**What is Sqoop?**

Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and external datastores such as relational databases, enterprise data warehouses.

Sqoop is used to import data from external datastores into Hadoop Distributed File System or related Hadoop eco-systems like Hive and HBase. Similarly, Sqoop can also be used to extract data from Hadoop or its eco-systems and export it to external datastores such as relational databases, enterprise data warehouses. Sqoop works with relational databases such as Teradata, Netezza, Oracle, MySQL, Postgres etc.

**Why is Sqoop used?**

For Hadoop developers, the interesting work starts after data is loaded into HDFS. Developers play around the data in order to find the magical insights concealed in that Big Data. For this, the data residing in the relational database management systems need to be transferred to HDFS, play around the data and might need to transfer back to relational database management systems. In reality of Big Data world, Developers feel the transferring of data between relational database systems and HDFS is not that interesting, tedious but too seldom required. Developers can always write custom scripts to transfer data in and out of Hadoop, but Apache Sqoop provides an alternative.

Sqoop automates most of the process, depends on the database to describe the schema of the data to be imported. Sqoop uses MapReduce framework to import and export the data, which provides parallel mechanism as well as fault tolerance. Sqoop makes developers life easy by providing command line interface. Developers just need to provide basic information like source, destination and database authentication details in the sqoop command. Sqoop takes care of remaining part.

Sqoop provides many salient features like:

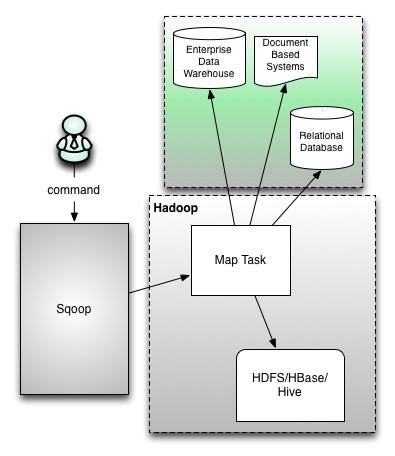
1. Full Load
2. Incremental Load
3. Parallel import/export
4. Import results of SQL query
5. Compression
6. Connectors for all major RDBMS Databases
7. Kerberos Security Integration
8. Load data directly into Hive/Hbase
9. Support for Accumulo

Sqoop is Robust, has great community support and contributions. Sqoop is widely used in most of the Big Data companies to transfer data between relational databases and Hadoop.

**Where is Sqoop used?**

Relational database systems are widely used to interact with the traditional business applications. So, relational database systems has become one of the sources that generate Big Data.

As we are dealing with Big Data, Hadoop stores and processes the Big Data using different processing frameworks like MapReduce, Hive, HBase, Cassandra, Pig etc and storage frameworks like HDFS to achieve benefit of distributed computing and distributed storage. In order to store and analyze the Big Data from relational databases, Data need to be transferred between database systems and Hadoop Distributed File System (HDFS). Here, Sqoop comes into picture. Sqoop acts like a intermediate layer between Hadoop and relational database systems.  You can import data and export data between relational database systems and Hadoop and its eco-systems directly using sqoop.



Sqoop provides command line interface to the end users. Sqoop can also be accessed using Java APIs. Sqoop command submitted by the end user is parsed by Sqoop and launches Hadoop Map only job to import or export data because Reduce phase is required only when aggregations are needed. Sqoop just imports and exports the data; it does not do any aggregations.

Sqoop parses the arguments provided in the command line and prepares the Map job. Map job launch multiple mappers depends on the number defined by user in the command line. For Sqoop import, each mapper task will be assigned with part of data to be imported based on key defined in the command line. Sqoop distributes the input data among the mappers equally to get high performance. Then each mapper creates connection with the database using JDBC and fetches the part of data assigned by Sqoop and writes it into HDFS or Hive or HBase based on the option provided in the command line.

## ****Basic Commands and Syntax for Sqoop****

### ****Sqoop-Import****

Sqoop import command imports a table from an RDBMS to HDFS. Each record from a table is considered as a separate record in HDFS. Records can be stored as text files, or in binary representation as Avro or SequenceFiles.

Syntax:

$ sqoop import --connect --table --username --password --target-dir

--connect        Takes JDBC url and connects to database  
--table             Source table name to be imported  
--username    Username to connect to database  
--password     Password of the connecting user  
--target-dir     Imports data to the specified directory

### ****Importing Selected Data from Table****

Syntax:

$ sqoop import --connect --table --username --password --columns --where

--columns       Selects subset of columns  
--where           Retrieves the data which satisfies the condition

### ****Importing Data from Query****

Syntax:

$ sqoop import --connect --table --username --password --query

--query           Executes the SQL query provided and imports the results

### ****Incremental Exports****

Syntax:

$ sqoop import --connect --table --username --password --incremental --check-column --last-value

Sqoop import supports two types of incremental imports:

1. Append
2. Lastmodified.

Append mode is to be used when new rows are continually being added with increasing values. Column should also be specified which is continually increasing with --check-column. Sqoop imports rows whose value is greater than the one specified with --last-value. Lastmodified mode is to be used when records of the table might be updated, and each such update will set the current timestamp value to a last-modified column. Records whose check column timestamp is more recent than the timestamp specified with --last-value are imported.

Notes:

1. In JDBC connection string, database host shouldn't be used as “localhost” as Sqoop launches mappers on multiple data nodes and the mapper will not able to connect to DB host.
2. “–password” parameter is insecure as any one can read it from command line. –P option can be used, which prompts for password in console. Otherwise, it is recommended to use –password-file pointing to the file containing password (Make sure you have revoked permission to unauthorized users).

Few arguments helpful with Sqoop import:

| **Argument** | **Description** |
| --- | --- |
| --num-mappers,-m | Mappers to Launch |
| --fields-terminated-by | Field Separator |
| --lines-terminated-by | End of line seprator |

### ****Importing Data into Hive****

Below mentioned Hive arguments is used with the sqoop import command to directly load data into Hive:

| **Argument** | **Description** |
| --- | --- |
| --hive-home | Override $HIVE\_HOME path |
| --hive-import | Import tables into Hive |
| --hive-overwrite | Overwrites existing Hive table data |
| --create-hive-table | Creates Hive table and fails if that table already exists |
| --hive-table | Sets the Hive table name to import |
| --hive-drop-import-delims | Drops delimiters like*\n*, *\r*, and *\01* from string fields |
| --hive-delims-replacement | Replaces delimiters like *\n*, *\r*, and *\01* from string fields with user defined delimiters |
| --hive-partition-key | Sets the Hive partition key |
| --hive-partition-value | Sets the Hive partition value |
| --map-column-hive | Overrides default mapping from SQL type datatypes to Hive datatypes |

Syntax:

$ sqoop import --connect --table --username --password --hive-import --hive-table

Specifying --hive-import, Sqoop imports data into Hive table rather than HDFS directory.

### ****Importing Data into HBase****

Below mentioned HBase arguments is used with the sqoop import command to directly load data into HBase:

| **Argument** | **Description** |
| --- | --- |
| --column-family | Sets column family for the import |
| --hbase-create-table | If specified, creates missing HBase tables and fails if already exists |
| --hbase-row-key | Specifies which column to use as the row key |
| --hbase-table | Imports to Hbase table |

Syntax:

$ sqoop import --connect --table --username --password --hbase-table

Specifying –hbase-table, Sqoop will import data into HBase rather than HDFS directory.

### ****Sqoop-Import-all-Tables****

The import-all-tables imports all tables in a RDBMS database to HDFS. Data from each table is stored in a separate directory in HDFS. Following conditions must be met in order to use sqoop-import-all-tables:

1. Each table should have a single-column primary key.

2. You should import all columns of each table.

3. You should not use splitting column, and should not check any conditions using where clause.

Generic Syntax:

$ sqoop import-all-tables (generic args) (import args)

$ sqoop-import-all-tables (generic args) (import args)

Sqoop specific arguments are similar with sqoop-import tool, but few options like --table, --split-by, --columns, and --where arguments are invalid.

Syntax:

$ sqoop-import-all-tables ---connect --username --password

### ****Sqoop-Export****

Sqoop export command exports a set of files in a HDFS directory back to RDBMS tables. The target table should already exist in the database.

Generic Syntax:

$ sqoop export (generic args) (export args)

$ sqoop-export (generic args) (export args)

Sqoop export command prepares INSERT statements with set of input data then hits the database. It is for exporting new records, If the table has unique value constant with primary key, export job fails as the insert statement fails. If you have updates,  you can use --update-key option. Then Sqoop prepares UPDATE statement which updates the existing row, not the INSERT statements as earlier.

Syntax:

$ sqoop-export ---connect --username --password --export-dir

### ****Sqoop-Job****

Sqoop job command allows us to create a job. Job remembers the parameters used to create job, so they can be invoked any time with same arguments.

Generic Syntax:

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

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

Sqoop-job makes work easy when we are using incremental import. The last value imported is stored in the job configuration of the sqoop-job, so for the next execution it directly uses from configuration and imports the data.

Sqoop-job options:

| **Argument** | **Description** |
| --- | --- |
| --create | Defines a new job with the specified job-id (name). Actual sqoop import command should be seperated by “--“ |
| --delete | Deletes a saved job. |
| --exec | Executes the saved job. |
| --show | Show the save job configuration |
| --list | Lists all the saved jobs |

Syntax:

$ sqoop job --create -- import --connect --table

### ****Sqoop-Codegen****

Sqoop-codegen command generates Java class files which encapsulate and interpret imported records. The Java definition of a record is initiated as part of the import process. For example, if Java source is lost, it can be recreated. New versions of a class can be created which use different delimiters between fields, and so on.

Generic Syntax:

$ sqoop codegen (generic args) (codegen args)

$ sqoop-codegen (generic args) (codegen args)

Syntax:

$ sqoop codegen --connect --table

|  |
| --- |
|  |

### ****Sqoop-Eval****

Sqoop-eval command allows users to quickly run simple SQL queries against a database and the results are printed on to the console. Generic Syntax:

$ sqoop eval (generic args) (eval args)

$ sqoop-eval (generic args) (eval args)

Syntax:

$ sqoop eval --connect --query "SQL query"

Using this, users can be sure that they are importing the data as expected.

### ****Sqoop-List-Database****

Used to list all the database available on RDBMS server. Generic Syntax:

$ sqoop list-databases (generic args) (list databases args)

$ sqoop-list-databases (generic args) (list databases args)

Syntax:

$ sqoop list-databases --connect

### ****Sqoop-List-Tables****

Used to list all the tables in a specified database. Generic Syntax:

$ sqoop list-tables (generic args) (list tables args)

$ sqoop-list-tables (generic args) (list tables args)

Syntax:

$ sqoop list-tables –connect