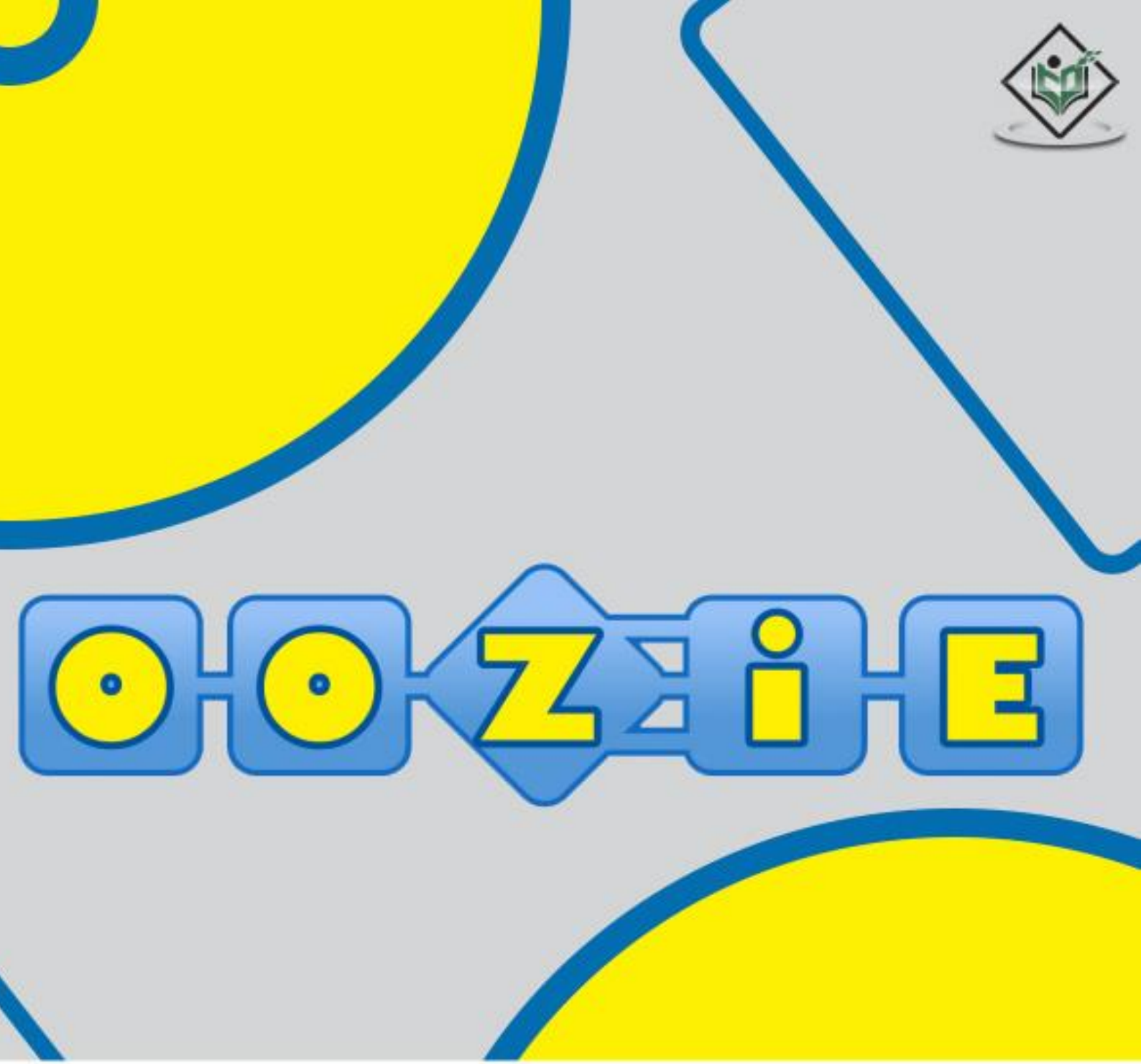


tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Apache Oozie is the tool in which all sort of programs can be pipelined in a desired order to work in Hadoop's distributed environment. Oozie also provides a mechanism to run the job at a given schedule.

This tutorial explains the scheduler system to run and manage Hadoop jobs called Apache Oozie. It is tightly integrated with Hadoop stack supporting various Hadoop jobs like Hive, Pig, Sqoop, as well as system specific jobs like Java and Shell.

This tutorial explores the fundamentals of Apache Oozie like workflow, coordinator, bundle and property file along with some examples. By the end of this tutorial, you will have enough understanding on scheduling and running Oozie jobs on Hadoop cluster in a distributed environment.

Audience

This tutorial has been prepared for professionals working with Big Data Analytics and want to understand about scheduling complex Hadoop jobs using Apache Oozie.

This tutorial is intended to make you comfortable in getting started with Oozie and does not detail each and every function available. For these details, Oozie documentation is the best place to visit.

Prerequisites

Before proceeding with this tutorial, you must have a conceptual understanding of Cron jobs and schedulers.

Copyright and Disclaimer

Table of Contents

1. Apache Oozie – Introduction

In this chapter, we will start with the fundamentals of Apache Oozie. Following is a detailed explanation about Oozie along with a few examples and screenshots for better understanding.

What is Apache Oozie?

Apache Oozie is a scheduler system to run and **manage Hadoop jobs** in a distributed environment. It allows to combine multiple complex jobs to be run in a sequential order to achieve a bigger task. Within a sequence of task, two or more jobs can also be programmed to run parallel to each other.

One of the main advantages of Oozie is that it is tightly integrated with Hadoop stack supporting various Hadoop jobs like **Hive, Pig, Sqoop** as well as system-specific jobs like **Java and Shell**.

Oozie is an **Open Source Java Web-Application** available under Apache license 2.0. It is responsible for triggering the workflow actions, which in turn uses the Hadoop execution engine to actually execute the task. Hence, Oozie is able to leverage the existing Hadoop machinery for load balancing, fail-over, etc.

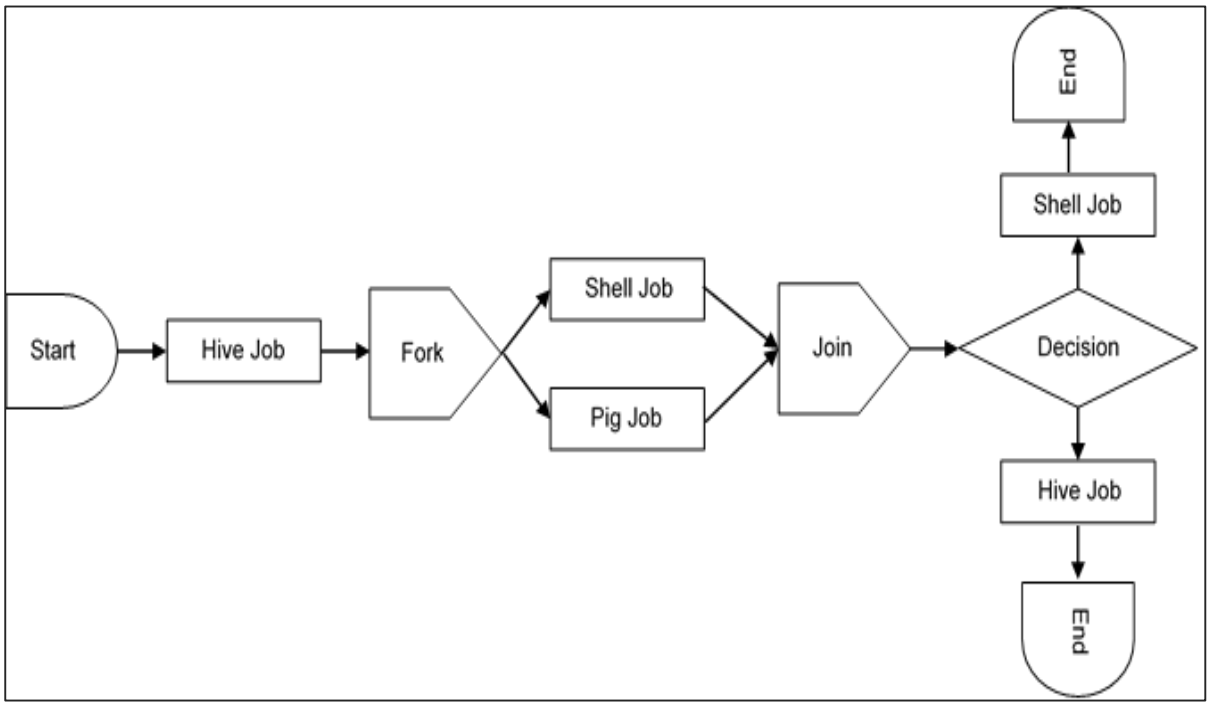
Oozie detects completion of tasks through callback and polling. When Oozie starts a task, it provides a unique **callback HTTP URL** to the task, and notifies that URL when it is complete. If the task fails to invoke the callback URL, Oozie can poll the task for completion.

Following three types of jobs are common in Oozie:

- **Oozie Workflow Jobs** — These are represented as Directed Acyclic Graphs (DAGs) to specify a sequence of actions to be executed.
- **Oozie Coordinator Jobs** — These consist of workflow jobs triggered by time and data availability.
- **Oozie Bundle** — These can be referred to as a package of multiple coordinator and workflow jobs.

We will look into each of these in detail in the following chapters.

A sample workflow with Controls (Start, Decision, Fork, Join and End) and Actions (Hive, Shell, Pig) will look like the following diagram:



Workflow will always start with a Start tag and end with an End tag.

Use-Cases of Apache Oozie

Apache Oozie is used by Hadoop system administrators to run complex log analysis on **HDFS**. Hadoop Developers use Oozie for performing ETL operations on data in a sequential order and saving the output in a specified format (Avro, ORC, etc.) in HDFS.

In an enterprise, Oozie jobs are scheduled as coordinators or bundles.

Oozie Editors

Before we dive into Oozie lets have a quick look at the available editors for Oozie.

Most of the time, you won't need an editor and will write the workflows using any popular text editors (like Notepad++, Sublime or Atom) as we will be doing in this tutorial.

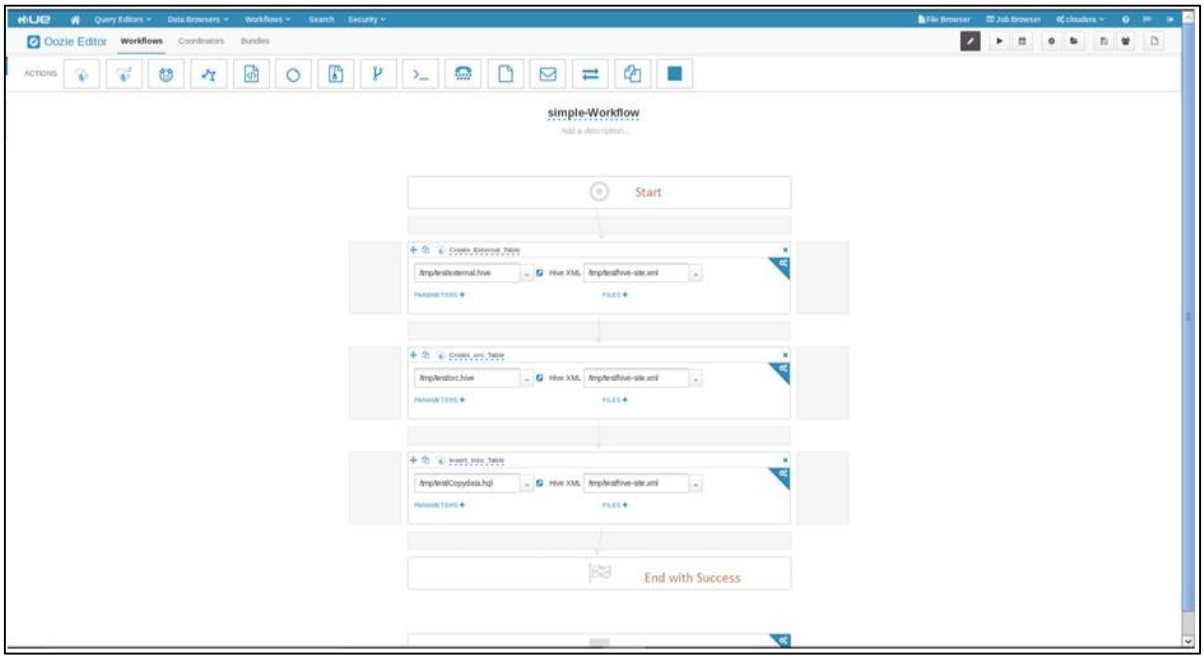
But as a beginner it makes some sense to create a workflow by the drag and drop method using the editor and then see how the workflow gets generated. Also, to map **GUI** with the actual **workflow.xml** created by the editor. This is the only section where we will discuss about Oozie editors and won't use it in our tutorial.

The most popular among Oozie editors is **Hue**.

Hue Editor for Oozie

This editor is very handy to use and is available with almost all Hadoop vendors' solutions.

The following screenshot shows an example workflow created by this editor.



You can drag and drop controls and actions and add your job inside these actions.

A good resource to learn more on this topic:

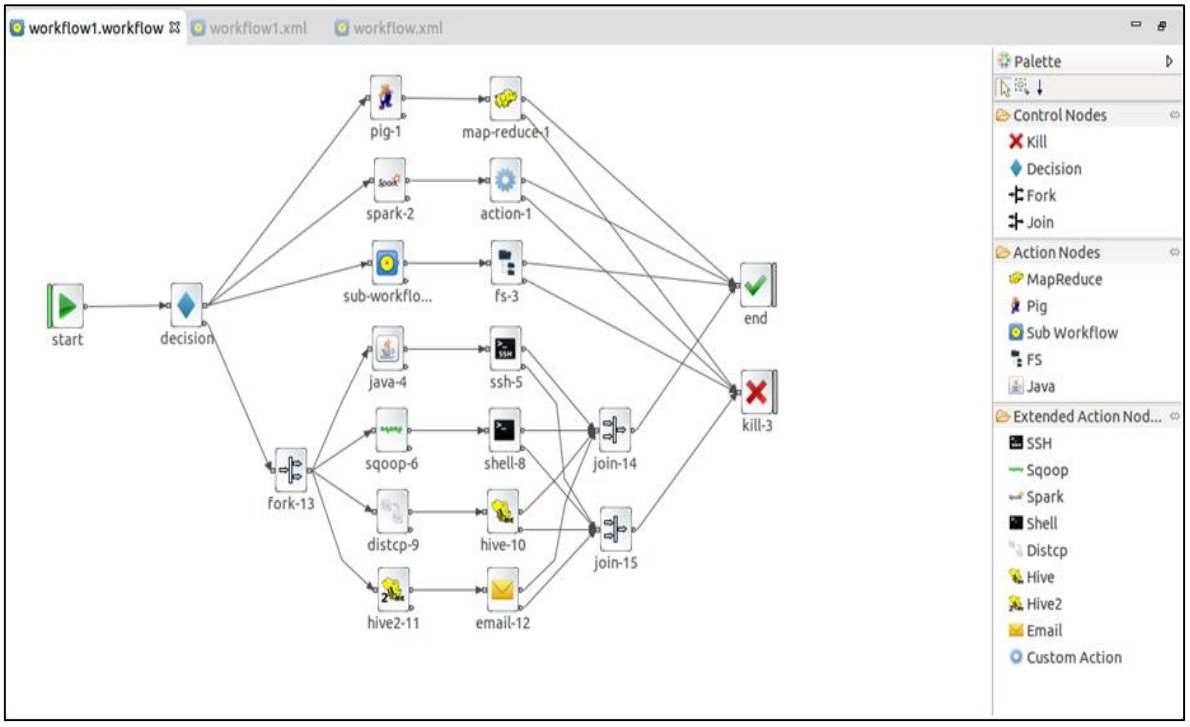
http://gethue.com/new-apache-oozie-workflow-coordinator-bundle-editors/

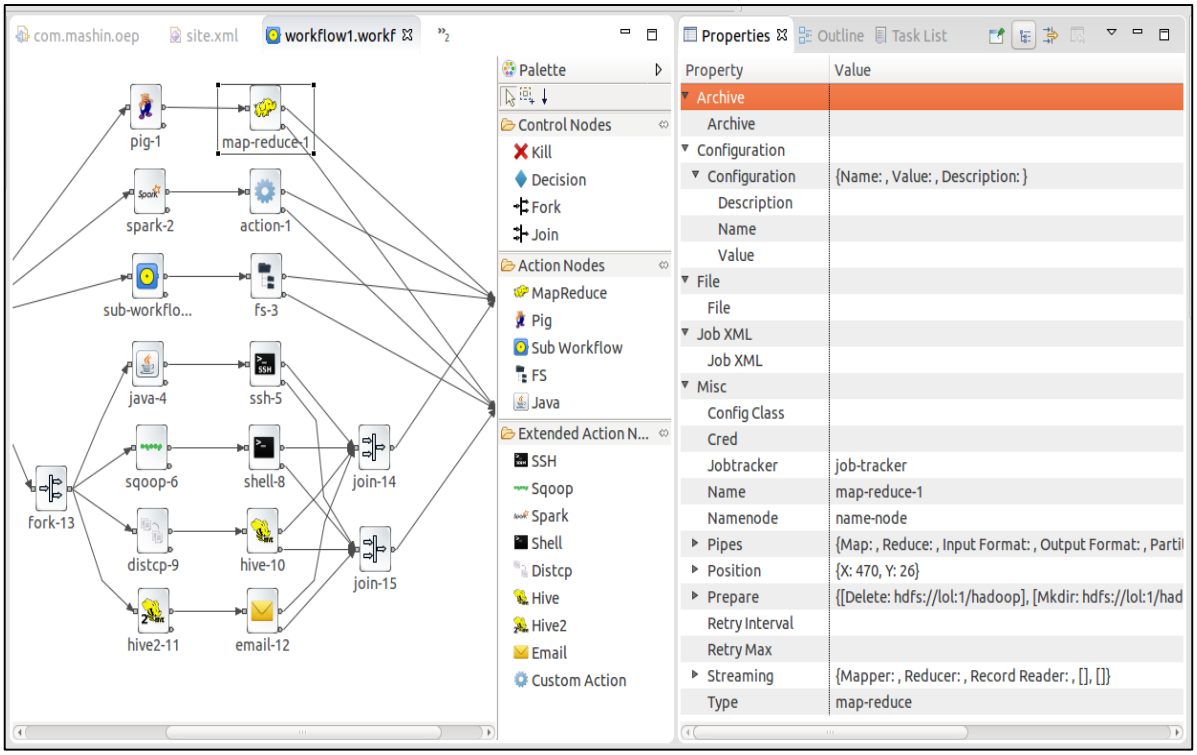
Oozie Eclipse Plugin (OEP)

Oozie Eclipse plugin (OEP) is an Eclipse plugin for editing Apache Oozie workflows graphically. It is a graphical editor for editing Apache Oozie workflows inside Eclipse.

Composing Apache Oozie workflows is becoming much simpler. It becomes a matter of drag-and-drop, a matter of connecting lines between the nodes.

The following screenshots are examples of OEP.





To learn more on OEP, you can visit http://oep.mashin.io/

Now let's go to our next lesson and start writing Oozie workflow.

2. Apache Oozie – Workflow

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). These parameters come from a configuration file called as property file. (More on this explained in the following chapters).

Let's learn by creating some examples.

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">
```

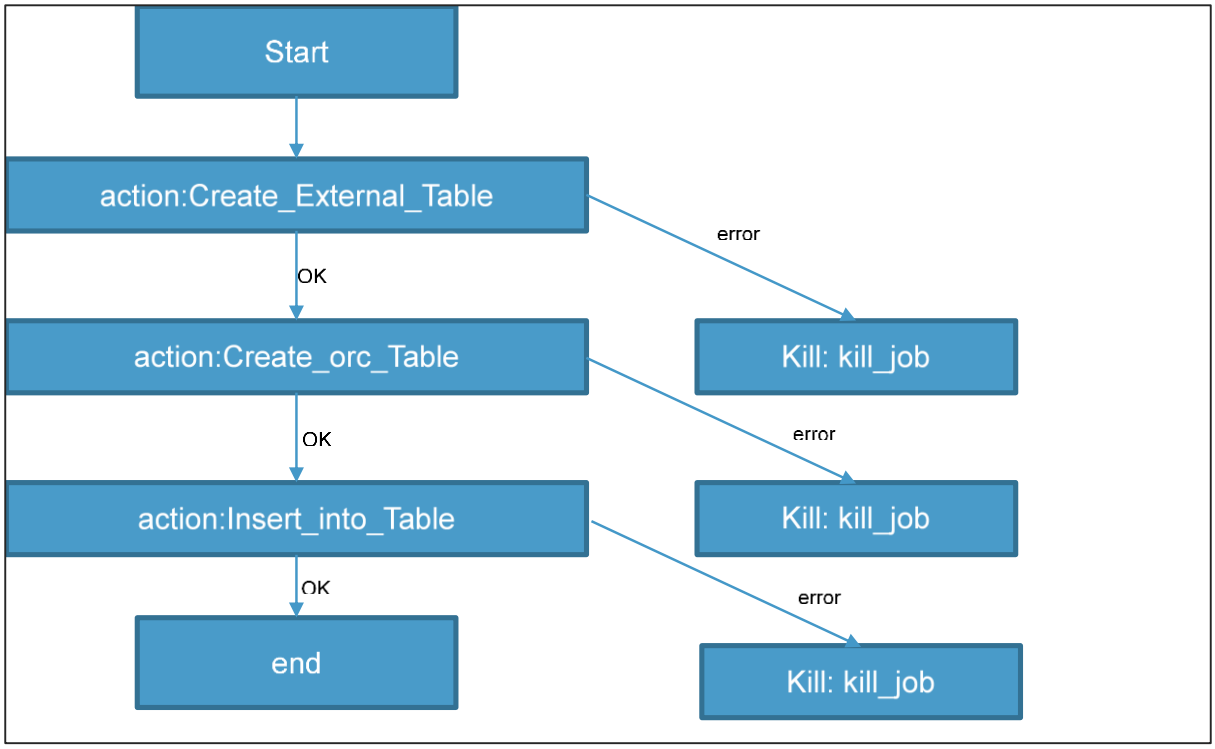
```
            <job-tracker>xyz.com:8088</job-tracker>
            <name-node>hdfs://rootname</name-node>
            <script>hdfs_path_of_script/orc.hive</script>
        </hive>
        <ok to="Insert_into_Table" />
        <error to="kill_job" />
    </action>

<!—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

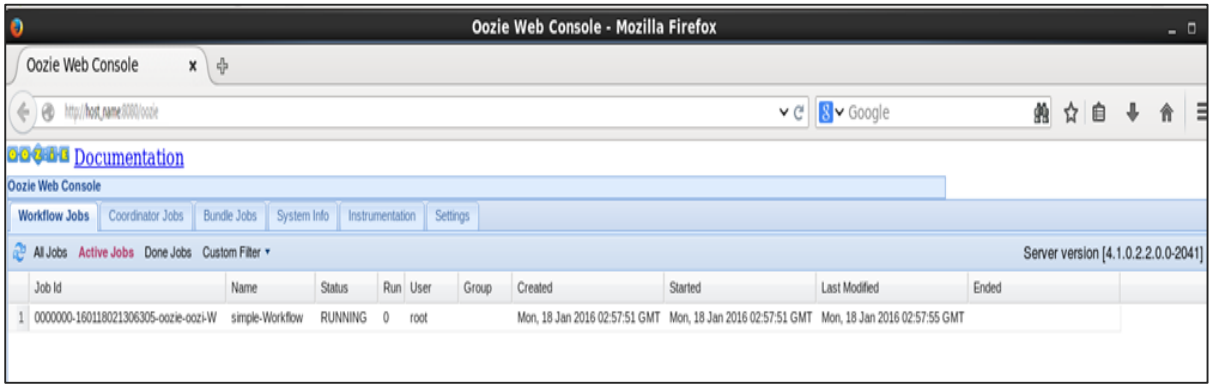
A topology runs in a distributed manner, on multiple worker nodes. Storm spreads the tasks evenly on all the worker nodes. The worker node's role is to listen for jobs and start or stop the processes whenever a new job arrives.

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/`



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.

Magic of Property File

Note that in the above example we have fixed the value of job-tracker, name-node, script and param by writing the exact value. This becomes hard to manage in many scenarios. This is where a **config file (.property file)** comes handy.

We will explore more on this in the following chapter.

3. Apache Oozie – Property File

Oozie workflows can be parameterized. The parameters come from a configuration file called as property file. We can run multiple jobs using same workflow by using multiple **.property** files (one property for each job).

Suppose we want to change the **jobtracker url** or change the script name or value of a param.

We can specify a **config file (.property)** and pass it while running the workflow.

Property File

Variables like **${nameNode}** can be passed within the workflow definition. The value of this variable will be replaced at the run time with the value defined in the '.properties' file.

Following is an example of a property file we will use in our workflow example.

File name -- job1.properties

```
#  proprties
nameNode=hdfs://rootname
jobTracker=xyz.com:8088
script_name_external=hdfs_path_of_script/external.hive
script_name_orc=hdfs_path_of_script/orc.hive
script_name_copy=hdfs_path_of_script/Copydata.hive
database=database_name
```

Now to use this property file we will have to update the workflow and pass the parameters in a workflow as shown in the following program.

```
<!-- This is a comment -->
<workflow-app xmlns="uri:oozie:workflow:0.4" name="simple-Workflow">
    <start to="Create_External_Table" />
    <action name="Create_External_Table">
        <hive xmlns="uri:oozie:hive-action:0.4">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <script>${script_name_external}</script>
        </hive>
        <ok to="Create_orc_Table" />
        <error to="kill_job" />
```

```
    </action>
    <action name="Create_orc_Table">
        <hive xmlns="uri:oozie:hive-action:0.4">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <script>${script_name_orc}</script>
        </hive>
        <ok to="Insert_into_Table" />
        <error to="kill_job" />
    </action>
    <action name="Insert_into_Table">
        <hive xmlns="uri:oozie:hive-action:0.4">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <script>${script_name_copy}</script>
            <param>${database}</param>
        </hive>
        <ok to="end" />
        <error to="kill_job" />
    </action>
    <kill name="kill_job">
        <message>Job failed</message>
    </kill>
    <end name="end" />
</workflow-app>
```

Now to use the property file in this workflow we will have to pass the **–config** while running the workflow.

```
oozie job --oozie http://host_name:8080/oozie --config
edgenode_path/job1.properties -D
oozie.wf.application.path=hdfs://Namenodepath/pathof_workflow_xml/workflow.xml
–run
```

Note: *The property file should be on the edge node (not in HDFS), whereas the workflow and hive scripts will be in HDFS.*

At run time, all the parameters in **${}** will be replaced by its corresponding value in the **.properties** file.

Also a single property file can have more parameters than required in a single workflow and no error will be thrown. This makes it possible to run more than one workflow by using the same properties file. But if the property file does not have a parameter required by a workflow then an error will occur.

4. Apache Oozie – Coordinator

Coordinator applications allow users to schedule complex workflows, including workflows that are scheduled regularly. Oozie Coordinator models the workflow execution triggers in the form of time, data or event predicates. The workflow job mentioned inside the **Coordinator** is started only after the given conditions are satisfied.

Coordinators

As done in the previous chapter for the workflow, let's learn concepts of coordinators with an example.

The first two hive actions of the workflow in our example creates the table. We don't need these step when we run the workflow in a coordinated manner each time with a given frequency. So let's modify the workflow which will then be called by our coordinator.

In a real life scenario, the external table will have a flowing data and as soon as the data is loaded in the external table, the data will be processed into ORC and from the file.

Modified Workflow:

```
<workflow-app xmlns="uri:oozie:workflow:0.4" name="simple-Workflow">
    <start to="Insert_into_Table" />
    <action name="Insert_into_Table">
        <hive xmlns="uri:oozie:hive-action:0.4">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <script>${script_name_copy}</script>
            <param>${database}</param>
        </hive>
        <ok to="end" />
        <error to="kill_job" />
    </action>
    <kill name="kill_job">
        <message>Job failed</message>
    </kill>
    <end name="end" />
</workflow-app>
```

Now let's write a simple coordinator to use this workflow.

```
<coordinator-app xmlns="uri:oozie:coordinator:0.2"
name="coord_copydata_from_external_orc" frequency="5 * * * *" start="2016-00-
18T01:00Z" end="2025-12-31T00:00Z"" timezone="America/Los_Angeles">
    <controls>
<timeout>1</timeout>
        <concurrency>1</concurrency>
        <execution>FIFO</execution>
        <throttle>1</throttle>
    </controls>
    <action>
        <workflow>
            <app-path>pathof_workflow_xml/workflow.xml</app-path>
        </workflow>
    </action>
</coordinator-app>
```

Definitions of the above given code is as follows:

- **start:** It means the start datetime for the job. Starting at this time the actions will be materialized.
- **end:** The end datetime for the job. When actions will stop being materialized.
- **timezone:** The timezone of the coordinator application.
- **frequency:** The frequency, in minutes, to materialize actions.

Control Information:

- **timeout:** The maximum time, in minutes, that a materialized action will be waiting for the additional conditions to be satisfied before being discarded. A timeout of 0 indicates that at the time of materialization all the other conditions must be satisfied, else the action will be discarded. A timeout of 0 indicates that if all the input events are not satisfied at the time of action materialization, the action should timeout immediately. A timeout of -1 indicates no timeout, the materialized action will wait forever for the other conditions to be satisfied. The default value is -1.
- **concurrency:** The maximum number of actions for this job that can be running at the same time. This value allows to materialize and submit multiple instances of the coordinator app, and allows operations to catchup on delayed processing. The default value is 1.
- **execution:** Specifies the execution order if multiple instances of the coordinator job have satisfied their execution criteria. Valid values are:
    - FIFO (oldest first) default.

- LIFO (newest first).
- LAST_ONLY (discards all older materializations).

(Ref of definitions: http://oozie.apache.org/docs/3.2.0-incubating/CoordinatorFunctionalSpec.html#a6.3._Synchronous_Coordinator_Application_Definition)

Above coordinator will run at a given frequency i.e. every 5th minute of an hour. (Similar to a cron job).

To run this coordinator, use the following command.

- **oozie job** -- oozie http://host_name:8080/oozie --config edgenode_path/job1.properties -D

- **oozie.wf.application.path=hdfs**: //Namenodepath/pathof_coordinator_xml/coordinator.xml -d "2 minute"` -run-d "2minute" will ensure that the coordinator starts only after 2 minutes of when the job was submitted.

The above coordinator will call the workflow which in turn will call the hive script. This script will insert the data from external table to hive the managed table.

Coordinator Job Status

Similar to the workflow, parameters can be passed to a coordinator also using the **.properties** file. These parameters are resolved using the configuration properties of Job configuration used to submit the coordinator job.

If a configuration property used in the definitions is not provided with the job configuration used to submit a coordinator job, the value of the parameter will be undefined and the job submission will fail.

At any time, a coordinator job is in one of the following statuses: **PREP, RUNNING, PREPSUSPENDED, SUSPENDED, PREPPAUSED, PAUSED, SUCCEEDED, DONWITHERROR, KILLED, FAILED**.

Valid coordinator job status transitions are:

- **PREP** – PREPSUSPENDED | PREPPAUSED | RUNNING | KILLED

- **RUNNING** – SUSPENDED | PAUSED | SUCCEEDED | DONWITHERROR | KILLED | FAILED

- **PREPSUSPENDED** – PREP | KILLED

- **SUSPENDED** – RUNNING | KILLED

- **PREPPAUSED** – PREP | KILLED

- **PAUSED** – SUSPENDED | RUNNING | KILLED

- When a coordinator job is submitted, Oozie parses the coordinator job XML. Oozie then creates a record for the coordinator with status **PREP** and returns a **unique ID**. The coordinator is also started immediately if the pause time is not set.

- When a user requests to suspend a coordinator job that is in status **PREP**, Oozie puts the job in the status **PREPSUSPEND**. Similarly, when the pause time reaches for a coordinator job with the status **PREP**, Oozie puts the job in the status **PREPPAUSED**.

- Conversely, when a user requests to resume a **PREPSUSPEND** coordinator job, Oozie puts the job in status **PREP**. And when the pause time is reset for a coordinator job and job status is **PREPPAUSED**, Oozie puts the job in status **PREP**.

- When a coordinator job starts, Oozie puts the job in status **RUNNING** and starts materializing workflow jobs based on the job frequency.

- When a user requests to kill a coordinator job, Oozie puts the job in status **KILLED** and it sends kill to all submitted workflow jobs. If any coordinator action finishes with not **KILLED**, Oozie puts the coordinator job into **DONEWITHERROR** .

- When a user requests to suspend a coordinator job that is in status **RUNNING**, Oozie puts the job in status **SUSPEND** and it suspends all the submitted workflow jobs.

- When pause time reaches for a coordinator job that is in status **RUNNING**, Oozie puts the job in status **PAUSED**.

Conversely, when a user requests to resume a **SUSPEND** coordinator job, Oozie puts the job in status **RUNNING**. And when pause time is reset for a coordinator job and job status is **PAUSED**, Oozie puts the job in status **RUNNING**.

A coordinator job creates workflow jobs (commonly coordinator actions) only for the duration of the coordinator job and only if the coordinator job is in **RUNNING** status. If the coordinator job has been suspended, when resumed it will create all the coordinator actions that should have been created during the time it was suspended, actions will not be lost, they will be delayed.

When the coordinator job materialization finishes and all the workflow jobs finish, Oozie updates the coordinator status accordingly. For example, if all the workflows are **SUCCEEDED**, Oozie puts the coordinator job into **SUCCEEDED** status. However, if any workflow job finishes with not **SUCCEEDED** (e.g. **KILLED** or **FAILED** or **TIMEOUT**), then Oozie puts the coordinator job into **DONEWITHERROR**. If all coordinator actions are **TIMEDOUT**, Oozie puts the coordinator job into **DONEWITHERROR**.

(**Reference:** http://oozie.apache.org/docs/)

Parametrization of a Coordinator

The workflow parameters can be passed to a coordinator as well using the **.properties** file. These parameters are resolved using the configuration properties of Job configuration used to submit the coordinator job.

If a configuration property used in the definition is not provided with the job configuration used to submit a coordinator job, the value of the parameter will be undefined and the job submission will fail.

5. Apache Oozie – Bundle

The **Oozie Bundle system** allows the user to define and execute a bunch of coordinator applications often called a data pipeline. There is no explicit dependency among the coordinator applications in a bundle. However, a user could use the data dependency of coordinator applications to create an implicit data application pipeline.

The user will be able to start/stop/suspend/resume/rerun in the bundle level resulting in a better and easy operational control.

Bundle

Let's extend our workflow and coordinator example to a bundle.

```
<bundle-app xmlns='uri:oozie:bundle:0.1'
name='bundle_copydata_from_external_orc'>
  <controls>
       <kick-off-time>${kickOffTime}</kick-off-time>
  </controls>
   <coordinator name='coord_copydata_from_external_orc' >
       <app-path>pathof_coordinator_xml</app-path>
       <configuration>
         <property>
              <name>startTime1</name>
              <value>time to start</value>
          </property>
      </configuration>
   </coordinator>
</bundle-app>
```

Kick-off-time: The time when a bundle should start and submit coordinator applications.

There can be more than one coordinator in a bundle.

Bundle Job Status

At any time, a bundle job is in one of the following status: PREP, RUNNING, PREPSUSPENDED, SUSPENDED, PREPPAUSED, PAUSED, SUCCEEDED, DONWITHERROR, KILLED, FAILED.

Valid bundle job status transitions are:

- **PREP** – PREPSUSPENDED | PREPPAUSED | RUNNING | KILLED
- **RUNNING** – SUSPENDED | PAUSED | SUCCEEDED | DONWITHERROR | KILLED | FAILED
- **PREPSUSPENDED** – PREP | KILLED
- **SUSPENDED** – RUNNING | KILLED
- **PREPPAUSED** – PREP | KILLED
- **PAUSED** – SUSPENDED | RUNNING | KILLED

- When a bundle job is submitted, Oozie parses the bundle job XML. Oozie then creates a record for the bundle with status **PREP** and returns a unique ID.
- When a user requests to suspend a bundle job that is in **PREP** state, Oozie puts the job in status **PREPSUSPEND**. Similarly, when the pause time reaches for a bundle job with **PREP** status, Oozie puts the job in status **PREPPAUSED**.
- Conversely, when a user requests to resume a PREPSUSPENDED bundle job, Oozie puts the job in status **PREP**. And when pause time is reset for a bundle job that is in **PREPPAUSED** state, Oozie puts the job in status **PREP**.
- There are two ways a bundle job could be started. * If **kick-off-time** (defined in the bundle xml) reaches. The default value is null, which means starts coordinators NOW. * If user sends a start request to START the bundle.
- When a bundle job starts, Oozie puts the job in status **RUNNING** and it submits the all coordinator jobs.
- When a user requests to kill a bundle job, Oozie puts the job in status **KILLED** and it sends kill to all submitted coordinator jobs.
- When a user requests to suspend a bundle job that is not in **PREP** status, Oozie puts the job in status **SUSPEND** and it suspends all submitted coordinator jobs.
- When pause time reaches for a bundle job that is not in **PREP** status, Oozie puts the job in status **PAUSED**. When the paused time is reset, Oozie puts back the job in status **RUNNING**.

When all the coordinator jobs finish, Oozie updates the bundle status accordingly. If all coordinators reach to the **same terminal state**, the bundle job status also moves to the same status. For example, if all coordinators are **SUCCEEDED**, Oozie puts the bundle job into **SUCCEEDED** status. However, if all coordinator jobs don't finish with the same status, Oozie puts the bundle job into **DONEWITHERROR**.

6. Apache Oozie – CLI and Extensions

By this time, you have a good understanding of Oozie workflows, coordinators and bundles. In the last part of this tutorial, let's touch base some of the other important concepts in Oozie.

Command Line Tools

We have seen a few commands earlier to run the jobs of workflow, coordinator and bundle. Oozie provides a command line utility, Oozie, to perform job and admin tasks.

```
oozie version : show client version
```

Following are some of the other job operations:

```
oozie job <OPTIONS> :

-action <arg>     coordinator rerun on action ids (requires -rerun);
coordinator log retrieval on action ids (requires -log)

-auth <arg>       select authentication type [SIMPLE|KERBEROS]

-change <arg>     change a coordinator/bundle job

-config <arg>     job configuration file '.xml' or '.properties'

-D <property=value>   set/override value for given property

-date <arg>       coordinator/bundle rerun on action dates (requires -rerun)

-definition <arg>     job definition

-doas <arg>       doAs user, impersonates as the specified user

-dryrun           Supported in Oozie-2.0 or later versions ONLY - dryrun or
test run a coordinator job, job is not queued

-info <arg>       info of a job

-kill <arg>       kill a job

-len <arg>        number of actions (default TOTAL ACTIONS, requires -info)

-localtime        use local time (default GMT)

-log <arg>        job log

-nocleanup        do not clean up output-events of the coordinator rerun
actions (requires -rerun)

-offset <arg>     job info offset of actions (default '1', requires -info)

-oozie <arg>      Oozie URL

-refresh          re-materialize the coordinator rerun actions (requires -
rerun)
```

```
-rerun <arg>      rerun a job (coordinator requires -action or -date; bundle
requires -coordinator or -date)
-resume <arg>     resume a job
-run              run a job
-start <arg>      start a job
-submit           submit a job
-suspend <arg>    suspend a job
-value <arg>      new endtime/concurrency/pausetime value for changing a
coordinator job; new pausetime value for changing a bundle job
-verbose          verbose mode
```

To check the status of the job, following commands can be used.

```
-auth <arg>    select authentication type [SIMPLE|KERBEROS]
-doas <arg>    doAs user, impersonates as the specified user.
-filter <arg>    user=<U>; name=<N>; group=<G>; status=<S>; ...
-jobtype <arg>   job type ('Supported in Oozie-2.0 or later versions ONLY -
coordinator' or 'wf' (default))
-len <arg>    number of jobs (default '100')
-localtime    use local time (default GMT)
-offset <arg>    jobs offset (default '1')
-oozie <arg>     Oozie URL
-verbose     verbose mode
```

For example: To check the status of the Oozie system you can run the following command:

```
oozie admin -oozie http://localhost:8080/oozie -status
```

Validating a Workflow XML –

```
oozie validate myApp/workflow.xml
```

It performs an XML Schema validation on the specified workflow XML file.

Action Extensions

We have seen hive extensions. Similarly, Oozie provides more action extensions few of them are as below:

Email Action

The email action allows sending emails in Oozie from a workflow application. An email action must provide to addresses, cc addresses (optional), a subject and a body. Multiple recipients of an email can be provided as comma separated addresses.

All the values specified in the email action can be parameterized (templated) using EL expressions.

Example:

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
    ...
    <action name="an-email">
        <email xmlns="uri:oozie:email-action:0.1">
            <to>julie@xyz.com,max@abc.com</to>
            <cc>jax@xyz.com</cc>
            <subject>Email notifications for ${wf:id()}</subject>
            <body>The wf ${wf:id()} successfully completed.</body>
        </email>
        <ok to="main_job"/>
        <error to="kill_job"/>
    </action>
    ...
</workflow-app>
```

Shell Action

The shell action runs a Shell command. The workflow job will wait until the Shell command completes before continuing to the next action.

To run the Shell job, you have to configure the shell action with the **=job-tracker=, name-node and Shell exec** elements as well as the necessary arguments and configuration. A shell action can be configured to create or delete HDFS directories before starting the Shell job.

The shell launcher configuration can be specified with a file, using the job-xml element, and inline, using the configuration elements.

Example:

How to run any shell script?

```
<workflow-app xmlns='uri:oozie:workflow:0.3' name='shell-wf'>
    <start to='shell1' />
    <action name='shell1'>
        <shell xmlns="uri:oozie:shell-action:0.1">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <file>path_of_file_name</file>
```

```
        </shell>
        <ok to="end" />
        <error to="fail" />
    </action>
    <kill name="fail">
        <message>Script failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message>
    </kill>
    <end name='end' />
</workflow-app>
```

Similarly, we can have many more actions like **ssh, sqoop, java action,** etc.

Additional Resources

Oozie official documentation website is the best resource to understand Oozie in detail.