Accadgild_Session_10_Assignment_10.1_Solution

Problem Statement

Explain in Brief:

- ✓ The workflow of Oozie and its Benefits
- ✓ The workflow of Sqoop and its Benefits

The workflow of Oozie and its Benefits

Workflow in Oozie is a sequence of actions arranged in a control dependency DAG (Direct Acyclic Graph). The actions are in controlled dependency as the next action can only run as per the output of current action. Subsequent actions are dependent on its previous action. A workflow action can be a Hive action, Pig action, Java action, Shell action, etc. There can be decision trees to decide how and on which condition a job should run.

A fork is used to run multiple jobs in parallel. Oozie workflows can be parameterized (variables like ${nameNode} can be passed within the workflow definition)

Example Workflow

Consider we want to load a data from external hive table to an ORC Hive table.

Step 1 – DDL for Hive external table (say external.hive)

```
Create external table external_table(
   name string,
   age int,
   address string,
   zip int
)
row format delimited
fields terminated by ','
stored as textfile
location '/test/abc';
```

Step 2 – DDL for Hive ORC table (say orc.hive)

```
Create Table orc_table(
   name string, -- Concate value of first name and last name with space as seperator
   yearofbirth int,
   age int, -- Current year minus year of birth
   address string,
   zip int
)
STORED AS ORC
;
```

Step 3 – Hive script to insert data from external table to ORC table (say Copydata.hql)

```
use ${database_name}; -- input from Oozie
insert into table orc_table
select
concat(first_name,' ',last_name) as name,
yearofbirth,
year(from_unixtime) --yearofbirth as age,
address,
zip
from external_table
;
```

Step 4 – Create a workflow to execute all the above three steps. (let's call it workflow.xml)

```
<!-- This is a comment -->

<workflow-app xmlns = "uri:oozie:workflow:0.4" name = "simple-Workflow">

   <start to = "Create_External_Table" />


   <!–Step 1 -->


   <action name = "Create_External_Table">

      <hive xmlns = "uri:oozie:hive-action:0.4">

         <job-tracker>xyz.com:8088</job-tracker>

         <name-node>hdfs://rootname</name-node>

         <script>hdfs_path_of_script/external.hive</script>

      </hive>


      <ok to = "Create_orc_Table" />

      <error to = "kill_job" />

   </action>


   <!–Step 2 -->


   <action name = "Create_orc_Table">
```

```
      <hive xmlns = "uri:oozie:hive-action:0.4">
         <job-tracker>xyz.com:8088</job-tracker>
         <name-node>hdfs://rootname</name-node>
         <script>hdfs_path_of_script/orc.hive</script>
      </hive>

      <ok to = "Insert_into_Table" />
      <error to = "kill_job" />
   </action>

   <!—Step 3 -->

   <action name = "Insert_into_Table">
      <hive xmlns = "uri:oozie:hive-action:0.4">
         <job-tracker>xyz.com:8088</job-tracker>
         <name-node>hdfs://rootname</name-node>
         <script>hdfs_path_of_script/Copydata.hive</script>
         <param>database_name</param>
      </hive>

      <ok to = "end" />
      <error to = "kill_job" />
   </action>

   <kill name = "kill_job">
      <message>Job failed</message>
   </kill>
```

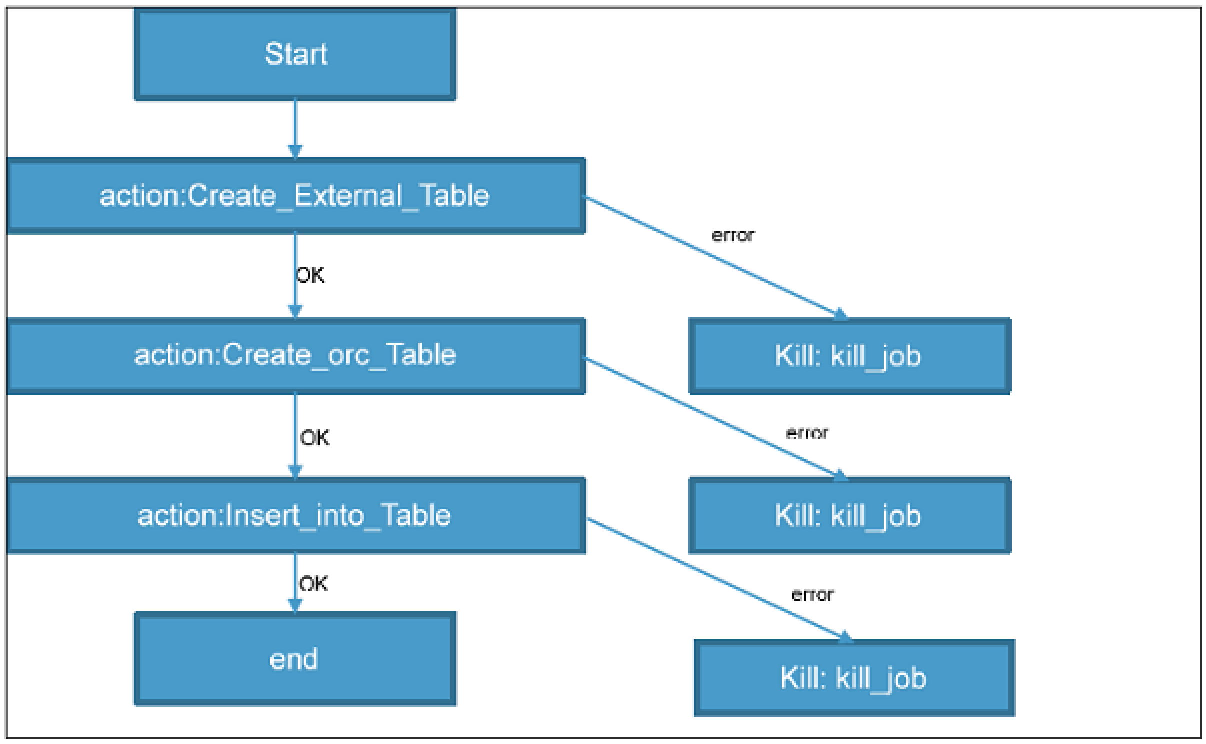
```
    <end name = "end" />



</workflow-app>
```

Explanation of the Above Example

Action Nodes in the above example defines the type of job that the node will run. Hive node inside the action node defines that the action is of type hive. This could also have been a pig, java, shell action, etc. as per the job you want to run.

Each type of action can have its own type of tags. In the above job we are defining the job tracker to us, name node details, script to use and the param entity. The Script tag defines the script we will be running for that hive action. The Param tag defines the values which we will pass into the hive script. (In this example we are passing database name in step 3).

The above workflow will translate into the following DAG.



Running the Workflow

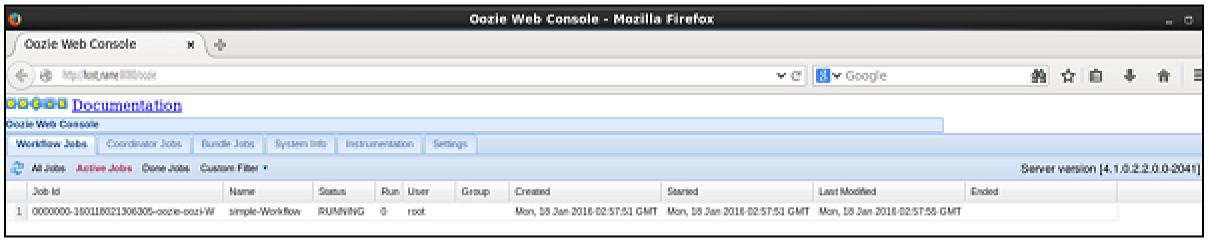
A topology runs in a distributed manner, on multiple worker nodes. Storm spreads the tasks evenly on all the worker nodes. The worker node's role is to listen for jobs and start or stop the processes whenever a new job arrives.

Note – The workflow and hive scripts should be placed in HDFS path before running the workflow.

```
oozie job --oozie http://host_name:8080/oozie -D
oozie.wf.application.path=hdfs://namenodepath/pathof_workflow_xml/workflow.xml-run
```

This will run the workflow once.

To check the status of job you can go to Oozie web console -- http://host_name:8080/



	Job Id	Name	Status	Run	User	Group	Created	Started	Last Modified	Ended
1	0000000-160118021306305-oozie-oozi-W	simple-Workflow	RUNNING	0	root		Mon, 18 Jan 2016 02:57:51 GMT	Mon, 18 Jan 2016 02:57:51 GMT	Mon, 18 Jan 2016 02:57:55 GMT	

By clicking on the job you will see the running job. You can also check the status using Command Line Interface (We will see this later). The possible states for workflow jobs are: PREP, RUNNING, SUSPENDED, SUCCEEDED, KILLED and FAILED.

In the case of an action start failure in a workflow job, depending on the type of failure, Oozie will attempt automatic retries. It will request a manual retry or it will fail the workflow job. Oozie can make HTTP callback notifications on action start/end/failure events and workflow end/failure events. In the case of a workflow job failure, the workflow job can be resubmitted skipping the previously completed actions. Before doing a resubmission the workflow application could be updated with a patch to fix a problem in the workflow application code.

Fork and Join Control Node in Workflow

In scenarios where we want to run multiple jobs parallel to each other, we can use Fork. When fork is used we have to use Join as an end node to fork. Basically Fork and Join work together. For each fork there should be a join. As Join assumes all the node are a child of a single fork.

(We also use fork and join for running multiple independent jobs for proper utilization of cluster).

In our above example, we can create two tables at the same time by running them parallel to each other instead of running them sequentially one after other. Such scenarios perfectly woks for implementing fork.

Let's see how fork is implemented.

Before running the workflow let's drop the tables.

```
Drop table if exist external_table;
Drop table if exist orc_table;
```

Now let’s see the workflow.

```
<workflow-app xmlns = "uri:oozie:workflow:0.4" name = "simple-Workflow">

   <start to = "fork_node" />


   <fork name = "fork_node">

      <path start = "Create_External_Table"/>

      <path start = "Create_orc_Table"/>

   </fork>


   <action name = "Create_External_Table">

      <hive xmlns = "uri:oozie:hive-action:0.4">

         <job-tracker>xyz.com:8088</job-tracker>

         <name-node>hdfs://rootname</name-node>

         <script>hdfs_path_of_script/external.hive</script>

      </hive>


      <ok to = "join_node" />

      <error to = "kill_job" />

   </action>


   <action name = "Create_orc_Table">

      <hive xmlns = "uri:oozie:hive-action:0.4">

         <job-tracker>xyz.com:8088</job-tracker>

         <name-node>hdfs://rootname</name-node>

         <script>hdfs_path_of_script/orc.hive</script>

      </hive>
      <ok to = "join_node" />

      <error to = "kill_job" />

   </action>


   <join name = "join_node" to = "Insert_into_Table"/>


   <action name = "Insert_into_Table">

      <hive xmlns = "uri:oozie:hive-action:0.4">

         <job-tracker>xyz.com:8088</job-tracker>

         <name-node>hdfs://rootname</name-node>

         <script>hdfs_path_of_script/Copydata.hive</script>

         <param>database_name</param>

      </hive>


      <ok to = "end" />

      <error to = "kill_job" />

   </action>


   <kill name = "kill_job">

      <message>Job failed</message>

   </kill>


   <end name = "end" />


</workflow-app>
```

The start node will get to fork and run all the actions mentioned in path for start. All the individual action nodes must go to join node after completion of its task. Until all the actions nodes complete and reach to join node the next action after join is not taken.

Decision Nodes in Workflow

We can add decision tags to check if we want to run an action based on the output of decision. In the above example, if we already have the hive table we won't need to create it again. In such a scenario, we can add a decision tag to not run the Create Table steps if the table already exists. The updated workflow with decision tags will be as shown in the following program.

In this example, we will use an HDFS EL Function fs:exists –

boolean fs:exists(String path)

It returns true or false depending on – if the specified path exists or not.

```
<workflow-app xmlns = "uri:oozie:workflow:0.4" name = "simple-Workflow">

   <start to = "external_table_exists" />


   <decision name = "external_table_exists">

      <switch>

         <case to = "Create_External_Table">${fs:exists('/test/abc') eq 'false'}

            </case>

         <default to = "orc_table_exists" />

      </switch>

   </decision>


   <action name = "Create_External_Table">

      <hive xmlns = "uri:oozie:hive-action:0.4">

         <job-tracker>xyz.com:8088</job-tracker>

         <name-node>hdfs://rootname</name-node>

         <script>hdfs_path_of_script/external.hive</script>

      </hive>
      <ok to = "orc_table_exists" />

      <error to = "kill_job" />

   </action>


   <decision name = "orc_table_exists">

      <switch>

         <case to = "Create_orc_Table">

            ${fs:exists('/apps/hive/warehouse/orc_table') eq 'false'}</case>

         <default to = "Insert_into_Table" />

      </switch>

   </decision>


   <action name = "Create_orc_Table">

      <hive xmlns = "uri:oozie:hive-action:0.4">

         <job-tracker>xyz.com:8088</job-tracker>

         <name-node>hdfs://rootname</name-node>

         <script>hdfs_path_of_script/orc.hive</script>

      </hive>


      <ok to = "Insert_into_Table" />

      <error to = "kill_job" />

   </action>


   <action name = "Insert_into_Table">

      <hive xmlns = "uri:oozie:hive-action:0.4">

         <job-tracker>xyz.com:8088</job-tracker>

         <name-node>hdfs://rootname</name-node>
```

```
            <script>hdfs_path_of_script/Copydata.hive</script>

            <param>database_name</param>

         </hive>


         <ok to = "end" />

         <error to = "kill_job" />

      </action>


      <kill name = "kill_job">

         <message>Job failed</message>

      </kill>


      <end name = "end" />


   </workflow-app>
```

Decision nodes have a switch tag similar to switch case. If the EL translates to success, then that switch case is executed.

This node also has a default tag. In case switch tag is not executed, the control moves to action mentioned in the default tag.

Benefits of Oozie

- ➔ Oozie is designed to scale in a Hadoop cluster. Each job will be launched from a different datanode. This means that the workflow load will be balanced and no single machine will become overburdened by launching workflows. This also means that the capacity to launch workflows will grow as the cluster grows.
- ➔ Oozie is well integrated with Hadoop security. This is especially important in a kerberized cluster. Oozie knows which user submitted the job and will launch all actions as that user, with the proper privileges. It will handle all the authentication details for the user as well.
- ➔ Oozie is the only workflow manager with built-in Hadoop actions, making workflow development, maintenance and troubleshooting easier.

- ➔ Oozie UI makes it easier to drill down to specific errors in the data nodes. Other systems would require significantly more work to correlate jobtracker jobs with the workflow actions.
- ➔ Oozie is proven to scale in some of the world's largest clusters. The white paper discusses a deployment at Yahoo! that can handle 1250 job submissions a minute.
- ➔ Oozie gets callbacks from MapReduce jobs so it knows when they finish and whether they hang without expensive polling. No other workflow manager can do this.
- ➔ Oozie Coordinator allows triggering actions when files arrive at HDFS. This will be challenging to implement anywhere else.
- ➔ Oozie is supported by Hadoop vendors. If there is ever an issue with how the workflow manager integrates with Hadoop – you can turn to the people who wrote the code for answers.

The workflow of Sqoop and its Benefits

Apache Sqoop is a Hadoop tool used for importing and exporting data between relational databases MySQL, Oracle, etc. and Hadoop clusters. Sqoop commands are structured around connecting to and importing or exporting data from various relational databases. It often uses JDBC to talk to these external database systems. Oozie's sqoop action helps users run Sqoop jobs as part of the workflow. The following elements are part of the Sqoop action

1. job-tracker (required)
2. name-node (required)
3. prepare
4. job-xml
5. configuration
6. command (required if arg is not used)
7. arg (required if command is not used)
8. file
9. archive

The action needs to know the JobTracker and the NameNode of the underlying Hadoop cluster where Oozie has to run the sqoop action .The <prepare> section is optional and is typically used as a preprocessor to delete output directories or HCatalog table partitions or to create some directories required for the action. This delete helps make the action repeatable and enables retries after failure.

The <job-xml> element or the <configuration> section can be used to capture all of the Hadoop job configuration properties.For hive action we will be using the <job-xml> tag to pass the hive-site.xml.This way, the hive-site.xml is just reused in its entirety and no additional configuration settings or special files are necessary.

Accadgild_Session_10_Assignment_10.1_Solution

Oozie also supports the <file> and <archive> elements for actions that need them. This is the native, Hadoop way of packaging libraries, archives, scripts, and other data files that jobs need, and Oozie provides the syntax to handle them.

The arguments to Sqoop are sent either through the <command> element in one line or broken down into many <arg> elements.

```
<action name="mySqoopAction">
<sqoop>
<command>import --connect jdbc:hsqldb:file:db.hsqldb --table
test_table --target-dir hdfs://localhost:8020/user/joe/sqoop_tbl -m 1</command>
</sqoop>
</action>
```

Lets look at an example of exporting data from a hive table into the oracle table

1. Oozie workflow xml – workflow.xml

An Oozie workflow is a multistage Hadoop job. A workflow is a collection of action and control nodes arranged in a directed acyclic graph (DAG) that captures control dependency where each action typically is a Hadoop job like a MapReduce, Pig, Hive, Sqoop, or Hadoop DistCp job. There can also be actions that are not Hadoop jobs like a Java application, a shell script, or an email notification. The order of the nodes in the workflow determines the execution order of these actions. An action does not start until the previous action in the workflow ends. Control nodes in a workflow are used to manage the execution flow of actions. The start and end control nodes define the start and end of a workflow. The fork and join control nodes allow executing actions in parallel. The decision control node is like a switch/case statement that can select a particular execution path within the workflow using information from the job itself.

```
<workflow-app name="HDFS_TO_ORACLE" xmlns="uri:oozie:workflow:0.4">
<global>
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
```

```
<configuration>
<property>
<name>mapred.job.queue.name</name>
<value>${queueName}</value>
</property>
</configuration>
</global>

<start to="SQOOP_EXPORT_TO_ORACLE" />

<action name="SQOOP_EXPORT_TO_ORACLE">
<sqoop xmlns="uri:oozie:sqoop-action:0.2">
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<arg>export</arg>
<arg>-Dmapred.child.java.opts=-Xmx4096m</arg>
<arg>--connect</arg>
<arg>${oracle_database}</arg>
<arg>--table</arg>
<arg>${oracle_table}</arg>
<arg>--columns</arg>
<arg>${oracle_columns}</arg>
<arg>--export-dir</arg>
<arg>${RESULTS_FOLDER}</arg>
<arg>--input-fields-terminated-by</arg>
<arg>,</arg>
<arg>--input-lines-terminated-by</arg>
<arg>'\n'</arg>
<arg>--username</arg>
<arg>${oracle_username}</arg>
<arg>--password</arg>
<arg>${oracle_password}</arg>
<arg>--m</arg>
<arg>${num_mappers}</arg>
</sqoop>
<ok to="Email_success" />
<error to="Email_failure" />
</action>
<action name="Email_success">
<email xmlns="uri:oozie:email-action:0.1">
<to>${success_emails}</to>
<subject>SUCCESS:update:${wf:id()}
</subject>
<body>Hi,

Database update is success for
workflow ID : ${wf:id()}

This is
auto-generated email. Please do not
reply to this email.

Thanks,
TimePassTechies.com
```

```
</body>
</email>
<ok to="end" />
<error to="kill" />
</action>
<action name="Email_failure">
<email xmlns="uri:oozie:email-action:0.1">
<to>${failure_emails}</to>
<subject>FAILURE:Database update :
${wf:id()}
</subject>
<body>Hi,
Database update is failed for
workflow ID : ${wf:id()}

This is
auto-generated email. Please do not
reply to this email.

Thanks,
TimePassTechies.com
</body>
</email>
<ok to="end" />
<error to="kill" />
</action>

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

2.Co-ordinator xml file – coordinator.xml

An Oozie coordinator schedules workflow executions based on a start-time and a frequency parameter, and it starts the workflow when all the necessary input data becomes available. If the input data is not available, the workflow execution is delayed until the input data becomes available. A coordinator is defined by a start and end time, a frequency, input and output data, and a workflow. A coordinator runs periodically from the start time until the end time.

Beginning at the start time, the coordinator job checks if the required input data is available. When the input data becomes available, a workflow is started to process the input data, which on completion, produces the corresponding output data. This process is repeated at every tick of the frequency until the end time of the coordinator job. If the input data is not available for a workflow run, the execution of the workflow job will be delayed until the input data becomes available. Normally, both the input and output data used for a workflow execution are aligned with the coordinator time frequency.

```
<coordinator-app
name="${coord_name}"
frequency="${coord:days(1)}"
start="${coord_start_time}"
end="${coord_end_time}"
timezone="IST"
xmlns="uri:oozie:coordinator:0.4"
xmlns:sla="uri:oozie:sla:0.2">
<action>
<workflow>
<app-path>${wf_workflow_path}</app-path>
<configuration>
<property>
<name>wf_exec_datetime</name>
<value>${coord:nominalTime()}</value>
</property>
</configuration>
</workflow>
<sla:info>
<sla:nominal-time>${coord:nominalTime()}</sla:nominal-time>
<sla:should-end>${60 * MINUTES}</sla:should-end>
<sla:alert-events>end_miss</sla:alert-events>
<sla:alert-contact>${wf_notification_email}</sla:alert-contact>
<sla:notification-msg>please check if Sqoop job for ${coord:nominalTime()} is running properly!</sla:notification-msg>
<sla:upstream-apps>${wf_hadoop_instance}</sla:upstream-apps>
</sla:info>
</action>
</coordinator-app>
```

3. coordinator property file to pass configuration – coordinator.properties

Finally the property file where the configuration parameters are passed from.

```
wf_hadoop_instance=HAAS_QUEUE
wf_hadoop_instance_queue=HAAS_QUEUE
nameNode=hdfs://nameservice
jobTracker=yarnRM

oozie.use.system.libpath=true
oozie.coord.application.path=/user/${queueName}/workflows/sqoop_export_oracle
workflowRoot=${nameNode}/user/${queueName}/workflows/sqoop_export_oracle
success_emails=test@google.com
failure_emails=test@google.com

start=2017-02-05T04:00Z
end=2026-01-06T23:30Z

num_mappers=10
RESULTS_FOLDER=${nameNode}/user/${queueName}/sqoop_results_oracle_db

oracle_columns=ID,DEVICE_ID,SUBELEMENT_ID,SERVICE_ID

oracle_database=jdbc:oracle:thin:@host:port/database
oracle_table=TEST.SERVICE_INFO
oracle_username=username
oracle_password=password
```

4. running the coordinator job

```
oozie job -oozie http://oozie_host:port/oozie -dryrun -config coordinator.properties

oozie job -oozie http://oozie_host:port/oozie -config coordinator.properties -run
```

Benefits of Sqoop

- Below are the advantages of Apache Sqoop, which is also the reason for choosing this technology in this layer.
- Allows the transfer of data with a variety of structured data stores like Postgres, Oracle, Teradata, and so on.
- Since the data is transferred and stored in Hadoop, Sqoop allows us to offload certain processing done in the ETL (Extract, Load and Transform) process into low-cost, fast, and effective Hadoop processes.
- Sqoop can execute the data transfer in parallel, so execution can be quick and more cost effective.
- Helps to integrate with sequential data from the mainframe. This helps not only to limit the usage of the mainframe, but also reduces the high cost in executing certain jobs using mainframe hardware.

Accadgild_Session_10_Assignment_10.1_Solution

Submitted By
Shishir Kumar Jha