FREE eBook

LEARNING

oozie

#oozie

Table of Contents

About

Chapter 1: Getting started with oozie

Remarks

Oozie is an Apache open source project, originally developed at Yahoo. Oozie is a general purpose scheduling system for multistage Hadoop jobs.

- Oozie allow to form a logical grouping of relevant Hadoop jobs into an entity called `Workflow`. The Oozie workflows are DAG (Directed cyclic graph) of actions.
- Oozie provides a way to schedule **Time** or **Data** dependent Workflow using an entity called `Coordinator`.
- Further you can combine the related Coordinators into an entity called `Bundle` and can be scheduled on a Oozie server for execution.

Oozie support most of the Hadoop Jobs as Oozie Action Nodes like: `MapRedude`, `Java`, `FileSystem` (HDFS operations), `Hive`, `Hive2`, `Pig`, `Spark`, `SSH`, `Shell`, `DistCp` and `Sqoop`. It provides a decision capability using a `Decision Control Node` action and Parallel execution of the jobs using `Fork-Join Control Node`. It allow users to configure email option for Success/Failure notification of the Workflow using `Email` action.

Versions

Oozie Version	Release Date
4.3.0	2016-12-02

Examples

Installation or Setup

Pre-requisites

This article demonstrated installing oozie-4.3.0 on Hadoop 2.7.3

1. Java 1.7+
2. Hadoop 2.x (here, 2.7.3)
3. Maven3+
4. Unix box

Step1: Dist file

Get oozie tar.gz file from http://www-eu.apache.org/dist/oozie/4.3.0/ and extract it

```
cd $HOME
tar -xvf oozie-4.3.0.tar.gz
```

Step2: Build Oozie

```
cd $HOME/oozie-4.3.0/bin
./mkdistro.sh -DskipTests
```

Step3: Server Installation

Copy the built binaries to the home directory as ‘oozie’

```
cd $HOME
cp -R $HOME/oozie-4.3.0/distro/target/oozie-4.3.0-distro/oozie-4.3.0 .
```

Step 3.1: libext Create libext directory inside oozie directory

```
 cd $HOME/oozie
 mkdir libext
```

Note: ExtJS (2.2+) library (optional, to enable Oozie webconsole) But, The ExtJS library is not bundled with Oozie because it uses a different license :(Now you need to put hadoop jars inside libext directory, else it will throw below error in oozie.log file

> WARN ActionStartXCommand:523 - SERVER[data01.teg.io] USER[hadoop] GROUP[-] TOKEN[] APP[map-reduce-wf] JOB[0000000-161215143751620-oozie-hado-W] ACTION[0000000-161215143751620-oozie-hado-W@mr-node] Error starting action [mr-node]. ErrorType [TRANSIENT], ErrorCode [JA009], Message [JA009: Cannot initialize Cluster. Please check your configuration for mapreduce.framework.name and the correspond server addresses.]

So, let's put below jars inside libext directory

```
cp $HADOOP_HOME/share/hadoop/common/*.jar oozie/libext/
cp $HADOOP_HOME/share/hadoop/common/lib/*.jar oozie/libext/
cp $HADOOP_HOME/share/hadoop/hdfs/*.jar oozie/libext/
cp $HADOOP_HOME/share/hadoop/hdfs/lib/*.jar oozie/libext/
cp $HADOOP_HOME/share/hadoop/mapreduce/*.jar oozie/libext/
cp $HADOOP_HOME/share/hadoop/mapreduce/lib/*.jar oozie/libext/
cp $HADOOP_HOME/share/hadoop/yarn/*.jar oozie/libext/
cp $HADOOP_HOME/share/hadoop/yarn/lib/*.jar oozie/libext/
```

Step 3.2: Oozie Impersonate To avoid impersonate error on oozie, modify core-site.xml like below

```
<!-- OOZIE -->
  <property>
    <name>hadoop.proxyuser.[OOZIE_SERVER_USER].hosts</name>
    <value>[OOZIE_SERVER_HOSTNAME]</value>
  </property>
  <property>
    <name>hadoop.proxyuser.[OOZIE_SERVER_USER].groups</name>
    <value>[USER_GROUPS_THAT_ALLOW_IMPERSONATION]</value>
  </property>
```

Assuming, my oozie user is huser and host is localhost and group is hadoop

```
<!-- OOZIE -->
  <property>
    <name>hadoop.proxyuser.huser.hosts</name>
    <value>localhost</value>
  </property>
  <property>
    <name>hadoop.proxyuser.huser.groups</name>
    <value>hadoop</value>
  </property>
```

Note : You can use * in all values, in case of confusion

Step 3.3: Prepare the war

```
cd $HOME/oozie/bin
./oozie-setup.sh prepare-war
```

This will create oozie.war file inside oozie directory. If this war will be used further, you may face this error :

> ERROR ActionStartXCommand:517 - SERVER[data01.teg.io] USER[hadoop] GROUP[-] TOKEN[] APP[map-reduce-wf] JOB[0000000-161220104605103-oozie-hado-W] ACTION[0000000-161220104605103-oozie-hado-W@mr-node] Error, java.lang.NoSuchFieldError: HADOOP_CLASSPATH

Why? because, The oozie compilation produced Hadoop 2.6.0 jars even when specifying Hadoop 2.7.3 with the option "-Dhadoop.version=2.7.3".

So, to avoid this error, copy the oozie.war file to a different directory

```
mkdir $HOME/oozie_war_dir
cp $HOME/oozie/oozie.war $HOME/oozie_war_dir
cd $HOME/oozie_war_dir
jar -xvf oozie.war
rm -f oozie.war/WEB-INF/lib/hadoop-*.jar
rm -f oozie.war/WEB-INF/lib/hive-*.jar
rm oozie.war
jar -cvf oozie.war ./*
cp oozie.war $HOME/oozie/
```

Then, regenerate the oozie.war binaries for oozie with a prepare-war

```
cd $HOME/oozie/bin
./oozie-setup.sh prepare-war
```

Step 3.4: Create sharelib on HDFS

```
cd $HOME/oozie/bin
./oozie-setup.sh sharelib create -fs hdfs://localhost:9000
```

Now, this sharelib set up may give you below error:

> org.apache.oozie.service.ServiceException: E0104: Could not fully initialize service [org.apache.oozie.service.ShareLibService], Not able to cache sharelib. An Admin needs to install the sharelib with oozie-setup.sh and issue the 'oozie admin' CLI command to update the sharelib

To avoid this, modify oozie-site.xml like below

```
cd $HOME/oozie
vi conf/oozie-site.xml
```

Add below property

```
<property>
    <name>oozie.service.HadoopAccessorService.hadoop.configurations</name>
    <value>*=/usr/local/hadoop/etc/hadoop/</value>
</property>
```

The value should be your $HADOOP_HOME/etc/hadoop, where all hadoop configuration files are present.

Step 3.5 : Create Oozie DB

```
cd $HOME/oozie
./bin/ooziedb.sh create -sqlfile oozie.sql -run
```

Step 3.6 : Start Daemon

To start Oozie as a daemon use the following command:

```
./bin/oozied.sh start
```

To stop

```
./bin/oozied.sh stop
```

check logs for errors, if any

```
cd $HOME/oozie/logs
tail -100f oozie.log
```

Use the following command to check the status of Oozie from command line:

```
$ ./bin/oozie admin -oozie http://localhost:11000/oozie -status
System mode: NORMAL
```

Step 4: Client Installation

```
$ cd
$ cp oozie/oozie-client-4.3.0.tar.gz .
$ tar -xvf oozie-client-4.3.0.tar.gz
$ mv oozie-client-3.3.2 oozie-client
$ cd bin
```

Add $HOME/oozie-client/bin to PATH variable in .bashrc file and restart your terminal or do

```
source $HOME/.bashrc
```

For more details on set up, you can refer this URL
https://oozie.apache.org/docs/4.3.0/DG_QuickStart.html

Now you can submit hadoop jobs to oozie in your terminal.

To run an example, you can follow this URL and set up your first example to run
https://oozie.apache.org/docs/4.3.0/DG_Examples.html

You may face below error while running the map reduce example in above URL

> java.io.IOException: java.net.ConnectException: Call From localhost.localdomain/127.0.0.1 to 0.0.0.0:10020 failed on connection exception: java.net.ConnectException: Connection refused; For more details see: http://wiki.apache.org/hadoop/ConnectionRefused

Solution: Start mr-jobhistory-server.sh

```
cd $HADOOP_HOME/sbin
./mr-jobhistory-server.sh start historyserver
```

Another point to note about modifying job.properties file is :

```
nameNode=hdfs://localhost:9000
jobTracker=localhost:8032
```

in your case, this can be different, as I am using apache hadoop, you may be using cloudera/hdp/anything

```
To run spark job, I have tried running in local[*], yarn-client and
yarn-cluster as master, but succeeded in local[*] only
```

Read Getting started with oozie online: https://riptutorial.com/oozie/topic/3437/getting-started-with-oozie

Chapter 2: Oozie 101

Examples

Oozie Architecture

Oozie is developed on a client-server architecture. Oozie server is a Java web application that runs Java servlet container within an embedded Apache Tomcat. Oozie provides three different type of clients to interact with the Oozie server: Command Line, Java Client API and HTTP REST API.

Oozie server does not store any in-memory information of the running jobs. It relies on RDBMS to store states and data of all the Oozie jobs. Every time it retrieves the job information from the database and stores updated information back into the database.

Oozie Server (can) sits outside of the Hadoop cluster and performs orchestration of the Hadoop jobs defined in a Oozie Workflow job.

Oozie Application Deployment

A simplest Oozie application is consists of a workflow logic file (workflow.xml), workflow properties file (job.properties/job.xml) and required JAR files, scripts and configuration files. Except the workflow properties file, all the other files should to be stored in a HDFS location. The workflow properties file should be available locally, from where Oozie application is submitted and started.

The HDFS directory, where workflow.xml is stored along with other scripts and configuration files, is called Oozie workflow application directory. All the JAR files should be stored under a /lib directory in the oozie application directory.

The more complex Oozie applications can consist of coordinators (coordinator.xml) and bundle (bundle.xml) logic files. These files are also stored in the HDFS into a respective Oozie application directory.

How to pass configuration with Oozie Proxy Job submission

When using the Oozie Proxy job submission API for submitting the Oozie `Hive`, `Pig`, and `Sqoop` actions. To pass any configuration to the action, is required to be in below format.

For Hive action:

- oozie.hive.options.size : The number of options you'll be passing to Hive action.
- oozie.hive.options.n : An argument to pass to Hive, the 'n' should be an integer starting with zero (0) to indicate the option number.

```
<property>
    <name>oozie.hive.options.1</name>
    <value>-Doozie.launcher.mapreduce.job.queuename=hive</value>
```

```
</property>
<property>
    <name>oozie.hive.options.0</name>
    <value>-Dmapreduce.job.queuename=hive</value>
</property>
<property>
    <name>oozie.hive.options.size</name>
    <value>2</value>
</property>
```

For Pig Action:

- oozie.pig.options.size : The number of options you'll be passing to Pig action.
- oozie.pig.options.n : An argument to pass to Pig, the 'n' should be an integer starting with zero (0) to indicate the option number.

```
<property>
    <name>oozie.pig.options.1</name>
    <value>-Doozie.launcher.mapreduce.job.queuename=pig</value>
</property>
<property>
    <name>oozie.pig.options.0</name>
    <value>-Dmapreduce.job.queuename=pig</value>
</property>
<property>
    <name>oozie.pig.options.size</name>
    <value>2</value>
</property>
```

For Sqoop Action:

- oozie.sqoop.options.size : The number of options you'll be passing to Sqoop Hadoop job.
- oozie.sqoop.options.n : An argument to pass to Sqoop. hadoop job conf, the 'n' should be an integer starting with zero(0) to indicate the option number.

```
<property>
    <name>oozie.sqoop.options.1</name>
    <value>-Doozie.launcher.mapreduce.job.queuename=sqoop</value>
</property>
<property>
    <name>oozie.sqoop.options.0</name>
    <value>-Dmapreduce.job.queuename=sqoop</value>
</property>
<property>
    <name>oozie.sqoop.options.size</name>
    <value>2</value>
</property>
```

Read Oozie 101 online: https://riptutorial.com/oozie/topic/4134/oozie-101

Chapter 3: Oozie data triggered coordinator

Introduction

A detailed explanation is given on oozie data triggered coordinator job with example.

Coordinator runs periodically from the start time until the end time. Beginning at start time, the coordinator job checks if input data is available. When the input data becomes available, a workflow is started to process the input data which on completion produces the required output data. This process is repeated at every tick of frequency until the end time of coordinator.

Remarks

```
    <done-flag>_SUCCESS</done_flag>
```

The above snippet in coordinator.xml for input dataset signals the presence of input data. That means coordinator action will be in WAITING state till _SUCCESS file is present in the given input directory. Once it is present, workflow will start execution.

Examples

oozie coordinator sample

The below coordinator job will trigger coordinator action once in a day that executes a workflow. The workflow has a shell script that moves input to output.

```
<coordinator-app name="log_process_coordinator" frequency="${coord:days(1)}" start="2017-04-
29T06:00Z" end="2018-04-29T23:25Z" timezone="UTC"
             xmlns="uri:oozie:coordinator:0.2">
<datasets>
    <dataset name="input_dataset" frequency="${coord:days(1)}" initial-instance="2017-04-
29T06:00Z" timezone="GMT">
        <uri-template>${nameNode}/mypath/coord_job_example/input/${YEAR}${MONTH}${DAY}</uri-
template>
        <done-flag>_SUCCESS</done-flag>
    </dataset>
    <dataset name="output_dataset" frequency="${coord:days(1)}" initial-instance="2017-04-
29T06:00Z" timezone="GMT">
        <uri-template>${nameNode}/mypath/coord_job_example/output/${YEAR}${MONTH}${DAY}</uri-
template>
        <done-flag>_SUCCESS</done-flag>
    </dataset>
</datasets>
<input-events>
    <data-in name="input_event" dataset="input_dataset">
        <instance>${coord:current(0)}</instance>
    </data-in>
</input-events>
 <output-events>
```

```
    <data-out name="output_event" dataset="output_dataset">
        <instance>${coord:current(0)}</instance>
    </data-out>
</output-events>
 <action>
    <workflow>
        <app-path>${workflowAppUri}</app-path>
        <configuration>
            <property>
                <name>jobTracker</name>
                <value>${jobTracker}</value>
            </property>
            <property>
                <name>nameNode</name>
                <value>${nameNode}</value>
            </property>
            <property>
                <name>pool.name</name>
                <value>${poolName}</value>
            </property>
            <property>
                <name>inputDir</name>
                <value>${coord:dataIn('input_event')}</value>
            </property>
             <property>
                <name>outputDir</name>
                <value>${coord:dataOut('output_event')}</value>
            </property>
        </configuration>
    </workflow>
</action>
```

</coordinator-app>

oozie workflow sample

```
<workflow-app xmlns="uri:oozie:workflow:0.4" name="shell-wf">
<start to="shell-node"/>
<action name="shell-node">
 <shell xmlns="uri:oozie:shell-action:0.2">
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<configuration>
 <property>
   <name>mapred.job.queue.name</name>
   <value>${poolName}</value>
 </property>
</configuration>
<exec>${myscript}</exec>
<argument>${inputDir}</argument>
<argument>${outputDir}</argument>
<file>${myscriptPath}</file>
<capture-output/>
</shell>
<ok to="end"/>
<error to="fail"/>
</action>
 <kill name="fail">
  <message>Shell action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]
  </message>
</kill>
 <end name="end"/>
</workflow-app>
```

job.properties sample

```
nameNode=hdfs://namenode:port
start=2016-04-12T06:00Z
end=2017-02-26T23:25Z
jobTracker=yourjobtracker
poolName=yourpool
oozie.coord.application.path=${nameNode}/hdfs_path/coord_job_example/coord
workflowAppUri=${oozie.coord.application.path}
myscript=myscript.sh
myscriptPath=${oozie.coord.application.path}/myscript.sh
```

shell script sample

```
inputDir=${1}
outputDir=${2}
hadoop fs -mkdir -p ${outputDir}
hadoop fs -cp ${inputDir}/* ${outputDir}/
```

submitting the coordinator job

Copy the script, coordinator.xml and workflow.xml into HDFS. coordinator.xml must be present in the directory specified by oozie.coord.application.path in job.properties. workflow.xml should be present in the directory specified by workflowAppUri. Once everything is in place, run the below command from shell

```
oozie job -oozie <oozie_url>/oozie/ -config job.properties
```

Read Oozie data triggered coordinator online: https://riptutorial.com/oozie/topic/9845/oozie-data-triggered-coordinator

Credits

S. No	Chapters	Contributors
1	Getting started with oozie	Community, Jyoti Ranjan, YoungHobbit
2	Oozie 101	YoungHobbit
3	Oozie data triggered coordinator	sunitha