**Explain in brief with their uses.**

**● Oozie Action and Decision Nodes**

**Decision 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.

#### **Action Nodes**

Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task. All computation/processing tasks triggered by an action node are remote to Oozie. No workflow application specific computation/processing task is executed within Oozie. 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.

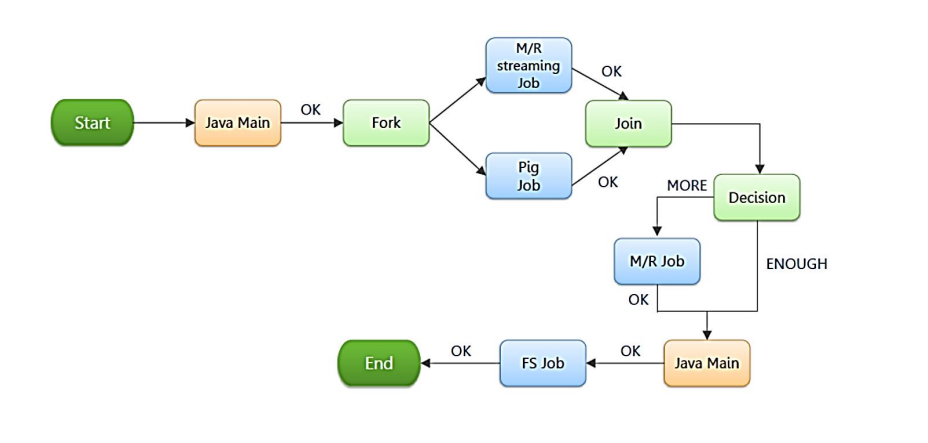
When a computation/processing tasks is started by Oozie, Oozie provides a unique callback URL to the task, the task should invoke the given URL to notify its completion.

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.

**● Oozie Workflow Nodes**

Workflow nodes are classified in control flow nodes and action nodes:

* **Control flow nodes:** nodes that control the start and end of the workflow and workflow job execution path.
* **Action nodes:** nodes that trigger the execution of a computation/processing task.



* **Control Flow nodes**
  + - **Start/end/kill**

**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.

**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.

**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.

* + **Decision**

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.

* + **Fork/join**

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.

* **Actions nodes**

**Map-Reduce Action**

The map-reduce action starts a Hadoop map/reduce job from a workflow. Hadoop jobs can be Java Map/Reduce jobs or streaming jobs.

A map-reduce action can be configured to perform file system cleanup and directory creation before starting the map reduce job. This capability enables Oozie to retry a Hadoop job in the situation of a transient failure (Hadoop checks the non-existence of the job output directory and then creates it when the Hadoop job is starting, thus a retry without cleanup of the job output directory would fail).

The workflow job will wait until the Hadoop map/reduce job completes before continuing to the next action in the workflow execution path.

The counters of the Hadoop job and job exit status (=FAILED=, KILLED or SUCCEEDED) must be available to the workflow job after the Hadoop jobs ends. This information can be used from within decision nodes and other actions configurations.

**Pig Action**

The pig action starts a Pig job.

The workflow job will wait until the pig job completes before continuing to the next action.

The pig action has to be configured with the job-tracker, name-node, pig script and the necessary parameters and configuration to run the Pig job.

A pig action can be configured to perform HDFS files/directories cleanup before starting the Pig job. This capability enables Oozie to retry a Pig job in the situation of a transient failure (Pig creates temporary directories for intermediate data, thus a retry without cleanup would fail).

Hadoop JobConf properties can be specified in a JobConf XML file bundled with the workflow application or they can be indicated inline in the pig action configuration.

The configuration properties are loaded in the following order, job-xml and configuration , and later values override earlier values.

**Fs (HDFS) action**

The fs action allows to manipulate files and directories in HDFS from a workflow application. The supported commands are move , delete and mkdir .

The FS commands are executed synchronously from within the FS action, the workflow job will wait until the specified file commands are completed before continuing to the next action.

Path names specified in the fs action can be parameterized (templatized) using EL expressions.

Each file path must specify the file system URI, for move operations, the target must not specified the system URI.

**Sub-workflow Action**

The sub-workflow action runs a child workflow job, the child workflow job can be in the same Oozie system or in another Oozie system.

The parent workflow job will wait until the child workflow job has completed.

**Java Action**

The java action will execute the public static void main(String[] args) method of the specified main Java class.

Java applications are executed in the Hadoop cluster as map-reduce job with a single Mapper task.

The workflow job will wait until the java application completes its execution before continuing to the next action.

The java action has to be configured with the job-tracker, name-node, main Java class, JVM options and arguments.

To indicate an ok action transition, the main Java class must complete gracefully the main method invocation.

● **Fork and Join**

**Fork and Join 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**

Web Console is a web-based application that allows to execute shell commands on a server directly from a browser (web-based SSH).

The application is very light, does not require any database and can be installed and configured in about 3 minutes. Installation process is really simple.

# Enabling the Oozie Web Console

* [Enabling the Oozie Web Console Using Cloudera Manager](https://www.cloudera.com/documentation/enterprise/5-5-x/topics/admin_oozie_console.html#concept_rxk_35t_2r)
* [Enabling the Oozie Web Console Using the Command Line](https://www.cloudera.com/documentation/enterprise/5-5-x/topics/admin_oozie_console.html#concept_fl3_35t_2r)

## **Enabling the Oozie Web Console Using Cloudera Manager**

1. Download [ext-2.2](http://tiny.cloudera.com/oozie-ext-2.2). Extract the contents of the file to /var/lib/oozie/ on the same host as the Oozie Server.
2. In the Cloudera Manager Admin Console, go to the Oozie service.
3. Click the Configuration tab.
4. Locate the Enable Oozie server web console property or search for it by typing its name in the Search box.
5. Select Enable Oozie server web console.
6. Click Save Changes to commit the changes.
7. Restart the Oozie service.

[**Enabling the Oozie Web Console Using the Command Line**](https://www.cloudera.com/documentation/enterprise/5-5-x/topics/admin_oozie_console.html#concept_fl3_35t_2r)

To enable the Oozie web console, download and add the ExtJS library to the Oozie server.

#### **Step 1: Download the Library**

Download the ExtJS version 2.2 library from <https://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.

#### **Step 3: Configure SPNEGO authentication (in Kerberos clusters only)**

The web console shares a port with the Oozie REST API, and the API allows modifications of Oozie jobs (kill, submission, and inspection). SPNEGO authentication ensures that the Kerberos realm trusts the client browser credentials and that configuration of the client web browser passes these credentials. If this configuration is not possible, use the Hue Oozie Dashboard instead of the Oozie Web Console.