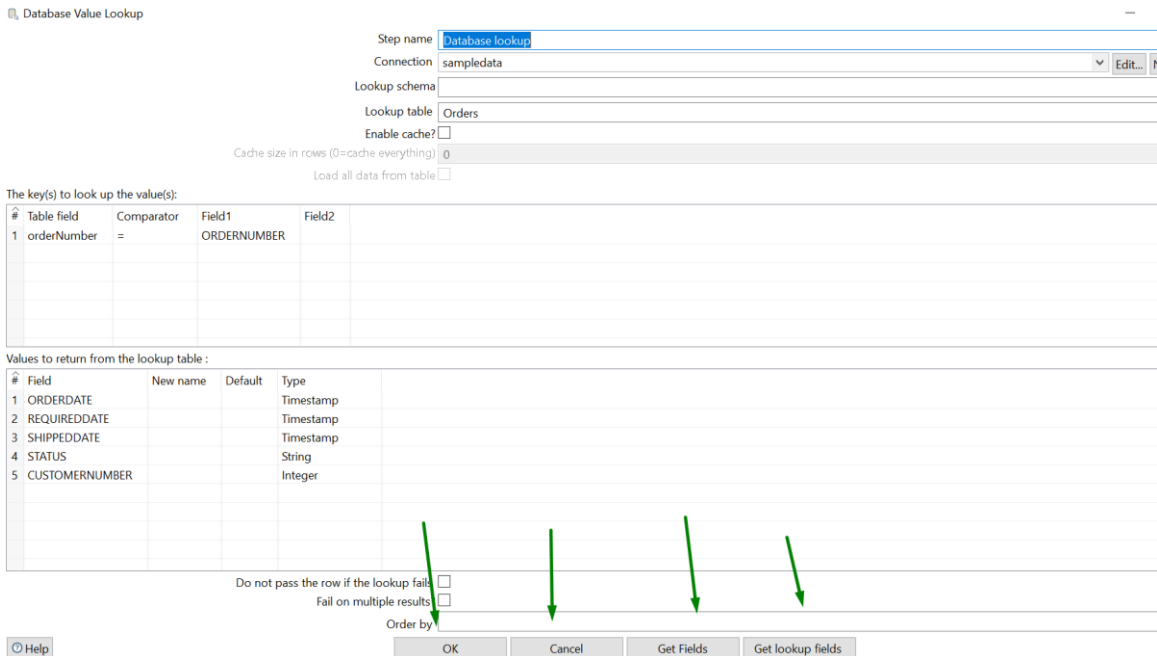
## Ilustración 4: Establecer conexión con mysql

Importante especificar Mysql 8+ si tenemos una versión de mysql superior o igual a la 8. Si no se especifica, no nos conectará

Posteriormente nuestra tarea será la de ir completando el ejercicio de ETL de Kettle. Todo es bastante similar exceptuando algunas cosas de interfaz como por ejemplo que no se vea el botón de OK en ciertas transformaciones como el database lookup.



## Ilustración 5: Database lookup con Hop faltan botones

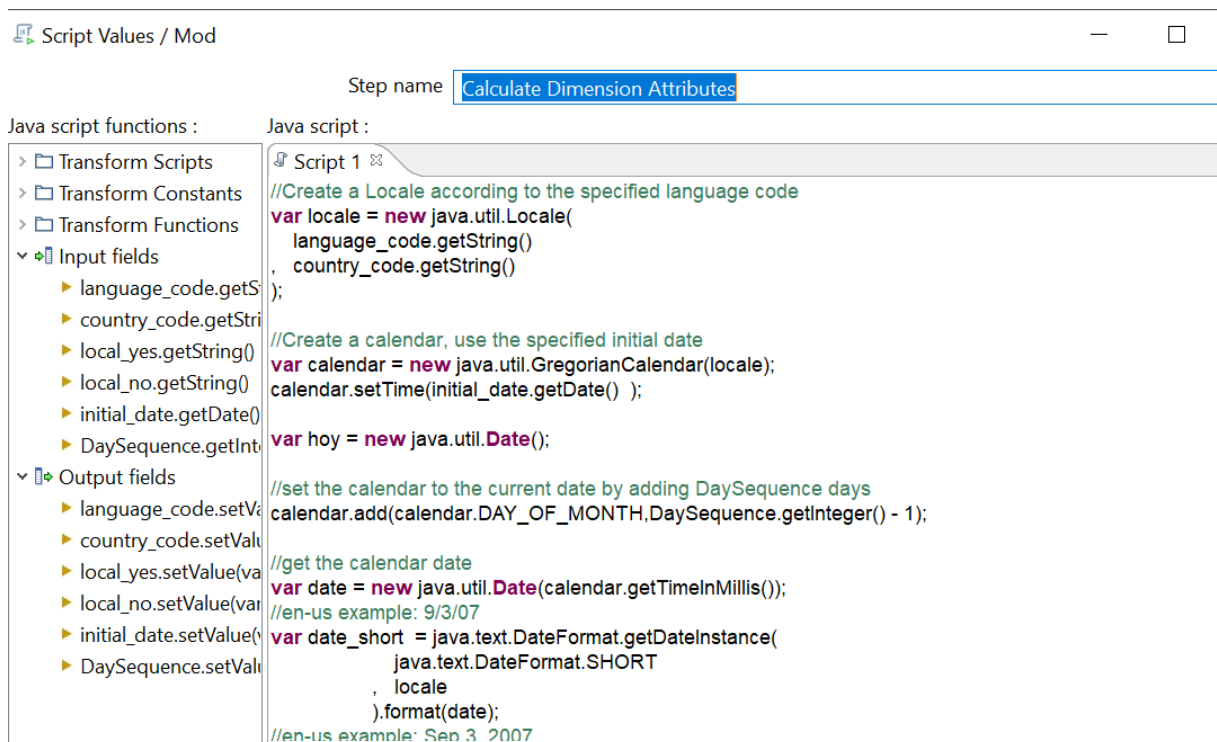


## Ilustración 6: Database lookup con Kettle

Otra peculiaridad que hemos encontrado es la interfaz a la hora de escribir código en javascript.

En Kettle sí que se te resaltan las variables y las funciones que se utilizan como en un IDE mientras que Hop no hace lo mismo.

Puede resultar una tontería, pero para muchos seguro que no es. Así mismo, los tipos Date dentro de esta transformación daban errores que en Kettle no sucedían por lo que tuvimos que en primera instancia cargar la fecha inicial como String y posteriormente pasarla a Date en el script.



The screenshot shows the 'Script Values / Mod' window in Kettle. The 'Step name' is 'Calculate Dimension Attributes'. The 'Java script functions' tree on the left shows 'Input fields' and 'Output fields'. The 'Java script' area contains the following code:

```
Script 1
//Create a Locale according to the specified language code
var locale = new java.util.Locale(
    language_code.getString()
    , country_code.getString()
);

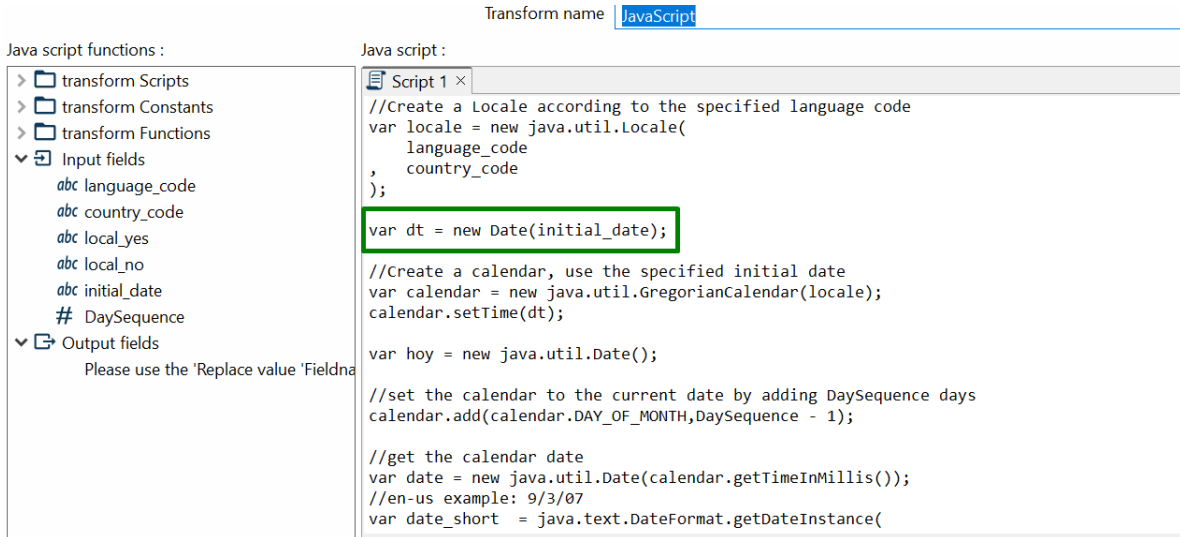
//Create a calendar, use the specified initial date
var calendar = new java.util.GregorianCalendar(locale);
calendar.setTime(initial_date.getDate() );

var hoy = new java.util.Date();

//set the calendar to the current date by adding DaySequence days
calendar.add(calendar.DAY_OF_MONTH,DaySequence.getInteger() - 1);

//get the calendar date
var date = new java.util.Date(calendar.getTimeInMillis());
//en-us example: 9/3/07
var date_short = java.text.DateFormat.getDateInstance(
    java.text.DateFormat.SHORT
    , locale
    ).format(date);
//en-us example: Sep 3, 2007
```

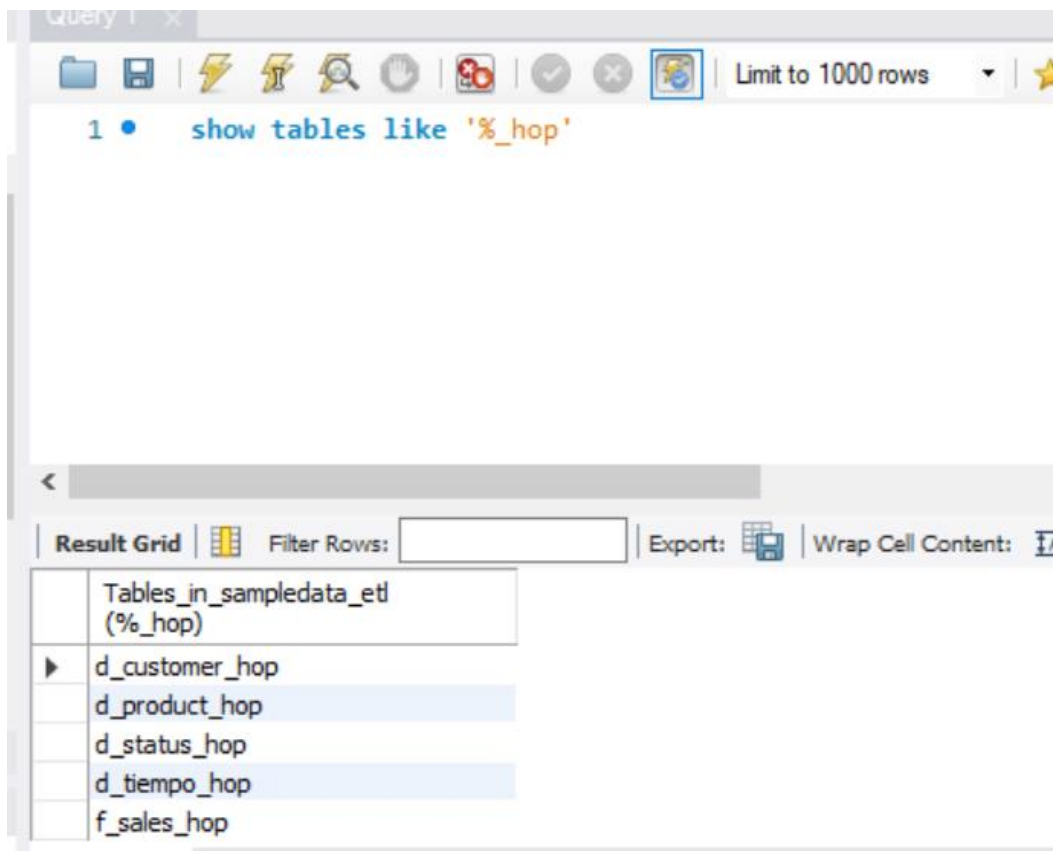
## Ilustración 7: Transformacion javascript de Kettle



## Ilustración 8: Transformación javascript de Hop

Por lo demás, tal y como se ha comentado, la ETL es igual que la de la formación de Kettle. Aquí podemos ver el resultado de esta en base de datos tras ejecutarla por completo. El nombre de las tablas lleva el sufijo \_hop para diferenciarlas de las ya existentes de la formación.





**Ilustración 9: Resultado de la ETL**

## 4. EJECUCIÓN DE PIPELINES CON APACHE SPARK

Una de las peculiaridades de Apache Hop es la ejecución de pipelines con diferentes engines. Uno de ellos es el de Apache Spark. Se realizarán pruebas de ejecución de los pipelines definidos previamente.



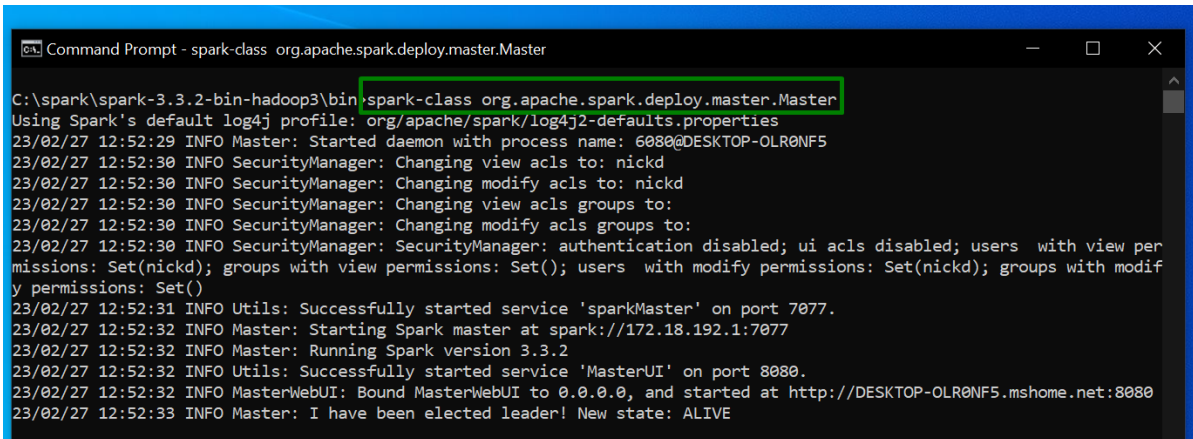
Se pueden iniciar pipelines de Apache Spark cuando Hop se ejecuta directamente sobre el master node. Esto significa que **podemos ejecutar Jobs de spark directamente desde Hop teniendo un cluster disponible en localhost**. Dado que la ejecución de un pipeline en Spark solo es posible desde Spark Master, es posible levantar un servidor Hop en el master. De esta manera podremos ejecutar de manera remota en el nodo master que queramos.

## Arrancar Spark

Antes de comenzar, cabe destacar que estas pruebas se han realizado en una maquina con sistema operativo Windows. Sin embargo, he hecho uso de WSL (subsystem for Linux) puesto que los comandos necesarios para la ejecución del pipeline son de Linux. Así que sería altamente recomendable el uso de Linux y no Windows.

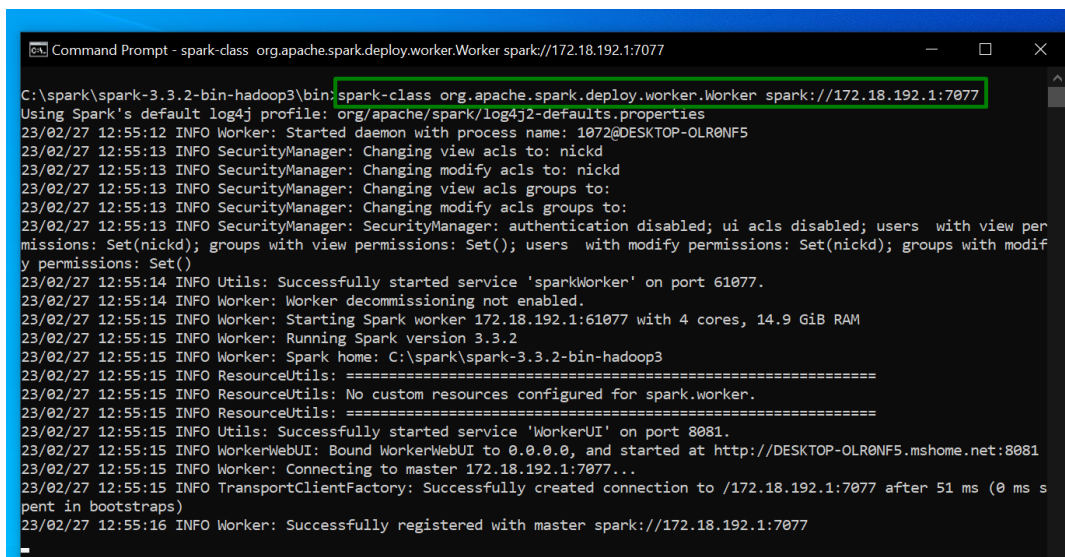
En primer lugar, hemos de tener instalado Spark en nuestra máquina. En mi caso, he instalado Spark para Windows, lo cual requiere de una serie de pasos adicionales. Estos pasos no los comentaré aquí puesto que la instalación de Spark no es relevante en este documento.

Una vez instalado todo, desplegaremos nuestro servidor de Spark local.



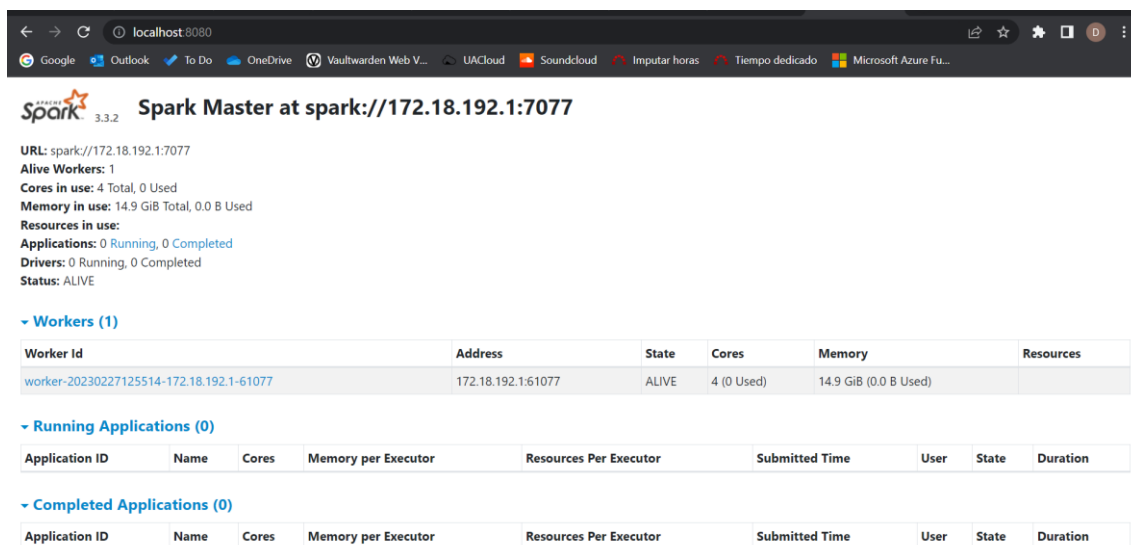
```
Command Prompt - spark-class org.apache.spark.deploy.master.Master
C:\spark\spark-3.3.2-bin-hadoop3\bin>spark-class org.apache.spark.deploy.master.Master
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
23/02/27 12:52:29 INFO Master: Started daemon with process name: 6080@DESKTOP-OLR0NF5
23/02/27 12:52:30 INFO SecurityManager: Changing view acls to: nickd
23/02/27 12:52:30 INFO SecurityManager: Changing modify acls to: nickd
23/02/27 12:52:30 INFO SecurityManager: Changing view acls groups to:
23/02/27 12:52:30 INFO SecurityManager: Changing modify acls groups to:
23/02/27 12:52:30 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view per
missions: Set(nickd); groups with view permissions: Set(); users with modify permissions: Set(nickd); groups with modif
y permissions: Set()
23/02/27 12:52:31 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
23/02/27 12:52:32 INFO Master: Starting Spark master at spark://172.18.192.1:7077
23/02/27 12:52:32 INFO Master: Running Spark version 3.3.2
23/02/27 12:52:32 INFO Utils: Successfully started service 'MasterUI' on port 8080.
23/02/27 12:52:32 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://DESKTOP-OLR0NF5.mshome.net:8080
23/02/27 12:52:33 INFO Master: I have been elected leader! New state: ALIVE
```

## Ilustración 10: Despliegue del master node de Spark



```
Command Prompt - spark-class org.apache.spark.deploy.worker.Worker spark://172.18.192.1:7077
C:\spark\spark-3.3.2-bin-hadoop3\bin>spark-class org.apache.spark.deploy.worker.Worker spark://172.18.192.1:7077
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
23/02/27 12:55:12 INFO Worker: Started daemon with process name: 1072@DESKTOP-OLR0NF5
23/02/27 12:55:13 INFO SecurityManager: Changing view acls to: nickd
23/02/27 12:55:13 INFO SecurityManager: Changing modify acls to: nickd
23/02/27 12:55:13 INFO SecurityManager: Changing view acls groups to:
23/02/27 12:55:13 INFO SecurityManager: Changing modify acls groups to:
23/02/27 12:55:13 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view per
missions: Set(nickd); groups with view permissions: Set(); users with modify permissions: Set(nickd); groups with modif
y permissions: Set()
23/02/27 12:55:14 INFO Utils: Successfully started service 'sparkWorker' on port 61077.
23/02/27 12:55:14 INFO Worker: Worker decommissioning not enabled.
23/02/27 12:55:15 INFO Worker: Starting Spark worker 172.18.192.1:61077 with 4 cores, 14.9 GiB RAM
23/02/27 12:55:15 INFO Worker: Running Spark version 3.3.2
23/02/27 12:55:15 INFO Worker: Spark home: C:\spark\spark-3.3.2-bin-hadoop3
23/02/27 12:55:15 INFO ResourceUtils: =====
23/02/27 12:55:15 INFO ResourceUtils: No custom resources configured for spark.worker.
23/02/27 12:55:15 INFO ResourceUtils: =====
23/02/27 12:55:15 INFO Utils: Successfully started service 'WorkerUI' on port 8081.
23/02/27 12:55:15 INFO WorkerWebUI: Bound WorkerWebUI to 0.0.0.0, and started at http://DESKTOP-OLR0NF5.mshome.net:8081
23/02/27 12:55:15 INFO Worker: Connecting to master 172.18.192.1:7077...
23/02/27 12:55:15 INFO TransportClientFactory: Successfully created connection to /172.18.192.1:7077 after 51 ms (0 ms s
pent in bootstraps)
23/02/27 12:55:16 INFO Worker: Successfully registered with master spark://172.18.192.1:7077
```

## Ilustración 11: Despliegue del worker de Spark



## Ilustración 12: webUI de Spark

# Configuración del pipeline runtime engine de Spark en Hop

Una vez desplegado nuestro servidor de Spark, tenemos que añadir una nueva configuración (runtime configuration) en Hop para que podamos utilizar este engine. Aquí especificaremos el nombre de la configuración, tipo de engine que emplea, URL del master node de Spark y la localización donde se guardarán los ficheros temporales que emplea Spark.

De esto último, la ruta ha de ser de tipo Linux. Es decir que si estás en Windows y especificas una ruta de

Windows (“C:\spark” por ejemplo), pues la run configuration no será correcta.



**Ilustración 13: Configuración del engine en Hop**

## Ejecución del pipeline

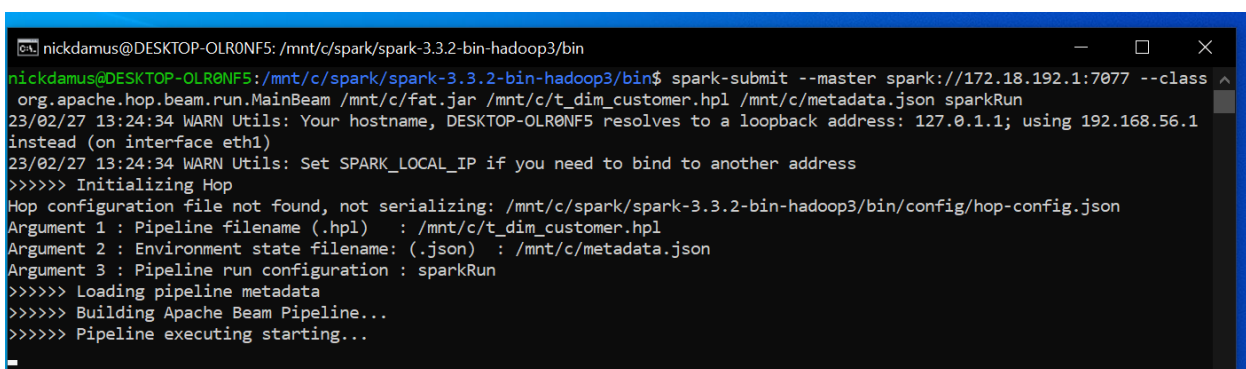
Para la ejecución del pipeline emplearemos spark-submit. **Spark-submit necesita que generemos 2 ficheros, un fat jar y los metadatos del proyecto.** El fat jar se puede generar en el menú de Hop haciendo click en Generate a Hop fat jar o mediante el comando:  
`sh hop-config.sh -fj /path/to/fat.jar`

Por otra parte, la generación de los metadatos se hará también en el menú de Hop.

Por último, nos queda ejecutar el pipeline mediante spark-submit. Para ello nos situaremos en el directorio /bin de nuestra instalación de Spark y ejecutaremos el comando con el siguiente prototipo:

```
spark-submit \  
  --master spark://master-host:7077 \  
  --class  
org.apache.hop.beam.run.MainBeam \  
  /path/hop-fat.jar \  
  /path/pipeline.hpl \  
  /path/metadata-export.json \  
  SparkRunConfig
```

Especificaremos la ruta en la que se encuentra el fichero .jar y .json que hemos generado previamente junto que el pipeline que queremos ejecutar. El último argumento es el nombre de la configuración en Hop creada previamente.



```
nickdamus@DESKTOP-OLR0NF5: /mnt/c/spark/spark-3.3.2-bin-hadoop3/bin  
nickdamus@DESKTOP-OLR0NF5:/mnt/c/spark/spark-3.3.2-bin-hadoop3/bin$ spark-submit --master spark://172.18.192.1:7077 --class  
org.apache.hop.beam.run.MainBeam /mnt/c/fat.jar /mnt/c/t_dim_customer.hpl /mnt/c/metadata.json sparkRun  
23/02/27 13:24:34 WARN Utils: Your hostname, DESKTOP-OLR0NF5 resolves to a loopback address: 127.0.1.1; using 192.168.56.1  
instead (on interface eth1)  
23/02/27 13:24:34 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address  
>>>>> Initializing Hop  
Hop configuration file not found, not serializing: /mnt/c/spark/spark-3.3.2-bin-hadoop3/bin/config/hop-config.json  
Argument 1 : Pipeline filename (.hpl) : /mnt/c/t_dim_customer.hpl  
Argument 2 : Environment state filename: (.json) : /mnt/c/metadata.json  
Argument 3 : Pipeline run configuration : sparkRun  
>>>>> Loading pipeline metadata  
>>>>> Building Apache Beam Pipeline...  
>>>>> Pipeline executing starting...
```

## Ilustración 14: Ejecución del pipeline

# Apache Hop: Ejecución de Pipelines con Apache Spark y Google Dataflow

Stratebi.com

```
23/02/27 13:26:11 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 16144 ms on 172.18.192.1 (executor 0) (1/1)
23/02/27 13:26:11 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
23/02/27 13:26:11 INFO DAGScheduler: ResultStage 1 (foreach at BoundedDataset.java:127) finished in 16.228 s
23/02/27 13:26:11 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
23/02/27 13:26:11 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
23/02/27 13:26:11 INFO DAGScheduler: Job 0 finished: foreach at BoundedDataset.java:127, took 69.453464 s
23/02/27 13:26:11 INFO SparkRunner: Batch pipeline execution complete.
23/02/27 13:26:12 INFO SparkUI: Stopped Spark web UI at http://192.168.56.1:4040
23/02/27 13:26:12 INFO StandaloneSchedulerBackend: Shutting down all executors
23/02/27 13:26:12 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
23/02/27 13:26:12 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
23/02/27 13:26:12 INFO MemoryStore: MemoryStore cleared
23/02/27 13:26:12 INFO BlockManager: BlockManager stopped
23/02/27 13:26:12 INFO BlockManagerMaster: BlockManagerMaster stopped
23/02/27 13:26:12 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
23/02/27 13:26:12 INFO SparkContext: Successfully stopped SparkContext
2023/02/27 13:26:12 - General - Beam pipeline execution has finished.
>>>>> Execution finished..
23/02/27 13:26:12 INFO ShutdownHookManager: Shutdown hook called
23/02/27 13:26:12 INFO ShutdownHookManager: Deleting directory /tmp/spark-bb6f3a4d-268f-4ec7-930d-ff23903adb17
23/02/27 13:26:12 INFO ShutdownHookManager: Deleting directory /tmp/spark-f99f3b25-45cc-4150-8650-81294cdf96b6
nickdamus@DESKTOP-OLR0NF5:/mnt/c/spark/spark-3.3.2-bin-hadoop3/bin$
```

## Ilustración 15: Comprobar resultado en terminal

The screenshot shows the Spark Master web UI for Spark 3.3.2 at spark://172.18.192.1:7077. The interface displays system metrics and a table of completed applications. A green box highlights the 'Completed Applications (1)' section, and a red arrow points to the 'BeamSparkPipelineRunConfiguration' application, which is in a 'FINISHED' state.

**Spark Master at spark://172.18.192.1:7077**

URL: spark://172.18.192.1:7077  
Alive Workers: 1  
Cores in use: 4 Total, 0 Used  
Memory in use: 14.9 GiB Total, 0.0 B Used  
Resources in use:  
Applications: 0 Running, 1 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

**Workers (1)**

Worker Id	Address	State	Cores	Memory	Resources
worker-20230227125514-172.18.192.1-61077	172.18.192.1:61077	ALIVE	4 (0 Used)	14.9 GiB (0.0 B Used)	

**Running Applications (0)**

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

**Completed Applications (1)**

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20230227132459-0000	BeamSparkPipelineRunConfiguration	4	1024.0 MiB		2023/02/27 13:24:59	nickdamus	FINISHED	1.2 min

## Ilustración 16: Comprobar resultado via Spark webUI



## 5. EJECUCIÓN DE PIPELINES CON GOOGLE DATAFLOW

Otra de las opciones es ejecutar pipelines empleando el engine de Google dataflow. En este caso la configuración es mucho más sencilla que con Spark. Para realizar las pruebas me he creado una cuenta con facturación en Google cloud.



### Google Cloud Dataflow

En primer lugar, **hemos de crear un Bucket en el cual subamos los archivos necesarios para la ejecución del pipeline.** Estos archivos son los metadatos del proyecto (Cómo obtenerlos se explica previamente), el fichero .hpl con la pipeline a ejecutar y un dataflow template.

Este dataflow template se trata de un json que contiene la definición utilizada por Google Dataflow para

configurar el pipeline que debe ejecutarse, contiene información como los parámetros necesarios etc. Este template se puede encontrar en la [documentación de Hop](#)

Una vez hecho esto y ya en Google dataflow, crearemos una nueva pipeline con los siguientes parámetros:

The screenshot shows the 'Edit Pipeline' configuration window in Google Dataflow. It contains the following fields and options:

- Pipeline name \***: my-hop-pipeline
- Dataflow template \***: Custom Template (with a dropdown arrow and a help icon)
- Execute a custom template that you've uploaded to Cloud Storage
- Template path \***: gs://[redacted]/hopFlexTemplateMetadata.json (with a 'BROWSE' button)
- Path to your template file stored in Cloud Storage
- This template allows you to start Hop pipelines on dataflow
- Schedule your pipeline**
  - Repeat \***: Daily (with a dropdown arrow)
  - At time \***: 12:00
  - Timezone \***: Central European Standard Time (CET) (with a dropdown arrow)



## Ilustración 17: Configuración del pipeline en Google Dataflow

Como vemos también podemos incluir una Schedule policy para que nuestro pipeline se ejecute según esta.

The screenshot shows a configuration form for a pipeline in Google Dataflow. At the top, there is a text input field for 'Cloud Scheduler service account email'. Below this is a section titled 'Required parameters'. The first parameter is 'Hop Metadata Location \*', with a value of 'gs://[redacted]/hop-metadata-export.json' and a description: 'Google storage location pointing to the Hop metadata file'. The second parameter is 'Hop Pipeline Location \*', with a value of 'gs://[redacted]/add-sequence-unique-id.hp|' and a description: 'Google storage location pointing to the pipeline you wish to start'. Below the parameters is an 'Encryption' section with a radio button selected for 'Google-managed encryption key' and the text 'No configuration required'.

## Ilustración 18: Configuración del pipeline en Google Dataflow

Por último, ejecutaremos el pipeline y esperaremos los resultados. Como es normal, el job se encola y se ejecuta por lo que tarda unos minutos en mostrar el resultado total.

Name	Type	End time	Elapsed time	Start time	Status	SDK version	ID
 <a href="#">my-hop-pipeline-mp-1677145203</a>	Batch	Feb 23, 2023, 10:46:30 AM	6 min 26 sec	Feb 23, 2023, 10:40:04 AM	Succeeded	 2.43.0	2023-02-23_01_40_03-8475610011520684021

## Ilustración 19: Resultado de la ejecución del pipeline

## 6. PLANIFICACIÓN DE LA ETL

La planificación y scheduling de los ETL's es algo que siempre hay que tener en cuenta. Con Apache Hop hay varias maneras de planificar la ejecución de la ETL.

Muy probablemente lo primero que se nos viene a la mente sería emplear la herramienta crontab para la planificación del proceso ETL. Si bien es cierto que esta es una muy buena opción, Apache Hop cuenta con otras maneras de hacer lo mismo.

### **Scheduling con Google Dataflow**

Tal y como hemos visto en la sección anterior, Hop permite la ejecución de pipelines con Google Dataflow. Si nos fijamos en la figura 17, vemos que al configurar el pipeline tenemos la opción de introducir una policy de Schedule, por lo que esta sería una opción válida.

### **Scheduling con Apache Airflow**

Sin embargo, una de las novedades que Hop aporta es la integración con Apache Airflow para la planificación de la ejecución de pipelines. Apache Airflow resumidamente es un job scheduler.

Para ello, debemos en primer lugar instalar el plugin

```
pip install airflow-hop-plugin
```

Igual que con la parte de Spark, no me centraré en los detalles de cómo instalar Apache Airflow y hacerlo funcionar. Se puede instalar mediante pip install o descargando la imagen de Docker.

Antes de arrancar nuestra instancia de Airflow, crearemos un DAG el cual orquestrará dos de nuestras pipelines definidas en Hop.

```
hopDag.py 4 X
C: > Users > nickd > airflow > dags > hopDag.py > ...
1  from airflow import DAG
2  from airflow_operators.python import PythonOperator
3  from datetime import datetime
4  from airflow_hop_operators import HopPipelineOperator
5  from airflow_hop_operators import HopWorkflowOperator
6
7  with DAG('hop_sample_dag', start_date=datetime(2023,3,1),
8         schedule_interval='@daily', catchup=False) as dag:
9      # Define a pipeline
10     first_pipe = HopPipelineOperator(
11         task_id='load_cutomers',
12         pipeline='default/t_dim_customer.hpl',
13         pipe_config='remote hop server',
14         project_name='default',
15         log_level='Basic')
16
17     # Define a pipeline
18     second_pipe = HopPipelineOperator(
19         task_id='load_status',
20         pipeline='default/t_dim_status.hpl',
21         pipe_config='remote hop server',
22         project_name='default',
23         log_level='Basic')
24
25     first_pipe >> second_pipe
```

Ilustración 20: DAG de ejemplo

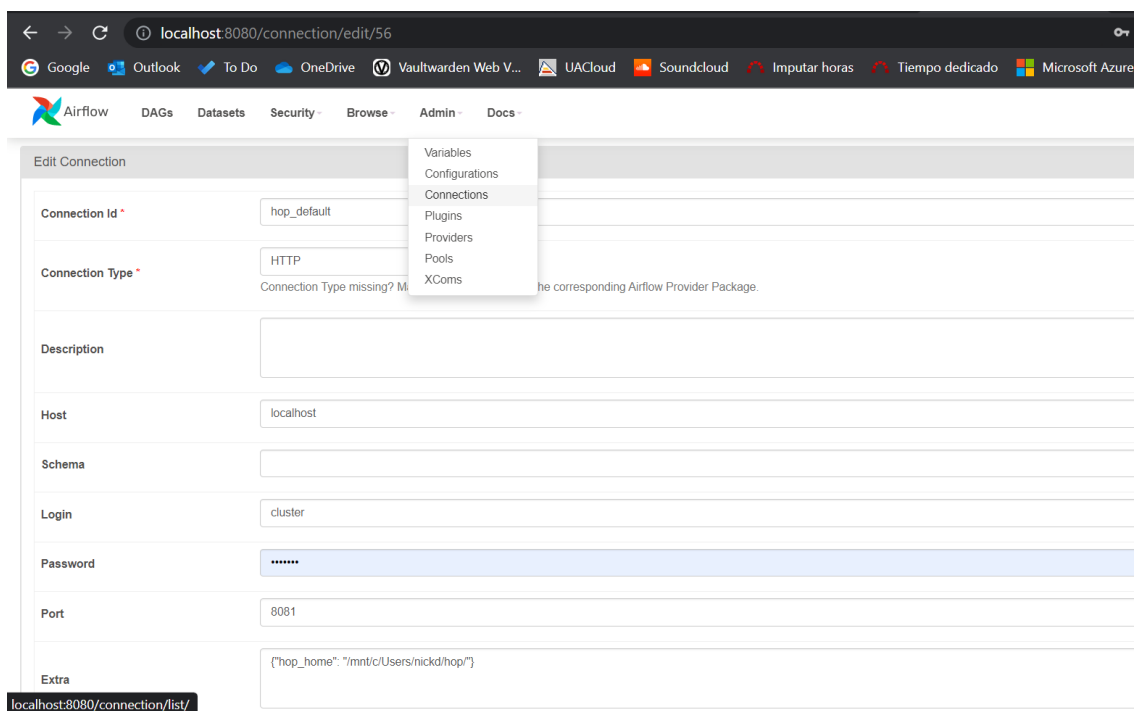
Situaremos el DAG dentro de la carpeta que hayamos definido en nuestra instalación para que sea visible via Airflow web-ui. En caso de querer cambiar esta localización habría que editar el fichero <PATH\_TO\_AIRFLOW>/airflow/airflow.cfg

Otro requisito importante es el de tener un hop-server activo y una conexión al mismo. Para instanciar un hop-server simplemente iremos a la carpeta donde tengamos la instalación de Hop y ejecutaremos:

```
./hop-server 127.0.0.1 <PORT>
```

En mi caso utilizaré el puerto 8081 puesto que Airflow-ui emplea el puerto 8080. De todos modos, para ambos casos puedo especificar el puerto en concreto a emplear.

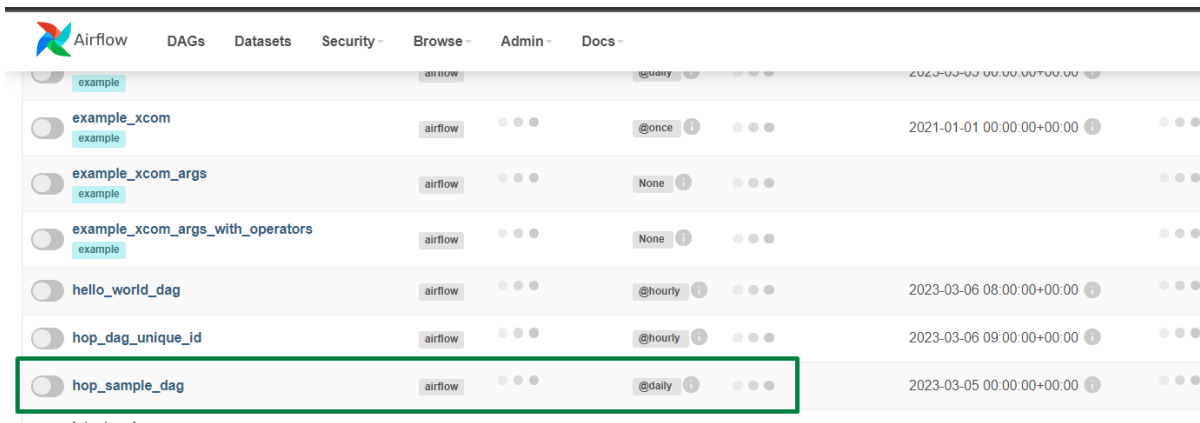
Ya en la interfaz web de Airflow, configuraremos una conexión



**Ilustración 21: Conexión a hop-server**



Por último, veremos que el DAG que hemos creado aparece en la lista de DAGS. Lo activaremos y esperaremos al final de la ejecución.



Toggle	DAG Name	Provider	Frequency	Last Run
<input type="checkbox"/>	example	airflow	@daily	2023-03-05 00:00:00+00:00
<input type="checkbox"/>	example_xcom	airflow	@once	2021-01-01 00:00:00+00:00
<input type="checkbox"/>	example_xcom_args	airflow	None	
<input type="checkbox"/>	example_xcom_args_with_operators	airflow	None	
<input type="checkbox"/>	hello_world_dag	airflow	@hourly	2023-03-06 08:00:00+00:00
<input type="checkbox"/>	hop_dag_unique_id	airflow	@hourly	2023-03-06 09:00:00+00:00
<input type="checkbox"/>	hop_sample_dag	airflow	@daily	2023-03-05 00:00:00+00:00

## Ilustración 22: Lista de DAG's

**Como vemos, Airflow es una herramienta potente para la planificación de pipelines.**

Sin embargo, una de las cosas que echo en falta es un log algo más descriptivo a la hora de ver algún fallo en un dag. Por otra parte, los tiempos de ejecución son un tanto largos incluso para pipelines sencillas. Así mismo, la interfaz web presenta algún que otro fallo por lo que tenía que recargar la página con frecuencia.

## 7. LOGS Y HOP-RUN

Con respecto a la parte de auditoría y logs, veo bastante similitud entre Apache Hop y la herramienta de Pentaho. Cuando hacemos uso de hop-ui, que es el equivalente a ejecutar el archivo Spoon.bat o Spoon.sh, tenemos una pestaña en la que nos aparecen métricas y logs del pipeline o workflow que hemos ejecutado, de la misma manera que con Pentaho. Así mismo, este log se guarda en el fichero <PATH\_TO\_HOP>/audit/hop-gui.txt

Ahora bien, podemos ejecutar pipelines sin necesidad de la interfaz gráfica que nos proporciona hop-gui haciendo uso del script hop-run, el cual nos presenta el log por terminal. Si queremos que este se guarde en un fichero, podemos redirigir su salida como en el ejemplo:

```
./hop-run.sh -j samples -r local -f  
${PROJECT_HOME}/transforms/switch-case-  
basic.hpl &> logPrueba.txt
```

Para más información acerca de hop-run [aquí](#).

## 8. BUENAS PRÁCTICAS

### Naming

- Dar nombres descriptivos a workflows y pipelines
- Cuando renombramos objetos de tipo metadata como conexiones a bases de datos, evitaremos palabras específicas de nuestro entorno como el país, región... Por ejemplo: Test database → CRM
- Emplear un naming standard para tus proyectos

### Variables

En cuanto a las variables del proyecto:

- Poner las variables en un fichero de configuración de las mismas
- Cuando hacemos referencia a ficheros y localización de estos, emplear preferiblemente expresiones como `${PROJECT_HOME}` antes que `${Internal.Entry.Current.Directory}` o `${Internal.Pipeline.Filename.Directory}`

### Gobernanza

- Emplear control de versiones
- Asegurarnos de tener backups
- Integración continua

- Tener diferentes entornos (producción, testing, desarrollo, integración)
- Realizar test sobre pipelines y ejecutarlos regularmente

## 9. CONCLUSIONES

Como vemos **Apache Hop es una herramienta bastante completa que además nos permite la ejecución de pipelines con distintos engines como con el de Spark o Google Dataflow entre otros.** Al tratarse de un fork de Pentaho Data Integration, el salto a Hop es bastante sencillo si ya tienes experiencia previa.

Un gran aspecto positivo es que, al tratarse de una herramienta nueva, parece que poco a poco irán trayendo novedades y mejoras sobre la misma, cosa que con PDI dejó de suceder hace un tiempo.

Sin embargo, y relacionado con lo anterior, la documentación de Hop aún es escasa lo cual podría dificultar ciertas tareas.

Por último y ya más relacionado con el aspecto visual y la interfaz, vemos que aún hay cosas que podrían mejorar en Apache Hop con respecto a PDI como lo comentado en el caso práctico.

