Big data in its raw form rarely satisfies the Hadoop developer’s data requirements for performing data processing tasks. Different extract/transform/load (ETL) and pre-processing operations are usually needed before starting any actual processing jobs. Oozie is a framework that helps automate this process and codify this work into repeatable units or workflows that can be reused over time without the need to write any new code or steps. Learn how Oozie can be used to create different types of workflows.

What is Oozie?
Apache Oozie is an open source project based on Java™ technology that simplifies the process of creating workflows and managing coordination among jobs. In principle, Oozie offers the ability to combine multiple jobs sequentially into one logical unit of work. One advantage of the Oozie framework is that it is fully integrated with the Apache Hadoop stack and supports Hadoop jobs for Apache MapReduce, Pig, Hive, and Sqoop. In addition, it can be used to schedule jobs specific to a system, such as Java programs. Therefore, using Oozie, Hadoop administrators are able to build complex data transformations that can combine the processing of different individual tasks and even sub-workflows. This ability allows for greater control over complex jobs and makes it easier to repeat those jobs at predetermined periods.

In practice, there are different types of Oozie jobs:

- **Oozie Workflow** jobs — Represented as directed acyclical graphs to specify a sequence of actions to be executed.
- **Oozie Coordinator** jobs — Represent Oozie workflow jobs triggered by time and data availability.
- **Oozie Bundle**— Facilitates packaging multiple coordinator and workflow jobs, and makes it easier to manage the life cycle of those jobs.

How does Oozie work?
An Oozie workflow is a collection of actions arranged in a directed acyclic graph (DAG). This graph can contain two types of nodes: control nodes and action nodes. Control nodes, which are used to define job chronology, provide the rules for beginning and ending a workflow and control the
workflow execution path with possible decision points known as fork and join nodes. *Action nodes* are used to trigger the execution of tasks. In particular, an action node can be a MapReduce job, a Pig application, a file system task, or a Java application. (The shell and ssh actions have been deprecated).

Oozie is a native Hadoop stack integration that supports all types of Hadoop jobs and is integrated with the Hadoop stack. In particular, Oozie is responsible for triggering the workflow actions, while the actual execution of the tasks is done using Hadoop MapReduce. Therefore, Oozie becomes able to leverage 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. Figure 1 illustrates a sample Oozie workflow that combines six action nodes (Pig script, MapReduce jobs, Java code, and HDFS task) and five control nodes (Start, Decision control, Fork, Join, and End). Oozie workflows can also be parameterized. When submitting a workflow job, values for the parameters must be provided. If the appropriate parameters are used, several identical workflow jobs can occur concurrently.

**Figure 1. Sample Oozie workflow**

In practice, it is sometimes necessary to run Oozie workflows on regular time intervals, but in coordination with other conditions, such as the availability of specific data or the completion of any other events or tasks. In these situations, Oozie Coordinator jobs allow the user to model workflow execution triggers in the form of the data, time, or event predicates where the workflow job is started after those predicates get satisfied. The Oozie Coordinator can also manage multiple workflows that are dependent on the outcome of subsequent workflows. The outputs of subsequent workflows become the input to the next workflow. This chain is called a *data application pipeline*.

Oozie workflow definition language is XML-based and it is called the *Hadoop Process Definition Language*. Oozie comes with a command-line program for submitting jobs. This command-line program interacts with the Oozie server using REST. To submit or run a job using the Oozie client, give Oozie the full path to your workflow.xml file in HDFS as a parameter to the client. Oozie does not have a notion of global properties. All properties, including the *jobtracker* and the *namenode*, must be submitted as part of every job run. Oozie uses an RDBMS for storing state.
Oozie in action

Use an Oozie workflow to run a recurring job. Oozie workflows are written as an XML file representing a directed acyclic graph. Let’s look at the following simple workflow example that chains two MapReduce jobs. The first job performs an initial ingestion of the data and the second job merges data of a given type.

Listing 1. Simple example of Oozie workflow

```xml
<workflow-app xmlns='uri:oozie:workflow:0.1' name='SimpleWorkflow'>
<start to='ingestor'/>
<action name='ingestor'>
  <java>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <configuration>
      <property>
        <name>mapred.job.queue.name</name>
        <value>default</value>
      </property>
    </configuration>
  </java>
  <ok to='merging'/>
  <error to='fail'/>
</action>
<fork name='merging'>
  <path start='mergeT1'/>
  <path start='mergeT2'/>
</fork>
<action name='mergeT1'>
  <java>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <configuration>
      <property>
        <name>mapred.job.queue.name</name>
        <value>default</value>
      </property>
    </configuration>
    <main-class>com.navteq.assetmgmt.hdfs.merge.MergerLoader</main-class>
  </java>
  <ok to='completed'/>
  <error to='fail'/>
</action>
<action name='mergeT2'>
  <java>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <configuration>
      <property>
        <name>mapred.job.queue.name</name>
        <value>default</value>
      </property>
    </configuration>
  </java>
</action>
</workflow-app>
```
This simple workflow defines three actions: ingestor, mergeT1, and mergeT2. Each action is implemented as a MapReduce job. As illustrated in Figure 2, the workflow starts with the start node, which transfers control to the ingestor action. Once the ingestor step completes, a fork control node is invoked, an action that starts the execution of mergeT1 and mergeT2 in parallel. Once both actions are completed, the join control node is invoked. On successful completion of join node, the control is passed to the end node, a step that ends the process. The <job-tracker> and <name-node> entities dictate the servers that the Hive job will connect to for executing its script.

**Figure 2. Illustration of the workflow of Listing 1**

[Diagram of workflow]

Let's now look at another Oozie workflow example that incorporates a Hive job.

**Listing 2. Oozie workflow incorporating a Hive job**

```xml
<workflow-app xmlns='uri:oozie:workflow:0.1' name='Hive-Workflow'>
  <start to='hive-job'/>
  <action name='hive-job'>
    <hive xmlns='uri:oozie:hive-action:0.4'>
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <prepare>
        <delete path='${workflowRoot}/output-data/hive'/>
        <mkdir path='${workflowRoot}/output-data'/>
      </prepare>
      <job-xml>${workflowRoot}/hive-site.xml</job-xml>
      <configuration>
        <property>
          <name>oozie.hive.defaults</name>
          <value>${workflowRoot}/hive-site.xml</value>
        </property>
      </configuration>
      <script>hive_script.q</script>
      <param>JSON_SERDE=${workflowRoot}/lib/hive-serdes-1.0-SNAPSHOT.jar</param>
      <param>PARTITION_PATH=${workflowInput}</param>
      <param>DATEHOUR=${dateHour}</param>
    </hive>
    <ok to='end'/>
    <error to='fail'/>
  </action>
  <kill name='fail'>
    <message>Hive failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>
  </kill>
  <end name='end'/>
</workflow-app>
```
In this workflow, we identify the action as a Hive action with this node: `<hive xmlns='uri:oozie:hive-action:0.4'>`. The `<job-xml>` entity is used to specify a configuration file for Hive.

Finally, let's check another Oozie example of scheduling recurring workflows.

### Listing 3. Oozie recurring workflows

```xml
<coordinator-app name='Hive-workflow' frequency='${coord:hours(1)}'
start='${jobStart}' end='${jobEnd}'>
timezone='UTC'
xmllns='uri:oozie:coordinator:0.1'>
<datasets>
<dataset name='InputData' frequency='${coord:hours(1)}'
initial-instance='${initialDataset}' timezone='America/Los_Angeles'>
<uri-template>
 hdfs://hadoop1:8020/user/flume/InputData/${YEAR}/${MONTH}/${DAY}/${HOUR}
</uri-template>
<done-flag></done-flag>
</dataset>
</datasets>
<input-events>
<data-in name='InputData ' dataset='Data'>
<instance>${coord:current(coord:tzOffset() / 60)}</instance>
</data-in>
<data-in name='readyIndicator' dataset='tweets'>
<instance>${coord:current(1 + (coord:tzOffset() / 60))}</instance>
</data-in>
</input-events>
<action>
<workflow>
<app-path>${workflowRoot}/ Hive-Workflow.xml</app-path>
<configuration>
<property>
<name>workflowInput</name>
<value>${coord:dataIn('InputData')}</value>
</property>
<property>
<name>dateHour</name>
<value>${coord:formatTime(coord:dateOffset(coord:nominalTime(), tzOffset, 'HOUR'), 'yyyyMMddHH')}</value>
</property>
</configuration>
</workflow>
</action>
</coordinator-app>
```

In this example, the Hive workflow of the previous example is configured to be executed on an hourly basis using the `coord:hours(1)` method. Specify a start time and end time for the job using the code `jobStart` and `jobEnd` variables. The datasets entity specifies the location of a set of input data. In this case, there is a dataset called `InputData`, which is updated every hour, as specified by the frequency. For each execution of the Hive workflow, there will be a separate instance of the input dataset, starting with the initial instance specified by the dataset. `YEAR`, `MONTH`, `DAY`, and `HOUR` are variables used to parameterize the URI template for the dataset. The `done` flag specifies a file that determines when the dataset is finished being generated.

**Conclusion**

Helping Hadoop users in chaining and automating the execution of big data processing tasks into a defined workflow is quite useful feature in real-world practices. In this article, you were
introduced to Oozie, an Apache open source project that simplifies the process of creating workflow and coordination between Hadoop-based jobs. However, Oozie is not the only project that can help you to achieve this goal. Other projects include Azkaban (written and open-sourced by LinkedIn), Luigi (Python-based workflow engine) and Cascading (supports any JVM-based language such as Java, JRuby, and Clojure). If you are ready to get started with Oozie and its various features and facilities, see Related topics for more information and resources.
Related topics

• Learn more about the Oozie Workflow Scheduler for Hadoop.
• Find out more about the Hadoop Project.
• Visit the Pig Project website for resources about Apache Pig.
• Gather information about the Hive Project.
• Learn more about the Sqoop Project.
• Check out the Azkaban Project.
• Learn more about the Luigi Project.
• Check out the Cascading Project.
• Find out about InfoSphere Streams in the article “An introduction to InfoSphere Streams.”
• Find resources to help you get started with InfoSphere BigInsights, IBM’s Hadoop-based offering that extends the value of open source Hadoop with features like Big SQL, text analytics, and BigSheets.
• Download InfoSphere BigInsights Quick Start Edition, available as a native software installation or as a VMware image.
• Find resources to help you in the developerWorks IBM Streams developer community, IBM’s high-performance computing platform that enables user-developed applications to rapidly ingest, analyze, and correlate information as it arrives from thousands of real-time sources.
• Download InfoSphere Streams, available as a native software installation or as a VMware image.
• Use InfoSphere Streams on IBM SmartCloud Enterprise.

© Copyright IBM Corporation 2013
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)