Explain in brief with their uses.

● Oozie Action and Decision Nodes

#### **Action Nodes:**

Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task.

##### *Action Basis*

The following sub-sections define common behavior and capabilities for all action types.

###### Action Computation/Processing Is Always Remote

All computation/processing tasks triggered by an action node are remote to Oozie. No workflow application specific computation/processing task is executed within Oozie.

###### *Actions Are Asynchronous*

All computation/processing tasks triggered by an action node are executed asynchronously by Oozie. For most types of computation/processing tasks triggered by workflow action, the workflow job has to wait until the computation/processing task completes before transitioning to the following node in the workflow.

The exception is the fs action that is handled as a synchronous action.

Oozie can detect completion of computation/processing tasks by two different means, callbacks and polling.

When a computation/processing tasks is started by Oozie, Oozie provides a unique callback URL to the task, the task should invoke the given URL to notify its completion.

For cases that the task failed to invoke the callback URL for any reason (i.e. a transient network failure) or when the type of task cannot invoke the callback URL upon completion, Oozie has a mechanism to poll computation/processing tasks for completion.

###### Actions Have 2 Transitions, =ok= and =error=

If a computation/processing task -triggered by a workflow- completes successfully, it transitions to ok .

If a computation/processing task -triggered by a workflow- fails to complete successfully, its transitions to error .

If a computation/processing task exits in error, there computation/processing task must provide error-code and error-message information to Oozie. This information can be used from decision nodes to implement a fine grain error handling at workflow application level.

Each action type must clearly define all the error codes it can produce.

###### *Action Recovery*

Oozie provides recovery capabilities when starting or ending actions.

Once an action starts successfully Oozie will not retry starting the action if the action fails during its execution. The assumption is that the external system (i.e. Hadoop) executing the action has enough resilience to recovery jobs once it has started (i.e. Hadoop task retries).

Depending on the nature of the failure, Oozie will have different recovery strategies.

If the failure is of transient nature, Oozie will perform retries after a pre-defined time interval. The number of retries and timer interval for a type of action must be pre-configured at Oozie level. Workflow jobs can override such configuration.

Examples of a transient failures are network problems or a remote system temporary unavailable.

If the failure is of non-transient nature, Oozie will suspend the workflow job until an manual or programmatic intervention resumes the workflow job and the action start or end is retried. It is the responsibility of an administrator or an external managing system to perform any necessary cleanup before resuming the workflow job.

If the failure is an error and a retry will not resolve the problem, Oozie will perform the error transition for the action.

**Decision Control Node**

A decision node enables a workflow to make a selection on the execution path to follow.

The behavior of a decision node can be seen as a switch-case statement.

A decision node consists of a list of predicates-transition pairs plus a default transition. Predicates are evaluated in order or appearance until one of them evaluates to true and the corresponding transition is taken. If none of the predicates evaluates to true the default transition is taken.

Predicates are JSP Expression Language (EL) expressions (refer to section 4.2 of this document) that resolve into a 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>

The name attribute in the decision node is the name of the decision node.

Each case elements contains a predicate an a transition name. The predicate ELs are evaluated in order until one returns true and the corresponding transition is taken.

The default element indicates the transition to take if none of the predicates evaluates to true .

All decision nodes must have a default element to avoid bringing the workflow into an error state if none of the predicates evaluates to 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>

**● Oozie Workflow Nodes**

A workflow job can have be in any of the following states:

PREP: When a workflow job is first create it will be in PREP state. The workflow job is defined but it is not running.

RUNNING: When a CREATED workflow job is started it goes into RUNNING state, it will remain in RUNNING state while it does not reach its end state, ends in error or it is suspended.

SUSPENDED: A RUNNING workflow job can be suspended, it will remain in SUSPENDED state until the workflow job is resumed or it is killed.

SUCCEEDED: When a RUNNING workflow job reaches the end node it ends reaching the SUCCEEDED final state.

KILLED: When a CREATED , RUNNING or SUSPENDED workflow job is killed by an administrator or the owner via a request to Oozie the workflow job ends reaching the KILLEDfinal state.

FAILED: When a RUNNING workflow job fails due to an unexpected error it ends reaching the FAILED final state.

**Workflow job state valid transitions:**

* --> PREP
* PREP --> RUNNING | KILLED
* RUNNING --> SUSPENDED | SUCCEEDED | KILLED | FAILED
* SUSPENDED --> RUNNING | KILLED

**Control Flow Nodes**

Control flow nodes define the beginning and the end of a workflow (the start , end and kill nodes) and provide a mechanism to control the workflow execution path (the decision , fork and join nodes).

**Start Control Node**

The start node is the entry point for a workflow job, it indicates the first workflow node the workflow job must transition to.

When a workflow is started, it automatically transitions to the node specified in the start .

A workflow definition must have 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>

**End Control Node**

The end node is the end for a workflow job, it indicates that the workflow job has completed successfully.

When a workflow job reaches the end it finishes successfully (SUCCEEDED).

If one or more actions started by the workflow job are executing when the end node is reached, the actions will be killed. In this scenario the workflow job is still considered as successfully run.

A workflow definition must have one end node.

**Syntax:**

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

...

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

...

</workflow-app>

The name attribute is the name of the transition to do to end the workflow job.

**Example:**

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

...

<end name="end"/>

</workflow-app>

**Kill Control Node**

The kill node allows a workflow job to kill itself.

When a workflow job reaches the kill it finishes in error (KILLED).

If one or more actions started by the workflow job are executing when the kill node is reached, the actions will be killed.

A workflow definition may have zero or more 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>

The name attribute in the kill node is the name of the Kill action node.

The content of the message element will be logged as the kill reason for the workflow job.

A kill node does not have transition elements because it ends the 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>

**Decision Control Node**

A decision node enables a workflow to make a selection on the execution path to follow.

The behavior of a decision node can be seen as a switch-case statement.

A decision node consists of a list of predicates-transition pairs plus a default transition. Predicates are evaluated in order or appearance until one of them evaluates to true and the corresponding transition is taken. If none of the predicates evaluates to true the default transition is taken.

Predicates are JSP Expression Language (EL) expressions (refer to section 4.2 of this document) that resolve into a 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>

The name attribute in the decision node is the name of the decision node.

Each case elements contains a predicate an a transition name. The predicate ELs are evaluated in order until one returns true and the corresponding transition is taken.

The default element indicates the transition to take if none of the predicates evaluates to true .

All decision nodes must have a default element to avoid bringing the workflow into an error state if none of the predicates evaluates to 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**

**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>

The name attribute in the fork node is the name of the workflow fork node. The start attribute in the path elements in the fork node indicate the name of the workflow node that will be part of the concurrent execution paths.

The name attribute in the join node is the name of the workflow join node. The to attribute in the join node indicates the name of the workflow node that will executed after all concurrent execution paths of the corresponding fork arrive to the 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>

**● Oozie Web Console**

Oozie provides a read-only Web based console that allows to allow to monitor Oozie system status, workflow applications status and workflow jobs status. The Web base console is implemented as a client of the Web Services API.

Enabling the Oozie Web Console

To enable Oozie's web console, you must download and add the ExtJS library to the Oozie server. If you have not already done this, proceed as follows.

Step 1: Download the Library

Download the ExtJS version 2.2 library from http://archive.cloudera.com/gplextras/misc/ext-2.2.zip and place it a convenient location.

Step 2: Install the Library

Extract the ext-2.2.zip file into /var/lib/oozie.

$ cd Downloads/

$ sudo cp -avr ext-2.2 /var/lib/oozie/