

Sqoop and Oozie



Data and Analytics Training

Big Data – Self-Study Guide (204)

© Copyright 2014 Avanade Inc. All Rights Reserved. This document contains confidential and proprietary information of Avanade and may be protected by patents, trademarks, copyrights, trade secrets, and/or other relevant state, federal, and foreign laws. Its receipt or possession does not convey any rights to reproduce, disclose its contents, or to manufacture, use or sell anything contained herein. Forwarding, reproducing, disclosing or using without specific written authorization of Avanade is strictly forbidden. The Avanade name and logo are registered trademarks in the US and other countries. Other brand and product names are trademarks of their respective owners.

This study guide serves as a basis for your self-study on the basics of Sqoop and Oozie. The study guide gives you an overview on the main topics and presents a number of internal and external resources you should use to get a deeper understanding of Sqoop and Oozie, the related terms and technical details.

Please especially use the external resources provided. They are a main part of this study guide and are required to get a good understanding of the discussed topics.

Estimated Completion Time

2 hours (including time for external resources)

Objectives

After completing this study guide, you will be able to understand

- The different ways of automating big data processing
- Different files used in Oozie and how they can be configured
- The concept behind Sqoop
- Some common command line options used in Sqoop
- Working with Sqoop using PowerShell

Contents

Sqoop 4

 Command line for Sqoop 5

 export 5

 import..... 6

 List all tables 7

 Validation 8

 Exporting Data with Sqoop using PowerShell 8

Oozie 12

 Understanding Oozie 12

 Oozie Workflow File..... 14

 Script Files 15

 Job.properties..... 15

 Running Oozie Workflow 16

Conclusion 22

Sqoop

While Hadoop is a natural choice for processing unstructured and semi-structured data, such as logs and files, there may also be a need to process structured data stored in relational databases.¹

Sqoop is a tool designed to transfer data between Hadoop clusters and relational databases. You can use it to import data from a relational database management system (RDBMS) such as SQL, MySQL or Oracle into the Hadoop Distributed File System (HDFS), transform the data in Hadoop with MapReduce or Hive, and then export the data back into an RDBMS.

Sqoop provides a pluggable connector mechanism for optimal connectivity to external systems. The Sqoop extension API provides a convenient framework for building new connectors which can be dropped into Sqoop installations to provide connectivity to various systems. Sqoop itself comes bundled with various connectors that can be used for popular database and data warehousing systems.²

Sqoop was originally created by Cloudera and is now an Apache project.

For supported Sqoop versions on HDInsight clusters, see this page:

<http://azure.microsoft.com/en-us/documentation/articles/hdinsight-component-versioning/>

With Sqoop, you can import data from a relational database system into HDFS. The input to the import process is a database table. Sqoop will read the table row-by-row into HDFS. The output of this import process is a set of files containing a copy of the imported table. The import process is performed in parallel. For this reason, the output will be in multiple files. These files may be delimited text files (for example, with commas or tabs separating each field), or binary Avro or SequenceFiles containing serialized record data.

A by-product of the import process is a generated Java class which can encapsulate one row of the imported table. This class is used during the import process by Sqoop itself. The Java source code for this class is also provided to you, for use in subsequent MapReduce processing of the data. This class can serialize and deserialize data to and from the SequenceFile format. It can also parse the delimited-text form of a record. These abilities allow you to quickly develop MapReduce applications that use the HDFS-stored records in your processing pipeline. You are also free to parse the delimited record data yourself, using any other tools you prefer.

¹ <http://azure.microsoft.com/en-us/documentation/articles/hdinsight-use-sqoop>

² <http://hortonworks.com/hadoop/sqoop/>

After manipulating the imported records (for example, with MapReduce or Hive) you may have a result data set which you can then export back to the relational database. Sqoop's export process will read a set of delimited text files from HDFS in parallel, parse them into records, and insert them as new rows in a target database table, for consumption by external applications or users.

Sqoop includes some other commands which allow you to inspect the database you are working with. For example, you can list the available database schemas (with the `sqoop-list-databases` tool) and tables within a schema (with the `sqoop-list-tables` tool). Sqoop also includes a primitive SQL execution shell (the `sqoop-eval` tool).

Most aspects of the import, code generation, and export processes can be customized. You can control the specific row range or columns imported. You can specify particular delimiters and escape characters for the file-based representation of the data, as well as the file format used. You can also control the class or package names used in generated code. Subsequent sections of this document explain how to specify these and other arguments to Sqoop.³

Sqoop jobs can be initiated from⁴:

- Hadoop command line on the remote desktop
- PowerShell, can be done remotely
- Custom application using .NET SDK for Hadoop
- An action within Oozie workflow. We can have an Oozie workflow that does all of the data processing it needs to do and then the last thing it does is export the data off to relational database so that our users can report from it.

The first two methods will be explained in the following sections.

Command line for Sqoop

To execute Sqoop commands, on the Hadoop command line you can access Sqoop from `%SQOOP_HOME%\bin\sqoop`

Some common commands for Sqoop are:

export

At minimum, you must specify `--connect`, `--export-dir`, and `--table`.⁵

³ http://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html#_introduction

⁴ <http://www.microsoftvirtualacademy.com/training-courses/implementing-big-data-analysis>

⁵ https://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html#_literal_sqoop_export_literal

```
sqoop export --connect <connectionString> --table <tableName> --export-dir  
<HDFS path>
```

The above command will export data from HDFS into the table in the database specified by the connection string.

Another basic export to populate a table named bar with validation enabled:

```
sqoop export --connect jdbc:mysql://db.example.com/foo --table  
bar --export-dir /results/bar_data --validate
```

An export that calls a stored procedure named barproc for every record in /results/bar_data would look like:

```
sqoop export --connect jdbc:mysql://db.example.com/foo --call  
barproc --export-dir /results/bar_data
```

import

At minimum, you must specify --connect and --table

```
sqoop import --connect <connectionString> --table <tableName>
```

The above command will import data into HDFS from the table in the database pointed to by the connection string. By default, Sqoop will import a table named foo to a directory named foo inside your home directory in HDFS. For example, if your username is someuser, then the import tool will write to /user/someuser/foo/(files)

This code sample will import the data to Hive:

```
sqoop import --connect jdbc:mysql://db.foo.com/corp --table  
EMPLOYEES --hive-import
```

Sqoop provides an incremental import mode which can be used to retrieve only rows newer than some previously-imported set of rows. The argument --check-column specifies the column to be examined when determining which rows to import. Sqoop determines new rows based on the mode passed to --incremental argument which can be either append or lastmodified. Append gets those rows which have row id for the check column greater than the maximum row id imported last time. Last modified mode gets those rows where the check column holds a timestamp more recent than the timestamp specified with --last-value argument.

Sqoop imports data in parallel from most database sources. You can specify the number of map tasks (parallel processes) to use to perform the import by using the `-m` or `--num-mappers` argument. Each of these arguments takes an integer value which corresponds to the degree of parallelism to employ. By default, four tasks are used. Some databases may see improved performance by increasing this value to 8 or 16. Do not increase the degree of parallelism greater than that available within your MapReduce cluster; tasks will run serially and will likely increase the amount of time required to perform the import. Likewise, do not increase the degree of parallelism higher than that which your database can reasonably support. Connecting 100 concurrent clients to your database may increase the load on the database server to a point where performance suffers as a result.⁶

This code sample will use 8 parallel tasks:

```
sqoop import --connect jdbc:mysql://db.foo.com/corp --table  
EMPLOYEES -m 8
```

When performing parallel imports, Sqoop needs a criterion by which it can split the workload. Sqoop uses a splitting column to split the workload. By default, Sqoop will identify the primary key column (if present) in a table and use it as the splitting column. The low and high values for the splitting column are retrieved from the database, and the map tasks operate on evenly-sized components of the total range.

If the actual values for the primary key are not uniformly distributed across its range, then this can result in unbalanced tasks. You should explicitly choose a different column with the `--split-by` argument. For example, `--split-by dept_id`. Sqoop cannot currently split on multi-column indices. If your table has no index column, or has a multi-column key, then you must also manually choose a splitting column.

```
sqoop import --connect jdbc:mysql://db.foo.com/corp --table  
EMPLOYEES --split-by dept_id
```

List all tables

The below code will list tables available in the "corp" database⁷:

```
sqoop list-tables --connect  
jdbc:mysql://database.example.com/corp
```

⁶ https://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html#_controlling_parallelism

⁷ https://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html#_literal_sqoop_list_tables_literal

Validation

Validate the data copied, either import or export by comparing the row counts from the source and the target post copy⁸.

Here's a basic import of a table named EMPLOYEES in the corp database that uses validation to validate the row counts:

```
sqoop import --connect jdbc:mysql://db.foo.com/corp --table  
EMPLOYEES --validate
```

And here's a basic export to populate a table named bar with validation enabled:

```
sqoop export --connect jdbc:mysql://db.example.com/foo --table  
bar --export-dir /results/bar_data -validate
```

Validation currently only validates data copied from a single table into HDFS. The following are the limitations in the current implementation:

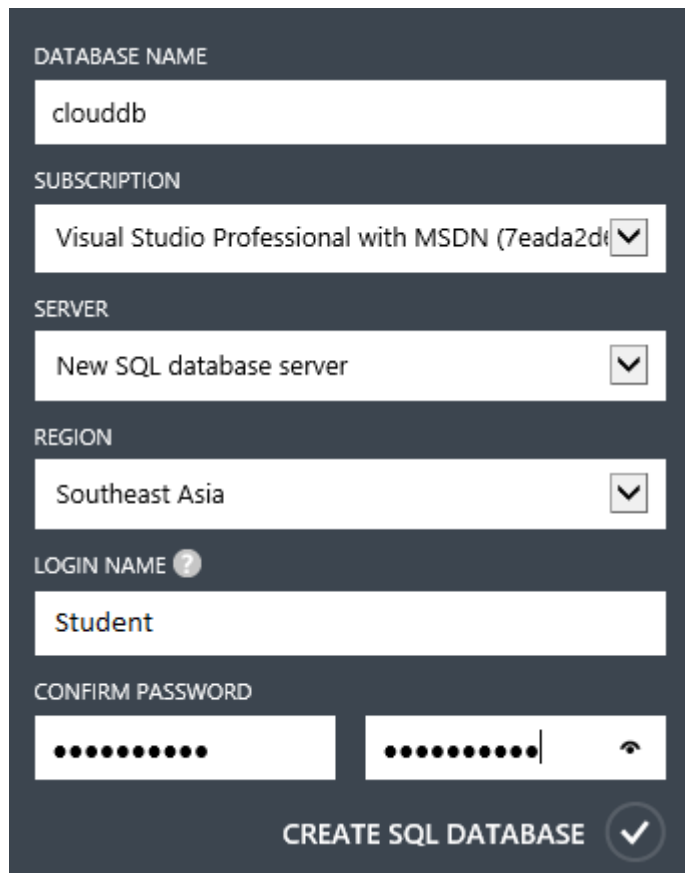
- all-tables option
- free-form query option
- Data imported into Hive, HBase or Accumulo
- table import with --where argument
- incremental imports

Exporting Data with Sqoop using PowerShell

We will first provision a Windows Azure SQL Database in our Windows Azure management portal. We are using Azure SQL database because it is an easy way to set up a relational database in the cloud, and we don't have to worry about any of the infrastructure. That is all handled for us by Windows Azure. This tutorial will equally work on a SQL Server in a Virtual Machine in Windows Azure or SQL Server running on premise as long as you've got the firewall configured to allow the appropriate connection using JDBC from your cluster.

In management portal, go to SQL DATABASES in left pane, click on NEW in the bottom bar, and QUICK CREATE the database as shown below:

⁸ <https://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html#validation>



DATABASE NAME

clouddb

SUBSCRIPTION

Visual Studio Professional with MSDN (7eada2d...)

SERVER

New SQL database server

REGION

Southeast Asia

LOGIN NAME ?

Student

CONFIRM PASSWORD

CREATE SQL DATABASE

1. Go to the database in the management portal once it is created, and click on DASHBOARD at the top. This will show you the SERVER NAME in the right pane when you scroll down.



We can connect to this server and manage from our SQL Server Management Studio to manage this instance.

2. Fire up SQL Server Management Studio, connect to this database server using the credentials you used to create this database.



This will connect to Windows Azure and will show up in Object Explorer as your Windows Azure SQL database.

3. Expand Tables under CloudDB for this connection, and you would find no user tables right now
4. We will now create a table to put our results into.

```
Create TABLE GrossDomesticProduct
(ReocrdID INTEGER PRIMARY KEY CLUSTERED,
CountryRegion VARCHAR(20),
[Year] CHAR(4),
GDPValue money);
```

One of the rules in Windows Azure SQL Database is that each table has to have some sort of clustered index. If you now refresh your tables in Object Explorer, this new table will appear.

5. Go to PowerShell to use Sqoop to transfer data from our HDFS storage across into our database.

```

$clusterName = "<clustername>"
$sqlDatabaseServerName = "ksohuccr7d"
$sqlDatabaseUserName = "Student"
$sqlDatabasePassword = "P@sswOrd"
$sqlDatabaseDatabaseName = "clouddb"
$tableName = "GrossDomesticProduct"
$hdfsPath = "/data/gdp"

$user = "$sqlDatabaseUserName@$sqlDatabaseServerName"
$serverPath = "sqlserver://$sqlDatabaseServerName.database.windows.net"

# Create a Sqoop job
$sqoopCommand = "export --connect jdbc:$serverPath;"
$sqoopCommand += "user=$user;password=$sqlDatabasePassword;database=$sqlDatabaseDatabaseName"
$sqoopCommand += " --table $tableName --export-dir $hdfsPath --input-fields-terminated-by \t"

$sqoopDef = New-AzureHDInsightSqoopJobDefinition -Command $sqoopCommand

# Submit the Sqoop job
$sqoopJob = Start-AzureHDInsightJob -Cluster $clusterName -JobDefinition $sqoopDef
write-Host "Sqoop job submitted..."
wait-AzureHDInsightJob -WaitTimeoutInSeconds 3600 -Job $sqoopJob
Get-AzureHDInsightJobOutput -Cluster $clusterName -JobId $sqoopJob.JobId -StandardError

```

Here, we are connecting to our cluster. We specify the SQL Server database server name, credentials to log in to that. We specify the table name that we want the data to transfer into. And the path in HDFS, that's where we want to transfer data out of.

Then we provide the actual Sqoop command, which is exactly pretty much the command we will execute at the command line, if we were to do this in the console.

It is an export command and we pass the connect switch that contains the connection string, a switch for the table that tells which table to import the data into, a switch export-dir which is the folder we want to take data out of, a switch for telling how the data is formatted, in our case it is terminated by tab.

Then we have got a pretty similar piece of code that we've seen all the way through this course. It is a `New-AzureHDInsightSqoopJobDefinition`. We've seen lots of these different type of job definitions. And that should begin to ring some bells. What you think would happen when we run some Sqoop job?

We can only guess that we are going to see a Map/Reduce. Which is indeed right. What Sqoop does under the cover is actually use the Map/Reduce engine to do this. So it is going to take the data and then there is additional code in Sqoop that will do the actual transfer.

Next we go ahead and start that job, and it is pretty much the same code we've seen before. Then we are going to wait for the job, we see the progress, and when it finishes we see the status information from the job.

Please have a look at this 5 minutes introduction video on Sqoop:

<http://youtu.be/HVyNjEMxSrM>

Oozie

Apache Oozie is a workflow/coordination system that manages Hadoop jobs. It is integrated with the Hadoop stack and supports Hadoop jobs for Apache MapReduce, Apache Pig, Apache Hive, and Apache Sqoop. It can also be used to schedule jobs specific to a system, like Java programs or shell scripts.⁹

In the module on HCatalog we've seen one aspect of making the solution bit more flexible, more reusable so that we could move folders around and not necessarily break the scripts that we've got, but one of the things we really want to do is to have the ability to automate a sequence of tasks. We might have multiple steps in our process, perhaps combining Pig jobs with Hive jobs and whole bunch of things that we need to do.

What we want to do is look at some options in automating these various steps. In other modules of this course we've been using PowerShell to run scripts and actually it turns out Windows Azure PowerShell is an incredibly powerful way for us to coordinate a sequence of tasks that do the big data processing: Everything from provisioning the cluster in the first place to uploading the data files, to uploading new jar files, Pig scripts, or Hive scripts to running Map/Reduce jobs and downloading the data back down to the on premise environment can be automated by using PowerShell.

Similarly if you wanted to build sort of custom applications, perhaps some kind of add-in for Excel or something like that, the Windows Azure HDInsight .NET SDK provides you with all those classes you need to build the .NET solution that will connect and do pretty much all the same things we do with PowerShell, like to connect to your Windows Azure subscription, run your HDInsight Map/Reduce jobs and get the data and all that kind of things.

There is actually, as part of the Hadoop environment, a technology designed specifically for automating workflows of tasks. And that is called Oozie.

Understanding Oozie

An Oozie Workflow is a collection of actions arranged in a Directed Acyclic Graph (DAG) . Control nodes define job chronology, setting rules for beginning and ending a workflow, which controls the

⁹ <http://azure.microsoft.com/en-us/documentation/articles/hdinsight-use-oozie>

workflow execution path with decision, fork and join nodes. Action nodes trigger the execution of tasks.

Oozie triggers workflow actions, but Hadoop MapReduce executes them. This allows Oozie to leverage other capabilities within the Hadoop stack to balance loads and handle failures.

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

Often it is necessary to run Oozie workflows on regular time intervals, but in coordination with unpredictable levels of data availability or events. In these circumstances, Oozie Coordinator allows you to model workflow execution triggers in the form of the data, time or event predicates. The workflow job is started after those predicates are satisfied.

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".¹⁰

- **Oozie Workflow** jobs are Directed Acyclical Graphs (DAGs), specifying a sequence of actions to execute. The Workflow job has to wait
- **Oozie Coordinator** jobs are recurrent Oozie Workflow jobs that are triggered by time and data availability.

We have an XML file that defines the workflow actions. And you usually define those actions as script: You probably have some script files that are HiveQL scripts, or Pig scripts that you are going to get Oozie workflow to execute as it goes through the workflow.

To coordinate all of this, we kick off the Oozie workflow from some sort of client, whether it be a machine in your environment or whether it is a remote desktop on the cluster. There is a job properties file that determines how the job would behave and it determines how many parameter values we are going to use.

If Oozie is going to run a Hive script in one of its actions, it needs to know how to set up the Hive environment and needs to know what the configuration settings it should use for Hive is. So we

¹⁰ <http://hortonworks.com/hadoop/oozie/>

usually give it some configuration file to tell it when it uses Hive, these are the number of reducers to use, these are the different parameters to set for the Hive environment.

Oozie Workflow File

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="MyWorkflow">
  <start to="FirstAction"/>
  <action name="FirstAction">
    <hive xmlns="uri:oozie:hive-action:0.2">
      <script>CreateTable.q</script>
      <param>TABLE_NAME=${tableName}</param>
      <param>LOCATION=${tableFolder}</param>
    </hive>
    <ok to="SecondAction"/>
    <error to="fail"/>
  </action>
  <action name="SecondAction">
    ...
  </action>
  <kill name="fail">
    <message>Workflow failed, error message[${wf.errorMessage(wf.lastErrorNode())}]</message>
  </kill>
  <end name="end"/>
</workflow-app>
```

Above is a fairly simple example of an Oozie workflow file. The workflow consists of actions and each action is a node in the XML document. In this example there is an action that runs a parameterized script, and you can see there are params in there where we are passing parameter values into the script. So, if we don't have to hard-code values in our script, we can pass them as parameters.

There is this notion of branching the workflow based on the outcome: There is the FirstAction in the sample and if it succeeds, we go on to the SecondAction, otherwise we go to fail if there is an error. And you can see in the end of the script there is a kill element with a name of fail. That's where we go if something goes wrong.

These files can become very complex, but this is a fairly simply conceptual example.

Script Files

```
DROP TABLE IF EXISTS ${TABLE_NAME};
CREATE EXTERNAL TABLE ${TABLE_NAME}
(Col1 STRING,
 Col2 FLOAT,
 Col3 FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION '${LOCATION}';
```

We pass parameters from the Oozie workflow file to the script file. Above is just a regular HiveQL file, but it has some parameters placeholders in there. So that allows us to be flexible, we don't have to hard code the table names, or the location or anything like that. We can pass that in as the parameters.

Job.properties

```
nameNode=wasb://my_container@my_storage_account.blob.core.windows.net
jobTracker=jobtrackerhost:9010
queueName=default
oozie.use.system.libpath=true

oozie.wf.application.path=/example/workflow/

tableName=ExampleTable
tableFolder=/example/ExampleTable
```

And the job properties file that we use on the client machine where we kick off the Oozie job contains the basic Oozie jobs settings, like Oozie endpoint which is an HTTP connection that we connect to.

We set up some settings, and can also specify some parameters at the end of that. Parameters at the end of this file tell the names the Oozie workflow should then pass to the script file. So this is a second level interaction. The parameters are defined in the job properties file, they are passed to the Oozie workflow file and then the Oozie workflow file, passes them on to the individual script that perform the actions.

Running Oozie Workflow

1. Create a Job.properties file as below:

```
nameNode=wasb://<container-name>@<storage-account>.blob.core.windows.net
jobTracker=jobtrackerhost:9010
queueName=default
oozie.use.system.libpath=true
oozie.wf.application.path=/adventureworks/oozieworkflow/
webDataTableName=webdata
webDataTableFolder=/adventureworks/webdata
logDataTableName=logdata
```

It contains the nameNode where the job is going to run, the path to where the Oozie workflow is stored which is /adventureworks/oozieworkflow in our case. It also contains some parameters, webDataTableName, webDataTableFolder which tells the location for this table and the logDataTableName

We can see the various parameters that we are going to pass into our Oozie job.

Let's look at the Oozie job now:

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="WebLogDataWorkflow">
  <start to="CreateWebDataTable" />
  <action name="CreateWebDataTable">
    <hive xmlns="uri:oozie:hive-action:0.2">
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <configuration>
        <property>
          <name>mapred.job.queue.name</name>
          <value>default</value>
        </property>
        <property>
          <name>oozie.hive.defaults</name>
```



```

        <value>hive-default.xml</value>
    </property>
</configuration>
<script>CreateWebDataTable.q</script>
<param>TABLE_NAME=${webDataTableName}</param>
<param>LOCATION=${webDataTableFolder}</param>
</hive>
<ok to="LoadData" />
<error to="fail" />
</action>
<action name="LoadData">
    <hive xmlns="uri:oozie:hive-action:0.2">
        <job-tracker>${jobTracker}</job-tracker>
        <name-node>${nameNode}</name-node>
        <configuration>
            <property>
                <name>mapred.job.queue.name</name>
                <value>default</value>
            </property>
            <property>
                <name>oozie.hive.defaults</name>
                <value>hive-default.xml</value>
            </property>
        </configuration>
        <script>LoadData.q</script>
        <param>SOURCE_TABLE_NAME=${logDataTableName}</param>
        <param>TARGET_TABLE_NAME=${webDataTableName}</param>
    </hive>
    <ok to="end" />
    <error to="fail" />
</action>
<kill name="fail">
    <message>Workflow failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message>
</kill>
<end name="end" />
</workflow-app>

```

The name of the workflow is WebLogDataWorkflow. The <start> tag tells us where to start. There are bunch of <action> tag in the workflow, a special action <kill> if something goes wrong and the <end> tag which marks the end of the workflow.

The `<start>` tag points to the first `<action>` tag. This action runs a Hive job. It uses `jobTracker` and `nameNode` that are passed in from the `job.properties` file. These are the internal settings that Oozie is going to use. The configuration for the Hive job uses default values, including `hive-default.xml` which has the settings that it uses to configure the Hive environment that it is going to use. This action tag runs the script file called `CreateWebDataTable.q` and it is going to pass a couple of parameters to that script. So, it takes the parameters from `job.properties` and passes them to `CreateWebDataTable.q` script. If this action succeeds, it executes the next action by name `LoadData` otherwise on failure it goes to `<kill>` tag by name "fail".

When we provision an HDInsight cluster, the installation of Hive under `C:\apps\dist\hive-0.13.0.2.1.8.0-2176` includes a `conf` folder. In this `conf` folder is a `hive-site.xml` which has the default configuration for Hive in our HDInsight cluster. If we would like to change the way Hive behaves, this is the XML that we would edit. For this tutorial, we have used this file and copied and renamed it to `hive-default.xml`

The `<action>` tag by name "LoadData" is similar in structure and easy to follow. Point to note over here in this action tag is that on succeeding, it goes to `<end>` element which completes the workflow.

The `<kill>` tag is recording the message in the log file which captures the error message for the last thing that went wrong.

So, it is a fairly rudimentary workflow, a very simple example of a workflow but you can see the power of this thing. We can define all of the steps we need to take, we can parameterise it and just re-execute it time-and-time-again with appropriate parameters.

2. Let's have a look at the script file "CreateWebDataTable.q"

```
DROP TABLE IF EXISTS ${TABLE_NAME};  
CREATE EXTERNAL TABLE ${TABLE_NAME}  
(LogDate STRING,  
PageHits INT,  
BytesRecvd INT,  
BytesSent INT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '  
STORED AS TEXTFILE LOCATION '${LOCATION}';
```

It creates a table using a parameter name that is being passed in to it. And it stores it in the location that is passed to it as a parameter. It is a fairly straight-forward HiveQL statement that is using parameters passed in to it from our workflow.xml file.

3. Let's have a look at LoadData.q file used by the workflow:

```
FROM ${SOURCE_TABLE_NAME}  
INSERT INTO TABLE ${TARGET_TABLE_NAME}  
SELECT log_date, COUNT(*), SUM(sc_bytes), SUM(cs_bytes)  
GROUP BY log_date  
ORDER BY log_date;
```

It takes data from source table and puts it into target table. It is again a straight-forward HiveQL statement with a couple of parameters to make it slightly flexible.

4. Lets put all these files up in our HDFS. Make sure that these files are in the following structure in your local hard drive

- ❖ C:\Scripts folder
 - oozieworkflow folder
 - workflow.xml
 - CreateWebDataTable.q
 - LoadData.q
 - hive-default.xml
 - job.properties file

Create a directory by the name adventureworks on HDFS

```
C:\>hadoop fs -mkdir /adventureworks
```

Then copy the oozieworkflow files to this folder that we just created and verify the copy was successful.

```
C:\>hadoop fs -copyFromLocal c:\scripts\oozieworkflow /adventureworks
C:\>hadoop fs -ls /adventureworks
Found 1 items
drwxr-xr-x - RemoteUser supergroup      0 2014-11-18 06:57 /adventureworks
/oozieworkflow
C:\>hadoop fs -ls /adventureworks/oozieworkflow
Found 4 items
-rw-r--r-- 1 RemoteUser supergroup      234 2014-11-18 06:57 /adventureworks
/oozieworkflow/CreateWebDataTable.q
-rw-r--r-- 1 RemoteUser supergroup      161 2014-11-18 06:57 /adventureworks
/oozieworkflow/LoadData.q
-rw-r--r-- 1 RemoteUser supergroup     7799 2014-11-18 06:57 /adventureworks
/oozieworkflow/hive-default.xml
-rw-r--r-- 1 RemoteUser supergroup     1660 2014-11-18 06:57 /adventureworks
/oozieworkflow/workflow.xml
```

5. Now we are ready to go and run our Oozie workflow. We use the system OOZIE_ROOT to access the folder where Oozie is installed. The command line syntax to execute this workflow is shown here:

```
C:\>%OOZIE_ROOT%\oozie-win-distro\bin\oozie job -oozie http://localhost:11000/oozie -config c:\scripts\job.properties -run_
```

We pass the command "job" to the oozie executable. We also pass some parameters into that job, like location of the server installation of Oozie that we want to use, that is, where the actual Hadoop cluster running Oozie is. We also specify the configuration for running the job which is contained in our job.properties file.

The configuration file includes the location of where the configuration file is, and includes all the parameters values that we are going to pass into it.

6. On executing, it runs asynchronously and returns a job id. Please note: Workflows can take days to run, depending on the volume of data

```
job: 0000001-140225190405629-oozie-hdp-w
```

7. We can track the progress using this job id from the browser by going to Oozie server

<http://localhost:11000/oozie/v2/job/<job-id-returned>?show=log>

This log file is very similar to what we see on the console window when we run the Map/Reduce job. Keep refreshing this page until you see status "DONE"

8. We can verify that the Oozie created table webdata and populated it using HiveQL.

We ran the Oozie workflow which did this automatically for us. The beauty of this would be when we want to call the table something different, or we want to store it in a different location, or any other type of thing. We can change these things in the job.properties file and that just get propagated through as our parameters. It is a pretty neat way of automating our tasks.

- Oozie allows us to run a series of map-reduce, pig, java & scripts actions a single workflow job
- It allows regular scheduling of workflow jobs
- Oozie does not have explicit support for resource control
- Oozie runs as server (multi user, multi workflows) and not as a standalone
- Oozie uses a SQL database to store internal information about the workflow/coordinator/bundle jobs it runs.
- By using a DB instead keeping the info about all running jobs in memory, Oozie can scale to several thousand concurrent jobs using modest hardware for both the Oozie server and the SQL database. Also, the transactional nature of a SQL DB allows Oozie jobs to continue running through restarts/crashes. Or, in other words, the SQL database is not used by Oozie jobs, it is only use by Oozie to keep track of the Oozie jobs.
- On failover, Oozie running workflows continue running from their current state

You can have a look at this video to get an additional introduction:

<http://www.youtube.com/watch?v=PCGCd771Hz0>

Conclusion

We saw working with Sqoop and Oozie was powerful and not terribly difficult either. And once you spend a bit of time just looking at the syntax, it is quite intimidating because the tools are fairly rudimentary, and you are dealing with command prompts and all that type of things, and even in the PowerShell environment, initially that can be intimidating, but it is amazing how intuitive it is. It is amazing how easy it is to get started with these tools and do some big data processing.

We've covered a lot of ground, and there is lot more ground that you could cover yourself. This course must have given you a good entry point to understanding what the overall story is, how the tools work. And now we leave you to go and do some further analysis and play with these tools.