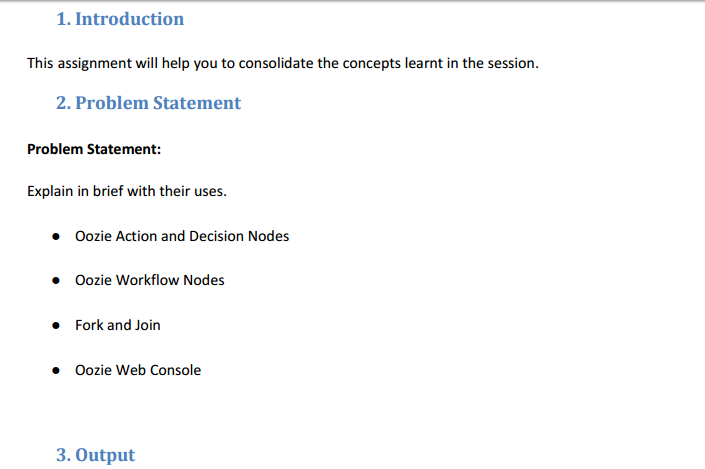
**Assignment 34.3**

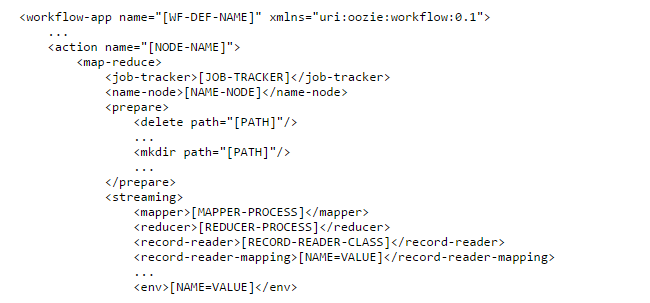


**Oozie action and decision nodes**

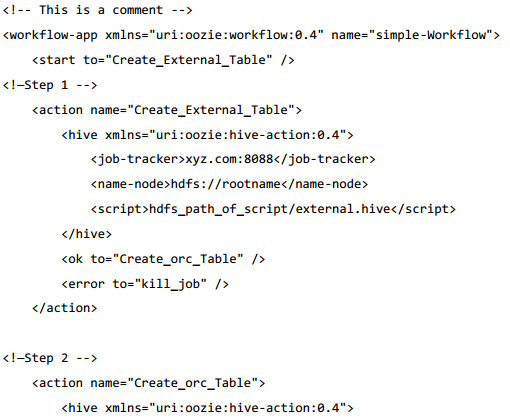
**Action Node**

* Action node - a workflow task.
* For example: moving files into HDFS, running MapReduce job, Pig or[Hive](http://www.guru99.com/hive-tutorials.html)job, importing data by using Sqoop or else running a shell script for programs which were written in Java.

**Example of map reduce action**

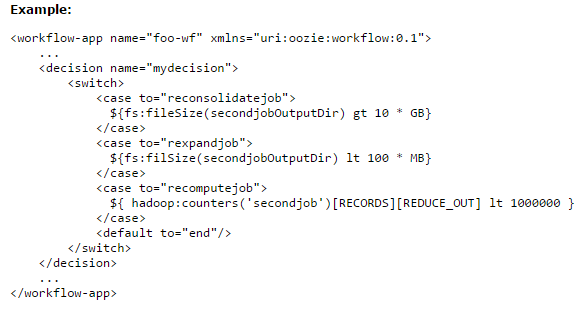


**Example of Hive Action node**

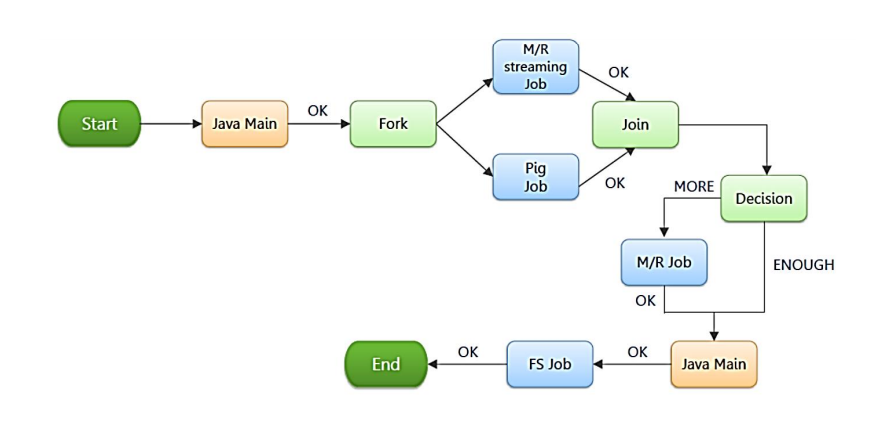


***Decision Node***

* In decision node they will add decision tag to check when one needs to run a action which is based on output of other decision.
* For example - hive table is already present so there will not be any need to create the table again.
* In such case, one need to add decision tag not run the Create Table steps if the table is already present.
* Behavior of each decision node will be made as switch-case statement.
* Decision node will consists a list of predicates-transition pairs adding to default transition.
* Predicates will be evaluated according to order or appearance.
* When one of them gets evaluated as true then the transition will take place.
* When no predicates evaluated as true then by default the transition will take place.



**Oozie workflow nodes**



Oozie workflow nodes will contain 2 nodes –

**Control Flow Nodes**

* Control nodes are used for defining a job chronology.
* It will provide rules for starting and ending the workflow
* Controlling workflow execution path will happen with possible decision points known as *fork and join nodes.*

*Control node achieves this by using 3 nodes*

* 1. **Start/end/kill**
  2. **Decision**
  3. **Fork/join**

**Start Control Node**

* Start node – acts as entry point for workflow job.
* Once workflow gets started will automatically gets translated to node which is specified in start .
* Workflow – will have at least one start node.

Syntax:

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<start to="[NODE-NAME]"/>

...

</workflow-app>

The to attribute is the name of first workflow node to execute.

Example:

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<start to="firstHadoopJob"/>

...

</workflow-app>

***Kill control Node***

* Kill node will allow workflow job to by kill itself.
* Once workflow job reach kill it will end up in error (KILLED).
* When more than one action gets started by the workflow job which were executing while kill node is reached then the actions will get killed.
* Workflow - may have more number of kill nodes.

Syntax:

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<kill name="[NODE-NAME]">

<message>[MESSAGE-TO-LOG]</message>

</kill>

...

</workflow-app>

* Name attribute present in kill node was given with the name of the Kill action node.
* Content of message element will be taken as the kill reason for all the workflow job.
* Kill node will not have any transition elements.
* Since it will ends workflow job, as KILLED .

**Example:**

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<kill name="killBecauseNoInput">

<message>Input unavailable</message>

</kill>

...

</workflow-app>

***End Control Node***

* End node acts as end for all workflow job.
* Which means that the workflow job have been completed successfully.
* Once the workflow job goes up to end will get finished successfully successfully (SUCCEEDED).
* When more actions gets started by workflow job while being executed .
* End node actions will be killed.
* In such case workflow job will be considered as run successfully.
* workflow – will have at least one end node.

Syntax:

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<end name="[NODE-NAME]"/>

...

</workflow-app>

* Name attribute will be of same name of transition in order to end the workflow job.

Example:

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<end name="end"/>

</workflow-app>

**Decision Control Node**

Decision node – will enables the workflow to select the execution path.

Decision node will act in such a way by which it can seen as switch-case statement.

It will have a list of predicates-transition pairs as well default transition.

Predicate will be evaluated in such a way that appearance until one of them were evaluated as true .

When at least one predicate is evaluated as true then the default transition will take place.

Predicates are JSP Expression Language (EL) expressions .

To resolve that into boolean value (true or false ).

For example:

${fs:fileSize('/usr/foo/myinputdir') gt 10 \* GB}

Syntax:

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<decision name="[NODE-NAME]">

<switch>

<case to="[NODE\_NAME]">[PREDICATE]</case>

...

<case to="[NODE\_NAME]">[PREDICATE]</case>

<default to="[NODE\_NAME]"/>

</switch>

</decision>

...

</workflow-app>

Name attribute which is in the decision node will be the name of the decision node.

Each case element will contain a predicate as transition name.

Predicate EL will be evaluated in such a order that until one is returned as true .

Default element will indicate that the transition take place and to take whether none of predicates were evaluated to true .

All the decision node will have a default element in order to avoid workflow getting into error if no predicate is evaluated as true.

**Example:**

<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">

...

<decision name="mydecision">

<switch>

<case to="reconsolidatejob">

${fs:fileSize(secondjobOutputDir) gt 10 \* GB}

</case>

<case to="rexpandjob">

${fs:filSize(secondjobOutputDir) lt 100 \* MB}

</case>

<case to="recomputejob">

${ hadoop:counters('secondjob')[RECORDS][REDUCE\_OUT] lt 1000000 }

</case>

<default to="end"/>

</switch>

</decision>

...

</workflow-app>

***Fork and Join Control Nodes***

A fork node splits one path of execution into multiple concurrent paths of execution.

A join node waits until every concurrent execution path of a previous fork node arrives to it.

The fork and join nodes must be used in pairs. The join node assumes concurrent execution paths are children of the same fork node.

Syntax:

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<fork name="[FORK-NODE-NAME]">

<path start="[NODE-NAME]" />

...

<path start="[NODE-NAME]" />

</fork>

...

<join name="[JOIN-NODE-NAME]" to="[NODE-NAME]" />

...

</workflow-app>

* Name attribute in fork node is named after workflow fork node.
* Start attribute present in the path elements in fork node will indicate name of workflow node which acts as a part of concurrent execution paths.
* Name attribute in join node is named after workflow join node.
* to attribute in join node will indicate name of workflow node which is executed after concurrent execution paths of join node.

Example:

<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">

...

<fork name="forking">

<path start="firstparalleljob"/>

<path start="secondparalleljob"/>

</fork>

<action name="firstparallejob">

<map-reduce>

<job-tracker>foo:9001</job-tracker>

<name-node>bar:9000</name-node>

<job-xml>job1.xml</job-xml>

</map-reduce>

<ok to="joining"/>

<error to="kill"/>

</action>

<action name="secondparalleljob">

<map-reduce>

<job-tracker>foo:9001</job-tracker>

<name-node>bar:9000</name-node>

<job-xml>job2.xml</job-xml>

</map-reduce>

<ok to="joining"/>

<error to="kill"/>

</action>

<join name="joining" to="nextaction"/>

...

</workflow-app>

***Action Nodes***

* Action node - represents workflow task
* For example - moving files into HDFS, running of MapReduce job, Pig or[Hive](http://www.guru99.com/hive-tutorials.html) jobs, importing data using Sqoop or running a shell script of program which were written in Java.

• Map-reduce

• Pig

• Hdfs

• Sub-workflow

• Java-run custom java code

In order to run oozie workflows we are in need of 2 files. :

**1. workflow.xml**

**2. job.properties**

*1. workflow.xml (stored in HDFS)*

• workflow.xml willcontain the structure of workflow.

*2. job.properties (stored in local)*

•job.properties will contains the configuration properties.

**Fork and join Control Nodes**

* Fork node will split one path of execution into multiple paths.
* Join node will wait until all the concurrent execution path of previous fork node reach them.
* Fork and join nodes should be always used in pairs.
* Join node will assume all the concurrent execution paths as children of fork node.

Syntax:

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">

...

<fork name="[FORK-NODE-NAME]">

<path start="[NODE-NAME]" />

...

<path start="[NODE-NAME]" />

</fork>

...

<join name="[JOIN-NODE-NAME]" to="[NODE-NAME]" />

...

</workflow-app>

* Name attribute will have the same name as workflow fork node.
* Start attribute path elements in fork node will have same name as workflow node .
* Which acts as a part of concurrent execution paths.
* Name attribute will join the node which has the name of workflow join node.
* to attribute in join node will indicate the name of workflow node.
* Its get executed only after concurrent execution paths .

**Example:**

<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">

...

<fork name="forking">

<path start="firstparalleljob"/>

<path start="secondparalleljob"/>

</fork>

<action name="firstparallejob">

<map-reduce>

<job-tracker>foo:9001</job-tracker>

<name-node>bar:9000</name-node>

<job-xml>job1.xml</job-xml>

</map-reduce>

<ok to="joining"/>

<error to="kill"/>

</action>

<action name="secondparalleljob">

<map-reduce>

<job-tracker>foo:9001</job-tracker>

<name-node>bar:9000</name-node>

<job-xml>job2.xml</job-xml>

</map-reduce>

<ok to="joining"/>

<error to="kill"/>

</action>

<join name="joining" to="nextaction"/>

...

</workflow-app>

*Oozie web console*

Oozie will provide read-only job.

Web based console will allow to see the syatem status of monitor Oozie all the workflow application status as well workflow jobs status.

Web base console gets implemented as client of Web Services API.

To enable the Oozie's web console which works with Cloudera Manager.

**Oozie Web Console to get enabled:**

1. We need to Download the [**ext-2.2**](http://extjs.com/deploy/ext-2.2.zip).
2. Then to extract those contents of file to the loacation /var/lib/oozie/libext.
3. We need to connect the Cloudera Manager Admin Console.
4. Then Click **Configuration** tab of the Oozie service
5. Later to enable **Enable Oozie server web console.**
6. Oozie user will have read/write permissions to the directory /var/lib/oozie/oozie-server/webapps.

For example

1. $ ls -l /var/lib/oozie/oozie-server/
2. total 12
3. drwxr-xr-x 3 oozie oozie 4096 Jul 13 05:42 conf
4. dr--r--r-- 4 oozie oozie 4096 Jul 13 13:15 webapps

drwxr-xr-x 3 oozie oozie 4096 Jul 13 05:42 work

1. Permissions should be set appropriately and should continue with next step.
2. Once permissions is modified,it can be done as:
3. $ chmod 755 /var/lib/oozie/oozie-server/webapps
4. $ ls -l /var/lib/oozie/oozie-server/
5. drwxr-xr-x 3 oozie oozie 4096 Jul 13 05:42 conf
6. drwxr-xr-x 4 oozie oozie 4096 Jul 13 13:15 webapps

drwxr-xr-x 3 oozie oozie 4096 Jul 13 05:42 work