Workflow report.

Oozie.

Since we can use any data for our spark job, I created a simple list of 5 strings:

- String1
- String1
- String2
- String2
- String2

I group by the name of the string, and count the number of strings in each group. Here's the program:

```scala
package test

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.count

object Main {
  def main(args: Array[String]): Unit = {
    if (args.length != 2) {
      return
    }

    val inputPath = args(0)
    val outputPath = args(1)

    val spark = SparkSession.builder()
      .appName("sparkTestApp")
      .getOrCreate()

    spark.read
      .csv(inputPath)
      .toDF("name")
      .groupBy("name")
      .agg(count("name").alias("count"))
      .write
      .csv(outputPath)

    spark.stop()
  }
}
```
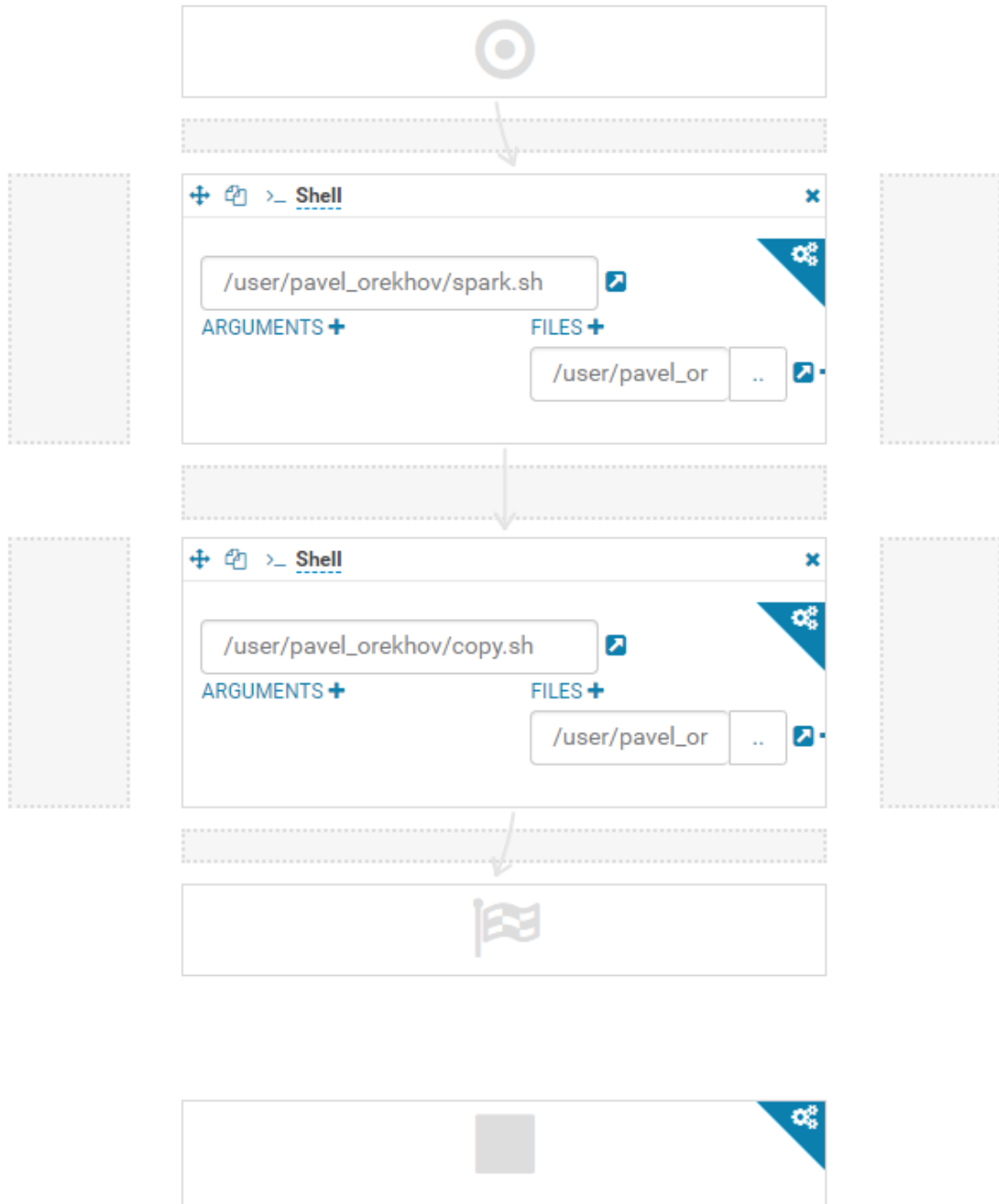
Cloudera has a UI for building workflows, here's my workflow:

My Workflow

Add a description...

Shell

/user/pavel_orekhov/spark.sh

ARGUMENTS +   FILES +

/user/pavel_or   ..

Shell

/user/pavel_orekhov/copy.sh

ARGUMENTS +   FILES +

/user/pavel_or   ..

As you can see, I first calculate the data and then copy it to a different folder. Here are my scripts for doing that:

Spark.sh – calculates, the contents are: **spark2-submit --class test.Main --master yarn-client hdfs:///user/pavel_orekhov/sparkTest.jar hdfs:///user/pavel_orekhov/data1.txt hdfs:///user/pavel_orekhov/output-data**

Copy.sh – copies, the contents are: **hdfs dfs -cp /user/pavel_orekhov/output-data /user/pavel_orekhov/output-data123**

Here's a screenshot, showing that both of my actions were successful:

And from the screenshot we can see the 2 of our folders:

	Name	Size	User	Group	Permissions	Date
☐	📁 ⬆		cloudera-scm	supergroup	drwxrwxrwx	December 26, 2018 12:53 AM
☐	📁 .		pavel_orekhov	supergroup	drwx------	January 28, 2019 07:14 AM
☐	📁 .Trash		pavel_orekhov	supergroup	drwx------	January 28, 2019 07:03 AM
☐	📁 .sparkStaging		pavel_orekhov	supergroup	drwxr-xr-x	January 28, 2019 04:18 AM
☐	📁 .staging		pavel_orekhov	supergroup	drwx------	January 28, 2019 06:36 AM
☐	📄 copy.sh	81 bytes	pavel_orekhov	supergroup	-rw-r--r--	January 28, 2019 07:10 AM
☐	📄 data1.txt	40 bytes	pavel_orekhov	supergroup	-rw-r--r--	January 28, 2019 05:45 AM
☐	📁 examples		pavel_orekhov	supergroup	drwxr-xr-x	January 25, 2019 05:13 AM
☐	📁 oozie-clou		pavel_orekhov	supergroup	drwxr-xr-x	January 28, 2019 06:36 AM
☐	📁 output-data		cloudera-scm	supergroup	drwxr-xr-x	January 28, 2019 07:13 AM
☐	📁 output-data123		cloudera-scm	supergroup	drwxr-xr-x	January 28, 2019 07:14 AM
☐	📄 spark.sh	170 bytes	pavel_orekhov	supergroup	-rw-r--r--	January 28, 2019 06:14 AM
☐	📄 sparkTest.jar	124.3 MB	pavel_orekhov	supergroup	-rw-r--r--	January 28, 2019 06:07 AM

And output:

/ user / pavel_orekhov / output-data123 /
part-00094-8287da36-3c0c-4e21-ac53-38ead125062b-c000.csv

```
string1,2
```

/ user / pavel_orekhov / output-data123 /
part-00130-8287da36-3c0c-4e21-ac53-38ead125062b-c000.csv

```
string2,3
```

We can also use oozie UI for monitoring. In the case of our cluster, it runs at http://ecsc00a02a04.epam.com:11000/oozie/

Here's the DAG for my job:

Job Info | Job Definition | Job Configuration | Job Log

start

shell-7dab

shell-5c07

End
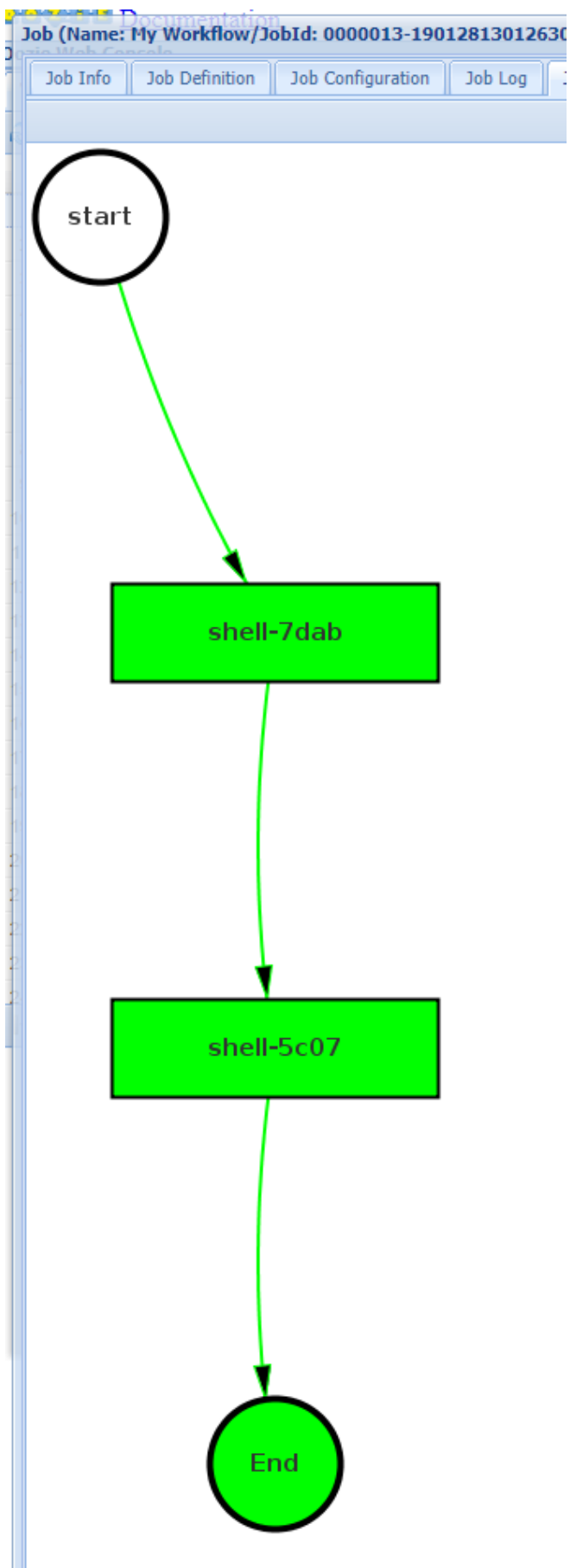
And here are the steps that my workflow took:

Actions

	Action Id	Name	Type	Status	Transition	StartTime	EndTime
1	0000013-190128130126308-oozie-clou-W@:start:	:start:	:START:	OK	shell-7dab	Mon, 28 Jan 2019 15:13:01 G…	Mon, 28 Jan 2019 15:13:01 G…
2	0000013-190128130126308-oozie-clou-W@shell-7dab	shell-7dab	shell	OK	shell-5c07	Mon, 28 Jan 2019 15:13:01 G…	Mon, 28 Jan 2019 15:14:01 G…
3	0000013-190128130126308-oozie-clou-W@shell-5c07	shell-5c07	shell	OK	End	Mon, 28 Jan 2019 15:14:01 G…	Mon, 28 Jan 2019 15:14:18 G…
4	0000013-190128130126308-oozie-clou-W@End	End	:END:	OK		Mon, 28 Jan 2019 15:14:18 G…	Mon, 28 Jan 2019 15:14:18 G…

Airflow.

Airflow can be easily installed by doing **pip3.6 SLUGIFY_USES_TEXT_UNIDECODE=yes install apache-airflow**
The most important thing to avoid errors during installation, is to use python version >=3.5.

The webserver to airflow can be launched like so: **airflow webserver -p 8081 -D**

Once again, since we can use any multi-column data, I created a simple list of strings:
person1,15
person2,16
person3,16
person4,23
person5,30

And here's the program that I run to process it:

```scala
package test

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.count
import com.databricks.spark.avro._

object Main {
  def main(args: Array[String]): Unit = {
    if (args.length != 2) {
      return
    }

    val inputPath = args(0)
    val outputPath = args(1)

    val spark = SparkSession.builder()
      .appName("sparkTestApp")
      .getOrCreate()

    spark.read
      .csv(inputPath)
      .toDF("name", "age")
      .groupBy("age")
      .agg(count("age").alias("count"))
      .write
      .avro(outputPath)

    spark.stop()
  }
}
```

As you can see, I save everything in the avro format.

In order to be able to run a DAG that transforms this data and then create a hive table on top of it, we need to write a definition file first, mine goes like this:

```python
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta


default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2015, 6, 1),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
```

```
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
}

dag = DAG(
    'pavel4', default_args=default_args, schedule_interval=timedelta(days=1))

# t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator(
    task_id='run_spark',
    bash_command='spark2-submit --class test.Main --master yarn-client hdfs:///user/pavel_orekhov/sparkTest.jar
hdfs:///user/pavel_orekhov/data1.txt hdfs:///user/pavel_orekhov/output-data',
    dag=dag)

t2 = BashOperator(
    task_id='create_hive_avro_table',
    bash_command='beeline -u jdbc:hive2://ecsc00a022c7.epam.com:10000/default -f /root/create_table.hql',
    retries=3,
    dag=dag)

t2.set_upstream(t1)
```

The contents of my **create_table.hql** script are the following:

```
CREATE EXTERNAL TABLE avro_hive_table
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
LOCATION
'/user/pavel_orekhov/output-data'
TBLPROPERTIES
('avro.schema.literal'='{
 "type" : "record",
 "name" : "topLevelRecord",
 "fields" : [ {
   "name" : "age",
   "type" : [ "string", "null" ]
 }, {
   "name" : "count",
   "type" : "long"
 } ]
}');
```

In order to instantiate this DAG, we have to run this script, by doing **python3.6 script.py**. My script is called **pavel4.py**. And we can
check that the DAG is in airflow's list of DAGs by doing **airflow list_dags**.

```
[2019-01-29 19:06:04,689] {models.py:273} INFO - Filling up the DagBag from /root/airflow/dags


----------------------------------------------------------------
DAGS
----------------------------------------------------------------
example_bash_operator
example_branch_dop_operator_v3
example_branch_operator
example_http_operator
example_passing_params_via_test_command
example_python_operator
example_short_circuit_operator
example_skip_dag
example_subdag_operator
example_subdag_operator.section-1
example_subdag_operator.section-2
example_trigger_controller_dag
example_trigger_target_dag
example_xcom
latest_only
latest_only_with_trigger
pavel
pavel2
pavel3
pavel4
test_utils
tutorial
```

As you can see, mine is there.

Once instantiated, the DAG can also be viewed in the webserver UI that airflow provides.

As you can see, **pavel4** is there.

Here's the screenshot of the successful execution of my pipeline:

We can verify that the data has been written into hive by doing **select * from avro_hive_table** from the beeline CLI.
Here's our table:

```
beeline> !connect jdbc:hive2://ecsc00a022c7.epam.com:10000/default "" "" ""
scan complete in 2ms
Connecting to jdbc:hive2://ecsc00a022c7.epam.com:10000/default
Connected to: Apache Hive (version 1.1.0-cdh5.12.2)
Driver: Hive JDBC (version 1.1.0-cdh5.12.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://ecsc00a022c7.epam.com:10000/d> show tables;
INFO  : Compiling command(queryId=cloudera-scm_20190129191414_ab4fd637-5502-4a7e-b5ed-b6d357b16946): show tables
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:tab_name, type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=cloudera-scm_20190129191414_ab4fd637-5502-4a7e-b5ed-b6d357b16946); Time taken: 0.04 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=cloudera-scm_20190129191414_ab4fd637-5502-4a7e-b5ed-b6d357b16946): show tables
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=cloudera-scm_20190129191414_ab4fd637-5502-4a7e-b5ed-b6d357b16946); Time taken: 0.168 seconds
INFO  : OK
+-------------------+--+
|     tab_name      |
+-------------------+--+
| airflow_result    |
| airports          |
| avro_hive_table   |
| carriers          |
| flights           |
| flow_result2      |
| flow_result3      |
+-------------------+--+
7 rows selected (0.387 seconds)
0: jdbc:hive2://ecsc00a022c7.epam.com:10000/d>
```

And here's our data:

```
0: jdbc:hive2://ecsc00a022c7.epam.com:10000/d> select * from avro_hive_table;
INFO  : Compiling command(queryId=cloudera-scm_20190129191515_ba145bc5-84db-4cda-953b-910491215866): select * from avro_hive_table
INFO  : Semantic Analysis Completed
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:avro_hive_table.age, type:string, comment:null), FieldSchema(name:av
)
INFO  : Completed compiling command(queryId=cloudera-scm_20190129191515_ba145bc5-84db-4cda-953b-910491215866); Time taken: 0.115 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=cloudera-scm_20190129191515_ba145bc5-84db-4cda-953b-910491215866): select * from avro_hive_table
INFO  : Completed executing command(queryId=cloudera-scm_20190129191515_ba145bc5-84db-4cda-953b-910491215866); Time taken: 0.0 seconds
INFO  : OK
+----------------------+-----------------------+--+
| avro_hive_table.age  | avro_hive_table.count  |
+----------------------+-----------------------+--+
| 15                   | 1                     |
| 30                   | 1                     |
| 16                   | 2                     |
| 23                   | 1                     |
+----------------------+-----------------------+--+
4 rows selected (1.327 seconds)
```