Authors:

Abhinav Meghmala

Manish Bajaj

Prabhu Om

Led By:

Nitin Goel, Sangamesh Gugwad & Anup Goel

BT e-Serv India Private Limited.

Abstract

Configure your details, run the application and get a fully-fledged coordinator and workflow scheduled in Oozie with your preferences and every action audited!

Hadoop   
data integration tool ver 1.0

Documentation & User guidebook

Table of Contents

[1.1 Objective 2](#_Toc449685799)

[1.2 Overview 2](#_Toc449685800)

[1.3 Architecture Overview 3](#_Toc449685801)

[1.4 Configuration file parameter description 3](#_Toc449685802)

[1.5 Directory structure 6](#_Toc449685803)

[1.6 Module description 7](#_Toc449685804)

[1.6.1 Constants 7](#_Toc449685805)

[1.6.2 Database 7](#_Toc449685806)

[1.6.3 Hive 9](#_Toc449685807)

[1.6.4 Sqoop 10](#_Toc449685808)

[1.6.5 Pig 10](#_Toc449685809)

[1.6.6 Shell 10](#_Toc449685810)

[1.6.7 Config 11](#_Toc449685811)

[1.6.8 Utilities 11](#_Toc449685812)

[1.6.9 Oozie 11](#_Toc449685813)

[1.6.10 FileSystem 11](#_Toc449685814)

[1.6.11 HTTPFS Client 12](#_Toc449685815)

[1.7 Job.properties file 12](#_Toc449685816)

[1.8 Running the application for importing data using sqoop 12](#_Toc449685817)

[1.9 Running the application for pushing/pulling file to/from HDFS using HTTPFS client 13](#_Toc449685818)

[1.10 Future Enhancements 15](#_Toc449685819)

[1.11 References 15](#_Toc449685820)

HADOOP DATA INTEGRATION TOOL (HDI Tool) 1.0

# Objective

Users who want to land their data from different database/filesystem sources in HaaS and perform specific analytics on them, have to go through a lot of challenges. Among these challenges the major ones are to understand the architecture of HaaS, get acquainted with the followed conventions, and then finally come up with their own scripts, workflows and endless testing for them before it goes live. This tool removes these hurdles and gives the ease to the user to have fully fledged scripts and workflows generated and most importantly tested before go live! All the user has to do is to input the required details in a configuration file and that’s it!

# 1.2 Overview

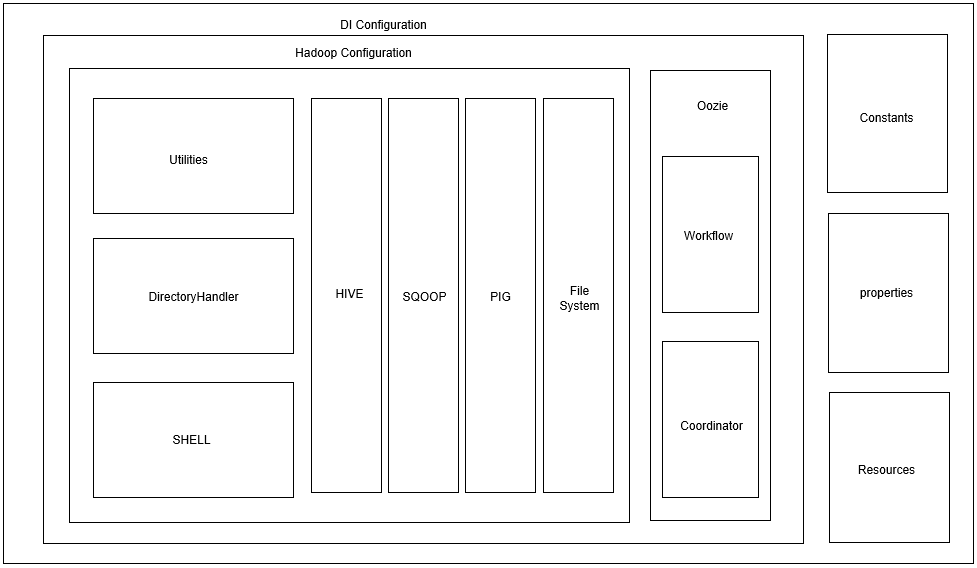
Hadoop data integration tool is designed for those users who want to land their data in HDFS and perform analytics on them. This tool takes a property/configuration file as input from the user and runs the application which validates the following –

* All the user provided details.
* Connection to source database.
* Sqoop import (for one record) as a check.
* Generates all required files and pushes them to HDFS.
* Generates a workflow to be scheduled in Oozie.

This tool gives the advantage to just run the application and have all the scripts, required files and the Oozie coordinator and workflow xml generated automatically which in turn will perform milestone/incremental pulls as AVRO data files, have HIVE table created over them.

One of the catching features of this tool is that it will give the user an AUDIT table for all the workflow actions, its failures if any along with more useful details which will help keep a track of the entire workflow.

# 1.3 Architecture Overview



# **1.4 Configuration file parameter description**



Parameter description –

|  |  |
| --- | --- |
| **PARAMETER** | **DESCRIPTION** |
| **Sqoop Import Parameters** | |
| import\_export\_flag | This parameter is used for sqoop. 1 for IMPORT and 2 for EXPORT. |
| database\_host | Source database host name. |
| datebase\_port | Source database port name. |
| database\_sid | Source database service ID name. |
| database\_username | Source database username name. |
| database\_password | Source database password name. |
| database\_tablename | Source database table name. |
| RDBMS | This parameter is for RDBMS source database type. As of now 1 for ORACLE. |
| DBDriver | For Oracle - oracle.jdbc.driver.OracleDriver. |
| split\_by\_column | This parameter will be used by sqoop .This has to be specified by the user if number of mappers are not specified. It is favoured to have this column a unique identifier say primary key. |
| last\_modified\_date\_column | This parameter will be used by sqoop for incremental import. A date column is suggested. |
| user\_selected\_columns | List of user specified columns if you don’t want to import all the columns. This will be used in sqoop. By default it is null, which means all the columns will be imported. |
| sqoopFileFormat | This parameter is to let you choose the file format in which you want the data in HDFS. 1-AVRO, 2-Sequence file, 3-Text. As of now the only and default option is AVRO (1) which will provide an automatic schema evolution and handling altering column structures. |
| is\_incremental | This parameter is to be used by sqoop. If true then the values will be imported incrementally. |
| incremental\_mode | This parameter will be used by sqoop. Incremental mode could be lastmodified or append depending on the need. |
| last\_value | This parameter is for sqoop incremental import based on a value post which the values will be imported from the source database. Note: This is a value not a column name. |
| direct\_flag | If true, then sqoop direct import will be used. |
| **Oozie Workflow** | |
| Environment\_Details | This parameter is to decide whether to run the workflow in a cluster or a VM. 1. |
| appNameNode | This name node will be used by the application JAR to run commands in sqoop/pig. |
| workflowNameNode | This name node will be used in the Oozie workflow.xml. |
| hive2\_jdbc\_url | Used in Oozie workflow.xml for Hive Action. |
| hive2\_server\_principal | Used in Oozie workflow.xml for Hive Action in case of Kerberos authentication. |
| jobTracker | This is name of the job tracker needed to run the Oozie workflow in HDFS (e.g. yarnRM). |
| cluster\_haas\_instance\_name | This parameter refers to the Instance or the Username in HDFS with which a respective database of similar name is already existing in Hive Metastore. |
| success\_email\_id | This parameter will be used in Oozie Email action for success alerts. |
| failure\_email\_id | This parameter will be used in Oozie Email action for failure alerts. |
| no\_of\_mappers | Will be used in sqoop –m <no\_of\_mappers> command parameter. |
| coordinator\_required | This parameter decides if you need a coordinator for your workflow (whether you want to schedule your workfow) if not, workflow will run only once. Required = true else false |
| workflow\_start\_time | If you choose coordinator\_required to be true, you will have to specify a start time for your scheduler, else it will fail. |
| workflow\_end\_time | If you choose coordinator\_required to be true, you will have to specify an end time for your scheduler, else it will fail. |
| time\_zone | If you choose coordinator\_required to be true, you will have to specify a timezome time for your scheduler, else it will fail. |
| concurrency | This parameter is for oozie coordinator for number of parallel threads you want to run at a time or number of parallel executions for the same workflow. By default it is 0. |
| throttle | This parameter decides how many maximum coordinator actions are allowed to be in WAITING state concurrently. |
| timeout | This parameter decides how long the coordinator action will be in **WAITING** or **READY** status before giving up on its execution. |
| frequency | This parameter is scheduling time in minutes. If 10 then the workflow will be triggered in every 10th minute. If 1440 then the workflow will be triggered every day. |
| retention\_period\_raw\_data | No of days you want the raw/sqoop imported data to be retained. |
| retention\_period\_processed\_data | No of days you want the data over which Hive table is created to be retained. |
| audit\_table\_name | Name of the Audit table created in hive. This table will capture the details such as workflow\_name, workflow\_id, status of job, job\_start\_time/end time etc. |
| housekeeping\_shell\_file | Shell file script name which will perform housekeeping based on the retention periods specified. |
| hdi\_audit\_table\_cols | Column names for Hive audit table. |
| audit\_shell\_file | Name of the script file which will be used for auditing. |
| audit\_log\_file\_path | Path of the log file in HDFS over which Hive Audit table will be created. |
| **Sqoop Export** | |
| export\_dir | Location of the export directory for sqoop export to destination database. |
| update\_key\_column | Anchor column to use for updates. Use a comma separated list of columns if there are more than one column. This parameter will be used by sqoop. |
| update\_mode | This parameter will be used by sqoop. Value of this parameter could be either ‘updateonly’ or ‘allowinsert’. By default it is ‘updateonly’. |
| columns\_name | Columns to export to table .It should be comma separated. |
| input\_null\_string | The string to be interpreted as null for string columns. |
| input-null-non-string | The string to be interpreted as null for non-string columns. |
| staging\_table | The table in which data will be staged before being inserted into the destination table. |
| clear\_staging\_table | Indicates that any data present in the staging table can be deleted. To be set either true or false. |
| stored\_procedure\_name | This parameter is to let sqoop export to know whether to use to use any stored procedure at destination database or not (in case of Oracle) |
| batch\_mode | Use batch mode for underlying statement execution. To be set either true or false. |

# 1.5 Directory structure

This application will create a specific directory structure in HDFS for the libraries, files, scripts and data which will be imported using sqoop. The followed convention is as below –

/user/<INSTANCE\_NAME>/<SOURCE\_TABLE\_NAME>/

Inside this we will have the following directories –

* **Workspace** – this directory will contain all script files for Hive, Pig & Shell. Ojdbc jar dependency for sqoop in case if it is not set in the sqoop lib path. This will be referred by Oozie workflow xml.
* **Raw\_data** – this directory will contain the data imported by sqoop as an AVRO file.
* **Password Repository** – This directory contains all the password files for respective databases and uses then in sqoop command in oozie workflow as well as in the application.

Following is the look and feel of the directory structure in HDFS -

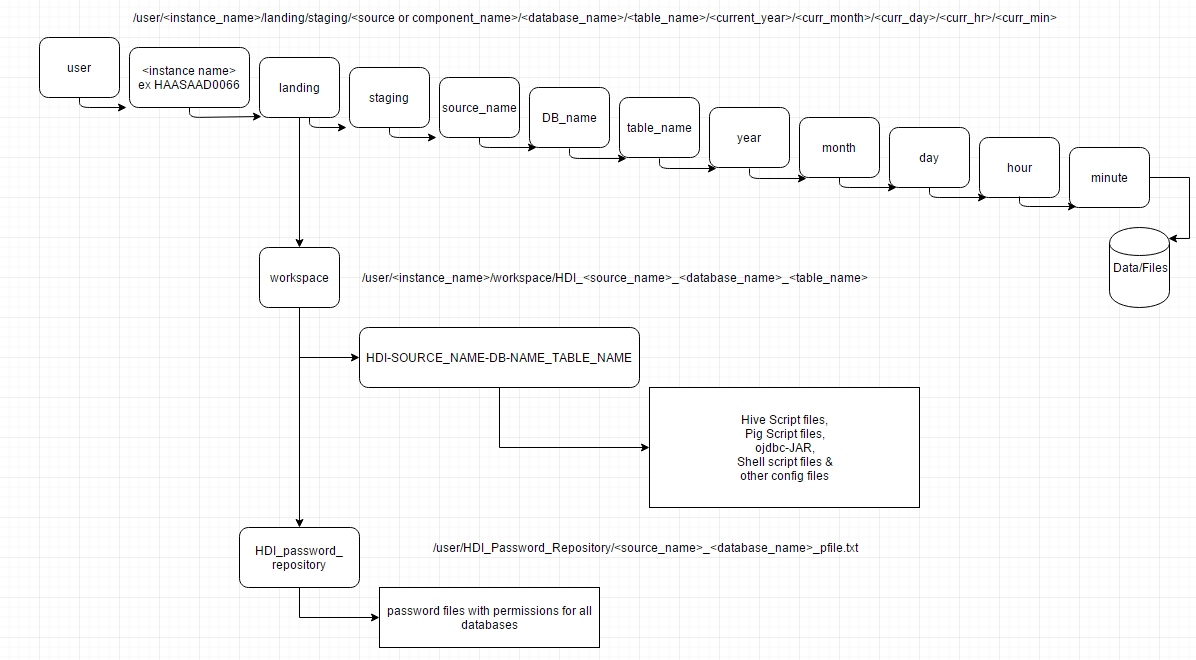


Fig 1.1 – Directory Structure in HDFS

# 1.6 Module description

## 1.6.1 Constants

This class contains all the constants which have been used throughout the application. For example the oozie workflow xmlns names, database property values etc. Some of these fields can be made configurable by keeping them in a properties file which the application could read at run time.

## 1.6.2 Database

This module contains an interface and its implementation.  
The implementation class connects to the database and checks if the user has catalogue permission, if the table exists, get table column details, constraint details, partition details. Store all the required table metadata in the job.properties file. This Class is called by DIConfigService class which has the main method and controls the execution/flow of the application.

Module description –

**package com.bt.dataintegration.database;**

Class – **DBConnectImpl**

SQL queries taken from **database.properties** file

Dependent on following classes –

**DIConfig** - POJO class for setting values read from configuration.properties file and setting them in this class.

**DIConfigService** – Service class which controls the flow of execution.

**public DBConnectImpl() -**

This is the default constructor for class. This module loads database.properties to be used by rest of the methods in this class.

**public Connection connect(DIConfig diConfig) –**

This method takes an object of DIConfig as parameter from service class and validates connection from source database using the jdbc driver. If connection is extablished, it returns the object of the Connection class.

**public boolean validateTable(Connection con, DIConfig conf) throws Exception –**

This method takes a Connection and DIConfig object as parameters from service class and validates if the table exists in the source database. This method throws an SQL exception is the table is not found and if found, returns true.

**public LinkedHashMap<String, List<String>> getColumnDetails(Connection con, DIConfig conf) throws SQLException –**

This method takes a Connection and DIConfig object from service class and retrieves the column names and respective datatypes from the source table, stores them in linked hash map to maintain the order of the columns and returns the same. This method throws an SQL exception of the column details are not found in the schema.

**public double getTableSize(Connection con, DIConfig conf) throws SQLException –**

This method takes a Connection and DIConfig object from service class and returns the size of the table specified in the configuration.properties file. Throws an SQL exception if query returns an error code.

**public Map<String, List<String>> getTablePartition(Connection con, DIConfig conf) throws SQLException –**

This method takes a Connection and DIConfig object from service class and returns the partition details for the user provided table. Throws an SQL exception if query fails at the database.

**public Map<String, String> getTableConstraints(Connection con, DIConfig conf) throws SQLException –**

This method takes a Connection and DIConfig object from service class and returns the constraint details (primary keys etc.) by querying to the database. Throws SQL exception if query fails.

**public int validateTabMetadata(Connection con,DIConfig conf) throws SQLException -**

This method takes a Connection and DIConfig object from service class and returns the count of the rows for the metadata entries found in the database for the table. This validates that the table has metadata entries in the database and validates its authenticity. Throws SQL exception if the query fails.

**public void storePassword(DIConfig conf) –**

This method stores the password of the source database provided by the user in a password file in HDFS which will be used by sqoop in –password-file parameter.

**public boolean validateCatalogPermission(Connection con,DIConfig conf) throws SQLException-**

This method takes a Connection and DIConfig object from service class and returns true if the user has a catalog permissions over it. Catalog permission is validated for sqoop direct option. If the user does not have a catalog permission, it is not eligible for sqoop direct.

**public String[] sqoopQueryBuilder(TableProperties tabProp , DIConfig conf) –**

This method takes a TableProperties and DIConfig object from service class and builds the query to be used in sqoop –query option foe milestone or incremental pulls. This method returns a String Array which holds the query.

**public void storeTableMetadata(TableProperties tabProp, DIConfig conf) –**

This method takes a TableProperties and DIConfig object from service class and stores the required data in key value pairs in job.properties file which will be used by Oozie to schedule and run coordinator and workflows respectively.

## 1.6.3 Hive

This module contains an Interface and its implementation.

The implementation class has query builder methods which will create the query strings for table creation and adding partitions and will push them as a script file in HDFS workspace location.

Before pushing the queries to HDFS, this module will try creating the table and adding partition in Hive and check its authenticity. If the table already exists then it will prompt the user for a new table name. After this has been validated, the table will be dropped (as it will be done in Oozie.)

Module Description –

**public String validateHiveTablePrivs(HadoopConfig hconf)**

This module creates a table in Hive with the provided table name in configuration.properties and adds a partition which finally validates that the same will be happening in Oozie. If the table already exists in Hive then it will prompt the user for a new table name and creates the table based on this name. Post all the checks are done, this table will be dropped as the same will be commenced in oozie.

**public String queryBuilder(HadoopConfig hconf)**

This module generates a hive query for create table in hive. This query is used by the above module.

**public String partitionBuilder(HadoopConfig hconf)**

This module generates the query for adding partition to the above created table. This query is used by validateHiveTablePrivs() method.

**public String partitionBuilderWorkflow(HadoopConfig hconf)**

This module generates the query to be used by oozie workflow.xml. Difference between this and the above module is that this query is generated with job.properties parameters. Example ${tableName}, ${partitionLocation} etc.

**public void writeHiveCreateQuery(String query, HadoopConfig hconf)**

This method writes the create table query which will be used by Oozie. This file will be created in the current directory where the application is running.

**public void writeHivePartitionQuery(String query, HadoopConfig hconf)**

This method writes the add partition to the created table query which will be used by Oozie. This file will be created in the current directory where the application is running.

**public void writeAuditTableCreateQuery(String query, HadoopConfig hconf)**

This method writes the create table query to maintain the audit logs which will be used by Oozie. This file will be created in the current directory where the application is running.

**public void copyHiveScriptsToHDFS(HadoopConfig hconf)**

This method copies all the above created scripts in HDFS location following the convention specified in directory structure in section 1.4

**public String queryBuilderWorkflow(HadoopConfig hconf)**

This method generates the create table query which will be used by Oozie.

**public String createAuditTableQueryBuilder(HadoopConfig hconf)**

This method generates the create audit table query which will be used by Oozie.

## 1.6.4 Sqoop

This module contains an Interface and its implementation.

The implementation class has methods to generate a sqoop command to extract only one row from the database.

Module Description –

**public boolean validateSqoopPropFileAccess()**

**public int sqoopImport(HadoopConfig conf) throws FileNotFoundException**

**public boolean validateFileFormat(Path location, Configuration conf)**

## 1.6.5 Pig

This module contains an Interface and its implementation.  
In the implementation class, a pig script is run to compress the data and a file pigCompress.pig is created and send to HDFS.

Module Description –

**public void compressData(HadoopConfig conf)**

**private void avscToHdfs(HadoopConfig conf)**

**public void generateAvscFile(HadoopConfig conf)**

## 1.6.6 Shell

This module contains an Interface and its implementation.  
In the implementation class, we create a text file unix\_date.txt. A shell script is written to capture the year, month, day, hour and minute with which the partition was created and save it to a file. A shell script to capture the audit details is also written to a file and send to HDFS.

Module Description –

**public void shellToHDFS(HadoopConfig conf)**

**public String refreshLastColValueShell()**

**public String updateLastColValueShell()**

**public String auditLogShell()**

**public String housekeepShell()**

## 1.6.7 Config

This module contains classes and respective methods for getting and setting configuration properties which is a user input configurable file to be used by the application. Also, this module has respective classes and methods for getting and setting values for those variables which will be read for config POJO and will be set with the variables needed in HDFS processing for e.g. nameNode, jobTracker, targetDirectories etc.

## 1.6.8 Utilities

The Utility class has generic methods used across the project. It has methods to read any property file, execute a shell command and clean up the workspace if an error has occurred.

## 1.6.9 Oozie

Oozie module has been divided in to three parts –

* Tags

This package contains POJO classes defining all the tags to be used in oozie workflow xml file.

* Main

This package populated values in the setter methods of the respective classed in the Tags module.

* Codegen

This package contains a class which populated the main workflow values finally and converts all the class definitions to an XML file using JAXB API.

This module will give two xml files. One is the coordinator.xml which will be used for scheduling the workflow as a job in HDFS. The other one will be the workflow.xml file which will have all the respective actions for shell, sqoop, hive, pig and email.

All these files will be saved in the workflow path in HDFS.

## 1.6.10 FileSystem

This module will be applicable for only file validations post any file/s pushed to HDFS.

The directory where the user has pushed the files must contain a success/control file of 0 byte size, the name of which has to be specified in the configuration.properties file. File type mask has to be provided in the config file as regex by the user Ex: \*cust\_\* etc.

This Class drives the following –

* File validations using the Hadoop FileSystem API
* Generates pig script for record level validations
* Generates hive scripts to create table over the valid data.

Apart from the above, this module also takes care of archiving data and rejecting invalid records in a separate directory which can be used for future reference.

Filesystem directories be categorized as following –

* Raw – which will contain the un-processed files.
* Temp – which will contain the valid files.
* Valid – which will contain the valid files with valid records.
* Archive – which will contain the valid files post processing.

Note: Housekeeping will be done over this archive directory.

## 1.6.11 HTTPFS Client

This module does file transfer based on Java HaaS Httpfs client which is a client utility written in java as part of HaaS common capabilities. This utility can be used to ingest the data into HaaS location using architecturally approved HTTPFS interface in HaaS.

This client is driven by below two files –

* + - haas-httpfs-config.properties
    - log4j.properties

The config file contains necessary cluster details required for Kerberos authentication.

This client can be used to push files from a server to HDFS or pull files from HDFS to a server.

Attached below is the detailed guide book to use this module.



# 1.7 Job.properties file



This file will be used to invoke the coordinator/workflow.xml in Oozie which will have the coordinator/workflow xml’s location in HDFS. The job.properties file will be created based on the inputs given by the user in the configuration.properties file. This file will be invoked using Oozie command at the edge/gateway node.

# 1.8 Running the application for importing data using sqoop

The application build comes as a zipped file which has the following –

* Configuration.properties
* Database.properties
* Ojdbc jar file
* Log4j.properties
* Application jar file
* run\_oozie\_workflow.sh

Steps –

1. Extract the zip file at any location in Unix server which the user has access to.
2. Copy hive-site.xml from /etc/hive/conf to the above location where the zip has been extracted.
3. Run the below in console –

java -cp \

hdi-tool-v1.0.jar:ojdbc6-11.2.0.3.jar:/opt/cloudera/parcels/CDH/jars/\* \ com.bt.dataintegration.property.config.DIConfigService

1. Post the application runs successfully, invoke the the run\_oozie\_workflow.sh file, which will trigger the workflow/coordinator.

Command – sh run\_oozie\_workflow.sh

For referencing any file or script, follow the locations specified in the section 1.3 (directory structure.)

# 1.9 Running the application for pushing/pulling file to/from HDFS using HTTPFS client

Prequisites:

Java JDK 1.7+

Any OS which supports Java JDK 1.7+

Step 1: Java Installation

Installation JDK 1.7 as private VM on the nominated OS system.

Go to the Installed java bin location using command/shell and type java –version. This should return the installed version of Java.

Step 2: Using FileSystem API from SKOOL.

The below two files need to be updated before running the final command -

**haas-httpfs-config.properties**

**log4j.properties**

log4j.properties is not necessarily updated unless you want your specific log formats or location to store the log file.

**haas-httpfs-config.properties 🡪** This is the configuration file which contains the credentials for Kerberos login and host and port of HaaS that host the REST services.This property file must in same location as **haas-httpfs-client-0.0.1.jar.**

The content of the properties file are:

***haas.authType = Kerberos***

***haas.realm = IUSER.IROOT.ADIDOM.COM***

***haas.kdc = iuuktvsmkp02.iuser.iroot.adidom.com***

**haas.servicePrincipal = HTTP/tplhc01g001.iuser.iroot.adidom.com**

**# tplhc01g001.iuser.iroot.adidom.com**

**haas.username = <CAD functional account ID>**

**haas.password = <Encrypted Password>**

**haas.httpfs.host = haas-1a.nat.bt.com**

**haas.httpfs.port = 14000**

**haas.kerberos.debug = true**

The haas.authType, haas.realm & haas.kdc properties will hardly change. So it is important that we don’t modify these properties.

Haas.servicePrincipal property must be sent to **HTTP/tplhc01g001.iuser.iroot.adidom.com** for RoBT cluster and **HTTP/tplhc01g004.iuser.iroot.adidom.com** for OR clsuter**.**

**haas.username 🡪** This is CAD functional account that have access service instance on HaaS.

**haas.password 🡪** This is encrypted CAD functional account. This is password is generated from java Httpfs client utility itself. The steps is mentioned below in the Password generation section.

**haas.httpfs.host & haas.httpfs.port 🡪** The host name & port where the REST services are hosted

For RoBT it is **haas-1a.nat.bt.com and port is 14000**

For OR it is **haas-1b.nat.bt.com and port is 14000**

**haas.kerberos.debug 🡪** This properties is used for debugging the Kerberos connection. It is can set to false as well.

**log4j.properties 🡪** This is log properties. The content is

log4j.appender.file=org.apache.log4j.RollingFileAppender

**log4j.appender.file.File=logs/haas-httpfs-client.log**

log4j.appender.file.MaxFileSize=20MB

log4j.appender.file.MaxBackupIndex=1

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

log4j.appender.file.File – must be set to the location in the OS for capturing the logs.

Onetime -Password –Generation 🡪 For generating the password for the first time **haas.password must be kept blank.**

**Steps to generate the password:**

**Open a command / Shell Set the private JVM java home**

**For Windows it should be**

**set JAVA\_HOME=<INSTALLED\_JAVA\_LOCATION>**

**set PATH=%JAVA\_HOME%\bin;%PATH%**

**For Non- Windows**

**export JAVA\_HOME=<INSTALLED\_JAVA\_LOCATION>**

**export PATH=$JAVA\_HOME/bin;$PATH**

**Run following command**

**java -cp skool.jar:/opt/cloudera/parcels/CDH/jars/\* com.bt.haas.httpfs.client.** **HaaSHTTPFSClient**

**This will prompt for password. Please enter the CAD functional password.**

**This will add password to the haas-httpfs-config.properties file.**

**Calling HTTPFS Rest Api using java Httpfs client you can get the list of commands supported in by java Httpfs client.**

**Run the commands**

**java -cp skool.jar:/opt/cloudera/parcels/CDH/jars/\* com.bt.haas.httpfs.client.** **HaaSHTTPFSClient -help**

This will list of commands with proper description of the commands .

For example: For ingesting a sample.txt file into HaaS you can run following command

**java -cp skool.jar:/opt/cloudera/parcels/CDH/jars/\* com.bt.haas.httpfs.client.** **HaaSHTTPFSClient –copyFromLocal /home/<CAD account ID>/sample.txt user/<HaaS Instance Name>/input\_files/**

**copyFromLocal** 🡪 This is file ingestion call

**/home/<CAD Account ID>/sample.txt** 🡪 this sample location of local file that need to be ingested.

**user/<HaaS Instance Name>/input\_files/** 🡪 location of destination HaaS location.

**You should get** HTTPFS Operation Success : 201 **after successful ingestion.**

# 1.10 Future Enhancements

* This tool is console driven as of now. In near future, this tool will have a lucrative user interface support.
* Support for exporting data to another RDBMS database.

# 1.11 References

Sqoop User Guide - <https://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html>

Hive User manual - <https://cwiki.apache.org/confluence/display/Hive/Tutorial>

Oozie Workflow - <https://oozie.apache.org/docs/3.2.0-incubating/WorkflowFunctionalSpec.html>

Functional Specifications

Oozie Coordinator Functional Specifications -

<https://oozie.apache.org/docs/3.1.3-incubating/CoordinatorFunctionalSpec.html>