Apache Sqoop

Release

October 31, 2016

Contents

Apache Sqoop is a tool designed for efficiently transferring data betweeen structured, semi-structured and unstructured data sources. Relational databases are examples of structured data sources with well defined schema for the data they store. Cassandra, Hbase are examples of semi-structured data sources and HDFS is an example of unstructured data source that Sqoop can support.

CHAPTER 1

Admin Guide

1.1 Installation

Sqoop ships as one binary package that incorporates two separate parts - client and server.

- **Server** You need to install server on single node in your cluster. This node will then serve as an entry point for all Sqoop clients.
- **Client** Clients can be installed on any number of machines.

1.1.1 Server installation

Copy the Sqoop artifact to the machine where you want to run Sqoop server. The Sqoop server acts as a Hadoop client, therefore Hadoop libraries (Yarn, Mapreduce, and HDFS jar files) and configuration files (`core-site.xml`, `mapreduce-site.xml`, ...) must be available on this node. You do not need to run any Hadoop related services - running the server on a "gateway" node is perfectly fine.

You should be able to list a HDFS for example:

```
hadoop dfs -ls
```

Sqoop currently supports Hadoop version 2.6.0 or later. To install the Sqoop server, decompress the tarball (in a location of your choosing) and set the newly created forder as your working directory.

```
# Decompress Sqoop distribution tarball
tar -xvf sqoop-<version>-bin-hadoop<hadoop-version>.tar.gz

# Move decompressed content to any location
mv sqoop-<version>-bin-hadoop<hadoop version>.tar.gz /usr/lib/sqoop

# Change working directory
cd /usr/lib/sqoop
```

Hadoop dependencies

Sqoop server needs following environmental variables pointing at Hadoop libraries - `$HADOOP_COMMON_HOME`, `$HADOOP_HDFS_HOME`, `$HADOOP_MAPRED_HOME` and `$HADOOP_YARN_HOME`. You have to make sure that those variables are defined and pointing to a valid Hadoop installation. Sqoop server will not start if Hadoop libraries can't be found.

The Sqoop server uses environment variables to find Hadoop libraries. If the environment variable $HADOOP_HOME is set, Sqoop will look for jars in the following locations: $HADOOP_HOME/share/hadoop/common, $HADOOP_HOME/share/hadoop/hdfs, $HADOOP_HOME/share/hadoop/mapreduce and $HADOOP_HOME/share/hadoop/yarn. You can specify where the Sqoop server should look for the common, hdfs, mapreduce, and yarn jars indepently with the $HADOOP_COMMON_HOME, $HADOOP_HDFS_HOME, $HADOOP_MAPRED_HOME and $HADOOP_YARN_HOME environment variables.

```
# Export HADOOP_HOME variable
export HADOOP_HOME=/...

# Or alternatively HADOOP_*_HOME variables
export HADOOP_COMMON_HOME=/...
export HADOOP_HDFS_HOME=/...
export HADOOP_MAPRED_HOME=/...
export HADOOP_YARN_HOME=/...
```

Note: If the environment $HADOOP_HOME is set, Sqoop will usee the following locations: $HADOOP_HOME/share/hadoop/common, $HADOOP_HOME/share/hadoop/hdfs, $HADOOP_HOME/share/hadoop/mapreduce and $HADOOP_HOME/share/hadoop/yarn.

Hadoop configuration

Sqoop server will need to impersonate users to access HDFS and other resources in or outside of the cluster as the user who started given job rather then user who is running the server. You need to configure Hadoop to explicitly allow this impersonation via so called proxyuser system. You need to create two properties in core-site.xml file - hadoop.proxyuser.$SERVER_USER.hosts and hadoop.proxyuser.$SERVER_USER.groups where $SERVER_USER is the user who will be running Sqoop 2 server. In most scenarios configuring * is sufficient. Please refer to Hadoop documentation for details how to use those properties.

Example fragment that needs to be present in core-site.xml file for case when server is running under sqoop2 user:

```
<property>
  <name>hadoop.proxyuser.sqoop2.hosts</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.sqoop2.groups</name>
  <value>*</value>
</property>
```

If you're running Sqoop 2 server under a so called system user (user with ID less then min.user.id - 1000 by default), then YARN will by default refuse to run Sqoop 2 jobs. You will need to add the user name who is running Sqoop 2 server (most likely user sqoop2) to a allowed.system.users property of container-executor.cfg. Please refer to YARN documentation for further details.

Example fragment that needs to be present in container-executor.cfg file for case when server is running under sqoop2 user:

```
allowed.system.users=sqoop2
```

Third party jars

To propagate any third party jars to Sqoop server classpath, create a directory anywhere on the file system and export it's location in `SQOOP_SERVER_EXTRA_LIB` variable.

```
# Create directory for extra jars
mkdir -p /var/lib/sqoop2/

# Copy all your JDBC drivers to this directory
cp mysql-jdbc*.jar /var/lib/sqoop2/
cp postgresql-jdbc*.jar /var/lib/sqoop2/

# And finally export this directory to SQOOP_SERVER_EXTRA_LIB
export SQOOP_SERVER_EXTRA_LIB=/var/lib/sqoop2/
```

Note: Sqoop doesn't ship with any JDBC drivers due to incompatible licenses. You will need to use this mechanism to install all JDBC drivers that are needed.

Configuring `PATH`

All user and administrator facing shell commands are stored in `bin/` directory. It's recommended to add this directory to your `$PATH` for easier execution, for example:

```
PATH=$PATH:`pwd`/bin/
```

The remainder of the Sqoop 2 documentation assumes that the shell commands are in your `$PATH`.

Configuring Server

Server configuration files are stored in `conf` directory. File `sqoop_bootstrap.properties` specifies which configuration provider should be used for loading configuration for rest of Sqoop server. Default value `PropertiesConfigurationProvider` should be sufficient.

Second configuration file called `sqoop.properties` contains remaining configuration properties that can affect Sqoop server. The configuration file is very well documented, so check if all configuration properties fits your environment. Default or very little tweaking should be sufficient in most common cases.

Repository Initialization

The metadata repository needs to be initialized before starting Sqoop 2 server for the first time. Use *Upgrade tool* to initialize the repository:

```
sqoop2-tool upgrade
```

You can verify if everything have been configured correctly using *Verify tool*:

```
sqoop2-tool verify
...
Verification was successful.
Tool class org.apache.sqoop.tools.tool.VerifyTool has finished correctly
```

Server Life Cycle

After installation and configuration you can start Sqoop server with following command:

```
sqoop2-server start
```

You can stop the server using the following command:

```
sqoop2-server stop
```

By default Sqoop server daemon use port `12000`. You can set `org.apache.sqoop.jetty.port` in configuration file `conf/sqoop.properties` to use different port.

1.1.2 Client installation

Just copy Sqoop distribution artifact on target machine and unzip it in desired location. You can start client with following command:

```
sqoop2-shell
```

You can find more documentation for Sqoop shell in Command Line Shell.

Note: Client is not acting as a Hadoop client and thus you do not need to be installed on node with Hadoop libraries and configuration files.

1.2 Tools

Tools are server commands that administrators can execute on the Sqoop server machine in order to perform various maintenance tasks. The tool execution will always perform a given task and finish. There are no long running services implemented as tools.

In order to perform the maintenance task each tool is suppose to do, they need to be executed in exactly the same environment as the main Sqoop server. The tool binary will take care of setting up the `CLASSPATH` and other environmental variables that might be required. However it's up to the administrator himself to run the tool under the same user as is used for the server. This is usually configured automatically for various Hadoop distributions (such as Apache Bigtop).

Note: Running tools while the Sqoop Server is also running is not recommended as it might lead to a data corruption and service disruption.

List of available tools:

- verify
- upgrade

To run the desired tool, execute binary `sqoop2-tool` with desired tool name. For example to run `verify` tool:

```
sqoop2-tool verify
```

Note: Stop the Sqoop Server before running Sqoop tools. Running tools while Sqoop Server is running can lead to a data corruption and service disruption.

1.2.1 Verify tool

The verify tool will verify Sqoop server configuration by starting all subsystems with the exception of servlets and tearing them down.

To run the `verify` tool:

```
sqoop2-tool verify
```

If the verification process succeeds, you should see messages like:

```
Verification was successful.
Tool class org.apache.sqoop.tools.tool.VerifyTool has finished correctly
```

If the verification process will find any inconsistencies, it will print out the following message instead:

```
Verification has failed, please check Server logs for further details.
Tool class org.apache.sqoop.tools.tool.VerifyTool has failed.
```

Further details why the verification has failed will be available in the Sqoop server log - same file as the Sqoop Server logs into.

1.2.2 Upgrade tool

Upgrades all versionable components inside Sqoop2. This includes structural changes inside the repository and stored metadata. Running this tool on Sqoop deployment that was already upgraded will have no effect.

To run the `upgrade` tool:

```
sqoop2-tool upgrade
```

Upon successful upgrade you should see following message:

```
Tool class org.apache.sqoop.tools.tool.UpgradeTool has finished correctly.
```

Execution failure will show the following message instead:

```
Tool class org.apache.sqoop.tools.tool.UpgradeTool has failed.
```

Further details why the upgrade process has failed will be available in the Sqoop server log - same file as the Sqoop Server logs into.

1.2.3 RepositoryDump

Writes the user-created contents of the Sqoop repository to a file in JSON format. This includes connections, jobs and submissions.

To run the `repositorydump` tool:

```
sqoop2-tool repositorydump -o repository.json
```

As an option, the administrator can choose to include sensitive information such as database connection passwords in the file:

```
sqoop2-tool repositorydump -o repository.json --include-sensitive
```

Upon successful execution, you should see the following message:

```
Tool class org.apache.sqoop.tools.tool.RepositoryDumpTool has finished correctly.
```

If repository dump has failed, you will see the following message instead:

```
Tool class org.apache.sqoop.tools.tool.RepositoryDumpTool has failed.
```

Further details why the upgrade process has failed will be available in the Sqoop server log - same file as the Sqoop Server logs into.

1.2.4 RepositoryLoad

Reads a json formatted file created by RepositoryDump and loads to current Sqoop repository.

To run the `repositoryLoad` tool:

```
sqoop2-tool repositoryload -i repository.json
```

Upon successful execution, you should see the following message:

```
Tool class org.apache.sqoop.tools.tool.RepositoryLoadTool has finished correctly.
```

If repository load failed you will see the following message instead:

```
Tool class org.apache.sqoop.tools.tool.RepositoryLoadTool has failed.
```

Or an exception. Further details why the upgrade process has failed will be available in the Sqoop server log - same file as the Sqoop Server logs into.

Note: If the repository dump was created without passwords (default), the connections will not contain a password and the jobs will fail to execute. In that case you'll need to manually update the connections and set the password.

Note: RepositoryLoad tool will always generate new connections, jobs and submissions from the file. Even when an identical objects already exists in repository.

1.2.5 RepositoryEncryption

Please see *Repository Encryption* for more details on repository encryption.

Sometimes we may want to change the password that is used to encrypt our data, generate a new key for our existing password, encrypt an existing unencrypted repository, or decrypt an existing encrypting repository. Sqoop 2 provides the Repository Encryption Tool to allow us to do this.

Before using the tool it is important to shut down the Sqoop 2 server.

All changes that the tool makes occur in a single transaction with the repository, which will prevent leaving the repository in a bad state.

The Repository Encryption Tool is very simple, it uses the exact same configuration specified above (with the exception of `useConf`). Configuration prefixed with a "-F" represents the existing repository state, configuration prefixed with a "-T" represents the desired repository state. If one of these configuration sets is left out that means unencrypted.

Changing the Password

In order to change the password, we need to specify the current configuration with the existing password and the desired configuration with the new password. It looks like this:

```
sqoop.sh tool repositoryencryption \
    -Forg.apache.sqoop.security.repo_encryption.password=old_password \
    -Forg.apache.sqoop.security.repo_encryption.hmac_algorithm=HmacSHA256 \
    -Forg.apache.sqoop.security.repo_encryption.cipher_algorithm=AES \
    -Forg.apache.sqoop.security.repo_encryption.cipher_key_size=16 \
    -Forg.apache.sqoop.security.repo_encryption.cipher_spec=AES/CBC/PKCS5Padding \
    -Forg.apache.sqoop.security.repo_encryption.initialization_vector_size=16 \
    -Forg.apache.sqoop.security.repo_encryption.pbkdf2_algorithm=PBKDF2WithHmacSHA1 \
    -Forg.apache.sqoop.security.repo_encryption.pbkdf2_rounds=4000 \
    -Torg.apache.sqoop.security.repo_encryption.password=new_password \
    -Torg.apache.sqoop.security.repo_encryption.hmac_algorithm=HmacSHA256 \
    -Torg.apache.sqoop.security.repo_encryption.cipher_algorithm=AES \
    -Torg.apache.sqoop.security.repo_encryption.cipher_key_size=16 \
    -Torg.apache.sqoop.security.repo_encryption.cipher_spec=AES/CBC/PKCS5Padding \
    -Torg.apache.sqoop.security.repo_encryption.initialization_vector_size=16 \
    -Torg.apache.sqoop.security.repo_encryption.pbkdf2_algorithm=PBKDF2WithHmacSHA1 \
    -Torg.apache.sqoop.security.repo_encryption.pbkdf2_rounds=4000
```

Generate a New Key for the Existing Password

Just like with the previous scenario you could copy the same configuration twice like this:

```
sqoop.sh tool repositoryencryption \
    -Forg.apache.sqoop.security.repo_encryption.password=password \
    -Forg.apache.sqoop.security.repo_encryption.hmac_algorithm=HmacSHA256 \
    -Forg.apache.sqoop.security.repo_encryption.cipher_algorithm=AES \
    -Forg.apache.sqoop.security.repo_encryption.cipher_key_size=16 \
    -Forg.apache.sqoop.security.repo_encryption.cipher_spec=AES/CBC/PKCS5Padding \
    -Forg.apache.sqoop.security.repo_encryption.initialization_vector_size=16 \
    -Forg.apache.sqoop.security.repo_encryption.pbkdf2_algorithm=PBKDF2WithHmacSHA1 \
    -Forg.apache.sqoop.security.repo_encryption.pbkdf2_rounds=4000 \
    -Torg.apache.sqoop.security.repo_encryption.password=password \
    -Torg.apache.sqoop.security.repo_encryption.hmac_algorithm=HmacSHA256 \
    -Torg.apache.sqoop.security.repo_encryption.cipher_algorithm=AES \
    -Torg.apache.sqoop.security.repo_encryption.cipher_key_size=16 \
    -Torg.apache.sqoop.security.repo_encryption.cipher_spec=AES/CBC/PKCS5Padding \
    -Torg.apache.sqoop.security.repo_encryption.initialization_vector_size=16 \
    -Torg.apache.sqoop.security.repo_encryption.pbkdf2_algorithm=PBKDF2WithHmacSHA1 \
    -Torg.apache.sqoop.security.repo_encryption.pbkdf2_rounds=4000
```

But we do have a shortcut to make this easier:

```
sqoop.sh tool repositoryencryption -FuseConf -TuseConf
```

The `useConf` option will read whatever configuration is already in the configured sqoop properties file and apply it for the specified direction.

Encrypting an Existing Unencrypted Repository

```
sqoop.sh tool repositoryencryption \
    -Torg.apache.sqoop.security.repo_encryption.password=password \
```

```
    -Torg.apache.sqoop.security.repo_encryption.hmac_algorithm=HmacSHA256 \
    -Torg.apache.sqoop.security.repo_encryption.cipher_algorithm=AES \
    -Torg.apache.sqoop.security.repo_encryption.cipher_key_size=16 \
    -Torg.apache.sqoop.security.repo_encryption.cipher_spec=AES/CBC/PKCS5Padding \
    -Torg.apache.sqoop.security.repo_encryption.initialization_vector_size=16 \
    -Torg.apache.sqoop.security.repo_encryption.pbkdf2_algorithm=PBKDF2WithHmacSHA1 \
    -Torg.apache.sqoop.security.repo_encryption.pbkdf2_rounds=4000
```

If the configuration for the encrypted repository has already been written to the sqoop properties file, one can simply execute:

```
sqoop.sh tool repositoryencryption -TuseConf
```

Decrypting an Existing Encrypted Repository

```
sqoop.sh tool repositoryencryption \
    -Forg.apache.sqoop.security.repo_encryption.password=password \
    -Forg.apache.sqoop.security.repo_encryption.hmac_algorithm=HmacSHA256 \
    -Forg.apache.sqoop.security.repo_encryption.cipher_algorithm=AES \
    -Forg.apache.sqoop.security.repo_encryption.cipher_key_size=16 \
    -Forg.apache.sqoop.security.repo_encryption.cipher_spec=AES/CBC/PKCS5Padding \
    -Forg.apache.sqoop.security.repo_encryption.initialization_vector_size=16 \
    -Forg.apache.sqoop.security.repo_encryption.pbkdf2_algorithm=PBKDF2WithHmacSHA1 \
    -Forg.apache.sqoop.security.repo_encryption.pbkdf2_rounds=4000
```

If the configuration for the encrypted repository has not yet been removed from the sqoop properties file, one can simply execute:

```
sqoop.sh tool repositoryencryption -FuseConf
```

1.3 Upgrade

This page describes procedure that you need to take in order to upgrade Sqoop from one release to a higher release. Upgrading both client and server component will be discussed separately.

Note: Only updates from one Sqoop 2 release to another are covered, starting with upgrades from version 1.99.2. This guide do not contain general information how to upgrade from Sqoop 1 to Sqoop 2.

1.3.1 Upgrading Server

As Sqoop server is using a database repository for persisting sqoop entities such as the connector, driver, links and jobs the repository schema might need to be updated as part of the server upgrade. In addition the configs and inputs described by the various connectors and the driver may also change with a new server version and might need a data upgrade.

There are two ways how to upgrade Sqoop entities in the repository, you can either execute upgrade tool or configure the sqoop server to perform all necessary upgrades on start up.

It's strongly advised to back up the repository before moving on to next steps. Backup instructions will vary depending on the repository implementation. For example, using MySQL as a repository will require a different back procedure than Apache Derby. Please follow the repositories' backup procedure.

Upgrading Server using upgrade tool

Preferred upgrade path is to explicitly run the *Upgrade tool*. First step is to however shutdown the server as having both the server and upgrade utility accessing the same repository might corrupt it:

```
sqoop2-server stop
```

When the server has been successfully stopped, you can update the server bits and simply run the upgrade tool:

```
sqoop2-tool upgrade
```

You should see that the upgrade process has been successful:

```
Tool class org.apache.sqoop.tools.tool.UpgradeTool has finished correctly.
```

In case of any failure, please take a look into *Upgrade tool* documentation page.

Upgrading Server on start-up

The capability of performing the upgrade has been built-in to the server, however is disabled by default to avoid any unintentional changes to the repository. You can start the repository schema upgrade procedure by stopping the server:

```
sqoop2-server stop
```

Before starting the server again you will need to enable the auto-upgrade feature that will perform all necessary changes during Sqoop Server start up.

You need to set the following property in configuration file `sqoop.properties` for the repository schema upgrade.

```
org.apache.sqoop.repository.schema.immutable=false
```

You need to set the following property in configuration file `sqoop.properties` for the connector config data upgrade.

```
org.apache.sqoop.connector.autoupgrade=true
```

You need to set the following property in configuration file `sqoop.properties` for the driver config data upgrade.

```
org.apache.sqoop.driver.autoupgrade=true
```

When all properties are set, start the sqoop server using the following command:

```
sqoop2-server start
```

All required actions will be performed automatically during the server bootstrap. It's strongly advised to set all three properties to their original values once the server has been successfully started and the upgrade has completed

1.3.2 Upgrading Client

Client do not require any manual steps during upgrade. Replacing the binaries with updated version is sufficient.

CHAPTER 2

User Guide

2.1 Command Line Shell

Sqoop 2 provides command line shell that is capable of communicating with Sqoop 2 server using REST interface. Client is able to run in two modes - interactive and batch mode. Commands `create`, `update` and `clone` are not currently supported in batch mode. Interactive mode supports all available commands.

You can start Sqoop 2 client in interactive mode using command `sqoop2-shell`:

```
sqoop2-shell
```

Batch mode can be started by adding additional argument representing path to your Sqoop client script:

```
sqoop2-shell /path/to/your/script.sqoop
```

Sqoop client script is expected to contain valid Sqoop client commands, empty lines and lines starting with `#` that are denoting comment lines. Comments and empty lines are ignored, all other lines are interpreted. Example script:

```
# Specify company server
set server --host sqoop2.company.net

# Executing given job
start job --name 1
```

Table of Contents

2.1.1 Resource file

Sqoop 2 client have ability to load resource files similarly as other command line tools. At the beginning of execution Sqoop client will check existence of file `.sqoop2rc` in home directory of currently logged user. If such file exists, it will be interpreted before any additional actions. This file is loaded in both interactive and batch mode. It can be used to execute any batch compatible commands.

Example resource file:

```
# Configure our Sqoop 2 server automatically
set server --host sqoop2.company.net

# Run in verbose mode by default
set option --name verbose --value true
```

2.1.2 Commands

Sqoop 2 contains several commands that will be documented in this section. Each command have one more functions that are accepting various arguments. Not all commands are supported in both interactive and batch mode.

Auxiliary Commands

Auxiliary commands are commands that are improving user experience and are running purely on client side. Thus they do not need working connection to the server.

- `exit` Exit client immediately. This command can be also executed by sending EOT (end of transmission) character. It's CTRL+D on most common Linux shells like Bash or Zsh.
- `history` Print out command history. Please note that Sqoop client is saving history from previous executions and thus you might see commands that you've executed in previous runs.
- `help` Show all available commands with short in-shell documentation.

```
sqoop:000> help
For information about Sqoop, visit: http://sqoop.apache.org/

Available commands:
  exit    (\x  ) Exit the shell
  history (\H  ) Display, manage and recall edit-line history
  help    (\h  ) Display this help message
  set     (\st ) Configure various client options and settings
  show    (\sh ) Display various objects and configuration options
  create  (\cr ) Create new object in Sqoop repository
  delete  (\d  ) Delete existing object in Sqoop repository
  update  (\up ) Update objects in Sqoop repository
  clone   (\cl ) Create new object based on existing one
  start   (\sta) Start job
  stop    (\stp) Stop job
  status  (\stu) Display status of a job
  enable  (\en ) Enable object in Sqoop repository
  disable (\di ) Disable object in Sqoop repository
```

Set Command

Set command allows to set various properties of the client. Similarly as auxiliary commands, set do not require connection to Sqoop server. Set commands is not used to reconfigure Sqoop server.

Available functions:

Function	Description
`server`	Set connection configuration for server
`option`	Set various client side options

Set Server Function

Configure connection to Sqoop server - host port and web application name. Available arguments:

Argument	Default value	Description
-h, --host	localhost	Server name (FQDN) where Sqoop server is running
-p, --port	12000	TCP Port
-w, --webapp	sqoop	Jetty's web application name
-u, --url		Sqoop Server in url format

Example:

```
set server --host sqoop2.company.net --port 80 --webapp sqoop
```

or

```
set server --url http://sqoop2.company.net:80/sqoop
```

Note: When --url option is given, --host, --port or --webapp option will be ignored.

Set Option Function

Configure Sqoop client related options. This function have two required arguments name and value. Name represents internal property name and value holds new value that should be set. List of available option names follows:

Option name	Default value	Description
verbose	false	Client will print additional information if verbose mode is enabled
poll-timeout	10000	Server poll timeout in milliseconds

Example:

```
set option --name verbose --value true
set option --name poll-timeout --value 20000
```

Show Command

Show commands displays various information as described below.

Available functions:

Function	Description
server	Display connection information to the sqoop server (host, port, webapp)
option	Display various client side options
version	Show client build version, with an option -all it shows server build version and supported api versions
connector	Show connector configurable and its related configs
driver	Show driver configurable and its related configs
link	Show links in sqoop
job	Show jobs in sqoop

Show Server Function

Show details about connection to Sqoop server.

Argument	Description
-a, --all	Show all connection related information (host, port, webapp)
-h, --host	Show host
-p, --port	Show port
-w, --webapp	Show web application name

Example:

```
show server --all
```

Show Option Function

Show values of various client side options. This function will show all client options when called without arguments.

Argument	Description
-n, --name	Show client option value with given name

Please check table in *Set Option Function* section to get a list of all supported option names.

Example:

```
show option --name verbose
```

Show Version Function

Show build versions of both client and server as well as the supported rest api versions.

Argument	Description
-a, --all	Show all versions (server, client, api)
-c, --client	Show client build version
-s, --server	Show server build version
-p, --api	Show supported api versions

Example:

```
show version --all
```

Show Connector Function

Show persisted connector configurable and its related configs used in creating associated link and job objects

Argument	Description
-a, --all	Show information for all connectors
-c, --cid <x>	Show information for connector with id <x>

Example:

```
show connector --all or show connector
```

Show Driver Function

Show persisted driver configurable and its related configs used in creating job objects

This function do not have any extra arguments. There is only one registered driver in sqoop

Example:

```
show driver
```

Show Link Function

Show persisted link objects.

Argument	Description
-a, --all	Show all available links
-n, --name <x>	Show link with name <x>

Example:

```
show link --all or show link --name linkName
```

Show Job Function

Show persisted job objects.

Argument	Description
-a, --all	Show all available jobs
-n, --name <x>	Show job with name <x>

Example:

```
show job --all or show job --name jobName
```

Show Submission Function

Show persisted job submission objects.

Argument	Description
-j, --job <x>	Show available submissions for given job name
-d, --detail	Show job submissions in full details

Example:

```
show submission
show submission --j jobName
show submission --job jobName --detail
```

Create Command

Creates new link and job objects. This command is supported only in interactive mode. It will ask user to enter the link config and job configs for from /to and driver when creating link and job objects respectively.

Available functions:

Function	Description
link	Create new link object
job	Create new job object

Create Link Function

Create new link object.

Argument	Description
-c, --connector <x>	Create new link object for connector with name <x>

Example:

```
create link --connector connectorName or create link -c connectorName
```

Create Job Function

Create new job object.

Argument	Description
-f, --from <x>	Create new job object with a FROM link with name <x>
-t, --to <t>	Create new job object with a TO link with name <x>

Example:

```
create job --from fromLinkName --to toLinkName or create job --f fromLinkName --t toLinkName
```

Update Command

Update commands allows you to edit link and job objects. This command is supported only in interactive mode.

Update Link Function

Update existing link object.

Argument	Description
-n, --name <x>	Update existing link with name <x>

Example:

```
update link --name linkName
```

Update Job Function

Update existing job object.

Argument	Description
-n, --name <x>	Update existing job object with name <x>

Example:

```
update job --name jobName
```

Delete Command

Deletes link and job objects from Sqoop server.

Delete Link Function

Delete existing link object.

Argument	Description
-n, --name <x>	Delete link object with name <x>

Example:

```
delete link --name linkName
```

Delete Job Function

Delete existing job object.

Argument	Description
-n, --name <x>	Delete job object with name <x>

Example:

```
delete job --name jobName
```

Clone Command

Clone command will load existing link or job object from Sqoop server and allow user in place updates that will result in creation of new link or job object. This command is not supported in batch mode.

Clone Link Function

Clone existing link object.

Argument	Description
-n, --name <x>	Clone link object with name <x>

Example:

```
clone link --name linkName
```

Clone Job Function

Clone existing job object.

Argument	Description
-n, --name <x>	Clone job object with name <x>

Example:

```
clone job --name jobName
```

Start Command

Start command will begin execution of an existing Sqoop job.

Start Job Function

Start job (submit new submission). Starting already running job is considered as invalid operation.

Argument	Description
-n, --name <x>	Start job with name <x>
-s, --synchronous	Synchoronous job execution

Example:

```
start job --name jobName
start job --name jobName --synchronous
```

Stop Command

Stop command will interrupt an job execution.

Stop Job Function

Interrupt running job.

Argument	Description
-n, --name <x>	Interrupt running job with name <x>

Example:

```
stop job --name jobName
```

Status Command

Status command will retrieve the last status of a job.

Status Job Function

Retrieve last status for given job.

Argument	Description
-n, --name <x>	Retrieve status for job with name <x>

Example:

```
status job --name jobName
```

2.2 Connectors

2.2.1 FTP Connector

The FTP connector supports moving data between an FTP server and other supported Sqoop2 connectors.

Currently only the TO direction is supported to write records to an FTP server. A FROM connector is pending (SQOOP-2127).

Contents

Usage

To use the FTP Connector, create a link for the connector and a job that uses the link.

Link Configuration

Inputs associated with the link configuration include:

Input	Type	Description	Example
FTP server hostname	String	Hostname for the FTP server. *Required*.	ftp.example.com
FTP server port	Integer	Port number for the FTP server. Defaults to 21. *Optional*.	2100
Username	String	The username to provide when connecting to the FTP server. *Required*.	sqoop
Password	String	The password to provide when connecting to the FTP server. *Required*	sqoop

Notes

1. The FTP connector will attempt to connect to the FTP server as part of the link validation process. If for some reason a connection can not be established, you'll see a corresponding warning message.

TO Job Configuration

Inputs associated with the Job configuration for the TO direction include:

Input	Type	Description	Example
Output directory	String	The location on the FTP server that the connector will write files to. *Required*	uploads

Notes

1. The *output directory* value needs to be an existing directory on the FTP server.

Loader

During the *loading* phase, the connector will create uniquely named files in the *output directory* for each partition of data received from the **FROM** connector.

2.2.2 Generic JDBC Connector

The Generic JDBC Connector can connect to any data source that adheres to the **JDBC 4** specification.

Contents

Usage

To use the Generic JDBC Connector, create a link for the connector and a job that uses the link.

Link Configuration

Inputs associated with the link configuration include:

Input	Type	Description	Example
JDBC Driver Class	String	The full class name of the JDBC driver. *Required* and accessible by the Sqoop server.	com.mysql.jdbc.Driver
JDBC Connection String	String	The JDBC connection string to use when connecting to the data source. *Required*. Connectivity upon creation is optional.	jdbc:mysql://localhost/test
Username	String	The username to provide when connecting to the data source. *Optional*. Connectivity upon creation is optional.	sqoop
Password	String	The password to provide when connecting to the data source. *Optional*. Connectivity upon creation is optional.	sqoop
JDBC Connection Properties	Map	A map of JDBC connection properties to pass to the JDBC driver *Optional*.	profileSQL=true&useFastDateParsing=false

FROM Job Configuration

Inputs associated with the Job configuration for the FROM direction include:

Input	Type	Description	Example
Schema name	String	The schema name the table is part of. *Optional*	sqoop
Table name	String	The table name to import data from. *Optional*. See note below.	test
Table SQL statement	String	The SQL statement used to perform a **free form query**. *Optional*. See notes below.	`SELECT COUNT(*) FROM test ${CONDITIONS}`
Table column names	String	Columns to extract from the JDBC data source. *Optional* Comma separated list of columns.	col1,col2
Partition column name	Map	The column name used to partition the data transfer process. *Optional*. Defaults to table's first column of primary key.	col1
Null value allowed for the partition column	Boolean	True or false depending on whether NULL values are allowed in data of the Partition column. *Optional*.	true
Boundary query	String	The query used to define an upper and lower boundary when partitioning. *Optional*.	

Notes

1. *Table name* and *Table SQL statement* are mutually exclusive. If *Table name* is provided, the *Table SQL statement* should not be provided. If *Table SQL statement* is provided then *Table name* should not be provided.
2. *Table column names* should be provided only if *Table name* is provided.
3. If there are columns with similar names, column aliases are required. For example: `SELECT table1.id as "i", table2.id as "j" FROM table1 INNER JOIN table2 ON table1.id = table2.id`.

TO Job Configuration

Inputs associated with the Job configuration for the TO direction include:

Input	Type	Description	Example
Schema name	String	The schema name the table is part of. *Optional*	sqoop
Table name	String	The table name to import data from. *Optional*. See note below.	test
Table SQL statement	String	The SQL statement used to perform a **free form query**. *Optional*. See note below.	`INSERT INTO test (col1, col2) VALUES (?, ?)`
Table column names	String	Columns to insert into the JDBC data source. *Optional* Comma separated list of columns.	col1,col2
Stage table name	String	The name of the table used as a *staging table*. *Optional*.	staging
Should clear stage table	Boolean	True or false depending on whether the staging table should be cleared after the data transfer has finished. *Optional*.	true

Notes

1. *Table name* and *Table SQL statement* are mutually exclusive. If *Table name* is provided, the *Table SQL statement* should not be provided. If *Table SQL statement* is provided then *Table name* should not be provided.

2. *Table column names* should be provided only if *Table name* is provided.

Partitioner

The Generic JDBC Connector partitioner generates conditions to be used by the extractor. It varies in how it partitions data transfer based on the partition column data type. Though, each strategy roughly takes on the following form:

```
(upper boundary - lower boundary) / (max partitions)
```

By default, the *primary key* will be used to partition the data unless otherwise specified.

The following data types are currently supported:

1. TINYINT
2. SMALLINT
3. INTEGER
4. BIGINT
5. REAL
6. FLOAT
7. DOUBLE
8. NUMERIC
9. DECIMAL
10. BIT
11. BOOLEAN
12. DATE
13. TIME
14. TIMESTAMP
15. CHAR
16. VARCHAR
17. LONGVARCHAR

Extractor

During the *extraction* phase, the JDBC data source is queried using SQL. This SQL will vary based on your configuration.

- If *Table name* is provided, then the SQL statement generated will take on the form `SELECT * FROM <table name>`.
- If *Table name* and *Columns* are provided, then the SQL statement generated will take on the form `SELECT <columns> FROM <table name>`.
- If *Table SQL statement* is provided, then the provided SQL statement will be used.

The conditions generated by the *partitioner* are appended to the end of the SQL query to query a section of data.

The Generic JDBC connector extracts CSV data usable by the *CSV Intermediate Data Format*.

Loader

During the *loading* phase, the JDBC data source is queried using SQL. This SQL will vary based on your configuration.

- If *Table name* is provided, then the SQL statement generated will take on the form `INSERT INTO <table name> (col1, col2, ...) VALUES (?,?,..)`.
- If *Table name* and *Columns* are provided, then the SQL statement generated will take on the form `INSERT INTO <table name> (<columns>) VALUES (?,?,..)`.
- If *Table SQL statement* is provided, then the provided SQL statement will be used.

This connector expects to receive CSV data consumable by the *CSV Intermediate Data Format*.

Destroyers

The Generic JDBC Connector performs two operations in the destroyer in the TO direction:

1. Copy the contents of the staging table to the desired table.
2. Clear the staging table.

No operations are performed in the FROM direction.

2.2.3 HDFS Connector

Contents

Usage

To use the HDFS Connector, create a link for the connector and a job that uses the link.

Link Configuration

Inputs associated with the link configuration include:

Input	Type	Description	Example
URI	String	The URI of the HDFS File System. *Optional*. See note below.	hdfs://example.com:8020/
Configuration directory	String	Path to the clusters configuration directory. *Optional*.	/etc/conf/hadoop

Notes

1. The specified URI will override the declared URI in your configuration.

FROM Job Configuration

Inputs associated with the Job configuration for the FROM direction include:

Input	Type	Description	Example
Input directory	String	The location in HDFS that the connector should look for files in. *Required*. See note below.	/tmp/sqoop2/hdfs
Null value	String	The value of NULL in the contents of each file extracted. *Optional*. See note below.	N
Override null value	Boolean	Tells the connector to replace the specified NULL value. *Optional*. See note below.	true

Notes

1. All files in *Input directory* will be extracted.
2. *Null value* and *override null value* should be used in conjunction. If *override null value* is not set to true, then *null value* will not be used when extracting data.

TO Job Configuration

Inputs associated with the Job configuration for the TO direction include:

Input	Type	Description	Example
Output directory	String	The location in HDFS that the connector will load files to. *Optional*	/tmp/sqoop2/hdfs
Output format	Enum	The format to output data to. *Optional*. See note below.	CSV
Compression	Enum	Compression class. *Optional*. See note below.	GZIP
Custom compression	String	Custom compression class. *Optional* Comma separated list of columns.	org.apache.sqoop.SqoopCompression
Null value	String	The value of NULL in the contents of each file loaded. *Optional*. See note below.	N
Override null value	Boolean	Tells the connector to replace the specified NULL value. *Optional*. See note below.	true
Append mode	Boolean	Append to an existing output directory. *Optional*.	true

Notes

1. *Output format* only supports CSV at the moment.
2. *Compression* supports all Hadoop compression classes.
3. *Null value* and *override null value* should be used in conjunction. If *override null value* is not set to true, then *null value* will not be used when loading data.

Partitioner

The HDFS Connector partitioner partitions based on total blocks in all files in the specified input directory. Blocks will try to be placed in splits based on the *node* and *rack* they reside in.

Extractor

During the *extraction* phase, the FileSystem API is used to query files from HDFS. The HDFS cluster used is the one defined by:

1. The HDFS URI in the link configuration
2. The Hadoop configuration in the link configuration
3. The Hadoop configuration used by the execution framework

The format of the data must be CSV. The NULL value in the CSV can be chosen via *null value*. For example:

```
1,\N
2,null
3,NULL
```

In the above example, if *null value* is set to N, then only the first row's NULL value will be inferred.

Loader

During the *loading* phase, HDFS is written to via the FileSystem API. The number of files created is equal to the number of loads that run. The format of the data currently can only be CSV. The NULL value in the CSV can be chosen via *null value*. For example:

Id	Value
1	NULL
2	value

If *null value* is set to N, then here's how the data will look like in HDFS:

```
1,\N
2,value
```

Destroyers

The HDFS TO destroyer moves all created files to the proper output directory.

2.2.4 Kafka Connector

Currently, only the TO direction is supported.

Contents

Usage

To use the Kafka Connector, create a link for the connector and a job that uses the link.

Link Configuration

Inputs associated with the link configuration include:

Input	Type	Description	Example
Broker list	String	Comma separated list of kafka brokers. *Required.*	example.com:10000,example.com:11000
Zookeeper connection	String	Comma separated list of zookeeper servers in your quorum. *Required.*	/etc/conf/hadoop

TO Job Configuration

Inputs associated with the Job configuration for the FROM direction include:

Input	Type	Description	Example
topic	String	The Kafka topic to transfer to. *Required.*	my topic

Loader

During the *loading* phase, Kafka is written to directly from each loader. The order in which data is loaded into Kafka is not guaranteed.

2.2.5 Kite Connector

Contents

Usage

To use the Kite Connector, create a link for the connector and a job that uses the link. For more information on Kite, checkout the kite documentation: http://kitesdk.org/docs/1.0.0/Kite-SDK-Guide.html.

Link Configuration

Inputs associated with the link configuration include:

Input	Type	Description	Example
authority	String	The authority of the kite dataset. *Optional.* See note below.	hdfs://example.com:8020/

Notes

1. The authority is useful for specifying Hive metastore or HDFS URI.

FROM Job Configuration

Inputs associated with the Job configuration for the FROM direction include:

Input	Type	Description	Example
URI	String	The Kite dataset URI to use. *Required*. See notes below.	dataset:hdfs:/tmp/ns/ds

Notes

1. The URI and the authority from the link configuration will be merged to create a complete dataset URI internally. If the given dataset URI contains authority, the authority from the link configuration will be ignored.
2. Only *hdfs* and *hive* are supported currently.

TO Job Configuration

Inputs associated with the Job configuration for the TO direction include:

Input	Type	Description	Example
URI	String	The Kite dataset URI to use. *Required*. See note below.	dataset:hdfs:/tmp/ns/ds
File format	Enum	The format of the data the kite dataset should write out. *Optional*. See note below.	PARQUET

Notes

1. The URI and the authority from the link configuration will be merged to create a complete dataset URI internally. If the given dataset URI contains authority, the authority from the link configuration will be ignored.
2. Only *hdfs* and *hive* are supported currently.

Partitioner

The kite connector only creates one partition currently.

Extractor

During the *extraction* phase, Kite is used to query a dataset. Since there is only one dataset to query, only a single reader is created to read the dataset.

NOTE: The avro schema kite generates will be slightly different than the original schema. This is because avro identifiers have strict naming requirements.

Loader

During the *loading* phase, Kite is used to write several temporary datasets. The number of temporary datasets is equivalent to the number of *loaders* that are being used.

Destroyers

The Kite connector TO destroyer merges all the temporary datasets into a single dataset.

2.2.6 SFTP Connector

The SFTP connector supports moving data between a Secure File Transfer Protocol (SFTP) server and other supported Sqoop2 connectors.

Currently only the TO direction is supported to write records to an SFTP server. A FROM connector is pending (SQOOP-2218).

Contents

Usage

Before executing a Sqoop2 job with the SFTP connector, set **mapreduce.task.classpath.user.precedence** to true in the Hadoop cluster config, for example:

```
<property>
  <name>mapreduce.task.classpath.user.precedence</name>
  <value>true</value>
</property>
```

This is required since the SFTP connector uses the JSch library (http://www.jcraft.com/jsch/) to provide SFTP functionality. Unfortunately Hadoop currently ships with an earlier version of this library which causes an issue with some SFTP servers. Setting this property ensures that the current version of the library packaged with this connector will appear first in the classpath.

To use the SFTP Connector, create a link for the connector and a job that uses the link.

Link Configuration

Inputs associated with the link configuration include:

Input	Type	Description	Example
SFTP server hostname	String	Hostname for the SFTP server. *Required*.	sftp.example.com
SFTP server port	Integer	Port number for the SFTP server. Defaults to 22. *Optional*.	2220
Username	String	The username to provide when connecting to the SFTP server. *Required*.	sqoop
Password	String	The password to provide when connecting to the SFTP server. *Required*	sqoop

Notes

1. The SFTP connector will attempt to connect to the SFTP server as part of the link validation process. If for some reason a connection can not be established, you'll see a corresponding error message.
2. Note that during connection, the SFTP connector explictly disables *StrictHostKeyChecking* to avoid "Unknown-HostKey" errors.

TO Job Configuration

Inputs associated with the Job configuration for the TO direction include:

Input	Type	Description	Example
Output directory	String	The location on the SFTP server that the connector will write files to. *Required*	uploads

Notes

1. The *output directory* value needs to be an existing directory on the SFTP server.

Loader

During the *loading* phase, the connector will create uniquely named files in the *output directory* for each partition of data received from the **FROM** connector.

2.3 Examples

This section contains various examples how Sqoop can be configured for various use cases.

2.3.1 S3 Import to HDFS

Contents

This section contains detailed description for example use case of transferring data from S3 to HDFS.

Use case

You have directory on S3 where some external process is creating new text files. New files are added to this directory, but existing files are never altered. They can only be removed after some period of time. Data from all new files needs to be transferred to a single HDFS directory. Preserving file names is not required and multiple source files can be merged to single file on HDFS.

Configuration

We will use HDFS connector for both `From` and `To` sides of the data transfer. In order to create link for S3 you need to have S3 bucket name and S3 access and secret keys. Please follow S3 documentation to retrieve S3 credentials if you don't have them already.

```
sqoop:000> create link -c hdfs-connector
```

- Our example uses `s3link` for the link name
- Specify HDFS URI in form of `s3a://$BUCKET_NAME` where `$BUCKET_NAME` is name of the S3 bucket
- Use Override configuration option and specify `fs.s3a.access.key` and `fs.s3a.secret.key` with your S3 access and secret key respectively.

Next step is to create link for HDFS

```
sqoop:000> create link -c hdfs-connector
```

Our example uses hdfslink for the link name If your Sqoop server is running on node that has HDFS and mapreduce client configuration deployed, you can safely keep all options blank and use defaults for them.

With having links for both HDFS and S3, you can create job that will transfer data from S3 to HDFS:

```
sqoop:000> create job -f s3link -t hdfslink
```

- Our example uses `s3import` for the job name
- Input directory should point to a directory inside your S3 bucket where new files are generated
- Make sure to choose mode `NEW_FILES` for Incremental type
- Final destination for the imported files can be specified in Output directory
- Make sure to enable Append mode, so that Sqoop can upload newly created files to the same directory on HDFS
- Configure the remaining options as you see fit

Then finally you can start the job by issuing following command:

```
sqoop:000> start job -j s3import
```

You can run the job `s3import` periodically and only newly created files will be transferred.

2.4 Sqoop 5 Minutes Demo

This page will walk you through the basic usage of Sqoop. You need to have installed and configured Sqoop server and client in order to follow this guide. Installation procedure is described in Installation. Please note that exact output shown in this page might differ from yours as Sqoop evolves. All major information should however remain the same.

Sqoop uses unique names or persistent ids to identify connectors, links, jobs and configs. We support querying a entity by its unique name or by its perisent database Id.

2.4.1 Starting Client

Start client in interactive mode using following command:

```
sqoop2-shell
```

Configure client to use your Sqoop server:

```
sqoop:000> set server --host your.host.com --port 12000 --webapp sqoop
```

Verify that connection is working by simple version checking:

```
sqoop:000> show version --all
client version:
  Sqoop 2.0.0-SNAPSHOT source revision 418c5f637c3f09b94ea7fc3b0a4610831373a25f
  Compiled by vbasavaraj on Mon Nov  3 08:18:21 PST 2014
server version:
  Sqoop 2.0.0-SNAPSHOT source revision 418c5f637c3f09b94ea7fc3b0a4610831373a25f
  Compiled by vbasavaraj on Mon Nov  3 08:18:21 PST 2014
API versions:
  [v1]
```

You should received similar output as shown above describing the sqoop client build version, the server build version and the supported versions for the rest API.

You can use the help command to check all the supported commands in the sqoop shell.

```
sqoop:000> help
For information about Sqoop, visit: http://sqoop.apache.org/

Available commands:
  exit    (\x  ) Exit the shell
  history (\H  ) Display, manage and recall edit-line history
  help    (\h  ) Display this help message
  set     (\st ) Configure various client options and settings
  show    (\sh ) Display various objects and configuration options
  create  (\cr ) Create new object in Sqoop repository
  delete  (\d  ) Delete existing object in Sqoop repository
  update  (\up ) Update objects in Sqoop repository
  clone   (\cl ) Create new object based on existing one
  start   (\sta) Start job
  stop    (\stp) Stop job
  status  (\stu) Display status of a job
  enable  (\en ) Enable object in Sqoop repository
  disable (\di ) Disable object in Sqoop repository
```

2.4.2 Creating Link Object

Check for the registered connectors on your Sqoop server:

```
sqoop:000> show connector
+------------------------+----------------+----------------------------------------------------------+---
|          Name          |    Version     |                          Class                           | Su
+------------------------+----------------+----------------------------------------------------------+---
| hdfs-connector         | 2.0.0-SNAPSHOT | org.apache.sqoop.connector.hdfs.HdfsConnector            | FR
| generic-jdbc-connector | 2.0.0-SNAPSHOT | org.apache.sqoop.connector.jdbc.GenericJdbcConnector     | FR
+------------------------+----------------+----------------------------------------------------------+---
```

Our example contains two connectors. The `generic-jdbc-connector` is a basic connector relying on the Java JDBC interface for communicating with data sources. It should work with the most common databases that are providing JDBC drivers. Please note that you must install JDBC drivers separately. They are not bundled in Sqoop due to incompatible licenses.

Generic JDBC Connector in our example has a name `generic-jdbc-connector` and we will use this value to create new link object for this connector. Note that the link name should be unique.

```
sqoop:000> create link -connector generic-jdbc-connector
Creating link for connector with name generic-jdbc-connector
Please fill following values to create new link object
Name: First Link

Link configuration
JDBC Driver Class: com.mysql.jdbc.Driver
JDBC Connection String: jdbc:mysql://mysql.server/database
Username: sqoop
Password: *****
JDBC Connection Properties:
There are currently 0 values in the map:
entry#protocol=tcp
New link was successfully created with validation status OK name First Link
```

Our new link object was created with assigned name First Link.

In the `show connector -all` we see that there is a hdfs-connector registered. Let us create another link object but this time for the hdfs-connector instead.

```
sqoop:000> create link -connector hdfs-connector
Creating link for connector with name hdfs-connector
Please fill following values to create new link object
Name: Second Link

Link configuration
HDFS URI: hdfs://nameservice1:8020/
New link was successfully created with validation status OK and name Second Link
```

2.4.3 Creating Job Object

Connectors implement the `From` for reading data from and/or `To` for writing data to. Generic JDBC Connector supports both of them List of supported directions for each connector might be seen in the output of `show connector -all` command above. In order to create a job we need to specifiy the `From` and `To` parts of the job uniquely identified by their link Ids. We already have 2 links created in the system, you can verify the same with the following command

```
sqoop:000> show link --all
2 link(s) to show:
link with name First Link (Enabled: true, Created by root at 11/4/14 4:27 PM, Updated by root at 11/4
Using Connector with name generic-jdbc-connector
  Link configuration
    JDBC Driver Class: com.mysql.jdbc.Driver
    JDBC Connection String: jdbc:mysql://mysql.ent.cloudera.com/sqoop
    Username: sqoop
    Password:
    JDBC Connection Properties:
      protocol = tcp
link with name Second Link (Enabled: true, Created by root at 11/4/14 4:38 PM, Updated by root at 11/
Using Connector with name hdfs-connector
  Link configuration
    HDFS URI: hdfs://nameservice1:8020/
```

Next, we can use the two link names to associate the `From` and `To` for the job.

```
 sqoop:000> create job -f "First Link" -t "Second Link"
 Creating job for links with from name First Link and to name Second Link
```

```
 Please fill following values to create new job object
 Name: Sqoopy

 FromJob configuration

  Schema name:(Required)sqoop
  Table name:(Required)sqoop
  Table SQL statement:(Optional)
  Table column names:(Optional)
  Partition column name:(Optional) id
  Null value allowed for the partition column:(Optional)
  Boundary query:(Optional)

ToJob configuration

  Output format:
   0 : TEXT_FILE
   1 : SEQUENCE_FILE
  Choose: 0
  Compression format:
   0 : NONE
   1 : DEFAULT
   2 : DEFLATE
   3 : GZIP
   4 : BZIP2
   5 : LZO
   6 : LZ4
   7 : SNAPPY
   8 : CUSTOM
  Choose: 0
  Custom compression format:(Optional)
  Output directory:(Required)/root/projects/sqoop

  Driver Config
  Extractors:(Optional) 2
  Loaders:(Optional) 2
  New job was successfully created with validation status OK  and name jobName
```

Our new job object was created with assigned name Sqoopy. Note that if null value is allowed for the partition column, at least 2 extractors are needed for Sqoop to carry out the data transfer. On specifying 1 extractor in this scenario, Sqoop shall ignore this setting and continue with 2 extractors.

2.4.4 Start Job (a.k.a Data transfer)

You can start a sqoop job with the following command:

```
sqoop:000> start job -name Sqoopy
Submission details
Job Name: Sqoopy
Server URL: http://localhost:12000/sqoop/
Created by: root
Creation date: 2014-11-04 19:43:29 PST
Lastly updated by: root
External ID: job_1412137947693_0001
  http://vbsqoop-1.ent.cloudera.com:8088/proxy/application_1412137947693_0001/
2014-11-04 19:43:29 PST: BOOTING  - Progress is not available
```

You can iteratively check your running job status with `status job` command:

```
sqoop:000> status job -n Sqoopy
Submission details
Job Name: Sqoopy
Server URL: http://localhost:12000/sqoop/
Created by: root
Creation date: 2014-11-04 19:43:29 PST
Lastly updated by: root
External ID: job_1412137947693_0001
  http://vbsqoop-1.ent.cloudera.com:8088/proxy/application_1412137947693_0001/
2014-11-04 20:09:16 PST: RUNNING  - 0.00 %
```

Alternatively you can start a sqoop job and observe job running status with the following command:

```
sqoop:000> start job -n Sqoopy -s
Submission details
Job Name: Sqoopy
Server URL: http://localhost:12000/sqoop/
Created by: root
Creation date: 2014-11-04 19:43:29 PST
Lastly updated by: root
External ID: job_1412137947693_0001
  http://vbsqoop-1.ent.cloudera.com:8088/proxy/application_1412137947693_0001/
2014-11-04 19:43:29 PST: BOOTING  - Progress is not available
2014-11-04 19:43:39 PST: RUNNING  - 0.00 %
2014-11-04 19:43:49 PST: RUNNING  - 10.00 %
```

And finally you can stop running the job at any time using `stop job` command:

```
sqoop:000> stop job -n Sqoopy
```

CHAPTER 3

Developer Guide

3.1 Building Sqoop2 from source code

This guide will show you how to build Sqoop2 from source code. Sqoop is using maven as build system. You you will need to use at least version 3.0 as older versions will not work correctly. All other dependencies will be downloaded by maven automatically. With exception of special JDBC drivers that are needed only for advanced integration tests.

3.1.1 Downloading source code

Sqoop project is using git as a revision control system hosted at Apache Software Foundation. You can clone entire repository using following command:

```
git clone https://git-wip-us.apache.org/repos/asf/sqoop.git sqoop2
```

Sqoop2 is currently developed in special branch `sqoop2` that you need to check out after clone:

```
cd sqoop2
git checkout sqoop2
```

3.1.2 Building project

You can use usual maven targets like `compile` or `package` to build the project. Sqoop supports one major Hadoop revision at the moment - 2.x. As compiled code for one Hadoop major version can't be used on another, you must compile Sqoop against appropriate Hadoop version.

```
mvn compile
```

Maven target `package` can be used to create Sqoop packages similar to the ones that are officially available for download. Sqoop will build only source tarball by default. You need to specify `-Pbinary` to build binary distribution.

```
mvn package -Pbinary
```

3.1.3 Running tests

Sqoop supports two different sets of tests. First smaller and much faster set is called **unit tests** and will be executed on maven target `test`. Second larger set of **integration tests** will be executed on maven target `integration-test`. Please note that integration tests might require manual steps for installing various JDBC drivers into your local maven cache.

Example for running unit tests:

```
mvn test
```

Example for running integration tests:

```
mvn integration-test
```

For the **unit tests**, there are two helpful profiles: **fast** and **slow**. The **fast** unit tests do not start or use any services. The **slow** unit tests, may start services or use an external service (ie. MySQL).

```
mvn test -Pfast,hadoop200
mvn test -Pslow,hadoop200
```

3.2 Sqoop Java Client API Guide

This document will explain how to use Sqoop Java Client API with external application. Client API allows you to execute the functions of sqoop commands. It requires Sqoop Client JAR and its dependencies.

The main class that provides wrapper methods for all the supported operations is the

```
public class SqoopClient {
  ...
}
```

Java Client API is explained using Generic JDBC Connector example. Before executing the application using the sqoop client API, check whether sqoop server is running.

3.2.1 Workflow

Given workflow has to be followed for executing a sqoop job in Sqoop server.

1. Create LINK object for a given connector name - Creates Link object and returns it
2. Create a JOB for a given “from” and “to” link name - Create Job object and returns it
3. Start the JOB for a given job name - Start Job on the server and creates a submission record

3.2.2 Project Dependencies

Here given maven dependency

```
<dependency>
  <groupId>org.apache.sqoop</groupId>
    <artifactId>sqoop-client</artifactId>
    <version>${requestedVersion}</version>
</dependency>
```

3.2.3 Initialization

First initialize the SqoopClient class with server URL as argument.

```
String url = "http://localhost:12000/sqoop/";
SqoopClient client = new SqoopClient(url);
```

Server URL value can be modfied by setting value to setServerUrl(String) method

```
client.setServerUrl(newUrl);
```

3.2.4 Link

Connectors provide the facility to interact with many data sources and thus can be used as a means to transfer data between them in Sqoop. The registered connector implementation will provide logic to read from and/or write to a data source that it represents. A connector can have one or more links associated with it. The java client API allows you to create, update and delete a link for any registered connector. Creating or updating a link requires you to populate the Link Config for that particular connector. Hence the first thing to do is get the list of registered connectors and select the connector for which you would like to create a link. Then you can get the list of all the config/inputs using *Display Config and Input Names For Connector* for that connector.

Save Link

First create a new link by invoking `createLink(connectorName)` method with connector name and it returns a MLink object with dummy id and the unfilled link config inputs for that connector. Then fill the config inputs with relevant values. Invoke `saveLink` passing it the filled MLink object.

```
// create a placeholder for link
MLink link = client.createLink("connectorName");
link.setName("Vampire");
link.setCreationUser("Buffy");
MLinkConfig linkConfig = link.getConnectorLinkConfig();
// fill in the link config values
linkConfig.getStringInput("linkConfig.connectionString").setValue("jdbc:mysql://localhost/my");
linkConfig.getStringInput("linkConfig.jdbcDriver").setValue("com.mysql.jdbc.Driver");
linkConfig.getStringInput("linkConfig.username").setValue("root");
linkConfig.getStringInput("linkConfig.password").setValue("root");
// save the link object that was filled
Status status = client.saveLink(link);
if(status.canProceed()) {
 System.out.println("Created Link with Link Name : " + link.getName());
} else {
 System.out.println("Something went wrong creating the link");
}
```

`status.canProceed()` returns true if status is OK or a WARNING. Before sending the status, the link config values are validated using the corresponding validator associated with th link config inputs.

On successful execution of the saveLink method, new link name is assigned to the link object else an exception is thrown. `link.getName()` method returns the unique name for this object persisted in the sqoop repository.

User can retrieve a link using the following methods

Method	Description
`getLink(linkName)`	Returns a link by name
`getLinks()`	Returns list of links in the sqoop

3.2.5 Job

A sqoop job holds the `From` and `To` parts for transferring data from the `From` data source to the `To` data source. Both the `From` and the `To` are uniquely identified by their corresponding connector Link Ids. i.e when creating a job we

have to specifiy the `FromLinkId` and the `ToLinkId`. Thus the pre-requisite for creating a job is to first create the links as described above.

Once the link names for the `From` and `To` are given, then the job configs for the associated connector for the link object have to be filled. You can get the list of all the from and to job config/inputs using *Display Config and Input Names For Connector* for that connector. A connector can have one or more links. We then use the links in the `From` and `To` direction to populate the corresponding `MFromConfig` and `MToConfig` respectively.

In addition to filling the job configs for the `From` and the `To` representing the link, we also need to fill the driver configs that control the job execution engine environment. For example, if the job execution engine happens to be the MapReduce we will specifiy the number of mappers to be used in reading data from the `From` data source.

Save Job

Here is the code to create and then save a job

```
String url = "http://localhost:12000/sqoop/";
SqoopClient client = new SqoopClient(url);
//Creating dummy job object
MJob job = client.createJob("fromLinkName", "toLinkName");
job.setName("Vampire");
job.setCreationUser("Buffy");
// set the "FROM" link job config values
MFromConfig fromJobConfig = job.getFromJobConfig();
fromJobConfig.getStringInput("fromJobConfig.schemaName").setValue("sqoop");
fromJobConfig.getStringInput("fromJobConfig.tableName").setValue("sqoop");
fromJobConfig.getStringInput("fromJobConfig.partitionColumn").setValue("id");
// set the "TO" link job config values
MToConfig toJobConfig = job.getToJobConfig();
toJobConfig.getStringInput("toJobConfig.outputDirectory").setValue("/usr/tmp");
// set the driver config values
MDriverConfig driverConfig = job.getDriverConfig();
driverConfig.getStringInput("throttlingConfig.numExtractors").setValue("3");

Status status = client.saveJob(job);
if(status.canProceed()) {
 System.out.println("Created Job with Job Name: "+ job.getName());
} else {
 System.out.println("Something went wrong creating the job");
}
```

User can retrieve a job using the following methods

Method	Description
`getJob(jobName)`	Returns a job by name
`getJobs()`	Returns list of jobs in the sqoop

List of status codes

Function	Description
`OK`	There are no issues, no warnings.
`WARNING`	Validated entity is correct enough to be proceed. Not a fatal error
`ERROR`	There are serious issues with validated entity. We can't proceed until reported issues will be resolved.

View Error or Warning valdiation message

In case of any WARNING AND ERROR status, user has to iterate the list of validation messages.

```
printMessage(link.getConnectorLinkConfig().getConfigs());

private static void printMessage(List<MConfig> configs) {
  for(MConfig config : configs) {
    List<MInput<?>> inputlist = config.getInputs();
    if (config.getValidationMessages() != null) {
     // print every validation message
     for(Message message : config.getValidationMessages()) {
      System.out.println("Config validation message: " + message.getMessage());
     }
    }
    for (MInput minput : inputlist) {
      if (minput.getValidationStatus() == Status.WARNING) {
       for(Message message : minput.getValidationMessages()) {
        System.out.println("Config Input Validation Warning: " + message.getMessage());
       }
    }
    else if (minput.getValidationStatus() == Status.ERROR) {
      for(Message message : minput.getValidationMessages()) {
       System.out.println("Config Input Validation Error: " + message.getMessage());
      }
     }
    }
   }
```

Updating link and job

After creating link or job in the repository, you can update or delete a link or job using the following functions

Method	Description
`updateLink(link)`	Invoke update with link and check status for any errors or warnings
`deleteLink(linkName)`	Delete link. Deletes only if specified link is not used by any job
`updateJob(job)`	Invoke update with job and check status for any errors or warnings
`deleteJob(jobName)`	Delete job

3.2.6 Job Start

Starting a job requires a job name. On successful start, getStatus() method returns “BOOTING” or “RUNNING”.

```
//Job start
MSubmission submission = client.startJob("jobName");
System.out.println("Job Submission Status : " + submission.getStatus());
if(submission.getStatus().isRunning() && submission.getProgress() != -1) {
  System.out.println("Progress : " + String.format("%.2f %%", submission.getProgress() * 100));
}
System.out.println("Hadoop job id :" + submission.getExternalId());
System.out.println("Job link : " + submission.getExternalLink());
Counters counters = submission.getCounters();
if(counters != null) {
  System.out.println("Counters:");
  for(CounterGroup group : counters) {
    System.out.print("\t");
```

```
    System.out.println(group.getName());
    for(Counter counter : group) {
      System.out.print("\t\t");
      System.out.print(counter.getName());
      System.out.print(": ");
      System.out.println(counter.getValue());
    }
  }
}
if(submission.getExceptionInfo() != null) {
  System.out.println("Exception info : " +submission.getExceptionInfo());
}


//Check job status for a running job
MSubmission submission = client.getJobStatus("jobName");
if(submission.getStatus().isRunning() && submission.getProgress() != -1) {
  System.out.println("Progress : " + String.format("%.2f %%", submission.getProgress() * 100));
}

//Stop a running job
submission.stopJob("jobName");
```

Above code block, job start is asynchronous. For synchronous job start, use `startJob(jobName, callback, pollTime)` method. If you are not interested in getting the job status, then invoke the same method with "null" as the value for the callback parameter and this returns the final job status. `pollTime` is the request interval for getting the job status from sqoop server and the value should be greater than zero. We will frequently hit the sqoop server if a low value is given for the `pollTime`. When a synchronous job is started with a non null callback, it first invokes the callback's `submitted(MSubmission)` method on successful start, after every poll time interval, it then invokes the `updated(MSubmission)` method on the callback API and finally on finishing the job executuon it invokes the `finished(MSubmission)` method on the callback API.

3.2.7 Display Config and Input Names For Connector

You can view the config/input names for the link and job config types per connector

```
String url = "http://localhost:12000/sqoop/";
SqoopClient client = new SqoopClient(url);
String connectorName = "connectorName";
// link config for connector
describe(client.getConnector(connectorName).getLinkConfig().getConfigs(), client.getConnectorConfigBu
// from job config for connector
describe(client.getConnector(connectorName).getFromConfig().getConfigs(), client.getConnectorConfigBu
// to job config for the connector
describe(client.getConnector(connectorName).getToConfig().getConfigs(), client.getConnectorConfigBund

void describe(List<MConfig> configs, ResourceBundle resource) {
  for (MConfig config : configs) {
    System.out.println(resource.getString(config.getLabelKey())+":");
    List<MInput<?>> inputs = config.getInputs();
    for (MInput input : inputs) {
      System.out.println(resource.getString(input.getLabelKey()) + " : " + input.getValue());
    }
    System.out.println();
  }
}
```

Above Sqoop 2 Client API tutorial explained how to create a link, create job and and then start the job.

3.3 Sqoop 2 Connector Development

This document describes how to implement a connector in the Sqoop 2 using the code sample from one of the built-in connectors (`GenericJdbcConnector`) as a reference. Sqoop 2 jobs support extraction from and/or loading to different data sources. Sqoop 2 connectors encapsulate the job lifecyle operations for extracting and/or loading data from and/or to different data sources. Each connector will primarily focus on a particular data source and its custom implementation for optimally reading and/or writing data in a distributed environment.

Contents

3.3.1 What is a Sqoop Connector?

Connectors provide the facility to interact with many data sources and thus can be used as a means to transfer data between them in Sqoop. The connector implementation will provide logic to read from and/or write to a data source that it represents. For instance the (`GenericJdbcConnector`) encapsulates the logic to read from and/or write to jdbc enabled relational data sources. The connector part that enables reading from a data source and transferring this data to internal Sqoop format is called the FROM and the part that enables writng data to a data source by transferring data from Sqoop format is called TO. In order to interact with these data sources, the connector will provide one or many config classes and input fields within it.

Broadly we support two main config types for connectors, link type represented by the enum `ConfigType.LINK` and job type represented by the enum `ConfigType.JOB`. Link config represents the properties to physically connect to the data source. Job config represent the properties that are required to invoke reading from and/or writing to particular dataset in the data source it connects to. If a connector supports both reading from and writing to, it will provide the `FromJobConfig` and `ToJobConfig` objects. Each of these config objects are custom to each

connector and can have one or more inputs associated with each of the Link, FromJob and ToJob config types. Hence we call the connectors as configurables i.e an entity that can provide configs for interacting with the data source it represents. As the connectors evolve over time to support new features in their data sources, the configs and inputs will change as well. Thus the connector API also provides methods for upgrading the config and input names and data related to these data sources across different versions.

The connectors implement logic for various stages of the extract/load process using the connector API described below. While extracting/reading data from the data-source the main stages are `Initializer`, `Partitioner`, `Extractor` and `Destroyer`. While loading/writitng data to the data source the main stages currently supported are `Initializer`, `Loader` and `Destroyer`. Each stage has its unique set of responsibilities that are explained in detail below. Since connectors understand the internals of the data source they represent, they work in tandem with the sqoop supported execution engines such as MapReduce or Spark (in future) to accomplish this process in a most optimal way.

When do we add a new connector?

You add a new connector when you need to extract/read data from a new data source, or load/write data into a new data source that is not supported yet in Sqoop 2. In addition to the connector API, Sqoop 2 also has an submission and execution engine interface. At the moment the only supported engine is MapReduce, but we may support additional engines in the future such as Spark. Since many parallel execution engines are capable of reading/writing data, there may be a question of whether adding support for a new data source should be done through the connector or the execution engine API.

Our guideline are as follows: Connectors should manage all data extract(reading) from and/or load(writing) into a data source. Submission and execution engine together manage the job submission and execution life cycle to read/write data from/to data sources in the most optimal way possible. If you need to support a new data store and details of linking to it and don't care how the process of reading/writing from/to happens then you are looking to add a connector and you should continue reading the below Connector API details to contribute new connectors to Sqoop 2.

3.3.2 Connector Implementation

The `SqoopConnector` class defines an API for the connectors that must be implemented by the connector developers. Each Connector must extend `SqoopConnector` and override the methods shown below.

```
public abstract String getVersion();
public abstract ResourceBundle getBundle(Locale locale);
public abstract Class getLinkConfigurationClass();
public abstract Class getJobConfigurationClass(Direction direction);
public abstract From getFrom();
public abstract To getTo();
public abstract ConnectorConfigurableUpgrader getConfigurableUpgrader(String oldConnectorVersion)
```

Connectors can optionally override the following methods:

```
public List<Direction> getSupportedDirections();
public Class<? extends IntermediateDataFormat<?>> getIntermediateDataFormat()
```

The `getVersion` method returns the current version of the connector It is important to provide a unique identifier every time a connector jar is released externally. In case of the Sqoop built-in connectors, the version refers to the Sqoop build/release version. External connectors can also use the same or similar mechanism to set this version. The version number is critical for the connector upgrade logic used in Sqoop

```
@Override
 public String getVersion() {
  return VersionInfo.getBuildVersion();
 }
```

The `getFrom` method returns *From* instance which is a `Transferable` entity that encapsulates the operations needed to read from the data source that the connector represents.

The `getTo` method returns *To* instance which is a `Transferable` entity that encapsulates the operations needed to write to the data source that the connector represents.

Methods such as `getBundle` , `getLinkConfigurationClass` , `getJobConfigurationClass` are related to *Configurations*

Since a connector represents a data source and it can support one of the two directions, either reading FROM its data source or writing to its data souurce or both, the `getSupportedDirections` method returns a list of directions that a connector will implement. This should be a subset of the values in the `Direction` enum we provide:

```
public List<Direction> getSupportedDirections() {
    return Arrays.asList(new Direction[]{
        Direction.FROM,
        Direction.TO
    });
}
```

From

The `getFrom` method returns *From* instance which is a `Transferable` entity that encapsulates the operations needed to read from the data source the connector represents. The built-in `GenericJdbcConnector` defines `From` like this.

```
private static final From FROM = new From(
      GenericJdbcFromInitializer.class,
      GenericJdbcPartitioner.class,
      GenericJdbcExtractor.class,
      GenericJdbcFromDestroyer.class);
...

@Override
public From getFrom() {
  return FROM;
}
```

Initializer and Destroyer

Initializer is instantiated before the submission of sqoop job to the execution engine and doing preparations such as connecting to the data source, creating temporary tables or adding dependent jar files. Initializers are executed as the first step in the sqoop job lifecyle. All interactions within an initializer are assumed to occur within a single thread, so state can be maintained between method calls (such as database connections). Here is the `Initializer` API.

```
public abstract void initialize(InitializerContext context, LinkConfiguration linkConfiguration,
    JobConfiguration jobConfiguration);

public List<String> getJars(InitializerContext context, LinkConfiguration linkConfiguration,
    JobConfiguration jobConfiguration){
     return new LinkedList<String>();
    }

public abstract Schema getSchema(InitializerContext context, LinkConfiguration linkConfiguration,
    JobConfiguration jobConfiguration) {
       return new NullSchema();
    }
```

In addition to the initialize() method where the job execution preparation activities occur, the `Initializer` can also implement the getSchema() method for the directions `FROM` and `TO` that it supports.

The getSchema() method is used by the sqoop system to match the data extracted/read by the `From` instance of connector data source with the data loaded/written to the `To` instance of the connector data source. In case of a relational database or columnar database, the returned Schema object will include collection of columns with their data types. If the data source is schema-less, such as a file, a default `NullSchema` will be used (i.e a Schema object without any columns).

NOTE: Sqoop 2 currently does not support extract and load between two connectors that represent schema-less data sources. We expect that atleast the `From` instance of the connector or the `To` instance of the connector in the sqoop job will have a schema. If both `From` and `To` have a associated non empty schema, Sqoop 2 will load data by column name, i.e, data in column "A" in `From` instance of the connector for the job will be loaded to column "A" in the `To` instance of the connector for that job.

`Destroyer` is instantiated after the execution engine finishes its processing. It is the last step in the sqoop job lifecyle, so pending clean up tasks such as dropping temporary tables and closing connections. The term destroyer is a little misleading. It represents the phase where the final output commits to the data source can also happen in case of the `TO` instance of the connector code.

Partitioner

The `Partitioner` creates `Partition` instances ranging from 1..N. The N is driven by a configuration as well. The default set of partitions created is set to 10 in the sqoop code. Here is the `Partitioner` API

`Partitioner` must implement the `getPartitions` method in the `Partitioner` API.

```
public abstract List<Partition> getPartitions(PartitionerContext context,
    LinkConfiguration linkConfiguration, FromJobConfiguration jobConfiguration);
```

`Partition` instances are passed to *Extractor* as the argument of `extract` method. *Extractor* determines which portion of the data to extract by a given partition.

There is no actual convention for Partition classes other than being actually `Writable` and `toString()` -able. Here is the `Partition` API

```
public abstract class Partition {
  public abstract void readFields(DataInput in) throws IOException;
  public abstract void write(DataOutput out) throws IOException;
  public abstract String toString();
}
```

Connectors can implement custom `Partition` classes. `GenericJdbcPartitioner` is one such example. It returns the `GenericJdbcPartition` objects.

Extractor

Extractor (E for ETL) extracts data from a given data source `Extractor` must implement the `extract` method in the `Extractor` API.

```
public abstract void extract(ExtractorContext context,
                             LinkConfiguration linkConfiguration,
                             JobConfiguration jobConfiguration,
                             SqoopPartition partition);
```

The `extract` method extracts data from the data source using the link and job configuration properties and writes it to the `SqoopMapDataWriter` (provided in the extractor context given to the extract method). The

`SqoopMapDataWriter` has the `SqoopWritable` thats holds the data read from the data source in the Intermediate Data Format representation

Extractors use Writer's provided by the ExtractorContext to send a record through the sqoop system.

```
context.getDataWriter().writeArrayRecord(array);
```

The extractor must iterate through the given partition in the `extract` method.

```
while (resultSet.next()) {
  ...
  context.getDataWriter().writeArrayRecord(array);
  ...
}
```

To

The `getTo` method returns `TO` instance which is a `Transferable` entity that encapsulates the operations needed to wtite data to the data source the connector represents. The built-in `GenericJdbcConnector` defines `To` like this.

```
private static final To TO = new To(
      GenericJdbcToInitializer.class,
      GenericJdbcLoader.class,
      GenericJdbcToDestroyer.class);
...

@Override
public To getTo() {
  return TO;
}
```

Initializer and Destroyer

Initializer and *Destroyer* of a `To` instance are used in a similar way to those of a `From` instance. Refer to the previous section for more details.

Loader

A loader (L for ETL) receives data from the `From` instance of the sqoop connector associated with the sqoop job and then loads it to an `TO` instance of the connector associated with the same sqoop job

`Loader` must implement `load` method of the `Loader` API

```
public abstract void load(LoaderContext context,
                          ConnectionConfiguration connectionConfiguration,
                          JobConfiguration jobConfiguration) throws Exception;
```

The `load` method reads data from `SqoopOutputFormatDataReader` (provided in the loader context of the load methods). It reads the data in the Intermediate Data Format representation and loads it to the data source.

Loader must iterate in the `load` method until the data from `DataReader` is exhausted.

```
while ((array = context.getDataReader().readArrayRecord()) != null) {
  ...
}
```

NOTE: we do not yet support a stage for connector developers to control how to balance the loading/writitng of data across the mutiple loaders. In future we may be adding this to the connector API to have custom logic to balance the loading across multiple reducers.

Sqoop Connector Identifier : sqoopconnector.properties

Every Sqoop 2 connector needs to have a sqoopconnector.properties in the packaged jar to be identified by Sqoop. A typical `sqoopconnector.properties` for a sqoop2 connector looks like below

```
# Sqoop Foo Connector Properties
org.apache.sqoop.connector.class = org.apache.sqoop.connector.foo.FooConnector
org.apache.sqoop.connector.name = sqoop-foo-connector
```

If the above file does not exist, then Sqoop will not load this jar and thus cannot be registered into Sqoop repository for creating Sqoop jobs

Sqoop Connector Build-time Dependencies

Sqoop provides the connector-sdk module identified by the package:`org.apache.sqoop.connector` It provides the public facing apis for the external connectors to extend from. It also provides common utilities that the connectors can utilize for converting data to and from the sqoop intermediate data format

The common-test module identified by the package `org.apache.sqoop.common.test` provides utilities used related to the built-in connectors such as the JDBC, HDFS, and Kafka connectors that can be used by the external connectors for creating the end-end integration test for sqoop jobs

The test module identified by the package `org.apache.sqoop.test` provides various minicluster utilites the integration test
a sqoop job with the given sqoop connector either using it as a `FROM` or `TO` data-source

Hence the pom.xml for the sqoop kite connector built using the kite-sdk might look something like below

```
 <dependencies>
  <!-- Sqoop modules -->
  <dependency>
    <groupId>org.apache.sqoop</groupId>
    <artifactId>connector-sdk</artifactId>
  </dependency>

  <!-- Testing specified modules -->
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <scope>test</scope>
  </dependency>
   <dependency>
     <groupId>org.apache.sqoop</groupId>
     <artifactId>sqoop-common-test</artifactId>
   </dependency>

   <dependency>
     <groupId>org.apache.sqoop</groupId>
     <artifactId>test</artifactId>
```

```
   </dependency>
  <!-- Connector required modules -->
  <dependency>
    <groupId>org.kitesdk</groupId>
    <artifactId>kite-data-core</artifactId>
  </dependency>
  ....
</dependencies>
```

Sqoop Connector Build

Sqoop 2 supports connectors to package their dependencies into the `lib` directory inside the connector jar to provide classpath isolation between connectors. Add the following to the pom.xml for the connector:

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>${maven-assembly-plugin.version}</version>
    <dependencies>
      <dependency>
        <groupId>org.apache.sqoop</groupId>
        <artifactId>sqoop-assemblies</artifactId>
        <version>${sqoop.version}</version>
      </dependency>
    </dependencies>
    <executions>
      <execution>
        <id>make-assembly</id>
        <phase>package</phase>
        <goals>
          <goal>single</goal>
        </goals>
        <configuration>
          <finalName>${project.artifactId}-${project.version}</finalName>
          <appendAssemblyId>false</appendAssemblyId>
          <descriptorRefs>
            <descriptorRef>sqoop-connector</descriptorRef>
          </descriptorRefs>
        </configuration>
      </execution>
    </executions>
  </plugin>
</plugins>
```

3.3.3 Configurables

Configurable registration

One of the currently supported configurable in Sqoop are the connectors. Sqoop 2 registers definitions of connectors from the file named `sqoopconnector.properties` which each connector implementation should provide to become available in Sqoop.

```
# Generic JDBC Connector Properties
org.apache.sqoop.connector.class = org.apache.sqoop.connector.jdbc.GenericJdbcConnector
org.apache.sqoop.connector.name = generic-jdbc-connector
```

Configurations

Implementations of `SqoopConnector` overrides methods such as `getLinkConfigurationClass` and `getJobConfigurationClass` returning configuration class.

```
@Override
public Class getLinkConfigurationClass() {
  return LinkConfiguration.class;
}

@Override
public Class getJobConfigurationClass(Direction direction) {
  switch (direction) {
    case FROM:
      return FromJobConfiguration.class;
    case TO:
      return ToJobConfiguration.class;
    default:
      return null;
  }
}
```

Configurations are represented by annotations defined in `org.apache.sqoop.model` package. Annotations such as `ConfigurationClass` , `ConfigClass` , `Config` and `Input` are provided for defining configuration objects for each connector.

`@ConfigurationClass` is a marker annotation for `ConfigurationClasses` that hold a group or lis of `ConfigClasses` annotated with the marker `@ConfigClass`

```
@ConfigurationClass
public class LinkConfiguration {

  @Config public LinkConfig linkConfig;

  public LinkConfiguration() {
    linkConfig = new LinkConfig();
  }
}
```

Each `ConfigClass` defines the different inputs it exposes for the link and job configs. These inputs are annotated with `@Input` and the user will be asked to fill in when they create a sqoop job and choose to use this instance of the connector for either the `From` or `To` part of the job.

```
@ConfigClass(validators = {@Validator(LinkConfig.ConfigValidator.class)})
public class LinkConfig {
  @Input(size = 128, validators = {@Validator(NotEmpty.class), @Validator(ClassAvailable.class)} )
  @Input(size = 128) public String jdbcDriver;
  @Input(size = 128) public String connectionString;
  @Input(size = 40)  public String username;
  @Input(size = 40, sensitive = true) public String password;
  @Input public Map<String, String> jdbcProperties;
}
```

Each `ConfigClass` and the inputs within the configs annotated with `Input` can specifiy validators via the `@Validator` annotation described below.

Configs and Inputs

As discussed above, `Input` provides a way to express the type of config parameter exposed. In addition it allows connector developer to add attributes that describe how the input will be used in the sqoop job. Here are the list of the supported attributes

Inputs associated with the link configuration include:

Attribute	Type	Description	Example
size	Integer	Describes the maximum size of the attribute value .	@Input(size = 128) public String driver
sensitive	Boolean	Describes if the input value should be hidden from display	@Input(sensitive = true) public String password
sensitiveKeyPattern	String	If the config paramteter is a map, this java regular expression (http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html) will be used to decide which keys are hidden from display.	@Input(sensitiveKeyPattern = ”.*sensitive”) public Map<String, String> sensitiveMap
editable	Enum	Describes the roles that can edit the value of this input	@Input(editable = ANY) public String value
overrides	String	Describes a list of other inputs this input can override in this config	@Input(overrides =”value”) public String lvalue

`Editable` Attribute: Possible values for the Enum InputEditable are USER_ONLY, CONNECTOR_ONLY, ANY. If an input says editable by USER_ONLY, then the connector code during the job run or upgrade cannot update the config input value. Similarly for a CONNECTOR_ONLY, user cannot update its value via the rest api or shell command line.

`Overrides` Attribute: USER_ONLY input attribute values cannot be overriden by other inputs.

Empty Configuration

If a connector does not have any configuration inputs to specify for the `ConfigType.LINK` or `ConfigType.JOB` it is recommended to return the `EmptyConfiguration` class in the `getLinkConfigurationClass()` or `getJobConfigurationClass(..)` methods.

```
@ConfigurationClass
public class EmptyConfiguration { }
```

Configuration ResourceBundle

The config and its corresponding input names, the input field description are represented in the config resource bundle defined per connector.

```
# jdbc driver
connection.jdbcDriver.label = JDBC Driver Class
connection.jdbcDriver.help = Enter the fully qualified class name of the JDBC \
                    driver that will be used for establishing this connection.

# connect string
connection.connectionString.label = JDBC Connection String
```

```
connection.connectionString.help = Enter the value of JDBC connection string to be \
                    used by this connector for creating connections.

...
```

Those resources are loaded by `getBundle` method of the `SqoopConnector`.

```
@Override
public ResourceBundle getBundle(Locale locale) {
  return ResourceBundle.getBundle(
  GenericJdbcConnectorConstants.RESOURCE_BUNDLE_NAME, locale);
}
```

Validations for Configs and Inputs

Validators validate the config objects and the inputs associated with the config objects. For config objects themselves we encourage developers to write custom valdiators for both the link and job config types.

```
@Input(size = 128, validators = {@Validator(value = StartsWith.class, strArg = "jdbc:")} )

@Input(size = 255, validators = { @Validator(NotEmpty.class) })
```

Sqoop 2 provides a list of standard input validators that can be used by different connectors for the link and job type configuration inputs.

```
public class NotEmpty extends AbstractValidator<String> {
@Override
public void validate(String instance) {
  if (instance == null || instance.isEmpty()) {
   addMessage(Status.ERROR, "Can't be null nor empty");
  }
 }
}
```

The validation logic is executed when users creating the sqoop jobs input values for the link and job configs associated with the `From` and `To` instances of the connectors associated with the job.

3.3.4 Loading External Connectors

Loading new connector say sqoop-foo-connector to the sqoop2, here are the steps to follow

1. Create a `sqoop-foo-connector.jar`. Make sure the jar contains the `sqoopconnector.properties` for it to be picked up by Sqoop
2. Add this jar to the `org.apache.sqoop.classpath.extra` property in the sqoop.properties located under the `conf` directory.

```
# Sqoop application classpath
# ":" separated list of jars to be included in sqoop.
#
org.apache.sqoop.classpath.extra=/path/to/connector.jar
```

3. Start the Sqoop 2 server and while initializing the server this jar should be loaded into the Sqoop 2's class path and registered into the Sqoop 2 repository

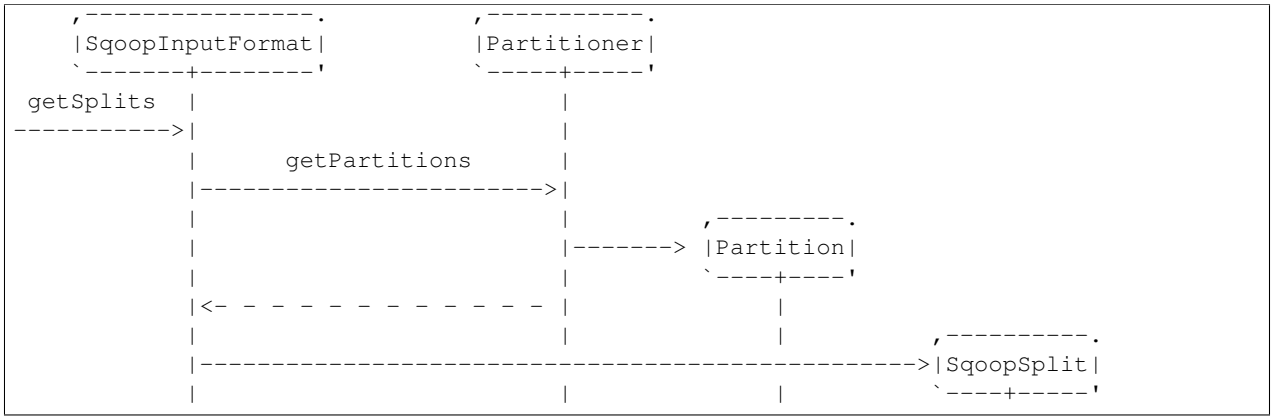
3.3.5 Sqoop 2 MapReduce Job Execution Lifecycle with Connector API

Sqoop 2 provides MapReduce utilities such as `SqoopMapper` and `SqoopReducer` that aid sqoop job execution.

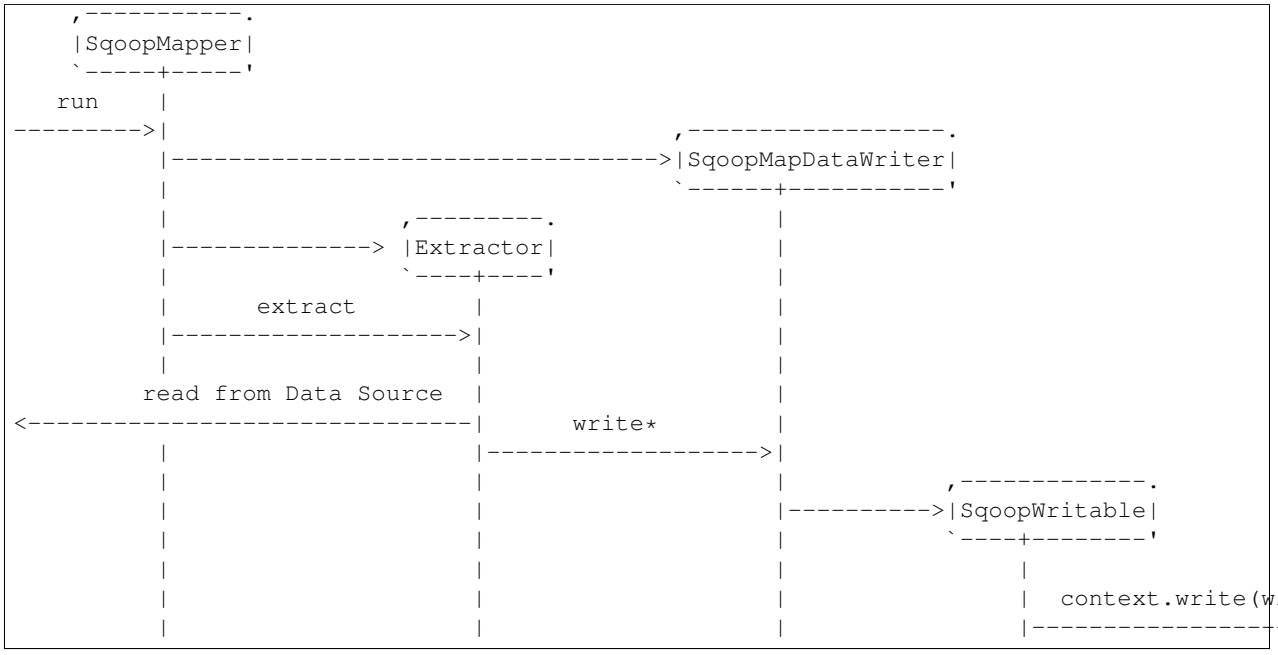
Note: Any class prefixed with Sqoop is a internal sqoop class provided for MapReduce and is not part of the conenector API. These internal classes work with the custom implementations of `Extractor`, `Partitioner` in the `From` instance and `Loader` in the `To` instance of the connector.

When reading from a data source, the `Extractor` provided by the `From` instance of the connector extracts data from a corresponding data source it represents and the `Loader`, provided by the TO instance of the connector, loads data into the data source it represents.

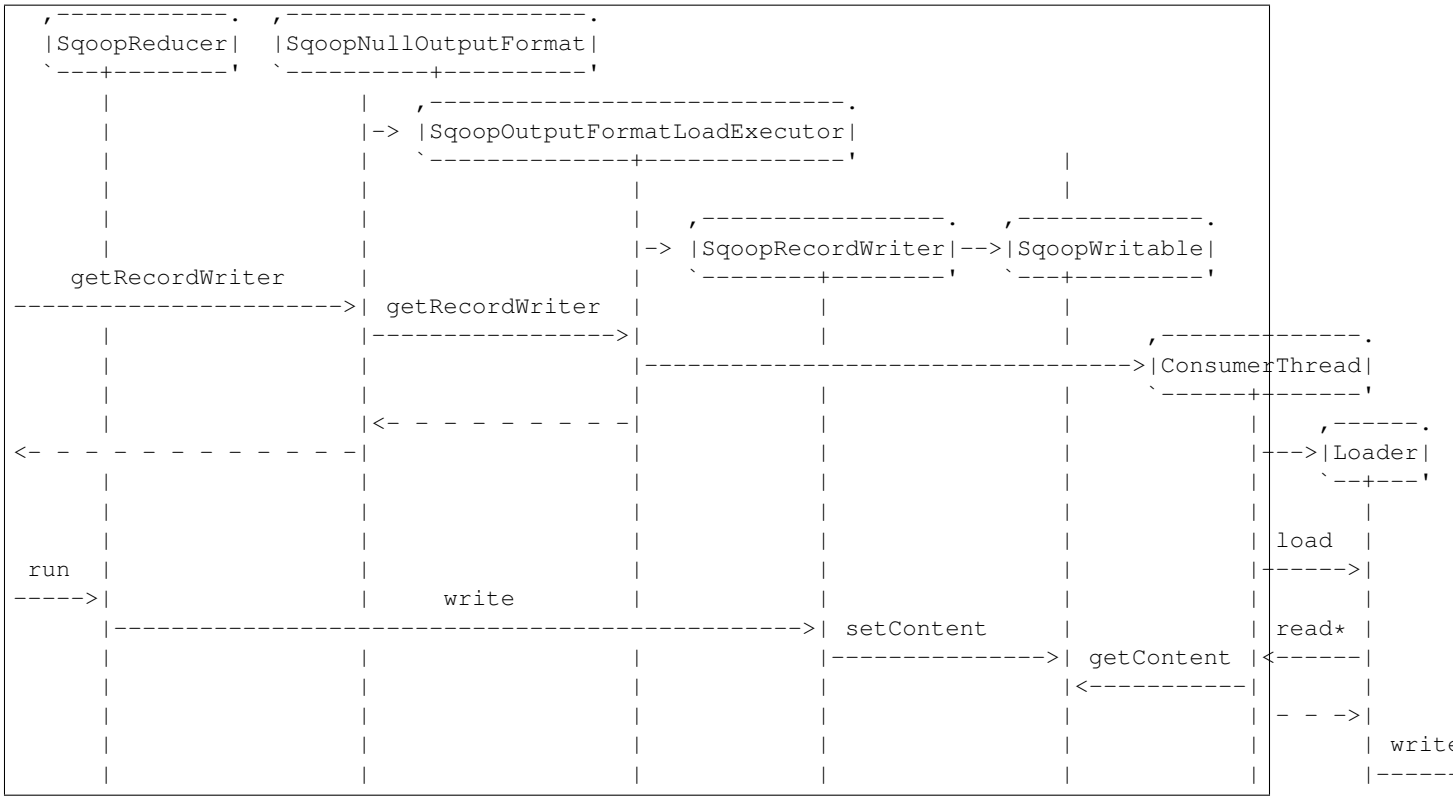
The diagram below describes the initialization phase of a job. `SqoopInputFormat` create splits using `Partitioner`.



The diagram below describes the map phase of a job. `SqoopMapper` invokes `From` connector's extractor's `extract` method.



The diagram below decribes the reduce phase of a job. `OutputFormat` invokes `To` connector's loader's `load` method (via `SqoopOutputFormatLoadExecutor`).



More details can be found in Sqoop MR Execution Engine

3.4 Sqoop 2 Development Environment Setup

This document describes you how to setup development environment for Sqoop 2.

3.4.1 System Requirement

Java

Sqoop has been developped and test only with JDK from Oracle and we require at least version 7 (we're not supporting JDK 1.6 and older releases).

Maven

Sqoop uses Maven 3 for building the project. Download Maven and its Installation instructions given in link.

3.4.2 Eclipse Setup

Steps for downloading source code are given in Building Sqoop2 from source code.

Sqoop 2 project has multiple modules where one module is depend on another module for e.g. sqoop 2 client module has sqoop 2 common module dependency. Follow below step for creating eclipse's project and classpath for each module.

```
//Install all package into local maven repository
mvn clean install -DskipTests

//Adding M2_REPO variable to eclipse workspace
mvn eclipse:configure-workspace -Declipse.workspace=<path-to-eclipse-workspace-dir-for-sqoop-2>

//Eclipse project creation with optional parameters
mvn eclipse:eclipse -DdownloadSources=true -DdownloadJavadocs=true
```

Alternatively, for manually adding M2_REPO classpath variable as maven repository path in eclipse-> window-> Java ->Classpath Variables ->Click "New" ->In new dialog box, input Name as M2_REPO and Path as $HOME/.m2/repository ->click Ok.

On successful execution of above maven commands, Then import the sqoop project modules into eclipse-> File -> Import ->General ->Existing Projects into Workspace-> Click Next-> Browse Sqoop 2 directory ($HOME/git/sqoop2) ->Click Ok ->Import dialog shows multiple projects (sqoop-client, sqoop-common, etc.) -> Select all modules -> click Finish.

3.5 Sqoop REST API Guide

This document will explain how you can use Sqoop REST API to build applications interacting with Sqoop server. The REST API covers all aspects of managing Sqoop jobs and allows you to build an app in any programming language using HTTP over JSON.

Table of Contents

3.5.1 Initialization

Before continuing further, make sure that the Sqoop server is running.

The Sqoop 2 server exposes its REST API via Jetty. By default the server is accessible over HTTP but it can be configured to use HTTPS, please see: *API TLS/SSL* for more information. The endpoints are registered under the `/sqoop` path and the port is configurable (the default is 12000). For example, if the host running the Sqoop 2 server is `example.com` and we are using the default port, we can reach the version endpoint by sending a GET request to:

```
http://example.com:12000/sqoop/v1/version
```

Certain requests might need to contain some additional query parameters and post data. These parameters could be given via the HTTP headers, request body or both. All the content in the HTTP body is in `JSON` format.

3.5.2 Understand Connector, Driver, Link and Job

To create and run a Sqoop Job, we need to provide config values for connecting to a data source and then processing the data in that data source. Processing might be either reading from or writing to the data source. Thus we have configurable entities such as the `From` and `To` parts of the connectors, the driver that each expose configs and one or more inputs within them.

For instance a connector that represents a relational data source such as MySQL will expose config classes for connecting to the database. Some of the relevant inputs are the connection string, driver class, the username and the password to connect to the database. These configs remain the same to read data from any of the tables within that database. Hence they are grouped under `LinkConfiguration`.

Each connector can support Reading from a data source and/or writing/to a data source it represents. Reading from and writing to a data source are represented by From and To respectively. Specific configurations are required to peform the job of reading from or writing to the data source. These are grouped in the `FromJobConfiguration` and `ToJobConfiguration` objects of the connector.

For instance, a connector that represents a relational data source such as MySQL will expose the table name to read from or the SQL query to use while reading data as a FromJobConfiguration. Similarly a connector that represents a data source such as HDFS, will expose the output directory to write to as a ToJobConfiguration.

3.5.3 Objects

This section covers all the objects that might exist in an API request and/or API response.

Configs and Inputs

Before creating any link for a connector or a job with associated `From` and `To` links, the first thing to do is getting familiar with all the configurations that the connector exposes.

Each config consists of the following information

Field	Description
`id`	The id of this config
`inputs`	A array of inputs of this config
`name`	The unique name of this config per connector
`type`	The type of this config (LINK/ JOB)

A typical config object is showing below:

```
 {
  id:7,
  inputs:[
    {
       id: 25,
       name: "throttlingConfig.numExtractors",
       type: "INTEGER",
       sensitive: false
    },
    {
       id: 26,
       name: "throttlingConfig.numLoaders",
       type: "INTEGER",
       sensitive: false
     }
  ],
  name: "throttlingConfig",
  type: "JOB"
}
```

Each input object in a config is structured below:

Field	Description
id	The id of this input
name	The unique name of this input per config
type	The data type of this input field
size	The length of this input field
sensitive	Whether this input contain sensitive information

To send a filled config in the request, you should always use config id and input id to map the values to their correspondig names. For example, the following request contains an input value `com.mysql.jdbc.Driver` with input id `7` inside a config with id `4` that belongs to a link with name `linkName`

```
link: {
      id : 3,
      name: "linkName",
      enabled: true,
      link-config-values: [{
          id: 4,
          inputs: [{
              id: 7,
              name: "linkConfig.jdbcDriver",
              value: "com.mysql.jdbc.Driver",
              type: "STRING",
              size: 128,
              sensitive: false
          }, {
              id: 8,
              name: "linkConfig.connectionString",
              value: "jdbc%3Amysql%3A%2F%2Fmysql.ent.cloudera.com%2Fsqoop",
              type: "STRING",
              size: 128,
              sensitive: false
          },
          ...
       }
     }
```

Exception Response

Each operation on Sqoop server might return an exception in the Http response. Remember to take this into account.The exception code and message could be found in both the header and body of the response.

Please jump to "Header Parameters" section to find how to get exception information from header.

In the body, the exception is expressed in `JSON` format. An example of the exception is:

```
{
  "message":"DERBYREPO_0030:Unable to load specific job metadata from repository - Couldn't find job
  "stack-trace":[
    {
      "file":"DerbyRepositoryHandler.java",
      "line":1111,
      "class":"org.apache.sqoop.repository.derby.DerbyRepositoryHandler",
      "method":"findJob"
    },
    {
      "file":"JdbcRepository.java",
      "line":451,
      "class":"org.apache.sqoop.repository.JdbcRepository$16",
      "method":"doIt"
    },
    {
      "file":"JdbcRepository.java",
      "line":90,
      "class":"org.apache.sqoop.repository.JdbcRepository",
      "method":"doWithConnection"
    },
    {
      "file":"JdbcRepository.java",
      "line":61,
      "class":"org.apache.sqoop.repository.JdbcRepository",
      "method":"doWithConnection"
    },
    {
      "file":"JdbcRepository.java",
      "line":448,
      "class":"org.apache.sqoop.repository.JdbcRepository",
      "method":"findJob"
    },
    {
      "file":"JobRequestHandler.java",
      "line":238,
      "class":"org.apache.sqoop.handler.JobRequestHandler",
      "method":"getJobs"
    }
  ],
  "class":"org.apache.sqoop.common.SqoopException"
}
```

Config and Input Validation Status Response

The config and the inputs associated with the connectors also provide custom validation rules for the values given to these input fields. Sqoop applies these custom validators and its corresponding valdation logic when config values for the LINK and JOB are posted.

An example of a OK status with the persisted ID:

```
{
   "id": 3,
   "validation-result": [
       {}
   ]
}
```

An example of ERROR status:

```
{
  "validation-result": [
    {
     "linkConfig": [
       {
         "message": "Invalid URI. URI must either be null or a valid URI. Here are a few valid exampl
         "status": "ERROR"
       }
     ]
   }
  ]
}
```

Job Submission Status Response

After starting a job, you could look up the running status of it. There could be 7 possible status:

Status	Description
BOOTING	In the middle of submitting the job
FAILURE_ON_SUBMIT	Unable to submit this job to remote cluster
RUNNING	The job is running now
SUCCEEDED	Job finished successfully
FAILED	Job failed
NEVER_EXECUTED	The job has never been executed since created
UNKNOWN	The status is unknown

3.5.4 Header Parameters

For all the responses, the following parameters in the HTTP message header are available:

Parameter	Required	Description
sqoop-error-code	false	The error code when some error happen in the server side for this request
sqoop-error-message	false	The explanation for a error code

So far, there are only these 2 parameters in the header of response message. They only exist when something bad happen in the server. And they always come along with an exception message in the response body.

3.5.5 REST APIs

The section elaborates all the rest apis that are supported by the Sqoop server.

For all Sqoop requests, the following request parameters will be added automatically. However, this user name is only in simple mode. In Kerberos mode, this user name will be ignored by Sqoop server and user name in UGI which is authenticated by Kerberos server will be used instead.

Parameter	Description
user.name	The name of the user who makes the requests

/version - [GET] - Get Sqoop Version

Get all the version metadata of Sqoop software in the server side.

- Method: GET
- Format: JSON
- Request Content: None
- Fields of Response:

Field	Description
source-revision	The revision number of Sqoop source code
api-versions	The version of network protocol
build-date	The Sqoop release date
user	The user who made the release
source-url	The url of the source code trunk
build-version	The version of Sqoop in the server side

- Response Example:

```
{
 source-url: "git://vbasavaraj.local/Users/vbasavaraj/Projects/SqoopRefactoring/sqoop2/common",
 source-revision: "418c5f637c3f09b94ea7fc3b0a4610831373a25f",
 build-version: "2.0.0-SNAPSHOT",
 api-versions: [
    "v1"
  ],
 user: "vbasavaraj",
 build-date: "Mon Nov 3 08:18:21 PST 2014"
}
```

/v1/connectors - [GET] Get all Connectors

Get all the connectors registered in Sqoop

- Method: GET
- Format: JSON
- Request Content: None
- Response Example

```
{
  connectors: [{
      id: 1,
      link-config: [],
      job-config: {},
      name: "hdfs-connector",
      class: "org.apache.sqoop.connector.hdfs.HdfsConnector",
      all-config-resources: {},
      version: "2.0.0-SNAPSHOT"
  }, {
      id: 2,
```

```
      link-config: [],
      job-config: {},
      name: "generic-jdbc-connector",
      class: "org.apache.sqoop.connector.jdbc.GenericJdbcConnector",
      all-config - resources: {},
      version: "2.0.0-SNAPSHOT"
  }]
}
```

/v1/connector/[cname] - [GET] - Get Connector

Provide the unique name of the connector in the url `[cname]` part.

- Method: `GET`
- Format: `JSON`
- Request Content: `None`
- Fields of Response:

Field	Description
`name`	The name for the connector (registered as a configurable)
`job-config`	Connector job config and inputs for both FROM and TO
`link-config`	Connector link config and inputs
`all-config-resources`	All config inputs labels and description for the given connector
`version`	The build version required for config and input data upgrades

- Response Example:

```
{
 connector: {
     id: 1,
     name: "connectorName",
     job-config: {
         TO: [{
             id: 3,
             inputs: [{
                 id: 3,
                 values: "TEXT_FILE,SEQUENCE_FILE",
                 name: "toJobConfig.outputFormat",
                 type: "ENUM",
                 sensitive: false
             }, {
                 id: 4,
                 values: "NONE,DEFAULT,DEFLATE,GZIP,BZIP2,LZO,LZ4,SNAPPY,CUSTOM",
                 name: "toJobConfig.compression",
                 type: "ENUM",
                 sensitive: false
             }, {
                 id: 5,
                 name: "toJobConfig.customCompression",
                 type: "STRING",
                 size: 255,
                 sensitive: false
             }, {
                 id: 6,
                 name: "toJobConfig.outputDirectory",
                 type: "STRING",
```

```
                size: 255,
                sensitive: false
            }],
            name: "toJobConfig",
            type: "JOB"
        }],
        FROM: [{
            id: 2,
            inputs: [{
                id: 2,
                name: "fromJobConfig.inputDirectory",
                type: "STRING",
                size: 255,
                sensitive: false
            }],
            name: "fromJobConfig",
            type: "JOB"
        }]
    },
    link-config: [{
        id: 1,
        inputs: [{
            id: 1,
            name: "linkConfig.uri",
            type: "STRING",
            size: 255,
            sensitive: false
        }],
        name: "linkConfig",
        type: "LINK"
    }],
    name: "hdfs-connector",
    class: "org.apache.sqoop.connector.hdfs.HdfsConnector",
    all-config-resources: {
        fromJobConfig.label: "From Job configuration",
            toJobConfig.ignored.label: "Ignored",
            fromJobConfig.help: "Specifies information required to get data from Hadoop ecosystem",
            toJobConfig.ignored.help: "This value is ignored",
            toJobConfig.label: "ToJob configuration",
            toJobConfig.storageType.label: "Storage type",
            fromJobConfig.inputDirectory.label: "Input directory",
            toJobConfig.outputFormat.label: "Output format",
            toJobConfig.outputDirectory.label: "Output directory",
            toJobConfig.outputDirectory.help: "Output directory for final data",
            toJobConfig.compression.help: "Compression that should be used for the data",
            toJobConfig.outputFormat.help: "Format in which data should be serialized",
            toJobConfig.customCompression.label: "Custom compression format",
            toJobConfig.compression.label: "Compression format",
            linkConfig.label: "Link configuration",
            toJobConfig.customCompression.help: "Full class name of the custom compression",
            toJobConfig.storageType.help: "Target on Hadoop ecosystem where to store data",
            linkConfig.help: "Here you supply information necessary to connect to HDFS",
            linkConfig.uri.help: "HDFS URI used to connect to HDFS",
            linkConfig.uri.label: "HDFS URI",
            fromJobConfig.inputDirectory.help: "Directory that should be exported",
            toJobConfig.help: "You must supply the information requested in order to get informatio
    },
    version: "2.0.0-SNAPSHOT"
```

```
  }
}
```

/v1/driver - [GET]- Get Sqoop Driver

Driver exposes configurations required for the job execution.

- Method: `GET`
- Format: `JSON`
- Request Content: `None`
- Fields of Response:

Field	Description
`id`	The id for the driver (registered as a configurable)
`job-config`	Driver job config and inputs
`version`	The build version of the driver
`all-config-resources`	Driver exposed config and input labels and description

- Response Example:

```
{
   id: 3,
   job-config: [{
       id: 7,
       inputs: [{
           id: 25,
           name: "throttlingConfig.numExtractors",
           type: "INTEGER",
           sensitive: false
       }, {
           id: 26,
           name: "throttlingConfig.numLoaders",
           type: "INTEGER",
           sensitive: false
       }],
       name: "throttlingConfig",
       type: "JOB"
   }],
   all-config-resources: {
       throttlingConfig.numExtractors.label: "Extractors",
           throttlingConfig.numLoaders.help: "Number of loaders that Sqoop will use",
           throttlingConfig.numLoaders.label: "Loaders",
           throttlingConfig.label: "Throttling resources",
           throttlingConfig.numExtractors.help: "Number of extractors that Sqoop will use",
           throttlingConfig.help: "Set throttling boundaries to not overload your systems"
   },
   version: "1"
}
```

/v1/links/ - [GET] Get all links

Get all the links created in Sqoop

- Method: `GET`

- Format: `JSON`
- Request Content: `None`
- Response Example

```
{
  links: [
    {
      id: 1,
      enabled: true,
      update-user: "root",
      link-config-values: [],
      name: "First Link",
      creation-date: 1415309361756,
      connector-name: "connectorName1",
      update-date: 1415309361756,
      creation-user: "root"
    },
    {
      id: 2,
      enabled: true,
      update-user: "root",
      link-config-values: [],
      name: "Second Link",
      creation-date: 1415309390807,
      connector-name: "connectorName2",
      update-date: 1415309390807,
      creation-user: "root"
    }
  ]
}
```

/v1/links?cname=[cname] - [GET] Get all links by Connector

Get all the links for a given connector identified by `[cname]` part.

/v1/link/[lname] - [GET] - Get Link

Provide the unique name of the link in the url `[lname]` part.

Get all the details of the link including the name, type and the corresponding config input values for the link

- Method: `GET`
- Format: `JSON`
- Request Content: `None`
- Response Example:

```
{
   link: {
       id: 1,
       enabled: true,
       link-config-values: [{
           id: 1,
           inputs: [{
               id: 1,
```

```
                name: "linkConfig.uri",
                value: "hdfs%3A%2F%2Fnamenode%3A8090",
                type: "STRING",
                size: 255,
                sensitive: false
            }],
            name: "linkConfig",
            type: "LINK"
        }],
        update-user: "root",
        name: "First Link",
        creation-date: 1415287846371,
        connector-name: "connectorName",
        update-date: 1415287846371,
        creation-user: "root"
    }
}
```

/v1/link - [POST] - Create Link

Create a new link object. Provide values to the link config inputs for the ones that are required.

- Method: `POST`
- Format: `JSON`
- Fields of Request:

Field	Description
`link`	The root of the post data in JSON
`id`	The id of the link can be left blank in the post data
`enabled`	Whether to enable this link (true/false)
`update-date`	The last updated time of this link
`creation-date`	The creation time of this link
`update-user`	The user who updated this link
`creation-user`	The user who created this link
`name`	The name of this link
`link-config-values`	Config input values for link config for the corresponding connector
`connector-id`	The id of the connector used for this link

- Request Example:

```
{
  link: {
      id: -1,
      enabled: true,
      link-config-values: [{
          id: 1,
          inputs: [{
              id: 1,
              name: "linkConfig.uri",
              value: "hdfs%3A%2F%2Fvbsqoop-1.ent.cloudera.com%3A8020%2Fuser%2Froot%2Fjob1",
              type: "STRING",
              size: 255,
              sensitive: false
          }],
          name: "testInput",
          type: "LINK"
```

```
      }],
      update-user: "root",
      name: "testLink",
      creation-date: 1415202223048,
      connector-name: "connectorName",
      update-date: 1415202223048,
      creation-user: "root"
  }
}
```

- Fields of Response:

Field	Description
`name`	The name assigned for this new created link
`validation-result`	The validation status for the link config inputs given in the post data

- ERROR Response Example:

```
{
  "validation-result": [
      {
          "linkConfig": [
              {
                  "message": "Invalid URI. URI must either be null or a valid URI. Here are a few val
                  "status": "ERROR"
              }
          ]
      }
  ]
}
```

/v1/link/[lname] - [PUT] - Update Link

Update an existing link object with name [lname]. To make the procedure of filling inputs easier, the general practice is get the link first and then change some of the values for the inputs.

- Method: `PUT`
- Format: `JSON`
- OK Response Example:

```
{
  "validation-result": [
      {}
  ]
}
```

/v1/link/[lname] - [DELETE] - Delete Link

Delete a link with name [lname]

- Method: `DELETE`
- Format: `JSON`
- Request Content: `None`
- Response Content: `None`

/v1/link/[lname]/enable - [PUT] - Enable Link

Enable a link with name `lname`

- Method: `PUT`
- Format: `JSON`
- Request Content: `None`
- Response Content: `None`

/v1/link/[lname]/disable - [PUT] - Disable Link

Disable a link with name `lname`

- Method: `PUT`
- Format: `JSON`
- Request Content: `None`
- Response Content: `None`

/v1/jobs/ - [GET] Get all jobs

Get all the jobs created in Sqoop

- Method: `GET`
- Format: `JSON`
- Request Content: `None`
- Response Example:

```
{
   jobs: [{
      driver-config-values: [],
          enabled: true,
          from-connector-name: "fromConnectorName",
          update-user: "root",
          to-config-values: [],
          to-connector-name: "toConnectorName",
          creation-date: 1415310157618,
          update-date: 1415310157618,
          creation-user: "root",
          id: 1,
          to-link-name: "toLinkName",
          from-config-values: [],
          name: "First Job",
          from-link-name: "fromLinkName"
     },{
      driver-config-values: [],
          enabled: true,
          from-connector-name: "fromConnectorName",
          update-user: "root",
          to-config-values: [],
          to-connector-name: "toConnectorName",
          creation-date: 1415310650600,
          update-date: 1415310650600,
```

```
          creation-user: "root",
          id: 2,
          to-link-name: "toLinkName",
          from-config-values: [],
          name: "Second Job",
          from-link-name: "fromLinkName"
      }]
}
```

/v1/jobs?cname=[cname] - [GET] Get all jobs by connector

Get all the jobs for a given connector identified by `[cname]` part.

/v1/job/[jname] - [GET] - Get Job

Provide the name of the job in the url [jname] part.

- Method: `GET`
- Format: `JSON`
- Request Content: `None`
- Response Example:

```
 {
   job: {
       driver-config-values: [{
               id: 7,
               inputs: [{
                   id: 25,
                   name: "throttlingConfig.numExtractors",
                   value: "3",
                   type: "INTEGER",
                   sensitive: false
               }, {
                   id: 26,
                   name: "throttlingConfig.numLoaders",
                   value: "3",
                   type: "INTEGER",
                   sensitive: false
               }],
               name: "throttlingConfig",
               type: "JOB"
           }],
           enabled: true,
           from-connector-name: "fromConnectorName",
           update-user: "root",
           to-config-values: [{
               id: 6,
               inputs: [{
                   id: 19,
                   name: "toJobConfig.schemaName",
                   type: "STRING",
                   size: 50,
                   sensitive: false
               }, {
```

```
                    id: 20,
                    name: "toJobConfig.tableName",
                    value: "text",
                    type: "STRING",
                    size: 2000,
                    sensitive: false
                }, {
                    id: 21,
                    name: "toJobConfig.sql",
                    type: "STRING",
                    size: 50,
                    sensitive: false
                }, {
                    id: 22,
                    name: "toJobConfig.columns",
                    type: "STRING",
                    size: 50,
                    sensitive: false
                }, {
                    id: 23,
                    name: "toJobConfig.stageTableName",
                    type: "STRING",
                    size: 2000,
                    sensitive: false
                }, {
                    id: 24,
                    name: "toJobConfig.shouldClearStageTable",
                    type: "BOOLEAN",
                    sensitive: false
                }],
                name: "toJobConfig",
                type: "JOB"
            }],
            to-connector-name: "toConnectorName",
            creation-date: 1415310157618,
            update-date: 1415310157618,
            creation-user: "root",
            id: 1,
            to-link-name: "toLinkName",
            from-config-values: [{
                id: 2,
                inputs: [{
                    id: 2,
                    name: "fromJobConfig.inputDirectory",
                    value: "hdfs%3A%2F%2Fvbsqoop-1.ent.cloudera.com%3A8020%2Fuser%2Froot%2Fjob1",
                    type: "STRING",
                    size: 255,
                    sensitive: false
                }],
                name: "fromJobConfig",
                type: "JOB"
            }],
            name: "First Job",
            from-link-name: "fromLinkName"
    }
}
```

/v1/job - [POST] - Create Job

Create a new job object with the corresponding config values.

- Method: `POST`
- Format: `JSON`
- Fields of Request:

Field	Description
`job`	The root of the post data in JSON
`from-link-name`	The name of the from link for the job
`to-link-name`	The name of the to link for the job
`id`	The id of the link can be left blank in the post data
`enabled`	Whether to enable this job (true/false)
`update-date`	The last updated time of this job
`creation-date`	The creation time of this job
`update-user`	The user who updated this job
`creation-user`	The uset who creates this job
`name`	The name of this job
`from-config-values`	Config input values for FROM part of the job
`to-config-values`	Config input values for TO part of the job
`driver-config-values`	Config input values for driver
`from-connector-name`	The name of the from connector for the job
`to-connector-name`	The name of the to connector for the job

- Request Example:

```
{
  job: {
    driver-config-values: [
      {
        id: 7,
        inputs: [
          {
            id: 25,
            name: "throttlingConfig.numExtractors",
            value: "3",
            type: "INTEGER",
            sensitive: false
          },
          {
            id: 26,
            name: "throttlingConfig.numLoaders",
            value: "3",
            type: "INTEGER",
            sensitive: false
          }
        ],
        name: "throttlingConfig",
        type: "JOB"
      }
    ],
    enabled: true,
    from-connector-name: "fromConnectorName",
    update-user: "root",
    to-config-values: [
      {
```

```
      id: 6,
      inputs: [
        {
          id: 19,
          name: "toJobConfig.schemaName",
          type: "STRING",
          size: 50,
          sensitive: false
        },
        {
          id: 20,
          name: "toJobConfig.tableName",
          value: "text",
          type: "STRING",
          size: 2000,
          sensitive: false
        },
        {
          id: 21,
          name: "toJobConfig.sql",
          type: "STRING",
          size: 50,
          sensitive: false
        },
        {
          id: 22,
          name: "toJobConfig.columns",
          type: "STRING",
          size: 50,
          sensitive: false
        },
        {
          id: 23,
          name: "toJobConfig.stageTableName",
          type: "STRING",
          size: 2000,
          sensitive: false
        },
        {
          id: 24,
          name: "toJobConfig.shouldClearStageTable",
          type: "BOOLEAN",
          sensitive: false
        }
      ],
      name: "toJobConfig",
      type: "JOB"
    }
  ],
  to-connector-name: "toConnectorName",
  creation-date: 1415310157618,
  update-date: 1415310157618,
  creation-user: "root",
  id: -1,
  to-link-name: "toLinkName",
  from-config-values: [
    {
      id: 2,
```

```
        inputs: [
          {
            id: 2,
            name: "fromJobConfig.inputDirectory",
            value: "hdfs%3A%2F%2Fvbsqoop-1.ent.cloudera.com%3A8020%2Fuser%2Froot%2Fjob1",
            type: "STRING",
            size: 255,
            sensitive: false
          }
        ],
        name: "fromJobConfig",
        type: "JOB"
      }
    ],
    name: "Test Job",
    from-link-name: "fromLinkName"
  }
}
```

- Fields of Response:

Field	Description
`name`	The name assigned for this new created job
`validation-result`	The validation status for the job config and driver config inputs in the post data

- ERROR Response Example:

```
{
  "validation-result": [
      {
          "linkConfig": [
              {
                  "message": "Invalid URI. URI must either be null or a valid URI. Here are a few val
                  "status": "ERROR"
              }
          ]
      }
  ]
}
```

/v1/job/[jname] - [PUT] - Update Job

Update an existing job object with name [jname]. To make the procedure of filling inputs easier, the general practice is get the existing job object first and then change some of the inputs.

- Method: `PUT`
- Format: `JSON`

The same as Create Job.

- OK Response Example:

```
{
  "validation-result": [
      {}
  ]
}
```

/v1/job/[jname] - [DELETE] - Delete Job

Delete a job with name `jname`.

- Method: `DELETE`
- Format: `JSON`
- Request Content: `None`
- Response Content: `None`

/v1/job/[jname]/enable - [PUT] - Enable Job

Enable a job with name `jname`.

- Method: `PUT`
- Format: `JSON`
- Request Content: `None`
- Response Content: `None`

/v1/job/[jname]/disable - [PUT] - Disable Job

Disable a job with name `jname`.

- Method: `PUT`
- Format: `JSON`
- Request Content: `None`
- Response Content: `None`

/v1/job/[jname]/start - [PUT]- Start Job

Start a job with name `[jname]` to trigger the job execution

- Method: `POST`
- Format: `JSON`
- Request Content: `None`
- Response Content: `Submission Record`
- BOOTING Response Example

```
{
  "submission": {
    "progress": -1,
    "last-update-date": 1415312531188,
    "external-id": "job_1412137947693_0004",
    "status": "BOOTING",
    "job": 2,
    "job-name": "jobName",
    "creation-date": 1415312531188,
    "to-schema": {
      "created": 1415312531426,
```

```
      "name": "HDFS file",
      "columns": []
    },
    "external-link": "http://vbsqoop-1.ent.cloudera.com:8088/proxy/application_1412137947693_0004/",
    "from-schema": {
      "created": 1415312531342,
      "name": "text",
      "columns": [
        {
          "name": "id",
          "nullable": true,
          "unsigned": null,
          "type": "FIXED_POINT",
          "size": null
        },
        {
          "name": "txt",
          "nullable": true,
          "type": "TEXT",
          "size": null
        }
      ]
    }
  }
}
```

- SUCCEEDED Response Example

```
{
  submission: {
    progress: -1,
    last-update-date: 1415312809485,
    external-id: "job_1412137947693_0004",
    status: "SUCCEEDED",
    job: 2,
    job-name: "jobName",
    creation-date: 1415312531188,
    external-link: "http://vbsqoop-1.ent.cloudera.com:8088/proxy/application_1412137947693_0004/",
    counters: {
      org.apache.hadoop.mapreduce.JobCounter: {
        SLOTS_MILLIS_MAPS: 373553,
        MB_MILLIS_MAPS: 382518272,
        TOTAL_LAUNCHED_MAPS: 10,
        MILLIS_MAPS: 373553,
        VCORES_MILLIS_MAPS: 373553,
        OTHER_LOCAL_MAPS: 10
      },
      org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter: {
        BYTES_WRITTEN: 0
      },
      org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter: {
        BYTES_READ: 0
      },
      org.apache.hadoop.mapreduce.TaskCounter: {
        MAP_INPUT_RECORDS: 0,
        MERGED_MAP_OUTPUTS: 0,
        PHYSICAL_MEMORY_BYTES: 4065599488,
        SPILLED_RECORDS: 0,
        COMMITTED_HEAP_BYTES: 3439853568,
```

```
        CPU_MILLISECONDS: 236900,
        FAILED_SHUFFLE: 0,
        VIRTUAL_MEMORY_BYTES: 15231422464,
        SPLIT_RAW_BYTES: 1187,
        MAP_OUTPUT_RECORDS: 1000000,
        GC_TIME_MILLIS: 7282
      },
      org.apache.hadoop.mapreduce.FileSystemCounter: {
        FILE_WRITE_OPS: 0,
        FILE_READ_OPS: 0,
        FILE_LARGE_READ_OPS: 0,
        FILE_BYTES_READ: 0,
        HDFS_BYTES_READ: 1187,
        FILE_BYTES_WRITTEN: 1191230,
        HDFS_LARGE_READ_OPS: 0,
        HDFS_WRITE_OPS: 10,
        HDFS_READ_OPS: 10,
        HDFS_BYTES_WRITTEN: 276389736
      },
      org.apache.sqoop.submission.counter.SqoopCounters: {
        ROWS_READ: 1000000
      }
    }
  }
}
```

- ERROR Response Example

```
{
  "submission": {
    "progress": -1,
    "last-update-date": 1415312390570,
    "status": "FAILURE_ON_SUBMIT",
    "error-summary": "org.apache.sqoop.common.SqoopException: GENERIC_HDFS_CONNECTOR_0000:Error occu
    "job": 1,
    "job-name": "jobName",
    "creation-date": 1415312390570,
    "to-schema": {
      "created": 1415312390797,
      "name": "text",
      "columns": [
        {
          "name": "id",
          "nullable": true,
          "unsigned": null,
          "type": "FIXED_POINT",
          "size": null
        },
        {
          "name": "txt",
          "nullable": true,
          "type": "TEXT",
          "size": null
        }
      ]
    },
    "from-schema": {
      "created": 1415312390778,
      "name": "HDFS file",
```

```
      "columns": [
      ]
    },
    "error-details": "org.apache.sqoop.common.SqoopException: GENERIC_HDFS_CONNECTOR_00"
  }
}
```

/v1/job/[jname]/stop - [PUT]- Stop Job

Stop a job with name `[jname]` to abort the running job.

- Method: `PUT`
- Format: `JSON`
- Request Content: `None`
- Response Content: `Submission Record`

/v1/job/[jname]/status - [GET]- Get Job Status

Get status of the running job with name `[jname]`

- Method: `GET`
- Format: `JSON`
- Request Content: `None`
- Response Content: `Submission Record`

```
{
    "submission": {
        "progress": 0.25,
        "last-update-date": 1415312603838,
        "external-id": "job_1412137947693_0004",
        "status": "RUNNING",
        "job": 2,
        "job-name": "jobName",
        "creation-date": 1415312531188,
        "external-link": "http://vbsqoop-1.ent.cloudera.com:8088/proxy/application_1412137947693_000
    }
}
```

/v1/submissions? - [GET] - Get all job Submissions

Get all the submissions for every job started in SQoop

/v1/submissions?jname=[jname] - [GET] - Get Submissions by Job

Retrieve all job submissions in the past for the given job. Each submission record will have details such as the status, counters and urls for those submissions.

Provide the name of the job in the url [jname] part.

- Method: `GET`

- Format: `JSON`
- Request Content: `None`
- Fields of Response:

Field	Description
`progress`	The progress of the running Sqoop job
`job`	The id of the Sqoop job
`job-name`	The name of the Sqoop job
`creation-date`	The submission timestamp
`last-update-date`	The timestamp of the last status update
`status`	The status of this job submission
`external-id`	The job id of Sqoop job running on Hadoop
`external-link`	The link to track the job status on Hadoop

- Response Example:

```
{
  submissions: [
    {
      progress: -1,
      last-update-date: 1415312809485,
      external-id: "job_1412137947693_0004",
      status: "SUCCEEDED",
      job: 2,
      job-name: "jobName",
      creation-date: 1415312531188,
      external-link: "http://vbsqoop-1.ent.cloudera.com:8088/proxy/application_1412137947693_0004/",
      counters: {
        org.apache.hadoop.mapreduce.JobCounter: {
          SLOTS_MILLIS_MAPS: 373553,
          MB_MILLIS_MAPS: 382518272,
          TOTAL_LAUNCHED_MAPS: 10,
          MILLIS_MAPS: 373553,
          VCORES_MILLIS_MAPS: 373553,
          OTHER_LOCAL_MAPS: 10
        },
        org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter: {
          BYTES_WRITTEN: 0
        },
        org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter: {
          BYTES_READ: 0
        },
        org.apache.hadoop.mapreduce.TaskCounter: {
          MAP_INPUT_RECORDS: 0,
          MERGED_MAP_OUTPUTS: 0,
          PHYSICAL_MEMORY_BYTES: 4065599488,
          SPILLED_RECORDS: 0,
          COMMITTED_HEAP_BYTES: 3439853568,
          CPU_MILLISECONDS: 236900,
          FAILED_SHUFFLE: 0,
          VIRTUAL_MEMORY_BYTES: 15231422464,
          SPLIT_RAW_BYTES: 1187,
          MAP_OUTPUT_RECORDS: 1000000,
          GC_TIME_MILLIS: 7282
        },
        org.apache.hadoop.mapreduce.FileSystemCounter: {
          FILE_WRITE_OPS: 0,
```

```
          FILE_READ_OPS: 0,
          FILE_LARGE_READ_OPS: 0,
          FILE_BYTES_READ: 0,
          HDFS_BYTES_READ: 1187,
          FILE_BYTES_WRITTEN: 1191230,
          HDFS_LARGE_READ_OPS: 0,
          HDFS_WRITE_OPS: 10,
          HDFS_READ_OPS: 10,
          HDFS_BYTES_WRITTEN: 276389736
        },
        org.apache.sqoop.submission.counter.SqoopCounters: {
          ROWS_READ: 1000000
        }
      }
    },
    {
      progress: -1,
      last-update-date: 1415312390570,
      status: "FAILURE_ON_SUBMIT",
      error-summary: "org.apache.sqoop.common.SqoopException: GENERIC_HDFS_CONNECTOR_0000:Error occu
      job: 1,
      job-name: "jobName",
      creation-date: 1415312390570,
      error-details: "org.apache.sqoop.common.SqoopException: GENERIC_HDFS_CONNECTOR_0000:Error occu
    }
  ]
}
```

/v1/authorization/roles/create - [POST] - Create Role

Create a new role object. Provide values to the link config inputs for the ones that are required.

- Method: `POST`
- Format: `JSON`
- Fields of Request:

Field	Description
`role`	The root of the post data in JSON
`name`	The name of this role

- Request Example:

```
{
  role: {
      name: "testRole",
  }
}
```

/v1/authorization/role/[role-name] - [DELETE] - Delete Role

Delete a role with name [role-name]

- Method: `DELETE`
- Format: `JSON`
- Request Content: `None`

- Response Content: `None`

/v1/authorization/roles?principal_type=[principal-type]&principal_name=[principal-name] - [GET] Get all Roles by Principal

Get all the roles or for a given principal identified by `[principal-type]` and `[principal-name]` part.

/v1/authorization/principals?role_name=[rname] - [GET] Get all Principals by Role

Get all the principals for a given role identified by `[rname]` part.

/v1/authorization/roles/grant - [PUT] - Grant a Role to a Principal

Grant a role with `[role-name]` to a principal with `[principal-type]` and `[principal-name]`.

- Method: `PUT`
- Format: `JSON`
- Fields of Request:

The same as Create Role and

Field	Description
`principals`	The root of the post data in JSON
`name`	The name of this principal
`type`	The type of this principal, ("USER", "GROUP", "ROLE")

- Request Example:

```
{
  roles: [{
      name: "testRole",
  }],
  principals: [{
      name: "testPrincipalName",
      type: "USER",
  }]
}
```

- Response Content: `None`

/v1/authorization/roles/revoke - [PUT] - Revoke a Role from a Principal

Revoke a role with `[role-name]` to a principal with `[principal-type]` and `[principal-name]`.

- Method: `PUT`
- Format: `JSON`
- Fields of Request:

The same as Grant Role

- Response Content: `None`

/v1/authorization/privileges/grant - [PUT] - Grant a Privilege to a Principal

Grant a privilege with `[resource-name]`, `[resource-type]`, `[action]` and `[with-grant-option]` to a principal with‘‘[principal-type]‘‘ and `[principal-name]`.

- Method: `PUT`
- Format: `JSON`
- Fields of Request:

The same as Principal and

Field	Description
`privileges`	The root of the post data in JSON
`resource-name`	The resource name of this privilege
`resource-type`	The resource type of this privilege, (“CONNECTOR”, “LINK”, “JOB”)
`action`	The action type of this privilege, (“READ”, “WRITE”, “ALL”)
`with-grant-option`	The resource type of this privilege

- Request Example:

```
{
  privileges: [{
      resource-name: "testResourceName",
      resource-type: "LINK",
      action: "READ",
      with-grant-option: false,
  }]
  principals: [{
      name: "testPrincipalName",
      type: "USER",
  }]
}
```

- Response Content: `None`

/v1/authorization/privileges/revoke - [PUT] - Revoke a Privilege to a Principal

Revoke a privilege with `[resource-name]`, `[resource-type]`, `[action]` and `[with-grant-option]` to a principal with‘‘[principal-type]‘‘ and `[principal-name]`.

- Method: `PUT`
- Format: `JSON`
- Fields of Request:

The same as Grant Privilege

- Response Content: `None`

/v1/authorization/privilieges?principal_type=[principal-type]&principal_name=[principal-name]&resource_type=[resource-type]&resource_name=[resource-name] - [GET] Get all Roles by Principal (and Resource)

Get all the privileges or for a given principal identified by `[principal-type]` and `[principal-name]` (and a given resource identified by `[resource-type]` and `[resource-name]`).

3.6 Repository

This repository contains additional information regarding Sqoop.

3.6.1 Sqoop Schema

The DDL queries that create the Sqoop repository schema in Derby database create the following tables:

SQ_SYSTEM

Store for various state information

SQ_SYSTEM
SQM_ID: BIGINT PK
SQM_KEY: VARCHAR(64)
SQM_VALUE: VARCHAR(64)

SQ_DIRECTION

Directions

SQ_DIRECTION	
SQD_ID: BIGINT PK AUTO-GEN	
SQD_NAME: VARCHAR(64)	“FROM”|”TO”

SQ_CONFIGURABLE

Configurable registration

SQ_CONFIGURABLE	
SQC_ID: BIGINT PK AUTO-GEN	
SQC_NAME: VARCHAR(64)	
SQC_CLASS: VARCHAR(255)	
SQC_TYPE: VARCHAR(32)	“CONNECTOR”|”DRIVER”
SQC_VERSION: VARCHAR(64)	

SQ_CONNECTOR_DIRECTIONS

Connector directions

SQ_CONNECTOR_DIRECTIONS	
SQCD_ID: BIGINT PK AUTO-GEN	
SQCD_CONNECTOR: BIGINT	FK SQCD_CONNECTOR(SQC_ID)
SQCD_DIRECTION: BIGINT	FK SQCD_DIRECTION(SQD_ID)

SQ_CONFIG

Config details

SQ_CONFIG	
SQ_CFG_ID: BIGINT PK AUTO-GEN	
SQ_CFG_CONNECTOR: BIGINT	FK SQ_CFG_CONNECTOR(SQC_ID), NULL for driver
SQ_CFG_NAME: VARCHAR(64)	
SQ_CFG_TYPE: VARCHAR(32)	“LINK”|”JOB”
SQ_CFG_INDEX: SMALLINT	

SQ_CONFIG_DIRECTIONS

Connector directions

SQ_CONNECTOR_DIRECTIONS	
SQCD_ID: BIGINT PK AUTO-GEN	
SQCD_CONFIG: BIGINT	FK SQCD_CONFIG(SQ_CFG_ID)
SQCD_DIRECTION: BIGINT	FK SQCD_DIRECTION(SQD_ID)

SQ_INPUT

Input details

SQ_INPUT	
SQI_ID: BIGINT PK AUTO-GEN	
SQI_NAME: VARCHAR(64)	
SQI_CONFIG: BIGINT	FK SQ_CONFIG(SQ_CFG_ID)
SQI_INDEX: SMALLINT	
SQI_TYPE: VARCHAR(32)	“STRING”|”MAP”
SQI_STRMASK: BOOLEAN	
SQI_STRLENGTH: SMALLINT	
SQI_ENUMVALS: VARCHAR(100)	

SQ_LINK

Stored links

SQ_LINK	
SQ_LNK_ID: BIGINT PK AUTO-GEN	
SQ_LNK_NAME: VARCHAR(64)	
SQ_LNK_CONNECTOR: BIGINT	FK SQ_CONNECTOR(SQC_ID)
SQ_LNK_CREATION_USER: VARCHAR(32)	
SQ_LNK_CREATION_DATE: TIMESTAMP	
SQ_LNK_UPDATE_USER: VARCHAR(32)	
SQ_LNK_UPDATE_DATE: TIMESTAMP	
SQ_LNK_ENABLED: BOOLEAN	

SQ_JOB

Stored jobs

SQ_JOB	
SQB_ID: BIGINT PK AUTO-GEN	
SQB_NAME: VARCHAR(64)	
SQB_FROM_LINK: BIGINT	FK SQ_LINK(SQ_LNK_ID)
SQB_TO_LINK: BIGINT	FK SQ_LINK(SQ_LNK_ID)
SQB_CREATION_USER: VARCHAR(32)	
SQB_CREATION_DATE: TIMESTAMP	
SQB_UPDATE_USER: VARCHAR(32)	
SQB_UPDATE_DATE: TIMESTAMP	
SQB_ENABLED: BOOLEAN	

SQ_LINK_INPUT

N:M relationship link and input

SQ_LINK_INPUT	
SQ_LNKI_LINK: BIGINT PK	FK SQ_LINK(SQ_LNK_ID)
SQ_LNKI_INPUT: BIGINT PK	FK SQ_INPUT(SQI_ID)
SQ_LNKI_VALUE: LONG VARCHAR	

SQ_JOB_INPUT

N:M relationship job and input

SQ_JOB_INPUT	
SQBI_JOB: BIGINT PK	FK SQ_JOB(SQB_ID)
SQBI_INPUT: BIGINT PK	FK SQ_INPUT(SQI_ID)
SQBI_VALUE: LONG VARCHAR	

SQ_SUBMISSION

List of submissions

SQ_JOB_SUBMISSION	
SQS_ID: BIGINT PK	
SQS_JOB: BIGINT	FK SQ_JOB(SQB_ID)
SQS_STATUS: VARCHAR(20)	
SQS_CREATION_USER: VARCHAR(32)	
SQS_CREATION_DATE: TIMESTAMP	
SQS_UPDATE_USER: VARCHAR(32)	
SQS_UPDATE_DATE: TIMESTAMP	
SQS_EXTERNAL_ID: VARCHAR(50)	
SQS_EXTERNAL_LINK: VARCHAR(150)	
SQS_EXCEPTION: VARCHAR(150)	
SQS_EXCEPTION_TRACE: VARCHAR(750)	

SQ_COUNTER_GROUP

List of counter groups

SQ_COUNTER_GROUP
SQG_ID: BIGINT PK
SQG_NAME: VARCHAR(75)

SQ_COUNTER

List of counters

SQ_COUNTER
SQR_ID: BIGINT PK
SQR_NAME: VARCHAR(75)

SQ_COUNTER_SUBMISSION

N:M Relationship

SQ_COUNTER_SUBMISSION	
SQRS_GROUP: BIGINT PK	FK SQ_COUNTER_GROUP(SQR_ID)
SQRS_COUNTER: BIGINT PK	FK SQ_COUNTER(SQR_ID)
SQRS_SUBMISSION: BIGINT PK	FK SQ_SUBMISSION(SQS_ID)
SQRS_VALUE: BIGINT	

CHAPTER 4

Security Guide

4.1 API TLS/SSL

Sqoop 2 offers an HTTP REST-like API as the mechanism by which clients can communicate with the Sqoop 2 server. The Sqoop 2 server and the Sqoop 2 shell have support for TLS/SSL.

4.1.1 Keystore Generation

Sqoop 2 uses the JKS format. Details on how to create JKS files can be found here: Generating a KeyStore and TrustStore

4.1.2 Server Configuration

All Sqoop 2 server TLS/SSL configuration occurs in the Sqoop configuration file, normally in `<Sqoop Folder>/conf/sqoop.properties`.

First, TLS must be enabled:

```
org.apache.sqoop.security.tls.enabled=true
```

A protocol should be specified. Please find a list of options here: Standard Algorithm Name Documentation

```
org.apache.sqoop.security.tls.protocol="TLSv1.2"
```

Configure the path to the JKS keystore:

```
org.apache.sqoop.security.tls.keystore=/Users/abe/mykeystore.jks
```

Configure the keystore password and the key manager password:

```
org.apache.sqoop.security.tls.keystore_password=keystorepassword
org.apache.sqoop.security.tls.keymanager_password=keymanagerpassword
```

Alternatively, the password can be specified using generators.

Generators are commands that the Sqoop propess will execute, and then retrieve the password from standard out. The generator will only be run if no standard password is configured.

```
org.apache.sqoop.security.tls.keystore_password_generator=echo keystorepassword
org.apache.sqoop.security.tls.keymanager_password=echo keymanagerpassword
```

4.1.3 Client/Shell Configuration

When using TLS on the Sqoop 2 server, especially with a self-signed certificate, it may be useful to specify a truststore for the client/shell to use.

The truststore for the shell is configured via a command. In practice, it may be useful to put this command inside the system sqoop rc file (`/etc/sqoop2/conf/sqoop2rc`) or the user's rc file (`~/.sqoop2rc`).

```
sqoop:000> set truststore --truststore /Users/abefine/keystore/node2.truststore
Truststore set successfully
```

You may also include a password. Passwords are not required for truststores.

```
sqoop:000> set truststore --truststore /Users/abefine/keystore/node2.truststore --truststore-passwor
Truststore set successfully
```

You may also use a password generator.

```
sqoop:000> set truststore --truststore /Users/abefine/keystore/node2.truststore --truststore-passwor
Truststore set successfully
```

4.2 Authentication and Authorization

Most Hadoop components, such as HDFS, Yarn, Hive, etc., have security frameworks, which support Simple, Kerberos and LDAP authentication. currently Sqoop 2 provides 2 types of authentication: simple and kerberos. The authentication module is pluggable, so more authentication types can be added. Additionally, a new role based access control is introduced in Sqoop 1.99.6. We recommend to use this capability in multi tenant environments, so that malicious users can't easily abuse your created link and job objects.

4.2.1 Simple Authentication

Configuration

Modify Sqoop configuration file, normally in <Sqoop Folder>/conf/sqoop.properties.

```
org.apache.sqoop.authentication.type=SIMPLE
org.apache.sqoop.authentication.handler=org.apache.sqoop.security.authentication.SimpleAuthenticatio
org.apache.sqoop.anonymous=true
```

- Simple authentication is used by default. Commenting out authentication configuration will yield the use of simple authentication.

Run command

Start Sqoop server as usual.

```
<Sqoop Folder>/bin/sqoop.sh server start
```

Start Sqoop client as usual.

```
<Sqoop Folder>/bin/sqoop.sh client
```

4.2.2 Kerberos Authentication

Kerberos is a computer network authentication protocol which works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Its designers aimed it primarily at a client–server model and it provides mutual authentication—both the user and the server verify each other's identity. Kerberos protocol messages are protected against eavesdropping and replay attacks.

Dependency

Set up a KDC server. Skip this step if KDC server exists. It's difficult to cover every way Kerberos can be setup (ie: there are cross realm setups and multi-trust environments). This section will describe how to setup the sqoop principals with a local deployment of MIT kerberos.

- All components which are Kerberos authenticated need one KDC server. If current Hadoop cluster uses Kerberos authentication, there should be a KDC server.
- If there is no KDC server, follow http://web.mit.edu/kerberos/krb5-devel/doc/admin/install_kdc.html to set up one.

Configure Hadoop cluster to use Kerberos authentication.

- Authentication type should be cluster level. All components must have the same authentication type: use Kerberos or not. In other words, Sqoop with Kerberos authentication could not communicate with other Hadoop components, such as HDFS, Yarn, Hive, etc., without Kerberos authentication, and vice versa.
- How to set up a Hadoop cluster with Kerberos authentication is out of the scope of this document. Follow the related links like https://hadoop.apache.org/docs/r2.5.0/hadoop-project-dist/hadoop-common/SecureMode.html

Create keytab and principal for Sqoop 2 via kadmin in command line.

```
addprinc -randkey HTTP/<FQDN>@<REALM>
addprinc -randkey sqoop/<FQDN>@<REALM>
xst -k /home/kerberos/sqoop.keytab HTTP/<FQDN>@<REALM>
xst -k /home/kerberos/sqoop.keytab sqoop/<FQDN>@<REALM>
```

- The <FQDN> should be replaced by the FQDN of the server, which could be found via "hostname -f" in command line.
- The <REALM> should be replaced by the realm name in krb5.conf file generated when installing the KDC server in the former step.
- The principal HTTP/<FQDN>@<REALM> is used in communication between Sqoop client and Sqoop server. Since Sqoop server is an http server, so the HTTP principal is a must during SPNEGO process, and it is case sensitive.
- Http request could be sent from other client like browser, wget or curl with SPNEGO support.
- The principal sqoop/<FQDN>@<REALM> is used in communication between Sqoop server and Hdfs/Yarn as the credential of Sqoop server.

Configuration

Modify Sqoop configuration file, normally in <Sqoop Folder>/conf/sqoop.properties.

```
org.apache.sqoop.authentication.type=KERBEROS
org.apache.sqoop.authentication.handler=org.apache.sqoop.security.authentication.KerberosAuthenticati
org.apache.sqoop.authentication.kerberos.principal=sqoop/_HOST@<REALM>
org.apache.sqoop.authentication.kerberos.keytab=/home/kerberos/sqoop.keytab
org.apache.sqoop.authentication.kerberos.http.principal=HTTP/_HOST@<REALM>
```

```
org.apache.sqoop.authentication.kerberos.http.keytab=/home/kerberos/sqoop.keytab
org.apache.sqoop.authentication.kerberos.proxyuser=true
```

- When _HOST is used as FQDN in principal, it will be replaced by the real FQDN. https://issues.apache.org/jira/browse/HADOOP-6632
- If parameter proxyuser is set true, Sqoop server will use proxy user mode (sqoop delegate real client user) to run Yarn job. If false, Sqoop server will use sqoop user to run Yarn job.

Run command

Set SQOOP2_HOST to FQDN.

```
export SQOOP2_HOST=$(hostname -f).
```

- The <FQDN> should be replaced by the FQDN of the server, which could be found via "hostname -f" in command line.

Start Sqoop server using sqoop user.

```
sudo -u sqoop <Sqoop Folder>/bin/sqoop.sh server start
```

Run kinit to generate ticket cache.

```
kinit HTTP/<FQDN>@<REALM> -kt /home/kerberos/sqoop.keytab
```

Start Sqoop client.

```
<Sqoop Folder>/bin/sqoop.sh client
```

Verify

If the Sqoop server has started successfully with Kerberos authentication, the following line will be in <@LOGDIR>/sqoop.log:

```
2014-12-04 15:02:58,038 INFO  security.KerberosAuthenticationHandler [org.apache.sqoop.security.authe
```

If the Sqoop client was able to communicate with the Sqoop server, the following will be in <@LOGDIR>/sqoop.log :

```
Refreshing Kerberos configuration
Acquire TGT from Cache
Principal is HTTP/<FQDN>@HADOOP.COM
null credentials from Ticket Cache
principal is HTTP/<FQDN>@HADOOP.COM
Will use keytab
Commit Succeeded
```

4.2.3 Customized Authentication

Users can create their own authentication modules. By performing the following steps:

- Create customized authentication handler extends abstract class AuthenticationHandler.
- Implement abstract function doInitialize and secureLogin in AuthenticationHandler.

```
public class MyAuthenticationHandler extends AuthenticationHandler {

  private static final Logger LOG = Logger.getLogger(MyAuthenticationHandler.class);

  public void doInitialize() {
    securityEnabled = true;
  }

  public void secureLogin() {
    LOG.info("Using customized authentication.");
  }
}
```

- Modify configuration org.apache.sqoop.authentication.handler in <Sqoop Folder>/conf/sqoop.properties and set it to the customized authentication handler class name.
- Restart the Sqoop server.

4.2.4 Authorization

Users, Groups, and Roles

At the core of Sqoop's authorization system are users, groups, and roles. Roles allow administrators to give a name to a set of grants which can be easily reused. A role may be assigned to users, groups, and other roles. For example, consider a system with the following users and groups.

```
<User>: <Groups>
user_all: group1, group2
user1: group1
user2: group2
```

Sqoop roles must be created manually before being used, unlike users and groups. Users and groups are managed by the login system (Linux, LDAP or Kerberos). When a user wants to access one resource (connector, link, connector), the Sqoop2 server will determine the username of this user and the groups associated. That information is then used to determine if the user should have access to this resource being requested, by comparing the required privileges of the Sqoop operation to the user privileges using the following rules.

- User privileges (Has the privilege been granted to the user?)
- Group privileges (Does the user belong to any groups that the privilege has been granted to?)
- Role privileges (Does the user or any of the groups that the user belongs to have a role that grants the privilege?)

Administrator

There is a special user: administrator, which can't be created, deleted by command. The only way to set administrator is to modify the configuration file. Administrator could run management commands to create/delete roles. However, administrator does not implicitly have all privileges. Administrator has to grant privilege to him/her if he/she needs to request the resource.

Role management commands

```
CREATE ROLE -role role_name
DROP ROLE -role role_name
SHOW ROLE
```

- Only the administrator has privilege for this.

Principal management commands

```
GRANT ROLE --principal-type principal_type --principal principal_name --role role_name
REVOKE ROLE --principal-type principal_type --principal principal_name --role role_name
SHOW ROLE --principal-type principal_type --principal principal_name
SHOW PRINCIPAL --role role_name
```

- principal_type: USER | GROUP | ROLE

Privilege management commands

```
GRANT PRIVILEGE --principal-type principal_type --principal principal_name --resource-type resource_t
REVOKE PRIVILEGE --principal-type principal_type --principal principal_name [--resource-type resource
SHOW PRIVILEGE -principal-type principal_type -principal principal_name [--resource-type resource_typ
```

- principal_type: USER | GROUP | ROLE
- resource_type: CONNECTOR | LINK | JOB
- action_type: ALL | READ | WRITE
- With with-grant in GRANT PRIVILEGE command, this principal could grant his/her privilege to other users.
- Without resource in REVOKE PRIVILEGE command, all privileges on this principal will be revoked.
- With with-grant in REVOKE PRIVILEGE command, only grant privilege on this principal will be removed. This principal has the privilege to access this resource, but he/she could not grant his/her privilege to others.
- Without resource in SHOW PRIVILEGE command, all privileges on this principal will be listed.

4.3 Repository Encryption

Sqoop 2 uses a database to store metadata about the various data sources it talks to, we call this database the repository.

The repository can store passwords and other pieces of information that are security sensitive, within the context of Sqoop 2, this information is referred to as sensitive inputs. Which inputs are considered sensitive is determined by the connector.

We support encrypting sensitive inputs in the repository using a provided password or password generator. Sqoop 2 uses the provided password and the provided key generation algorithm (such as PBKDF2) to generate a key to encrypt sensitive inputs and another hmac key to verify their integrity.

Only the sensitive inputs are encrypted. If an input is not defined as sensitive by the connector, it is NOT encrypted.

4.3.1 Server Configuration

Note: This configuration will allow a new Sqoop instance to encrypt information or read from an already encrypted repository. It will not encrypt sensitive inputs in an existing repository. For instructions on how to encrypt an existing repository, please look here: *RepositoryEncryption*

First, repository encryption must be enabled.

```
org.apache.sqoop.security.repo_encryption.enabled=true
```

Then we configure the password:

```
org.apache.sqoop.security.repo_encryption.password=supersecret
```

Or the password generator:

```
org.apache.sqoop.security.repo_encryption.password_generator=echo supersecret
```

The plaintext password is always given preference to the password generator if both are present.

Then we can configure the HMAC algorithm. Please find the list of possibilities here: Standard Algorithm Name Documentation - Mac We can store digests with up to 1024 bits.

```
org.apache.sqoop.security.repo_encryption.hmac_algorithm=HmacSHA256
```

Then we configure the cipher algorithm. Possibilities can be found here: Standard Algorithm Name Documentation - Cipher

```
org.apache.sqoop.security.repo_encryption.cipher_algorithm=AES
```

Then we configure the key size for the cipher in bytes. We can store up to 1024 bit keys.

```
org.apache.sqoop.security.repo_encryption.cipher_key_size=16
```

Next we need to specify the cipher transformation. The options for this field are listed here: Cipher (Java Platform SE 7)

```
org.apache.sqoop.security.repo_encryption.cipher_spec=AES/CBC/PKCS5Padding
```

The size of the initialization vector to use in bytes. We support up to 1024 bit initialization vectors.

```
org.apache.sqoop.security.repo_encryption.initialization_vector_size=16
```

Next we need to specfy the algorithm for secret key generation. Please refer to: Standard Algorithm Name Documentation - SecretKeyFactory

```
org.apache.sqoop.security.repo_encryption.pbkdf2_algorithm=PBKDF2WithHmacSHA1
```

Finally specify the number of rounds/iterations for the generation of a key from a password.

```
org.apache.sqoop.security.repo_encryption.pbkdf2_rounds=4000
```

CHAPTER 5

Administrator Guide

If you are a admin trying to set up Sqoop, check out the links below

- Sqoop Server and Client Installation
- Sqoop Server Upgrade
- Sqoop Tools

CHAPTER 6

User Guide

If you are excited to start using Sqoop you can follow the links below to get a quick overview of the system

- Sqoop 5 Minute Demo
- Command Line Shell Usage Guide
- Connectors

CHAPTER 7

Developer Guide

If you are keen on contributing to Sqoop and get your hands dirty building connectors or interesting UI/applications for Sqoop internals check out the links below

- Building Sqoop 2
- Sqoop Development Environment Setup
- Developing a Sqoop Connector with Connector API
- Developing Sqoop application with REST API
- Developing Sqoop application using Sqoop Java Client API
- Repository

CHAPTER 8

Security:

- Security Guide

CHAPTER 9

License

Sqoop is licensed under Apache Software License v2.