

PRÁCTICA SESIÓN 7 - INGESTA DE DATOS

Resumen

En este documento se trata de explicar mediante ejercicios prácticos el funcionamiento de SQOOP, FLUME y NIFI, con los que el alumno podrá experimentar el funcionamiento básico de la ingesta de datos en el entorno de Hadoop con diferentes herramientas.

Contenido

| | |
|---|----|
| Contenido..... | 1 |
| 1. SQOOP..... | 2 |
| 1.1. Ejemplo 1 – Importar tablas con Sqoop..... | 2 |
| 1.2. Ejemplo 2 – Compresión y formato Avro..... | 4 |
| 2. FLUME..... | 5 |
| 2.1 Ejemplos operaciones básicas con Flume..... | 5 |
| 3. NIFI..... | 8 |
| 3.1. Instalación..... | 8 |
| 3.2. Mover datos..... | 10 |

1. SQOOP

1.1. Ejemplo 1 – Importar tablas con Sqoop

Con SQOOP podemos importar/exportar desde una BBDD a nuestro HDFS y viceversa. En esta práctica vamos a trabajar con mysql que ya tenemos instalada en nuestra máquina virtual. Cuando se lanza SQOOP convierte las marcas de tiempo de nuestra base de datos origen y las convierte a la hora del sistema servidor por lo que tenemos que especificar en nuestra base de datos la zona horaria.

La forma de realizarlo es muy simple, simplemente editamos el fichero mysqld.cnf que se encuentra en /etc/mysql/mysql.conf.d/

```
/etc/mysql/mysql.conf.d$ sudo nano mysqld.cnf
```

Y añadimos la siguiente propiedad:

```
default_time_zone = Europe/Madrid
```

Ahora solo nos queda reiniciar el servicio de mysql:

```
sudo service mysql restart
```

OJO: Esto en la sesión práctica nos dio error, si al cambiar la propiedad del fichero da error al reiniciar mysql, hay que pasar el serverTimezone por parámetro en la cadena JDBC

```
jdbc:mysql://localhost/ejemplo?serverTimezone=Europe/Madrid
```

Una vez realizado el paso anterior, vamos a crear un par de tablas en mysql, para ello abrimos la Shell de mysql:

```
sudo mysql
```

Y creamos una base de datos que se va a llamar ejemplo

```
mysql> create database ejemplo;  
mysql> use ejemplo;
```

Creamos un par de tablas:

```
mysql> CREATE TABLE profesores(  
    id MEDIUMINT NOT NULL AUTO_INCREMENT,  
    nombre CHAR(30) NOT NULL,  
    edad INTEGER(30),  
    materia CHAR(30),  
    PRIMARY KEY (id) );
```

```
mysql> CREATE TABLE profesores2(  
    id MEDIUMINT NOT NULL AUTO_INCREMENT,  
    nombre CHAR(30) NOT NULL,  
    edad INTEGER(30),  
    materia CHAR(30),  
    PRIMARY KEY (id) );
```

Hacemos una inserción en la primera tabla:

```
INSERT INTO profesores (nombre, edad, materia) VALUES  
("Carlos", 24, "Matematicas"),  
("Pedro", 32, "Lenguaje"),  
("Juan", 35, "Tecnologia"),  
("Jose", 48, "Matematicas"),  
("Paula", 24, "Informatica"),  
("Susana", 32, "Informatica"),  
("Lorena", 54, "Informatica");
```

Arrancamos hdfs y yarn:

```
start-dfs.sh  
start-yarn.sh
```

Con el comando SQOOP listamos todas las tablas de ejemplo (NO COPIAR Y PEGAR)

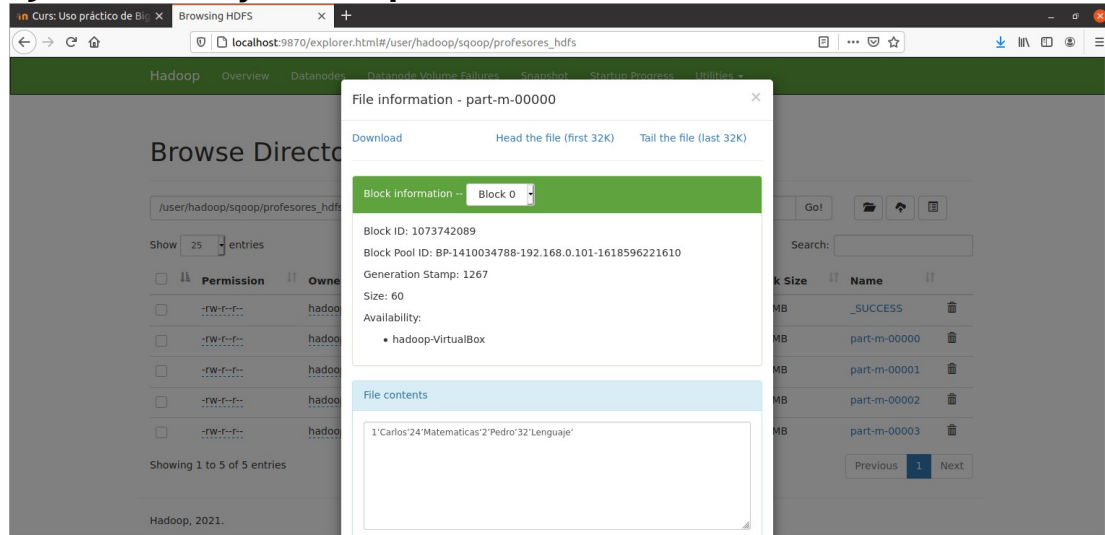
```
sqoop list-tables --connect  
jdbc:mysql://localhost/ejemplo?serverTimezone=Europe/Madr  
id --username=hadoop --password=hadoop
```

Importamos los datos de nuestra BBDD a HDFS mediante el comando IMPORT:

```
sqoop import --connect jdbc:mysql://localhost/ejemplo?  
serverTimezone=Europe/Madrid  
--username=hadoop --password=hadoop  
--table=profesores --driver=com.mysql.jdbc.Driver  
--target-dir=/user/hadoop/sqoop/profesores_hdfs  
--fields-terminated-by=',' --lines-terminated-by '\n'
```

A través del navegador web de ficheros de hdfs localhost:9870 podremos comprobar en el directorio /user/hadoop/sqoop que ha creado el directorio que hemos especificado junto con los archivos:

EJERCICIO [Adjuntar captura]



Ahora vamos a hacer el paso contrario, desde hdfs vamos a exportar los ficheros a nuestra otra tabla que habíamos creado en mysql para ello lanzamos la siguiente orden:

```
sqoop export --connect jdbc:mysql://localhost/ejemplo?  
serverTimezone=Europe/Madrid  
--username=hadoop  
--password=hadoop  
--table=profesores2  
--export-dir=/user/hadoop/sqoop/profesores_hdfs  
--columns="id,nombre,edad,materia"
```

Con la opción --columns="columna1,columna2" restringimos el número de columnas.

Con la opción `--where="campo>valor"` restringimos las condiciones de importación

```
sqoop import --connect jdbc:mysql://localhost/ejemplo?  
serverTimezone=Europe/Madrid  
--username=hadoop --password=hadoop  
--table=profesores  
--driver=com.mysql.jdbc.Driver  
--target-dir=/user/hadoop/sqoop/where_hdfs  
--columns="nombre,edad"  
--where="edad>30"
```

1.2. Ejemplo 2 - Compresión y formato Avro

Avro es un formato de almacenamiento basado en filas para Hadoop que es usado ampliamente como plataforma de serialización.

Snappy es una biblioteca de compresión y descompresión de datos rápida que es utilizado con frecuencia en proyectos Big Data.

Para que funcione la serialización con Avro hay que copiar el fichero .jar que viene en el directorio de Sqoop para Avro con el siguiente comando:

```
cp $SQOOP_HOME/lib/avro-1.8.1.jar  
$HADOOP_HOME/share/hadoop/common/lib/  
rm $HADOOP_HOME/share/hadoop/common/lib/avro-1.7.7.jar
```

Ahora ya podemos lanzar el siguiente comando:

```
sqoop import --connect jdbc:mysql://localhost/ejemplo?  
serverTimezone=Europe/Madrid  
--username=hadoop --password=hadoop  
--table=profesores  
--driver=com.mysql.jdbc.Driver  
--target-dir=/comprimido_hdfs  
--compress  
--compression-codec  
org.apache.hadoop.io.compress.SnappyCodec  
--as-avrodatafile
```

Probamos a hacer la ingesta con compresión BZip2 y formato secuencial:

```
sqoop import --connect jdbc:mysql://localhost/ejemplo?  
serverTimezone=Europe/Madrid  
--username=hadoop --password=hadoop  
--table=profesores  
--driver=com.mysql.jdbc.Driver  
--target-dir=/BZip_hdfs  
--compress  
--compression-codec  
org.apache.hadoop.io.compress.BZip2Codec  
--as-sequencefile
```

2. FLUME

2.1 Ejemplos operaciones básicas con Flume

Vamos a ver con un par de ejemplos cómo podemos a través de flume, generar, recopilar y agregar datos desde un origen a nuestro sistema HDFS.

- Nos movemos hasta el directorio \$FLUME_HOME y creamos un fichero llamado seq_gen.conf en el directorio /conf con el siguiente contenido:

```
#Nombramos a los componentes del agente  
SeqGenAgent.sources = SeqSource  
SeqGenAgent.channels = MemChannel  
SeqGenAgent.sinks = HDFS  
  
# Describimos el tipo de origen  
SeqGenAgent.sources.SeqSource.type = seq  
  
# Describimos el destino  
SeqGenAgent.sinks.HDFS.type = hdfs  
SeqGenAgent.sinks.HDFS.hdfs.path = hdfs://hadoop-  
VirtualBox:9000/user/hadoop/seqgen_data/  
SeqGenAgent.sinks.HDFS.hdfs.filePrefix = log  
SeqGenAgent.sinks.HDFS.hdfs.rollInterval = 0
```

```
SeqGenAgent.sinks.HDFS.hdfs.rollCount = 10000
SeqGenAgent.sinks.HDFS.hdfs.fileType = DataStream

# Describimos la configuración del canal
SeqGenAgent.channels.MemChannel.type = memory
SeqGenAgent.channels.MemChannel.capacity = 1000
SeqGenAgent.channels.MemChannel.transactionCapacity = 100

# Unimos el origen y el destino a través del canal
SeqGenAgent.sources.SeqSource.channels = MemChannel
SeqGenAgent.sinks.HDFS.channel = MemChannel
```

Ejecutamos el siguiente comando:

```
./bin/flume-ng agent --conf ./conf/ --conf-file
./conf/seq_gen.conf --name SeqGenAgent
```

Ahora vamos a crear otro ejemplo de generación de información. En el mismo directorio \$FLUME_HOME/conf, creamos un nuevo fichero con el nombre *netcat.conf* y copiamos el siguiente código:

```
#Nombramos a los componentes del agente
NetcatAgent.sources = Netcat
NetcatAgent.channels = MemChannel
NetcatAgent.sinks = HDFS

# Describimos el tipo de origen
NetcatAgent.sources.Netcat.type = netcat
NetcatAgent.sources.Netcat.bind = localhost
NetcatAgent.sources.Netcat.port = 44444
NetcatAgent.sources.Netcat.channels = MemChannel

# Describimos el destino
NetcatAgent.sinks.HDFS.type=hdfs
NetcatAgent.sinks.HDFS.hdfs.path=hdfs://hadoop-VirtualBox:9000/
```



```
user/hadoop/net_data/  
NetcatAgent.sinks.HDFS.hdfs.writeFormat=Text  
NetcatAgent.sinks.HDFS.hdfs.fileType=DataStream  
NetcatAgent.sinks.HDFS.channel=MemChannel  
  
# Unimos el origen y el destino a través del canal  
NetcatAgent.channels.MemChannel.type = memory  
NetcatAgent.channels.MemChannel.capacity = 1000  
NetcatAgent.channels.MemChannel.transactionCapacity = 100
```

Lanzamos al agente:

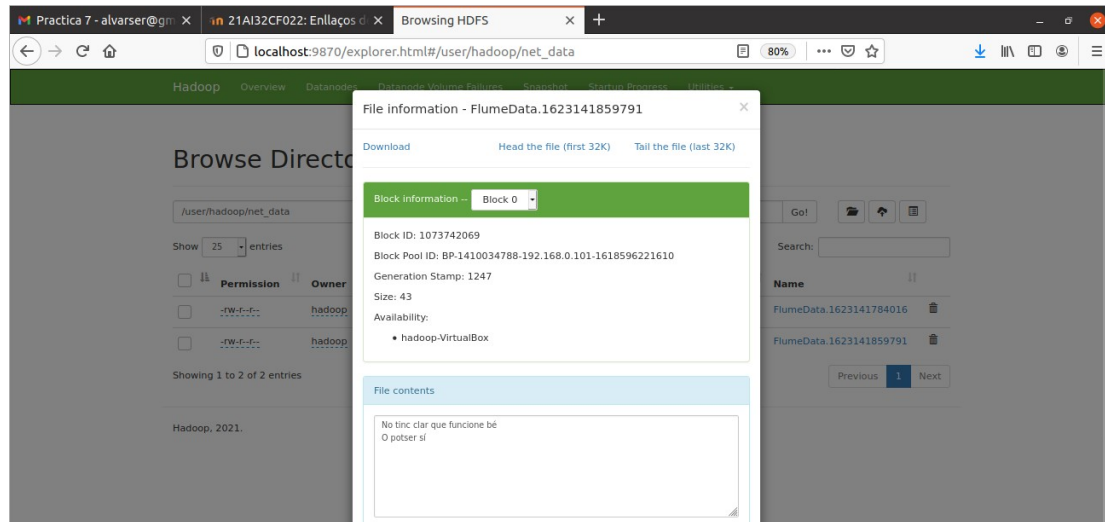
```
./bin/flume-ng agent --conf ./conf/ --conf-file  
./conf/netcat.conf --name NetcatAgent -  
Dflume.root.logger=INFO,console
```

En una nueva pestaña introducimos el siguiente comando y escribimos

```
hadoop@hadoop-VirtualBox:/opt/hadoop/flume/conf$ curl telnet://localhost:44444  
Hola mundo  
OK
```

Nos vamos al navegador web de HDFS y comprobamos que se ha creado el fichero:

[Ejercicio] Adjuntar captura de pantalla

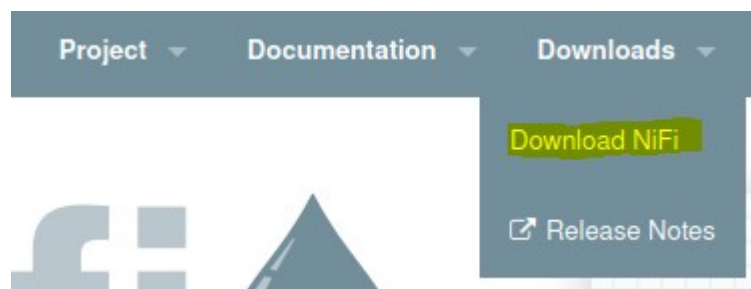


3. NIFI

3.1. Instalación

Para instalar Nifi accedemos al siguiente enlace: <https://nifi.apache.org>

Una vez dentro seleccionamos la opción de descarga:



Y seleccionamos el binario de la versión 1.13.2

Releases

. 1.13.2

- Released March 19, 2021
- Sources:
 - [nifi-1.13.2-source-release.zip](#) (asc, sha256, sha512)
- Binaries
 - [nifi-1.13.2-bin.tar.gz](#) (asc, sha256, sha512)
 - [nifi-1.13.2-bin.zip](#) (asc, sha256, sha512)
 - [nifi-toolkit-1.13.2-bin.tar.gz](#) (asc, sha256, sha512)
 - [nifi-toolkit-1.13.2-bin.zip](#) (asc, sha256, sha512)
- Release Notes
- Migration Guidance

A continuación seleccionamos el mirror que nos sugiere para realizar la descarga:



COMMUNITY-LED

Projects ▾

People ▾

We suggest the following mirror site for your download:

<https://ftp.cixug.es/apache/nifi/1.13.2/nifi-1.13.2-bin.tar.gz>

Una vez se ha realizado la descarga descomprimos el archivo:

```
tar xvf nifi-1.13.2-bin.tar.gz
```

Para ejecutar nifi, tendremos que ejecutar el script nifi.sh que se encuentra en el directorio bin:

```
./nifi.sh start
```

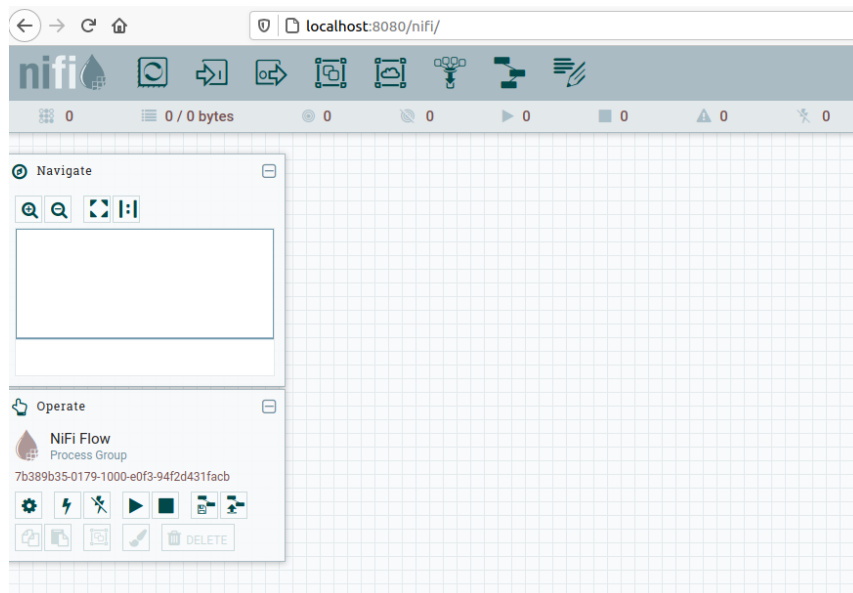
Si vemos el siguiente mensaje es que nifi ha arrancado correctamente:

```
hadoop@hadoop-VirtualBox:~/Descargas/nifi-1.13.2/bin$ ./nifi.sh start

Java home: /usr/lib/jvm/java-8-openjdk-amd64
NiFi home: /home/hadoop/Descargas/nifi-1.13.2

Bootstrap Config File: /home/hadoop/Descargas/nifi-1.13.2/conf/bootstrap.conf
```

Para acceder al entorno de trabajo introducimos en el navegador:
<http://localhost:8080/nifi>



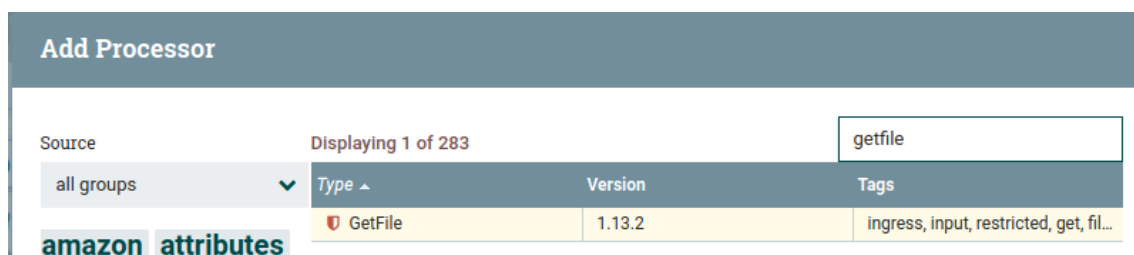
3.2. Mover datos

Vamos a hacer un pequeño ejercicio con NiFi para familiarizarnos con el entorno desarrollando un flujo sencillo que mueva un fichero de un directorio a otro.



- I. Seleccionamos un procesador y lo colocamos en nuestra área de trabajo



- II. Indicamos el tipo de procesador getfile



- III. Damos doble **click** sobre el elemento gráfico que representa nuestro procesador.

| | | |
|---|---|---|
|  |  | GetFile GetFile 1.13.2 org.apache.nifi - nifi-standard-nar |
| In | 0 (0 bytes) | 5 min |
| Read/Write | 0 bytes / 0 bytes | 5 min |
| Out | 0 (0 bytes) | 5 min |
| Tasks/Time | 0 / 00:00:00.000 | 5 min |

- IV. En **properties** indicamos el directorio de entrada de donde tendrá que recoger el fichero, en este caso:

Configure Processor

 Invalid


SETTINGS SCHEDULING **PROPERTIES** COMMENTS


Required field

| Property | Value |
|-----------------|---|
| Input Directory |  /home/hadoop/Documentos/in/ |

- V. Añadimos un nuevo procesador de tipo PutFile

Add Processor

Source all groups  Displaying 1 of 283

| Type | Version | Tags |
|---|---------|--|
|  PutFile | 1.13.2 | restricted, files, archive, copy, p... |

amazon attributes
avro aws azure

- VI. En **properties** indicamos el directorio de salida

Configure Processor

Invalid

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property

Value

Directory



/home/hadoop/Documentos/out/

VII. Para este ejemplo sencillo “autoterminaremos” las 2 relaciones:

Configure Processor

Invalid

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Name

PutFile

☒ Enabled

Automatically Terminate Relationships

☒ failure

Files that could not be written to the output directory for some reason are transferred to this relationship

Id

7b502505-0179-1000-ab17-9da5f82d40e6

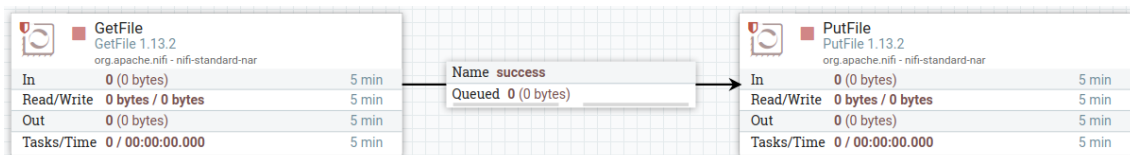
☒ success

Files that have been successfully written to the output directory are transferred to this relationship

Type

PutFile 1.13.2

VIII. Ya solo nos queda unir los dos procesadores



IX. Antes de arrancarlo, creamos un pequeño fichero en el directorio que hemos puesto como entrada

```
echo "Hola" >> Documentos/in/ejemplo1.txt
```

X. Con el segundo botón sobre el procesados le damos a **START**

GetFile
GetFile 1.13.2
org.apache.nifi - nifi-standard

| | |
|------------|--------------------|
| In | 0 (0 bytes) |
| Read/Write | 0 bytes / 0 bytes |
| Out | 0 (0 bytes) |
| Tasks/Time | 298 / 00:00:00.321 |

- Configure
- Start**
- Disable
- View data provenance
- View status history
- View usage
- View connections
- Center in view
- Change color

XI. Y vemos que tenemos fichero en la cola

GetFile
GetFile 1.13.2
org.apache.nifi - nifi-standard-nar

| | | |
|------------|--------------------|-------|
| In | 0 (0 bytes) | 5 min |
| Read/Write | 0 bytes / 0 bytes | 5 min |
| Out | 0 (0 bytes) | 5 min |
| Tasks/Time | 300 / 00:00:00.328 | 5 min |

Name success
Queued 1 (5 bytes)

XII. Damos a START al proceso PUTFILE y vemos como el fichero pasa de la cola a este proceso marcando la entrada:

PutFile
PutFile 1.13.2
org.apache.nifi - nifi-standard-nar

| | | |
|------------|-------------------|-------|
| In | 1 (5 bytes) | 5 min |
| Read/Write | 5 bytes / 5 bytes | 5 min |
| Out | 0 (0 bytes) | 5 min |
| Tasks/Time | 1 / 00:00:00.059 | 5 min |

Name success
Queued 0 (0 bytes)

XIII. Comprobamos los directorios /in y /out para comprobar que ha movido los ficheros.