Explain in Brief:

● The workflow of Oozie and its Benefits.

Oozie is a workflow manager and scheduler. Hadoop need its own special workflow scheduler.

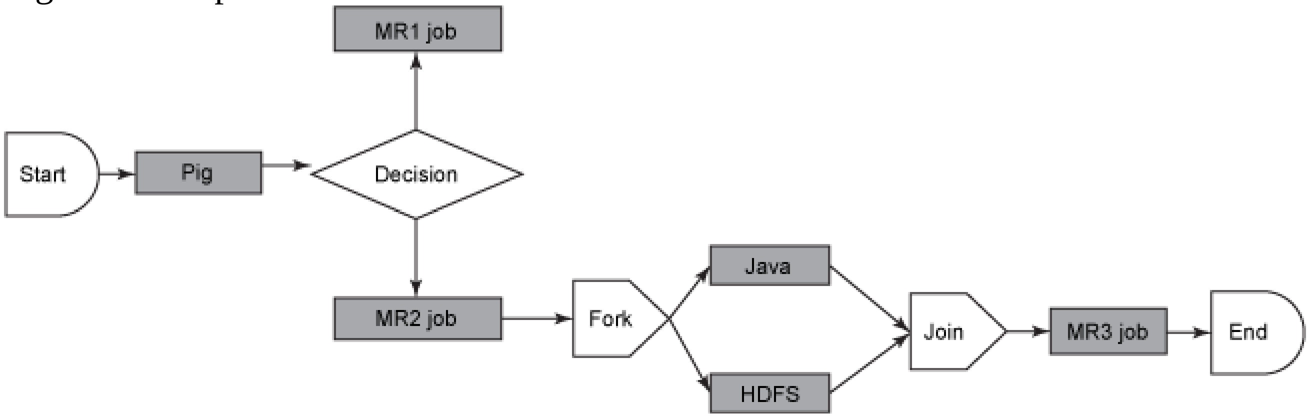
Workflow in Oozie is a sequence of actions arranged in a control dependency **DAG (Direct Acyclic Graph)**. The actions are in controlled dependency as the next action can only run as per the output of current action. Subsequent actions are dependent on its previous action. A workflow action can be a **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

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 scrip, MapReduce jobs, Java code, and HDFS task) and five control nodes (Start, Decision control, Fork, Join, and End). Oozie workflows can be also 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

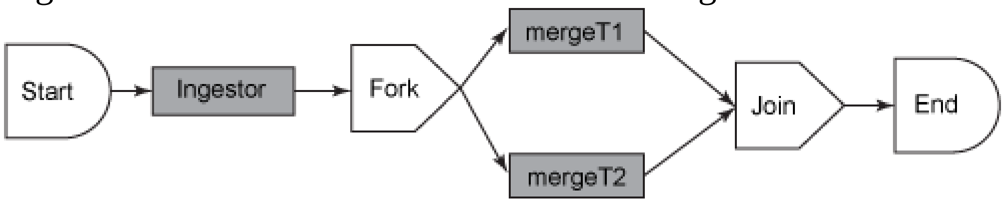
```
<workflow-app xmlns='uri:oozie:workflow:0.1' name='SimpleWorkflow'>
    <start to='ingestor'/>
```

```
<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>
        <arg>${driveID}</arg>
    </java>
    <ok to='merging'/>
    <error to='fail'/>
</action>
<fork name='merging'>
    <path start='mergeT1'/>
    <path start='mergeT2'/>
</fork>
<action name='mergeT1'>
    <java>
        <job-tracker>${jobTracker}</job-tracker>
        <name-node>${nameNode}</name-node>
        <configuration>
            <property>
                <name>mapred.job.queue.name</name>
                <value>default</value>
            </property>
        </configuration>
        <arg>-drive</arg>
        <arg>${driveID}</arg>
        <arg>-type</arg>
        <arg>T1</arg>
    </java>
    <ok to='completed'/>
    <error to='fail'/>
</action>
<action name='mergeT2'>
    <java>
        <job-tracker>${jobTracker}</job-tracker>
```

```
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>default</value>
                </property>
            </configuration>
            <main-class>com.navteq.assetmgmt.hdfs.merge.MergerLoader</main-class>
            <arg>-drive</arg>
            <arg>${driveID}</arg>
            <arg>-type</arg>
            <arg>T2</arg>
        </java>
        <ok to='completed'/>
        <error to='fail'/>
    </action>
    <join name='completed' to='end'/>
    <kill name='fail'>
        <message>Java failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</mes
    </kill>
    <end name='end'/>
</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



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

Listing 2. Oozie workflow incorporating a Hive job

```
<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=${workflowRoot}/lib/hive-serdes-1.0-SNAPSHOT.jar</p
            <param>PARTITION_PATH=${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())}]</me
    </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

```
<coordinator-app name='Hive-workglow' frequency='${coord:hours(1)}'
start='${jobStart}' end='${jobEnd}'
timezone='UTC'
```

```
xmlns='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}/${DAY}/${HO
    </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, 'H
      </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 codejobStart and jobEndvariables. 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.

As usual, it depends. In general, you can keep using any workflow scheduler that works for you. No need to change, really.
However, Oozie does have some benefits that are worth considering:

1. Oozie is designed to scale in a Hadoop cluster. Each job will be launched from a different datanode. This means that the workflow load will be balanced and no single machine will become overburdened by launching workflows. This also means that the capacity to launch workflows will grow as the cluster grows.
2. Oozie is well integrated with Hadoop security. This is especially important in a kerberized cluster. Oozie knows which user submitted the job and will launch all actions as that user, with the proper privileges. It will handle all the authentication details for the user as well.
3. Oozie is the only workflow manager with built-in Hadoop actions, making workflow development, maintenance and troubleshooting easier.
4. Oozie UI makes it easier to drill down to specific errors in the data nodes. Other systems would require significantly more work to correlate jobtracker jobs with the workflow actions.
5. Oozie is proven to scale in some of the world's largest clusters. The **white paper** discusses a deployment at Yahoo! that can handle 1250 job submissions a minute.
6. Oozie gets callbacks from MapReduce jobs so it knows when they finish and whether they hang without expensive polling. No other workflow manager can do this.
7. Oozie Coordinator allows triggering actions when files arrive at HDFS. This will be challenging to implement anywhere else.

8. Oozie is supported by Hadoop vendors. If there is ever an issue with how the workflow manager integrates with Hadoop

● The workflow of Sqoop and its Benefits

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. This is a brief tutorial that explains how to make use of Sqoop in Hadoop ecosystem.

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. It is provided by the Apache Software Foundation.

How Sqoop Works?

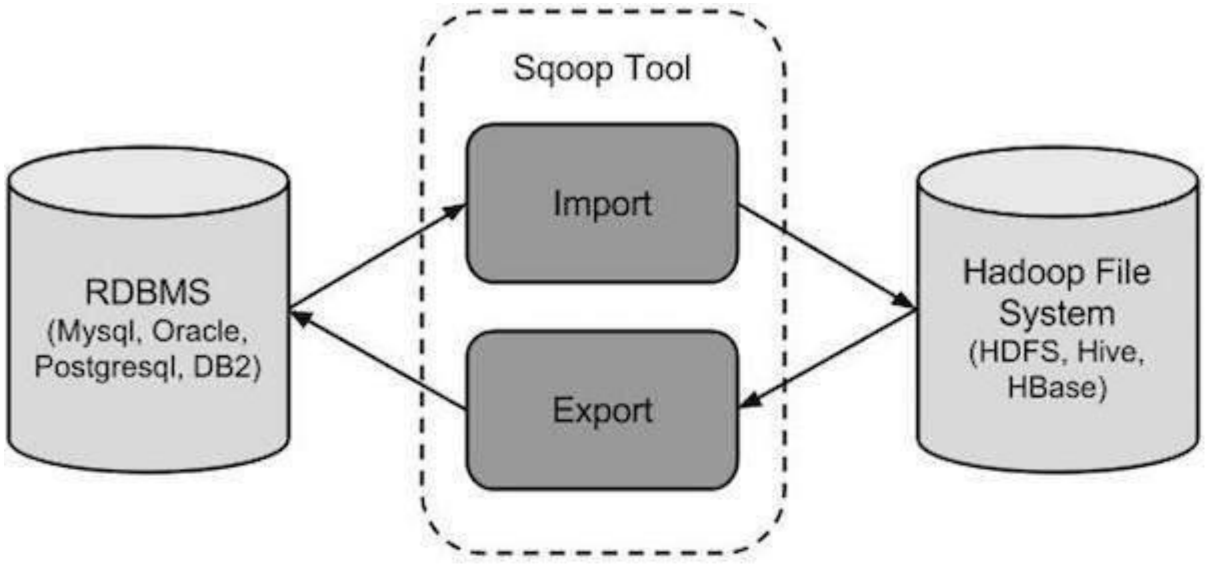
Sqoop Working

Step 1: Sqoop send the request to Relational DB to send the return the metadata informationabout the table(Metadata here is the data about the table in relational DB).

Step 2: From the received information it will generate the java classes (Reason why you shouldhave Java configured before get it working-Sqoop internally uses JDBC API to generate data).

Step 3: Now Sqoop (As its written in java ?tries to package the compiled classes to beable togenerate table structure) , post compiling creates jar file(Java packaging standard).

The following image describes the workflow of Sqoop.



Sqoop Import

The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

Sqoop Export

The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

Apache Sqoop does the following to integrate bulk data movement between Hadoop and structured datastores:

Function	**Benefit**
Import sequential datasets from mainframe	Satisfies the growing need to move data from mainframe to HDFS
Import direct to ORCFiles	Improved compression and light-weight indexing for improved query performance

Function	Benefit
Data imports	Moves certain data from external stores and EDWs into Hadoop to optimize cost-effectiveness of combined data storage and processing
Parallel data transfer	For faster performance and optimal system utilization
Fast data copies	From external systems into Hadoop
Efficient data analysis	Improves efficiency of data analysis by combining structured data with unstructured data in a schema-on-read data lake
Load balancing	Mitigates excessive storage and processing loads to other systems