

**Solution-**

1. **Apache Oozie Action and Decision Nodes-**

**Actions Nodes-**

1. **Map-reduce** - 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).
2. **Pig -** 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.
3. **Hdfs -** 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.
4. **Sub-workflow -** 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.
5. **Java-run custom java code-** 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.

**Decision Nodes-**

1. **Start/end/kill** - 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- 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).

Kill- 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.

1. **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.
2. **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.

1. **Oozie Workflow Nodes-**

**Control Flow**

* Start/end/kill
* Decision
* Fork/join Actions
* Map-reduce
* Pig
* Hdfs
* Sub-workflow
* Java-run custom java code

**Oozie Workflow Nodes**

To run oozie workflows, two files are needed.

1. workflow.xml (stored in HDFS)

• It contains the structure of workflow.

1. job.properties (stored in local)

• It contains the configuration properties.

1. **Fork and 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.

**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.

1. **Oozie Web Console-**

The Oozie Web Services API is a HTTP REST JSON API.

All responses are in UTF-8 .

Assuming Oozie is runing at OOZIE\_URL , the following web services end points are supported:

* /versions
* /v0/admin
* /v0/job
* /v0/jobs