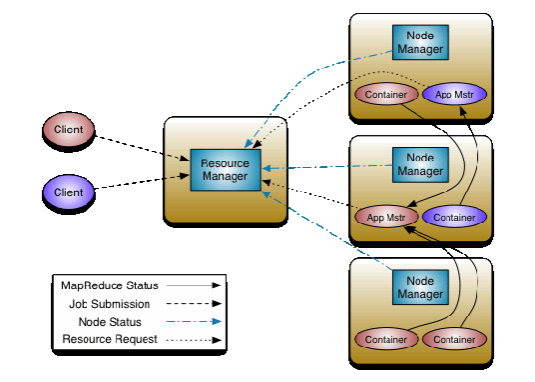
**Hadoop**

Distributed processing framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models

The other name of Hadoop YARN is Yet Another Resource Negotiator (YARN). In a Hadoop cluster, there is a need to manage resources at global level and to manage at a node level.

YARN or ResourceManager is responsible for managing global resources among all nodes. It is the ultimate authority to allocate resources among all applications.

Whereas, NodeManager is responsible for managing resources at machine level. See following image from Apache. Here ResourceManager is YARN.



As per Apache Documentation, there are two main components of ResourceManager/YARN:

* **Scheduler**: Scheduler is responsible for allocating resources. It does not monitor the progress. It performs its scheduling function on the basis of resource requirements of the applications; it does so based on the abstract notion of a resource *Container* which incorporates elements such as memory, cpu, disk, network etc.
* **ApplicationsManager**: ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.

SQOOP

$ sqoop <generic arguments> <import arguments>

SQOOP + MySQL

Start the MySQL service

$sudo service mysqld start

$mysql -u root -p root

Create a Database, if needed in MySQL

$show databases;

$use custdb;

$show tables;

| **Argument** | **Description** |
| --- | --- |
| --connect <jdbc-uri> | Specify JDBC connect string |
| --connection-manager <class-name> | Specify connection manager class to use |
| --driver <class-name> | Manually specify JDBC driver class to use |
| --hadoop-mapred-home <dir> | Override $HADOOP\_MAPRED\_HOME |
| --help | Print usage instructions |
| --password-file | Set path for a file containing the authentication password |
| -P | Read password from console |
| --password <password> | Set authentication password |
| --username <username> | Set authentication username |
| --verbose | Print more information while working |
| --connection-param-file <filename> | Optional properties file that provides connection parameters |
| --relaxed-isolation | Set connection transaction isolation to read uncommitted for the mappers. |

**Import control arguments:**

| **Argument** | **Description** |
| --- | --- |
| --append | Append data to an existing dataset in HDFS |
| --as-avrodatafile | Imports data to Avro Data Files |
| --as-sequencefile | Imports data to SequenceFiles |
| --as-textfile | Imports data as plain text (default) |
| --as-parquetfile | Imports data to Parquet Files |
| --boundary-query <statement> | Boundary query to use for creating splits |
| --columns <col,col,col…> | Columns to import from table |
| --delete-target-dir | Delete the import target directory if it exists |
| --direct | Use direct connector if exists for the database |
| --fetch-size <n> | Number of entries to read from database at once. |
| --inline-lob-limit <n> | Set the maximum size for an inline LOB |
| -m,--num-mappers <n> | Use *n* map tasks to import in parallel |
| -e,--query <statement> | Import the results of *statement*. |
| --split-by <column-name> | Column of the table used to split work units. Cannot be used with --autoreset-to-one-mapper option. |
| --autoreset-to-one-mapper | Import should use one mapper if a table has no primary key and no split-by column is provided. Cannot be used with --split-by <col> option. |
| --table <table-name> | Table to read |
| --target-dir <dir> | HDFS destination dir |
| --warehouse-dir <dir> | HDFS parent for table destination |
| --where <where clause> | WHERE clause to use during import |
| -z,--compress | Enable compression |
| --compression-codec <c> | Use Hadoop codec (default gzip) |
| --null-string <null-string> | The string to be written for a null value for string columns |
| --null-non-string <null-string> | The string to be written for a null value for non-string columns |

**Obtaining SQOOP version**

$sqoop-version

$sqoop version

Both of the above commands yields the same results.

The current sqoop version is 1.4.7

List of databases in MySQL:

Sqoop command:

$sqoop list-databases –connect jdbc:mysql://localhost –username root -P

$ sqoop import \

--connect jdbc:mysql://localhost/custdb \

--username root \

-P \

-m 1 \

--delete-target-dir \

--direct \

-table customer \

To list the databases in MySQL

$sqoop list-databases \

--connect jdbc:mysql://localhost \

--username root \

-P

The option list-databases will fetch all the databases created in mysql

To list the tables of MySQL DB

$sqoop list-tables \

--connect jdbc:mysql://localhost/custdb \

--username root \

--password root

Custdb is the database

Jdbc:mysql is the sql driver

Import data from MySQL to HDFS with one mapper:

$sqoop import \

--connect jdbc:mysql://localhost/custdb \

--username root \

-P \

-table customer \

--targer-dir customer\_bala \

--delete-target-dir \

--direct \

-m 1

Below are some of the common performance improvement techniques for Sqoop:

* split-by and boundary-query
* direct
* fetch-size
* num-mapper

For MySQL or PostgreSQL it relates to bulk loader/unloader utilities (i.e. completetely bypassing JDBC)

* --direct is only supported in mysql and postgresql.
* Sqoop’s direct mode does not support imports of BLOB, CLOB, or LONGVARBINARY columns

--direct - Use direct import fast path

[By supplying the --direct argument](https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#_controlling_the_import_process), you are specifying that Sqoop should attempt the direct import channel. This channel may be higher performance than using JDBC.

**For MySQL:**

MySQL Direct Connector allows faster import and export to/from MySQL using mysqldump and mysqlimport tools functionality instead of SQL selects and inserts

**Overriding parameters:**

For example:

* -D option in command line:

-D mapreduce.map.memory.mb=8192

* The mapred-site.xml can be used to override the default settings
* The map reduce default settings are available in mapred-default.xml

Check whether the below import works?

$sqoop import \

--connect jdbc:mysql://localhost/custdb \

--username root \

--password root \

-table customer \

-m 2;

The above command will throw an error since 2 mappers are used with split-by

Error:

Error during import: No primary key could be found for table customer. Please specify one with –split-by or perform a sequential import with ‘-m 1’

Speculative Execution:

In Hadoop, Speculative Execution is a process that takes place during the slower execution of a task at a node. In this process, the master node starts executing another instance of the same task on another node. And the task, which is finished first is accepted and the execution of the other is stopped by killing that.

**What is UBER mode in Hadoop2?**

Normally mappers and reducers will run by ResourceManager (RM), RM will create separate container for mapper and reducer. Uber configuration, will allow to run mapper and reducers in the same process as the ApplicationMaster (AM).

**Uber jobs :**

Uber jobs are jobs that are executed within the MapReduce ApplicationMaster. Rather then communicate with RM to create the mapper and reducer containers. The AM runs the map and reduce tasks within its own process and avoided the overhead of launching and communicate with remote containers.

**Why**

If you have a small dataset or you want to run MapReduce on small amount of data, Uber configuration will help you out, by reducing additional time that MapReduce normally spends in mapper and reducers phase.

**Can I configure an *Uber* for all MapReduce job?**

As of now, map-only jobs and jobs with one reducer are supported.

**JPS**

JVM Process Status

Tool to check the java processes that are running

**EVAL in SQOOP**

It allows users to execute user-defined queries against respective database servers and preview the result in the console. So, the user can expect the resultant table data to import. Using eval, we can evaluate any type of SQL query that can be either DDL or DML statement.

Example:

$sqoop eval \

--connect jdbc:mysl://localhost/custdb \

--username root \

--password root \

--query “create table test(rank int)”

$sqoop eval \

--connect jdbc:mysl://localhost/custdb \

--username root \

--password root \

--query “select \* from customer”

$sqoop eval \

--connect jdbc:mysl://localhost/custdb \

--username root \

--password root \

--query “insert into customer values (‘Bala’, ‘Muthu’, ‘Erode’, 38, ‘2020-09-28’, 10000)”

--fields-terminated-by

--lines-terminated-by

Changing the delimiters in sqoop

Example:

$sqoop-import \

--connect jdbc:mysql://localhost/custdb \

--username root \

-P \

-table customer\

--fields-terminated-by ‘|’ \

--lines-terminated-by ‘\n’ \

--direct \

-m 1

What is the significance of $conditions clause in sqoop import command?

Example:

select col1, col2 from test\_table where \$CONDITIONS

Sqoop performs highly efficient data transfers by inheriting Hadoop’s parallelism.

* To help Sqoop split your query into multiple chunks that can be transferred in parallel, you need to include the $CONDITIONS placeholder in the where clause of your query.
* Sqoop will automatically substitute this placeholder with the generated conditions specifying which slice of data should be transferred by each individual task.
* While you could skip $CONDITIONS by forcing Sqoop to run only one job using the --num-mappers 1 param‐ eter, such a limitation would have a severe performance impact.

Example:

If you run a parallel import, the map tasks will execute your query with different values substituted in for $CONDITIONS. one mapper may execute "select bla from foo WHERE (id >=0 AND id < 10000)", and the next mapper may execute "select bla from foo WHERE (id >= 10000 AND id < 20000)" and so on.

If you are issuing the query wrapped with double quotes ("), you will have to use \$CONDITIONS instead of just $CONDITIONS to disallow your shell from treating it as a shell variable. For example, a double quoted query may look like: "SELECT \* FROM x WHERE a='foo' AND \$CONDITIONS"

$ sqoop import \

--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \

--split-by a.id --target-dir /user/foo/joinresults

$ sqoop import \

--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \

-m 1 --target-dir /user/foo/joinresults

CDC is Change Data Capture -

The CDC methods will enable you to extract and load only the new or changed records form the source, rather than loading the entire records from the source. Also called as delta or incremental load.

SCD Type 2 (Slowly Changing Dimension Type 2)

This lets you store/preserve the history of changed records of selected dimensions as per your choice. The transaction table / source table will mostly have only the current value and is used in certain cases where in the history of a certain dimension is required for analysis purpose.

**Incremental load**

Sqoop provides an incremental import mode which can be used to retrieve only rows newer than some previously-imported set of rows.

The following arguments control incremental imports:

Incremental import arguments:

| **Argument** | **Description** |
| --- | --- |
| --check-column (col) | Specifies the column to be examined when determining which rows to import. (the column should not be of type CHAR/NCHAR/VARCHAR/VARNCHAR/ LONGVARCHAR/LONGNVARCHAR) |
| --incremental (mode) | Specifies how Sqoop determines which rows are new. Legal values for mode include append and lastmodified. |
| --last-value (value) | Specifies the maximum value of the check column from the previous import. |

Example:

$sqoop import \

--connect jdbc:mysql://localhost/custdb \

--username root \

-P \

-table customer \

-m 1\

-incremental append or lastmodified \

--check-column <column\_name> \

--last-value <value>

--direct

--last-value column should not be of type CHAR/NCHAR/VARCHAR/VARNCHAR/ LONGVARCHAR/LONGNVARCHAR)

--check-column city \

--last-value ‘pune’

Will end up with the below error:

Character column can’t be used to determine which rows to incrementally import

Sqoop jobs:

Imports and exports can be repeatedly performed by issuing the same command multiple times. Especially when using the incremental import capability, this is an expected scenario.

Sqoop allows you to define *saved jobs* which make this process easier. A saved job records the configuration information required to execute a Sqoop command at a later time.

By default, job descriptions are saved to a private repository stored in $HOME/.sqoop/.

Syntax:

$ sqoop job (generic-args) (job-args) [-- [subtool-name] (subtool-args)]

$ sqoop-job (generic-args) (job-args) [-- [subtool-name] (subtool-args)]

Although the Hadoop generic arguments must preceed any job arguments, the job arguments can be entered in any order with respect to one another.

Job management options:

| **Argument** | **Description** |
| --- | --- |
| --create <job-id> | Define a new saved job with the specified job-id (name). A second Sqoop command-line, separated by a -- should be specified; this defines the saved job. |
| --delete <job-id> | Delete a saved job. |
| --exec <job-id> | Given a job defined with --create, run the saved job. |
| --show <job-id> | Show the parameters for a saved job. |
| --list | List all saved jobs |

Creating saved jobs is done with the --create action. This operation requires a -- followed by a tool name and its arguments. The tool and its arguments will form the basis of the saved job. Consider:

$ sqoop job --create myjob -- import --connect jdbc:mysql://example.com/db \

--table mytable

This creates a job named myjob which can be executed later. The job is not run. This job is now available in the list of saved jobs:

$ sqoop job --list

Available jobs:

myjob

We can inspect the configuration of a job with the show action:

$ sqoop job --show myjob

Job: myjob

Tool: import

Options:

----------------------------

direct.import = false

codegen.input.delimiters.record = 0

hdfs.append.dir = false

db.table = mytable

...

And if we are satisfied with it, we can run the job with exec:

$ sqoop job --exec myjob

10/08/19 13:08:45 INFO tool.CodeGenTool: Beginning code generation

...

The exec action allows you to override arguments of the saved job by supplying them after a --. For example, if the database were changed to require a username, we could specify the username and password with:

$ sqoop job --exec myjob -- --username someuser -P

Enter password:

...

Example:

$sqoop job \

--create <job\_name> \

-- \

import \

--connect jdbc:mysql://localhost/custdb \

-table customer \

-m 1 \

-direct \

--target-dir <dir\_name> \

--append

**List of jobs**

$sqoop job --list

**Show jobs**

$sqoop job --show <job\_nam>

**Delete jobs**

$sqoop job --delete <job\_name>

**Execute jobs**

$sqoop job --exec <job\_name>

Export data to MySQL (from HDFS) using sqoop:

Additional use cases – sqoop

* Import all data from the source by deleting target dir (Delete and load).
* Application:
  + When the source data is a rapidly growing small dimension table that undergoes several inserts/updates/deletes.
  + For Eg. Product table that may change regularly and small.
  + sqoop import --connect jdbc:mysql://127.0.0.1/custdb --username root -P -table customer -m 3 \ --split-by custid --target-dir sqoop\_import --delete-target-dir --direct;

**Alternate:**

$sqoop import \

--connect jdbc:mysql://localhost/custdb \

--username root \

-P \

-table customer \

-m 1\

--delete-target-dir \

--direct

[Sqoop : Incremental Imports using Last-Modified mode](https://shalishvj.wordpress.com/2014/08/12/sqoop-incremental-imports-using-last-modified-mode/)

Sqoop is a tool designed to transfer data between Hadoop and relational databases.  
Incremental imports mode can be used to retrieve only rows newer than some previously-imported set of rows.

**Why & When Last-Modified mode ??**

* “lastmodified,” works on time-stamped data.
* use this when**rows** of the source table **may be updated**
* and each such **update** will set the value of a **last-modified** **column to the current timestamp**
* **Rows where the check column holds a timestamp more recent than the timestamp specified with –last-value are imported**

**Note :-**

Oracle Timestamp format : **‘DD-Mon-RR HH24:MI:**SS.FF**’**

Sqoop Timestamp format : **‘YYYY-MM-DD HH24:MI:**SS.FF**’**

Specify  **–last-value** in **Double Quotes**. ie, timestamp in this case.