Apache Spark Component Guide

Also available as:

« Prev Next »

Chapter 7. Automating Spark Jobs with Oozie Spark Action

If you use Apache Spark as part of a complex workflow with multiple processing steps, triggers, and interdependencies, consider using Apache Oozie to automate jobs. Oozie is a workflow engine that executes sequences of actions structured as directed acyclic graphs (DAGs). Each action is an individual unit of work, such as a Spark job or Hive query.

The Oozie "Spark action" runs a Spark job as part of an Oozie workflow. The workflow waits until the Spark job completes before continuing to the next action.

For additional information about Spark action, see the Apache Oozie Spark Action Extension documentation. For general information about Oozie, see Using HDP for Workflow and Scheduling with Oozie. For general information about using Workflow Manager, see the Workflow Management Guide.

Note

In HDP 2.6, Oozie works with either Spark 1 or Spark 2 (not side-by-side deployments). You can configure Spark 2 through manual steps (not Ambari).

Support for yarn-client execution mode for Oozie Spark action will be removed in a future release. Oozie will continue to support yarn-cluster execution mode for Oozie Spark action.

Configuring Oozie Spark Action for Spark 1

To place a Spark job into an Oozie workflow, you need two configuration files:

- A workflow XML file that defines workflow logic and parameters for running the Spark job. Some of the elements in a Spark action are specific to Spark; others are common to many types of actions.
- A `job.properties` file for configuring the Oozie job.

You can configure a Spark action manually, or on an Ambari-managed cluster you can use the Spark action editor in the Ambari Oozie Workflow Manager (WFM). The Workflow Manager is designed to help build powerful workflows.

For two examples that use Oozie Workflow Manager--one that creates a new Spark action, and another that imports and runs an existing Spark workflow--see the Hortonworks Community Connection article Apache Ambari Workflow Manager View for Apache Oozie: Part 7 (Spark Action & PySpark).

Here is the basic structure of a workflow definition XML file for a Spark action:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.3">
    ...
    <action name="[NODE-NAME]">
        <spark xmlns="uri:oozie:spark-action:0.1">
            <job-tracker>[JOB-TRACKER]</job-tracker>
            <name-node>[NAME-NODE]</name-node>
            <prepare>
               <delete path="[PATH]"/>
               ...
               <mkdir path="[PATH]"/>
               ...
            </prepare>
            <job-xml>[SPARK SETTINGS FILE]</job-xml>
            <configuration>
                <property>
                    <name>[PROPERTY-NAME]</name>
                    <value>[PROPERTY-VALUE]</value>
                </property>
                ...
            </configuration>
            <master>[SPARK MASTER URL]</master>
            <mode>[SPARK MODE]</mode>
            <name>[SPARK JOB NAME]</name>
            <class>[SPARK MAIN CLASS]</class>
            <jar>[SPARK DEPENDENCIES JAR / PYTHON FILE]</jar>
            <spark-opts>[SPARK-OPTIONS]</spark-opts>
            <arg>[ARG-VALUE]</arg>
                ...
            <arg>[ARG-VALUE]</arg>
            ...
        </spark>
        <ok to="[NODE-NAME]"/>
        <error to="[NODE-NAME]"/>
    </action>
    ...
</workflow-app>
```

The following examples show a workflow definition XML file and an Oozie job configuration file for running a SparkPi job (Spark version 1.x).

Sample `Workflow.xml` file for `SparkPi` app:

```
<workflow-app xmlns='uri:oozie:workflow:0.5' name='SparkWordCount'>
      <start to='spark-node' />
        <action name='spark-node'>
          <spark xmlns="uri:oozie:spark-action:0.1">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <prepare>
              <delete path="${nameNode}/user/${wf:user()}/${examplesRoot}/output-
data"/>
            </prepare>
            <master>${master}</master>
            <name>SparkPi</name>
            <class>org.apache.spark.examples.SparkPi</class>
            <jar>lib/spark-examples.jar</jar>
            <spark-opts>--executor-memory 20G --num-executors 50</spark-opts>
            <arg>value=10</arg>
          </spark>
          <ok to="end" />
          <error to="fail" />
        </action>
        <kill name="fail">
          <message>Workflow failed, error
            message[${wf:errorMessage(wf:lastErrorNode())}] </message>
        </kill>
        <end name='end' />
    </workflow-app>
```

Sample `Job.properties` file for `SparkPi` app:

```
nameNode=hdfs://host:8020
jobTracker=host:8050
queueName=default
examplesRoot=examples
oozie.use.system.libpath=true
oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/apps/pyspark
master=yarn-cluster
```

Configuring Oozie Spark Action for Spark 2

To use Oozie Spark action with Spark 2 jobs, create a `spark2` ShareLib directory, copy associated files into it, and then point Oozie to `spark2`. (The Oozie ShareLib is a set of libraries that allow jobs to run on any node in a cluster.)

1. Create a `spark2` ShareLib directory under the Oozie ShareLib directory associated with the `oozie` service user:

   ```
   hdfs dfs -mkdir /user/oozie/share/lib/lib_<ts>/spark2
   ```

2. Copy `spark2` jar files from the `spark2` jar directory to the Oozie `spark2` ShareLib:

   ```
   hdfs dfs -put \
       /usr/hdp/current/spark2-client/jars/* \
       /user/oozie/share/lib/lib_<ts>/spark2/
   ```

3. Copy the `oozie-sharelib-spark` jar file from the `spark` ShareLib directory to the `spark2` ShareLib directory:

```
hdfs dfs -cp \
    /user/oozie/share/lib/lib_<ts>/spark/oozie-sharelib-spark-*.jar \
    /user/oozie/share/lib/lib_<ts>/spark2/
```

4. Copy the `hive-site.xml` file for Spark2 to the `spark2` ShareLib:

```
hdfs dfs -put \
        /usr/hdp/current/spark2-client/conf/hive-site.xml \
    /user/oozie/share/lib/lib_<ts>/spark2/
```

5. Copy Python libraries to the `spark2` ShareLib:

```
hdfs dfs -put \
    /usr/hdp/current/spark2-client/python/lib/py* \
    /user/oozie/share/lib/lib_<ts>/spark2/
```

6. Run the Oozie `sharelibupdate` command:

```
oozie admin –sharelibupdate
```

To verify the configuration, run the Oozie `shareliblist` command. You should see `spark2` in the results.

```
oozie admin –shareliblist spark2
```

To run a Spark job with the `spark2` ShareLib, set the following properties in the `job.properties` file:

```
oozie.action.sharelib.for.spark=spark2
oozie.action.sharelib.for.spark.exclusion=oozie/jackson
```

The following examples show a workflow definition XML file, an Oozie job configuration file, and a Python script for running a Spark2-Pi job.

Sample `Workflow.xml` file for `spark2-Pi`:

```
<workflow-app xmlns='uri:oozie:workflow:0.5' name='SparkPythonPi'>
          <start to='spark-node' />

          <action name='spark-node'>
            <spark xmlns="uri:oozie:spark-action:0.1">
              <job-tracker>${jobTracker}</job-tracker>
              <name-node>${nameNode}</name-node>
              <master>${master}</master>
              <name>Python-Spark-Pi</name>
              <jar>pi.py</jar>
            </spark>
            <ok to="end" />
            <error to="fail" />
          </action>

          <kill name="fail">
            <message>Workflow failed, error message
[${wf:errorMessage(wf:lastErrorNode())}]</message>
          </kill>
          <end name='end' />
        </workflow-app>
```

Sample `Job.properties` file for `spark2-Pi`:

```
nameNode=hdfs://host:8020
jobTracker=host:8050
queueName=default
examplesRoot=examples
oozie.use.system.libpath=true
oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/apps/pyspark
master=yarn-cluster
oozie.action.sharelib.for.spark=spark2
```

Sample Python script, `lib/pi.py`:

```
import sys
from random import random
from operator import add
from pyspark import SparkContext

if __name__ == "__main__":
"""
Usage: pi [partitions]
"""
sc = SparkContext(appName="Python-Spark-Pi")
partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
n = 100000 * partitions

def f(_):
x = random() * 2 - 1
```

```
y = random() * 2 - 1
return 1 if x ** 2 + y ** 2 < 1 else 0

count = sc.parallelize(range(1, n + 1), partitions).map(f).reduce(add)
print("Pi is roughly %f" % (4.0 * count / n))

sc.stop()
```