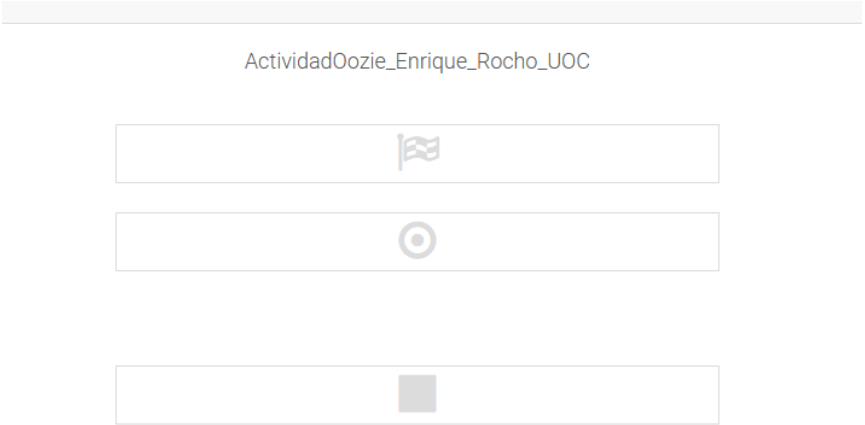
Habilitaremos un entorno de edición que nos permite plantear el grafo de acciones de nuestro flujo de trabajo (workflow). Deberemos crear nuestro workflow asignando el nombre ActividadOozie_<vuestro_nombre>_UOC donde planteamos una serie de tareas sencillas.

Primero realizamos el cambio de nombre del workflow:



Para revisar el funcionamiento del código crearemos una carpeta nueva para los documentos que necesitaremos:

```
hadoop fs -mkdir /user/erocho/Oozie
```

```
erocho@Cloudera01:~$ hadoop fs -mkdir /user/erocho/Oozie
erocho@Cloudera01:~$ hdfs dfs -ls /user/erocho
Found 15 items
drwx------   - erocho erocho          0 2023-04-26 12:00 /user/erocho/.Trash
drwxr-xr-x   - erocho erocho          0 2023-06-09 17:43 /user/erocho/.sparkStaging
drwx------   - erocho erocho          0 2023-04-24 16:48 /user/erocho/.staging
drwxr-xr-x   - erocho erocho          0 2023-05-27 10:55 /user/erocho/DATA_LAKE
drwxrwx---   - erocho erocho          0 2023-04-25 10:39 /user/erocho/HDFS
drwxr-xr-x   - erocho erocho          0 2023-06-20 17:28 /user/erocho/Oozie
-rw-r--r--   3 erocho erocho       5295 2023-05-02 20:15 /user/erocho/captura_flume
drwxr-xr-x   - erocho erocho          0 2023-05-15 11:36 /user/erocho/checkpoint
drwxr-xr-x   - erocho erocho          0 2023-04-24 16:29 /user/erocho/d_pais
drwxr-xr-x   - erocho erocho          0 2023-04-24 16:35 /user/erocho/d_tipo_habitacion
drwxr-xr-x   - erocho erocho          0 2023-04-25 11:17 /user/erocho/data
drwxr-xr-x   - erocho erocho          0 2023-04-24 16:48 /user/erocho/h_reserva
drwxrwx---   - erocho erocho          0 2023-04-28 11:30 /user/erocho/tweets_recuperados
drwxrwx---   - erocho erocho          0 2023-04-25 10:44 /user/erocho/user
drwxr-xr-x   - erocho erocho          0 2023-05-04 12:45 /user/erocho/yarn
```

3.1 Creación de un flujo de trabajo mediante acción shell. (2 puntos)

a)Mostrar y explicar el código bash y su correcto funcionamiento que se incorporará a posteriori en la tarea Oozie (0,5 puntos). El correcto funcionamiento debe mostrarse en la línea de comandos.

Primero revisaremos si el código que vamos a usar funciona, con una carpeta más accesible para ver los resultado en /user/erocho/Oozie.

Revisamos el siguiente código:

```
top_output=$(top -b -n 1 | head -n $(($(tput lines) - 7)) | tail -n +8 | head
-n 10)
echo "$top_output" > top_output.txt
hdfs dfs -put -f top_output.txt /user/erocho/Oozie/top_procesos.txt
```

Si lo vemos en detalle, la siguiente línea muestra la información que necesitamos y la asigna a top_output

```
top -b -n 1 | head -n $(($(tput lines) - 7)) | tail -n +8 | head -n 10
```

La siguiente línea pasa el resultado a txt:

```
echo "$top_output" > top_output.txt
```

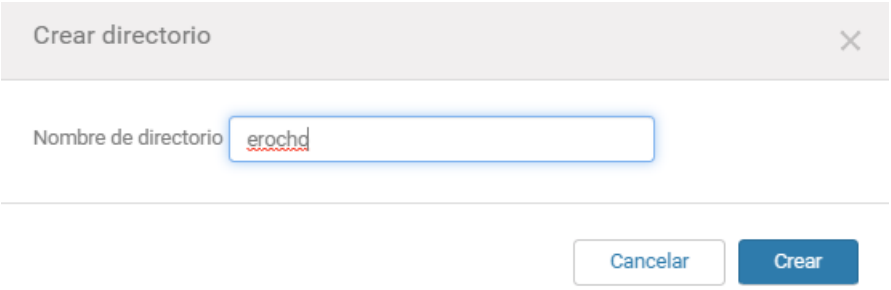
Y finalmente la siguiente guarda el txt en hdfs:

```
hdfs dfs -put top_output.txt /user/erocho/Oozie/top_procesos.txt
```

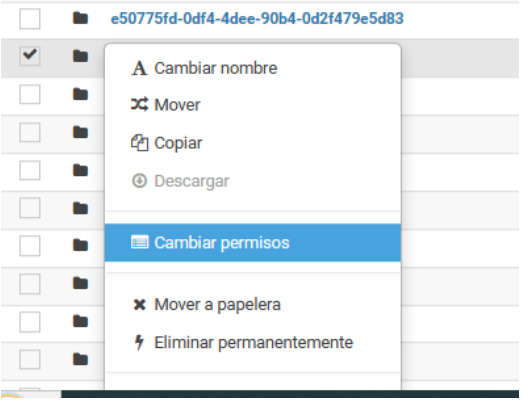
Si ejecutamos la línea en bash obtenemos:

```
erocho@Cloudera01:~$ top -b -n 1 | head -n $(($(tput lines) - 7)) | tail -n +8 | head -n 10
29119 erocho    20   0   43720   4212   3180 R  11,8  0,0   0:00.03 top
 5610 yarn      20   0 3452700 1,023g  30936 S   5,9  1,4 880:32.82 java
10078 impala    20   0 30,033g 6,336g  49308 S   5,9  8,7   2001:36 impalad
    1 root      20   0  185300   5812   3820 S   0,0  0,0   0:48.96 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:01.51 kthreadd
    3 root      20   0       0      0      0 S   0,0  0,0   0:14.42 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0,0  0,0 107:06.67 rcu_sched
    8 root      20   0       0      0      0 S   0,0  0,0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:03.89 migration/0
```

Ahora ya vamos a preparar lo que necesitamos en Hue, donde creamos una carpeta en tmp con nuestro usuario:



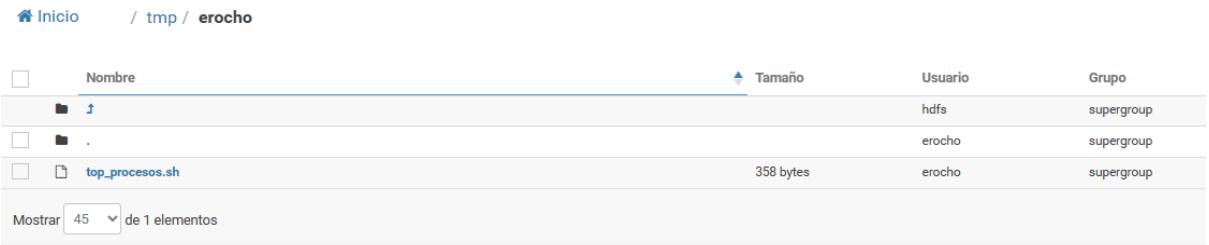
Al nuevo directorio le cambiamos los permisos:



Respecto al código de bash, ahora ya hemos hecho cambio en la ruta para que las acciones se realicen en el directorio de hdfs tmp/erocho en HUE:

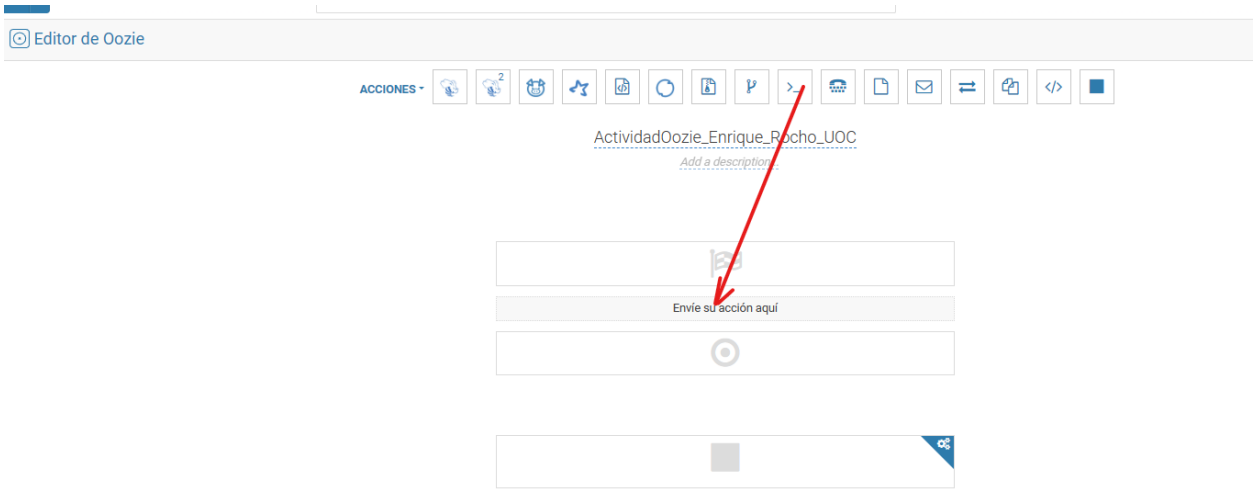
```
top_procesos=$(top -b -n 1 | head -n $(($(tput lines) - 7)) | tail -n +8 |
head -n 10)
echo "$top_procesos" > /tmp/top_procesos.txt
hdfs dfs -put -f /tmp/top_procesos.txt /tmp/erocho/top_procesos.txt
```

Entramos en la carpeta y cargamos el fichero sh que hemos creado. Le cambiamos los permisos también:

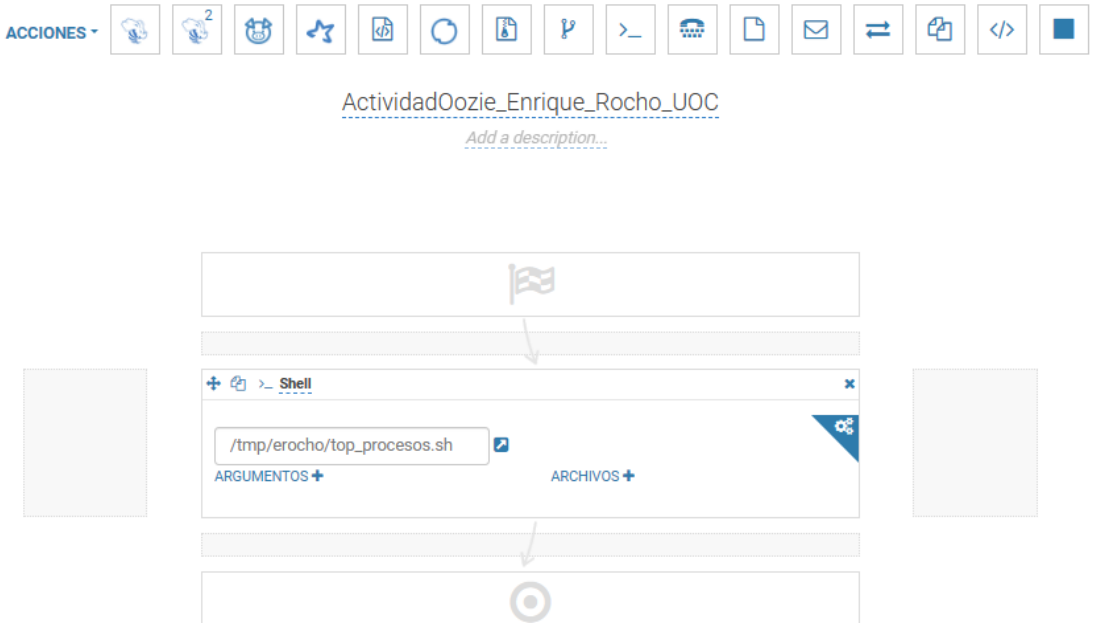


b) Incorporar el código bash del apartado a) en una tarea Oozie y mostrar mediante una captura de pantalla el fichero XML con la descripción del proceso que se acaba de crear. (0,5 puntos)

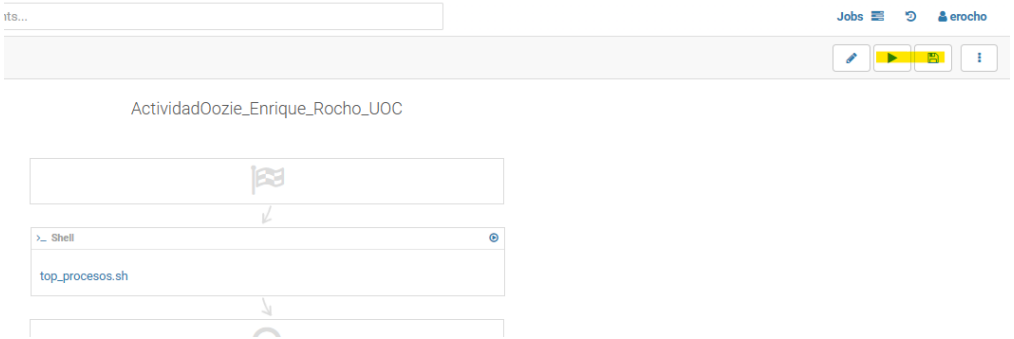
Arrastramos “Shell” a la zona de “Envíe su Acción Aquí”:



En la siguiente ventana de Shell Command buscamos el fichero top_procesos.sh que está dentro de la carpeta tmp/erocho . Nos aparecerá ya la acción que hemos creado:



Guardamos y seleccionamos “enviar” para que empiece el proceso:



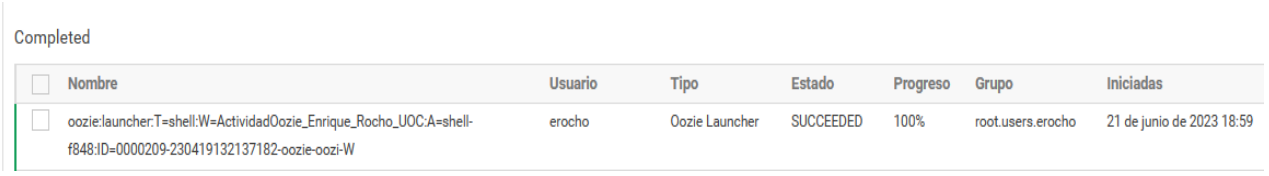
Se nos abrirá la ventana de Workflows con la información de la ejecución. Aparecerá en verde si se ha ejecutado correctamente y estado “Succeeded”:

Revisamos también el XML que se ha generado:

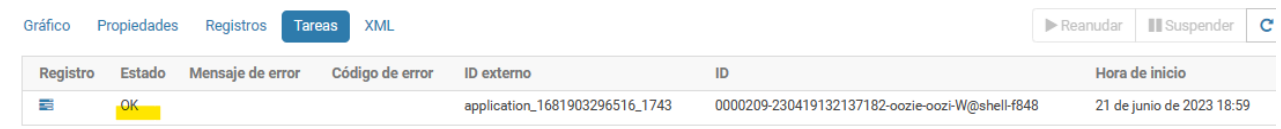
c) Mostrar el correcto funcionamiento del mismo. Esta información está disponible en diversos apartados. Pero es habitual consultarlo con la opción Job Browser > Task (0,5 puntos)

Podemos ver de varias maneras si el proceso ha funcionado correctamente. Una de ellas es revisando si el progreso aparece en verde y si el Estado aparece como “Succeeded”:

También podemos ver en Job Browser un resumen general de los procesos con datos similares a los que hemos visto antes:

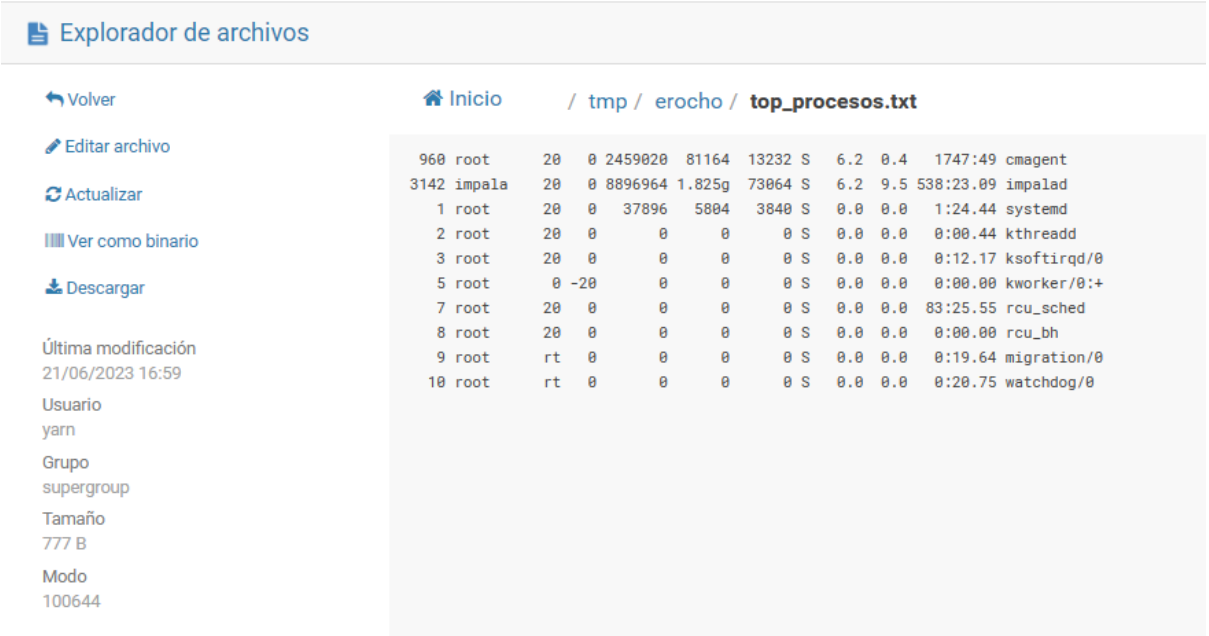


Además en Workflow-Tareas, se nos indicará Estado: Ok si ha funcionado:



d) Mostrar una captura del contenido almacenado en HDFS /tmp/<usuario> con los 10 procesos con más consumo de memoria. (0,5 puntos)

Si vamos al explorador de archivos y buscamos el fichero que se ha generado con el nombre top_procesos.txt podremos ver el resultado:



3.2 Incorporación de flujo de trabajo Spark (3 puntos)

a) A partir de los datos almacenados en HDFS vamos a realizar un un proceso distribuido con Apache Spark que calcule la memoria consumida por los procesos lanzados de cada usuario. En este apartado NO añadiremos todavía nuestro programa Spark al flujo de trabajo de Oozie, primeramente, vamos a implementarlo en un archivo Python. A nivel indicativo, el programa debe obtener un pair RDD con el usuario y total de memoria consumida por los procesos que ha lanzado en el sistema. Se pide mostrar el código Python que realiza la lectura del fichero HDFS mediante Spark y agrupa la memoria consumida por usuario. Adjuntar la salida de este. (1 punto)

El código que vamos a usar es:

```
import findspark
findspark.init()
import pyspark
import random
sc = pyspark.SparkContext(master="local[1]", appName="erocho")

import psutil

# Listamos los procesos
processes = psutil.process_iter(attrs=['name', 'username', 'memory_percent'])

# Creamos el RDD
rdd = sc.parallelize(processes)

# Map de los procesos a un par RDD de user y memory usage
pair_rdd = rdd.map(lambda x: (x.info['username'],
float(x.info['memory_percent'])))

# Memoria agregada por user
aggregated_rdd = pair_rdd.reduceByKey(lambda a, b: a + b)

# Ordenar por memoria descending
sorted_rdd = aggregated_rdd.sortBy(lambda x: x[1], ascending=False)

# Print
for user, memory_usage in sorted_rdd.collect():
    print("u: {}, {}".format(user, memory_usage))

# Stop SparkContext
sc.stop()
```

La información que obtenemos es:

```
u: cloudera-scm, 12.983733147700661
u: impala, 9.569225247569909
u: hdfs, 4.235798658804982
u: yarn, 2.6055327130278934
u: hive, 1.9994142262519998
u: oozie, 1.7540489401786792
u: hbase, 1.7203118014173677
u: spark, 1.574208140373261
u: erocho, 1.4948782956503506
u: hue, 1.4367504705766305
u: jorengap, 1.3167468302693122
u: rperezdelg, 1.289068158168112
u: mgomezrico, 1.2088810325718116
u: cesc, 1.1477370424740332
u: mapred, 0.8943373618738865
u: zookeeper, 0.8467478187559777
u: solr, 0.6475293348185203
u: dperezcala, 0.6416904156991546
u: rperezmartinez01, 0.6318160036879805
u: root, 0.6098612496133741
u: cristianhdezhdez, 0.3228760226014784
u: dadler, 0.32264602041593493
u: mysql, 0.3080513362787204
u: ccarbonellg, 0.13336990368083476
u: postgres, 0.05970124911573308
u: sgraul, 0.02481409942670387
u: postfix, 0.02053814970455435
u: guillemgimenez, 0.015650603261755085
u: ncorbera0, 0.008578036056292617
u: lbernardezc, 0.008489171575514449
u: drodriguezgarcia0123, 0.008457807641122153
u: rmoralmartin, 0.008457807641122153
u: abuendiap, 0.008452580318723438
u: messagebus, 0.00500777485796973
u: syslog, 0.004777772672426235
u: nagios, 0.003078892892843602
u: systemd-timesync, 0.0030684382480461706
u: uuidd, 0.0018034262275569489
```

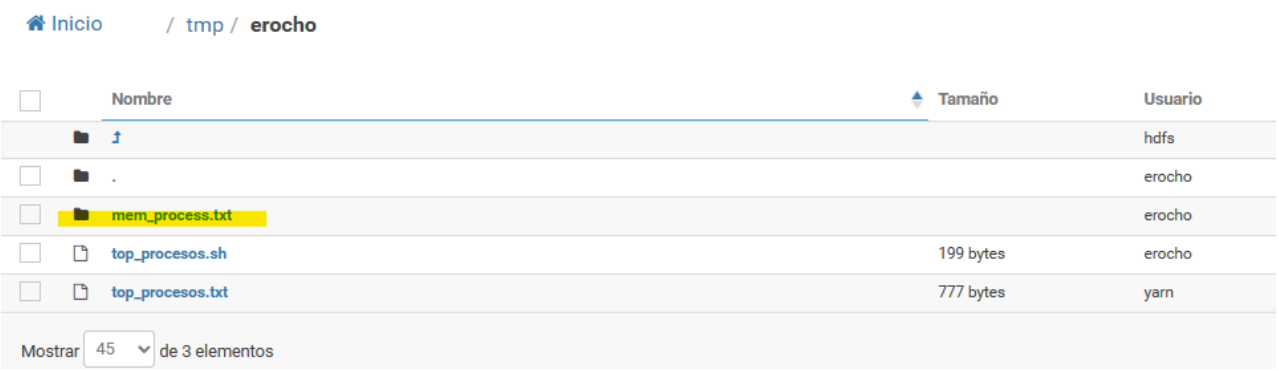
b) Los resultados deberán almacenarse en un directorio del HDFS para posteriormente evaluar su salida y almacenarla en estructura tabular. Una vez más el HDFS juega un papel

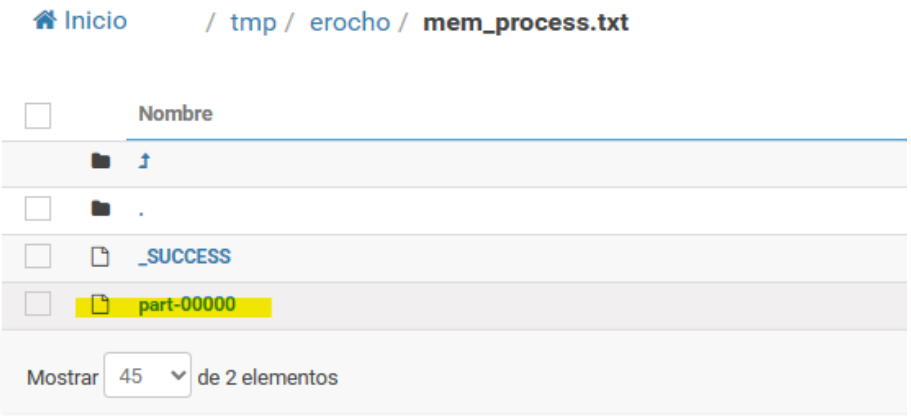
importante en poder interactuar entre las distintas tareas. Mostrar el contenido del fichero HDFS generado, tenéis un ejemplo en la Figura 9 (1 punto)

Hemos hecho un cambio en el código anterior, añadiendo la siguiente línea en lugar del print:

```
sorted_rdd.saveAsTextFile('/tmp/erocho/mem_process.txt')
```

Si revisamos en Hue, veremos que se ha creado una carpeta nueva, y el contenido que habíamos visto previamente en Jupyter:





Inicio / tmp / erocho / mem_process.txt / **part-00000**

```
('cloudera-scm', 12.964595920398963)
('impala', 9.569382067241872)
('hdfs', 4.235798658804982)
('yarn', 2.6055327130278934)
('hive', 1.9994142262519998)
('oozie', 1.7540489401786792)
('hbase', 1.7203536199965574)
('spark', 1.5721956212497556)
('erocho', 1.5379671141829647)
('hue', 1.4427932552695462)
('jorengap', 1.3167468302693122)
('rperezdelg', 1.289068158168112)
('mgomezrico', 1.2093096730085062)
('cesc', 1.1477370424740332)
('mapred', 0.8943373618738865)
('zookeeper', 0.8467478187559777)
('solr', 0.6475293348185203)
('dperezcala', 0.6416904156991546)
('rperezmartinez01', 0.6318160036879805)
('root', 0.6209431730986515)
('cristianhdezhdez', 0.3228760226014784)
```

c) Una vez validado el correcto funcionamiento de los programas, en este apartado vamos a combinar dos tareas en un flujo de trabajo de Oozie. Vamos a añadir a nuestro shell, el programa Spark que acabamos de crear. Muestra una captura de pantalla con el flujo de trabajo creado y con el resultado de la ejecución de los dos procesos anteriores en el job preview, en una captura de pantalla parecida a la figura 10. (1 punto)

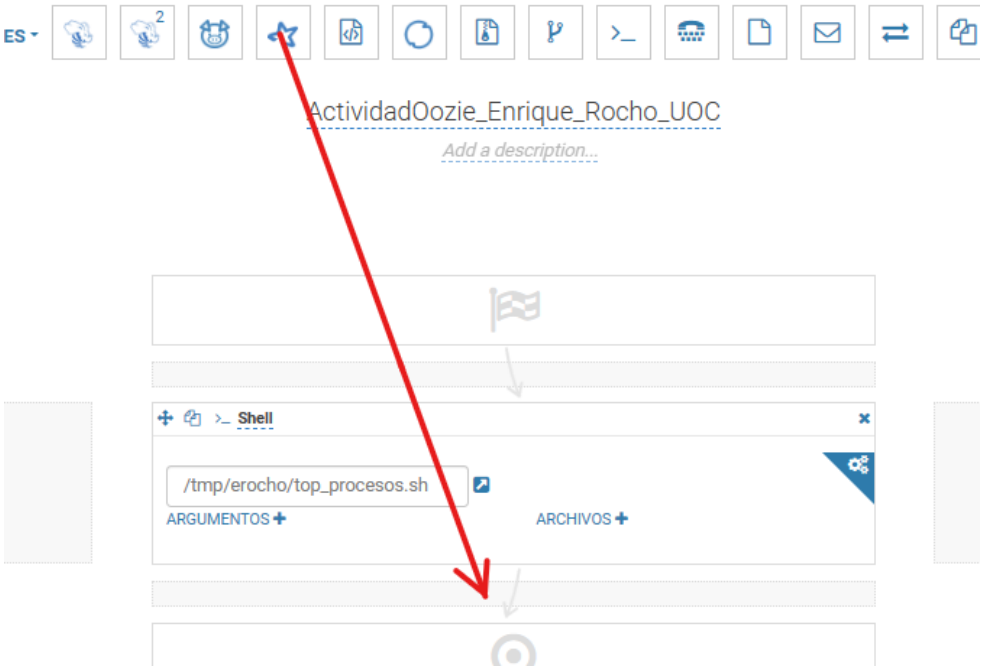
Primero subimos el programa a Hue y cambiamos los permisos:

Inicio / tmp / **erocho**

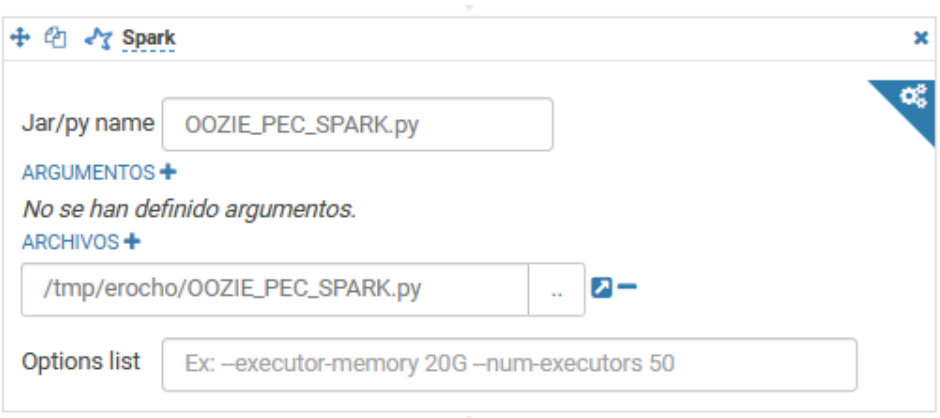
	Nombre
	↑
	.
	OOZIE_PEC_SPARK.py
	mem_process.txt
	top_procesos.sh
	top_procesos.txt

Mostrar 45 de 4 elementos

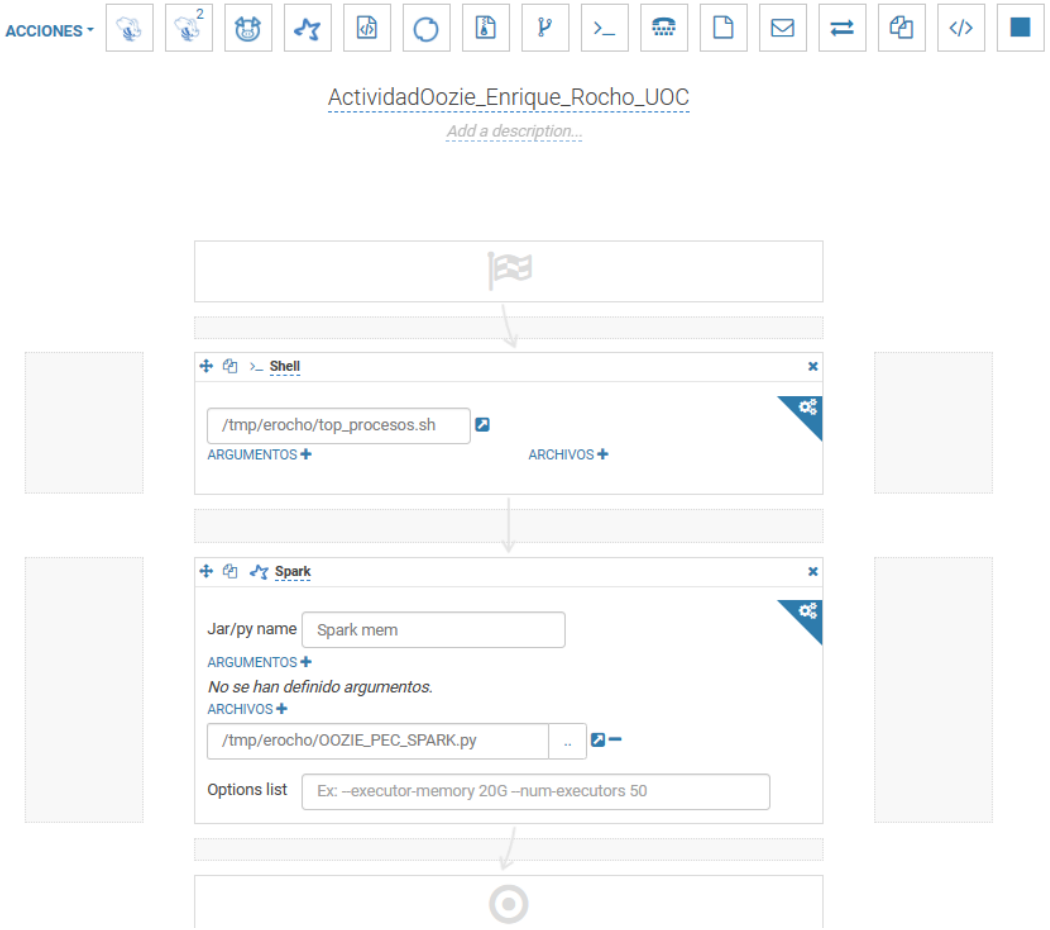
A continuación vamos al editor de Oozie y añadimos un proceso Spark:



Ahora cargamos el programa:

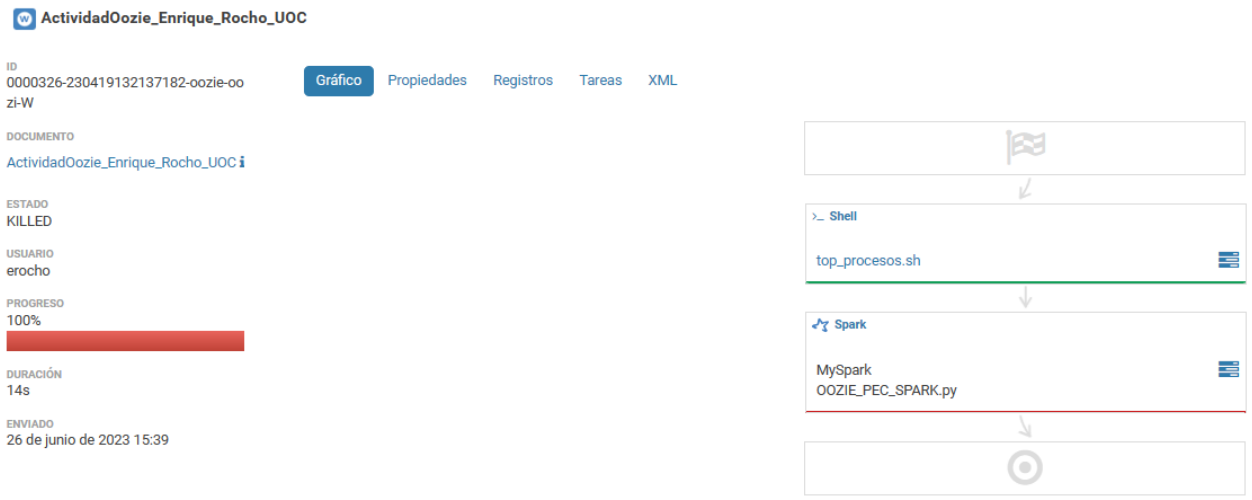


El flujo quedaría así:



Si ahora seleccionamos enviar, se abrirá la ventana de Job Browser>Workflows.

Veremos que en este caso el proceso shell top_procesos.sh se ha ejecutado correctamente, pero no el proceso Spark OOZIE_PEC_SPARK.py



Si revisamos Tareas, tendremos una descripción de cada una de ellas, donde podremos revisar el mensaje de error del proceso Spark:

Gráfico Propiedades Registros Tareas XML ▶ Reanudar

Registro	Estado	Mensaje de error	Código de error	ID externo	ID
	OK			application_1681903296516_1964	0000326-230419132137182-oozie-oozi-W@shell-f848
	ERROR	Main Class [org.apache.oozie.action.hadoop.SparkMain], exit code [1]	1	application_1681903296516_1965	0000326-230419132137182-oozie-oozi-W@spark-7c11

3.3 Almacenaje en MySQL (2 puntos)

En este apartado de la práctica se pide insertar los datos contenidos en el fichero con la agrupación por usuario de la memoria consumida en una tabla generada bajo la instancia MySQL disponible en el entorno. Para ello vamos a realizar un segundo programa con Spark que permita leer el fichero con el resultado agregado y almacenar los datos en una tabla de MySQL. Debemos limitar dicha consulta a los primeros 5 elementos, por tanto, en la base de datos no vamos a guardar los 10 usuarios que más memoria consumen, tan solo los primeros cinco. Los datos se deben almacenar en la tabla PEC_OOZIE_output de la base de datos PEC_OOZIE. La siguiente salida nos permite observar la estructura de los datos a almacenar. La columna usuario refiere al usuario que se está monitorizando y la columna user, al alumno que está ejecutando el ejercicio: allí debe aparecer vuestro nombre.

Para realizar este ejercicio debéis utilizar el conector de Spark para enlazar con bases de datos con JDBC3 (revisar el link en el pie de página). Debéis utilizar el driver de conexión disponible en la ruta /var/lib/sqoop/mysql-connector-java-8.0.26.jar. La base de datos se denomina PEC_OOZIE y la tabla PEC_OOZIE_output. Por tanto debéis especificar (entre otras) en Spark las opciones siguientes,

```
... .option("url", "jdbc:mysql://eimtcld2.uoclabs.uoc.es/PEC_OOZIE")\
 .option("dbtable", "PEC_OOZIE.PEC_OOZIE_output")
…
```

Debéis mostrar también el contenido del MySQL asociado a vuestro usuario. En el video explicativo del aula hay un ejemplo al respecto.

Usaremos el siguiente código:

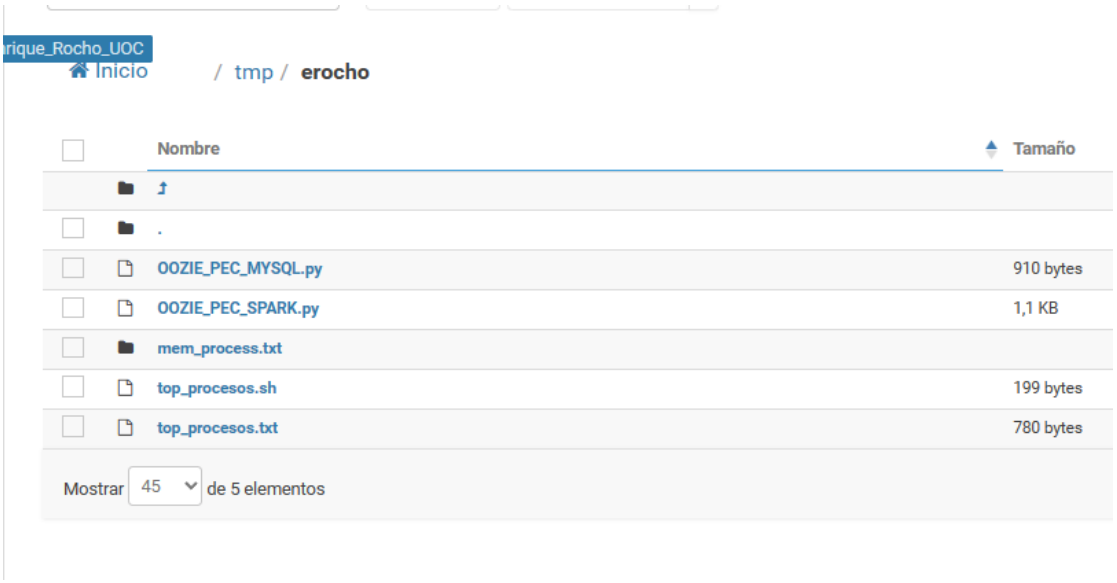
```
import re
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import lit, current_timestamp, regexp_replace,
row_number
from pyspark.sql.window import Window
from pyspark.sql.functions import desc

# crear sesion
spark = SparkSession.builder \
    .appName("MySQLInt") \
    .config("spark.jars", "/var/lib/sqoop/mysql-connector-java-8.0.26.jar") \
    .getOrCreate()

# Leer fichero
data =
spark.read.text("hdfs://eimtcld2.uoclabs.uoc.es/tmp/erocho/mem_process.txt/pa
rt-00000")
# Extraer columnas
```

```
data = data.selectExpr("split(value, ',')[0] as usuario", "split(value,
',')[1] as memoria")

# Quitar parentesis de columnas
data = data.withColumn("usuario", regexp_replace("usuario", r"\(|\)", ""))
data = data.withColumn("memoria", regexp_replace("memoria", r"\(|\)", ""))

# Añadir user
user = "erocho"
data_user = data.withColumn("User", lit(user))

# Añadir time
data_time = data_user.withColumn("Time", current_timestamp())

# Limitar resultado a 5 por memoria
window = Window.orderBy(desc("memoria"))
top_5_datos = data_time.withColumn("rank",
row_number().over(window)).filter("rank <= 5").drop("rank")

# Guardar mysql
top_5_datos.write.format("jdbc") \
    .option("url", "jdbc:mysql://eimtcld2.uoclabs.uoc.es/PEC_OOZIE") \
    .option("dbtable", "PEC_OOZIE.PEC_OOZIE_output") \
    .option("driver", "com.mysql.jdbc.Driver") \
    .option("user", "erocho") \
    .option("password", "jes4IJb6") \
    .mode("overwrite") \
    .save()

spark.stop()
```

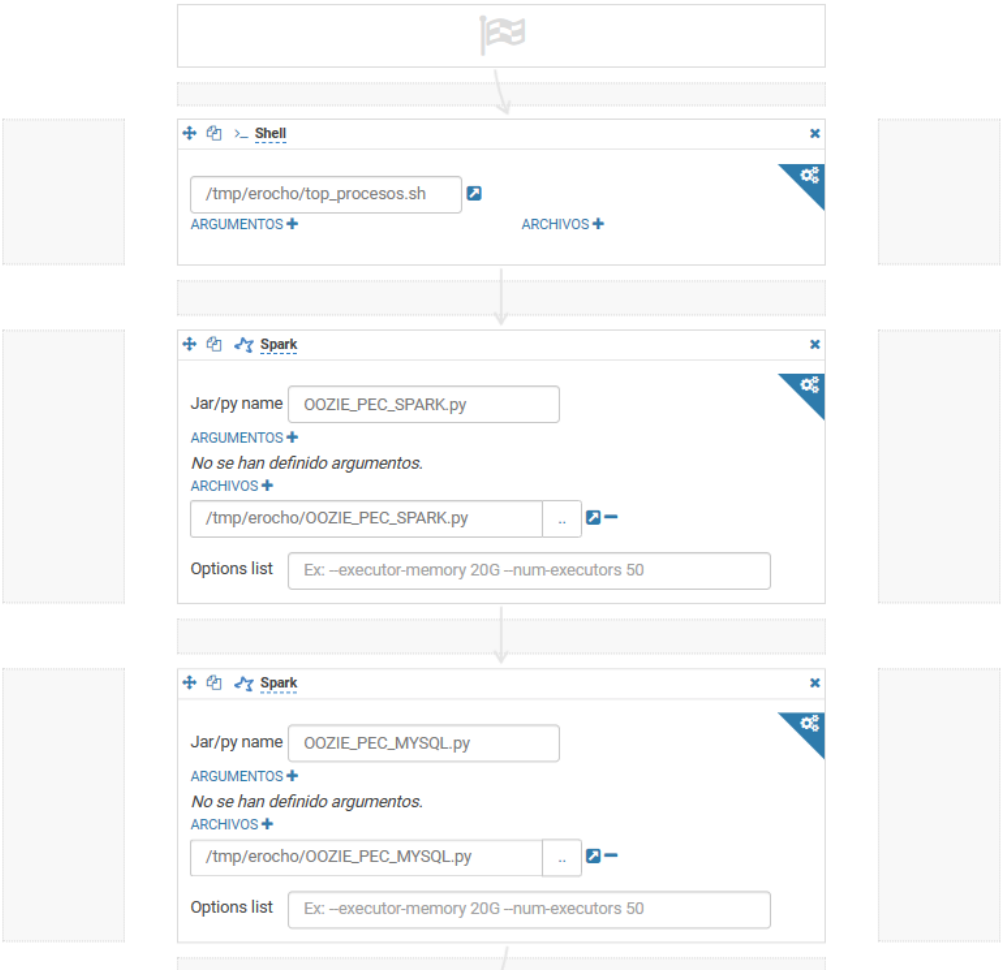
Si revisamos en mysql desde un terminal, obtenemos:

```
mysql> select * from PEC_OOZIE_output;
+----------------+---------------------+--------+---------------------+
| usuario        | memoria             | User   | Time                |
+----------------+---------------------+--------+---------------------+
| 'impala'       |  9.569251384181904  | erocho | 2023-06-27 16:18:11 |
| 'hdfs'         |  4.243331230381532  | erocho | 2023-06-27 16:18:11 |
| 'yarn'         |  3.1474335441355636 | erocho | 2023-06-27 16:18:11 |
| 'hive'         |  2.004244272148413  | erocho | 2023-06-27 16:18:11 |
| 'cloudera-scm' |  12.995599169545745 | erocho | 2023-06-27 16:18:11 |
+----------------+---------------------+--------+---------------------+
5 rows in set (0.00 sec)
```

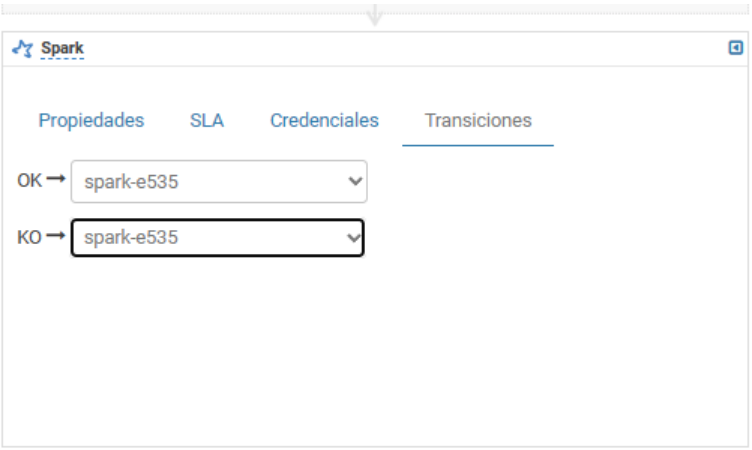
Ahora subiremos al directorio de hdfs con Hue el programa que hemos creado, OOZIE_PEC_MYSQL.py:



Añadiremos este nuevo programa al Workflow después del programa Spark anterior:



Para evitar el error del anterior programa Spark, al no ser esto producción real, se ha optado por poner como salida del OOZIE_PEC_SPARK al programa de SQL tanto para OK como para KO, de este modo el flujo no parará con el error, sino que seguirá con la ejecución de Spark SQL:



Igualmente obtendremos el mismo error para este programa que para el anterior:

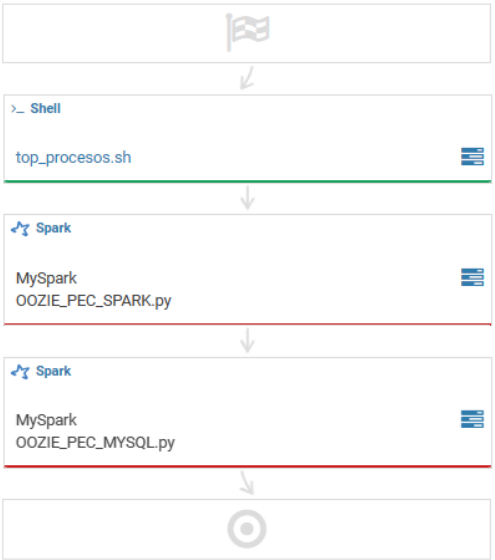


Gráfico Propiedades Registros Tareas XML

Registro	Estado	Mensaje de error	Código de error	ID externo	ID
	OK			application_1681903296516_2094	000041 W@shel
	ERROR	Main Class [org.apache.oozie.action.hadoop.SparkMain], exit code [1]	1	application_1681903296516_2095	000041 W@spa
	ERROR	Main Class [org.apache.oozie.action.hadoop.SparkMain], exit code [1]	1	application_1681903296516_2096	000041 W@spa

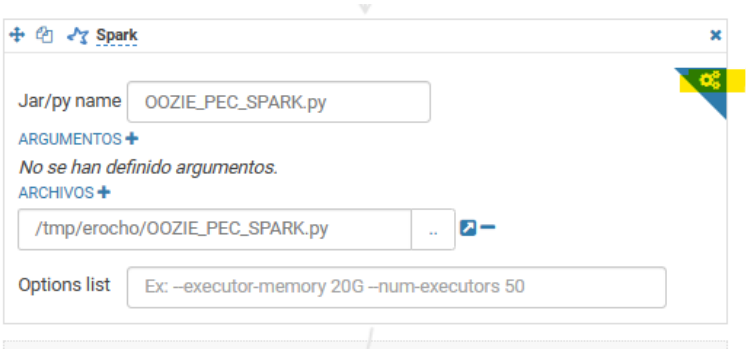
3.4 Ejecución en Oozie y ejecución periódica (3 puntos)

a) Para acabar nuestra actividad vamos a definir un flujo condicional de ejecución con Oozie. En concreto vamos a utilizar las transiciones entre tareas para el tratamiento de errores. En nuestro caso, la ejecución del programa Spark que escribe un fichero con el pair RDD (usuario, memoria) genera un error (KO) si el directorio ya se ha sido creado. Este error se soluciona garantizando que el directorio no exista. Así, queremos automatizar el proceso de borrar el directorio:

/tmp/<usuario>/<salida_3_2_b>

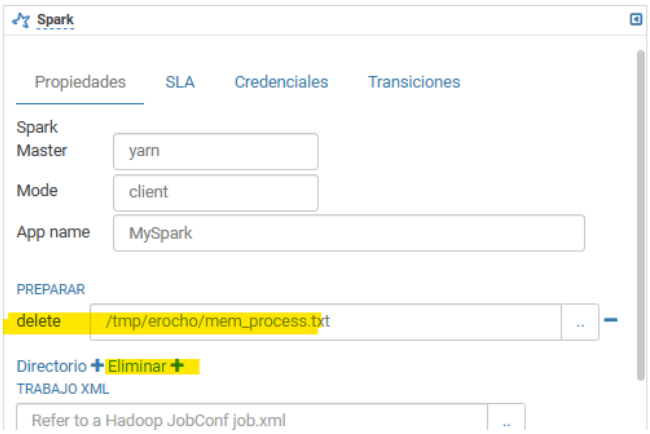
que se genera en la ejecución del programa del ejercicio 3.2.b. Adjuntar la captura de pantalla del flujo condicional, y el detalle de configuración de esta. (2 puntos)

Para eliminar el directorio que el proceso Spark está creando, tenemos que ir a editar el Workflow y acceder a las propiedades de este proceso:



Aquí podremos añadir la opción de eliminar, y el directorio al que queremos que afecte esta acción:

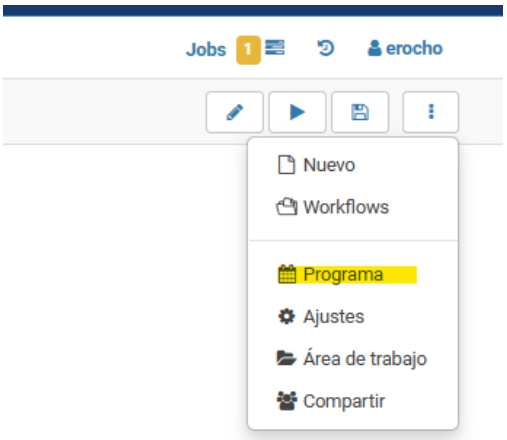
`/tmp/erocho/mem_process.txt`



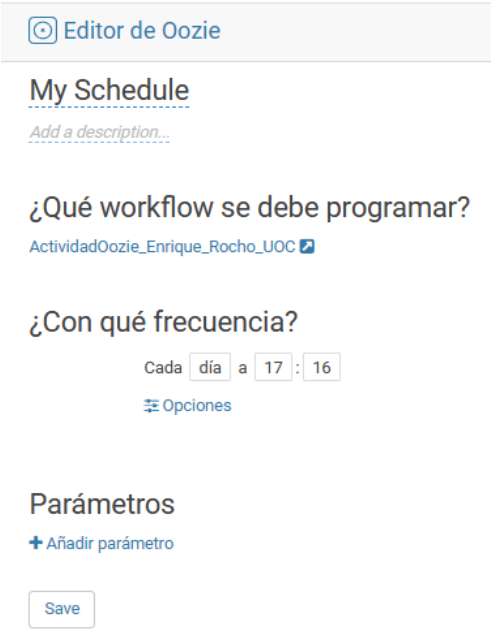
Sin embargo en este caso al fallar el proceso Spark del pair RDD, el fichero se eliminará igualmente y el siguiente Spark Mysql no podría extraer información al haberse eliminado ese fichero.

b) El último punto de la actividad consiste en definir este flujo de trabajo para que se ejecute periódicamente todos los días a hora determinada. Tenéis que definir la configuración de la planificación de la tarea (0,5 punto), y ejecutarla mostrando el resultado de esta (0,5 punto).

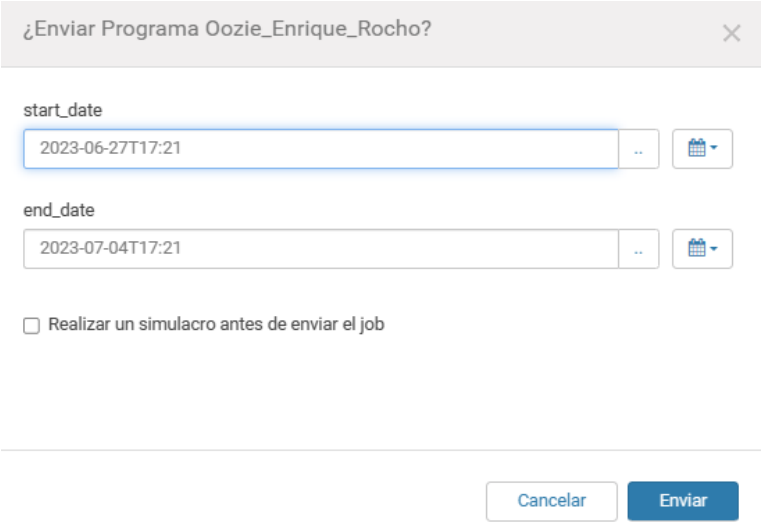
Desde la parte derecha de Workflow vamos a Programa:



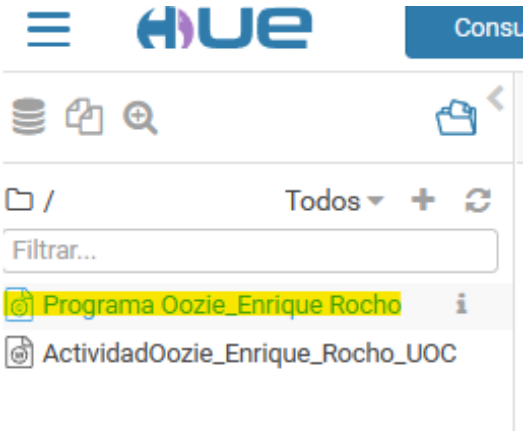
Se nos dará a elegir el workflow y el tiempo:



Seleccionamos Enviar y nos aparecerá una ventana donde elegiremos el margen de tiempo durante el cual el programa estará activo:



En el explorador de Hue encontraremos el Programa:



En Job Browser>Programa podremos ver los programas que tenemos:

Si entramos en uno, veremos su estado en detalle:

Job Browser Jobs Consultas Workflows Programas Bundles SLAs

Programa Oozie_Enrique Rocho

ID
0000432-230419132137182-oozie-oozi-C

DOCUMENTO
Programa Oozie_Enrique Rocho

TIPO
schedule

ESTADO
RUNNING

USUARIO
erocho

PROGRESO
14%

ENVIADO
27 Jun 2023 17:30:00

SIGUIENTE EJECUCIÓN
Wed, 28 Jun 2023 18:22:00

ACCIONES TOTALES
1

HORA DE FINALIZACIÓN
Tue, 04 Jul 2023 17:30:00

Tareas Registros Propiedades XML

Estado	Título	tipo	errorMess
WAITING	1-27 Jun 2023 18:22:00	schedule-task	

Una vez llegado el momento de la ejecución, obtendremos el resultado del programa, que en este caso aparece como Killed, ya que ya hemos visto que había errores al ejecutar los pasos de Spark:

Programa Oozie_Enrique Rocho

ID
0000432-230419132137182-oozie-oozi-C

DOCUMENTO
Programa Oozie_Enrique Rocho

TIPO
schedule

ESTADO
RUNNING

USUARIO
erocho

PROGRESO
14%

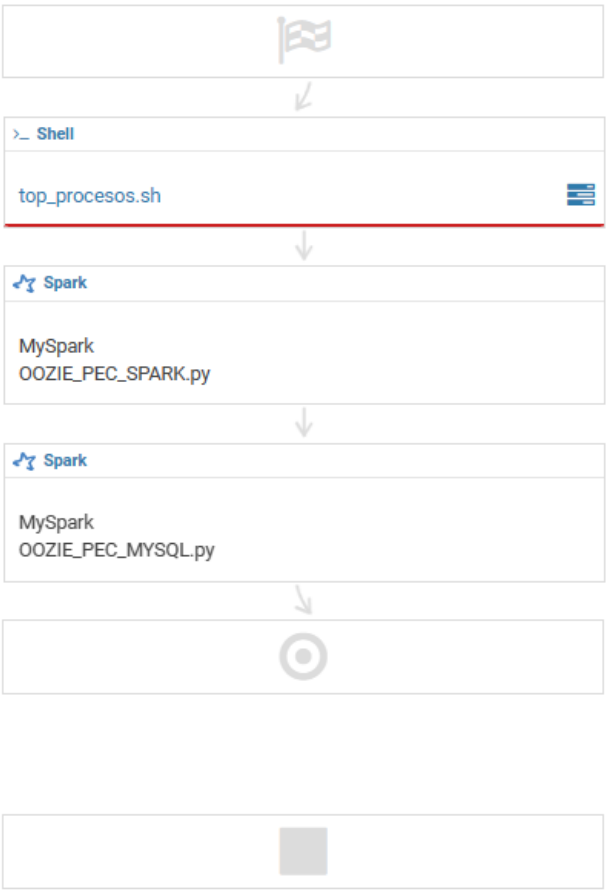
ENVIADO
27 Jun 2023 17:30:00

SIGUIENTE EJECUCIÓN
Wed, 28 Jun 2023 18:22:00

ACCIONES TOTALES
1

HORA DE FINALIZACIÓN
Tue, 04 Jul 2023 17:30:00

Tareas Registros Propiedades XML

Estado	Título	tipo	errorMessage	missingDependencies	número	errorCode
KILLED	1-27 Jun 2023 18:22:00	schedule-task			1	

Podemos también ver el detalle del workflow: