Hadoop Foreign Data Wrapper Guide

Version hadoop_data_adapter

Built at 2020-12-10T15:31:51

1 Requirements Overview

Supported Versions

The Hadoop Foreign Data Wrapper is certified with PostgreSQL and EDB Postgres Advanced Server 9.5 and above.

Supported Platforms

The Hadoop Foreign Data Wrapper is supported on the following platforms:

Linux x86-64

- RHEL 8.x/7.x/6.x
- CentOS 8.x/7.x/6.x
- OEL 8.x/7.x/6.x
- Ubuntu 18.04+
- Debian 9.6.x

Linux on IBM Power8/9 (LE)

- RHEL 7.x

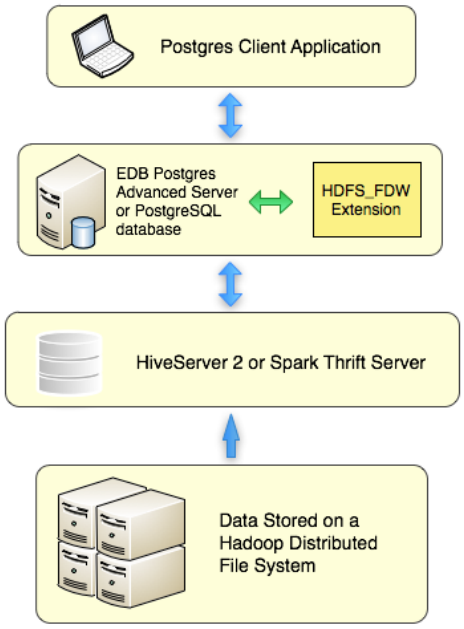
The Hadoop Foreign Data Wrapper supports use of the Hadoop file system using a HiveServer2 interface or Apache Spark using the Spark Thrift Server.

2 Architecture Overview

Hadoop is a framework that allows you to store a large data set in a distributed file system.

The Hadoop data wrapper provides an interface between a Hadoop file system and a

Postgres database. The Hadoop data wrapper transforms a Postgres SELECT statement into a query that is understood by the HiveQL or Spark SQL interface.



When possible, the Foreign Data Wrapper asks the Hive or Spark server to perform the actions associated with the WHERE clause of a SELECT statement. Pushing down the WHERE clause improves performance by decreasing the amount of data moving across the network.

3 Supported Authentication Methods

The Hadoop Foreign Data Wrapper supports NOSASL and LDAP authentication modes. To use NOSASL, do not specify any OPTIONS while creating user mapping. For LDAP authentication mode, specify username and password in OPTIONS while creating user mapping.

Using LDAP Authentication

When using the Hadoop Foreign Data Wrapper with LDAP authentication, you must first configure the Hive Server or Spark Server to use LDAP authentication. The configured server must provide a hive-site.xml file that includes the connection details for the LDAP server. For example:

```
<property>
 <name>hive.server2.authentication</name>
 <value>LDAP</value>
 <description>
  Expects one of [nosasl, none, ldap, kerberos, pam, custom].
  Client authentication types.
    NONE: no authentication check
    LDAP: LDAP/AD based authentication
    KERBEROS: Kerberos/GSSAPI authentication
    CUSTOM: Custom authentication provider
          (Use with property hive.server2.custom.authentication.class)
    PAM: Pluggable authentication module
    NOSASL:  Raw transport
 </description>
</property>
<property>
 <name>hive.server2.authentication.ldap.url</name>
 <value>ldap://localhost</value>
 <description>LDAP connection URL</description>
</property>
<property>
 <name>hive.server2.authentication.ldap.baseDN</name>
 <value>ou=People,dc=itzgeek,dc=local</value>
 <description>LDAP base DN</description>
</property>
```

Then, when starting the hive server, include the path to the `hive-site.xml` file in the command. For example:

```
./hive --config path_to_hive-site.xml_file --service hiveServer2
```

Where *path_to_hive-site.xml_file* specifies the complete path to the `hive-site.xml` file.

When creating the user mapping, you must provide the name of a registered LDAP user and the corresponding password as options. For details, see Create User Mapping.

Using NOSASL Authentication

When using NOSASL authentication with the Hadoop Foreign Data Wrapper, set the authorization to None, and the authentication method to NOSASL on the Hive Server or Spark Server. For example, if you start the Hive Server at the command line, include the hive.server2.authentication configuration parameter in the command:

```
hive --service hiveserver2 --hiveconf hive.server2.authentication=NOSASL
```

4 Installing the Hadoop Foreign Data Wrapper

The Hadoop Foreign Data Wrapper can be installed with an RPM package. During the installation process, the installer will satisfy software prerequisites.

Installing the Hadoop Foreign Data Wrapper using an RPM Package

The RPM installation package for the Hadoop Foreign Data Wrapper is available from the EDB repository. Before installing the forein data wrapper, you must install the epel-release package:

- On RHEL or CentOS 7:

  ```
  yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
  ```

- On RHEL or CentOS 8:

  ```
  dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
  ```

Please note that you may need to enable the [extras] repository definition in the CentOS-Base.repo file (located in /etc/yum.repos.d).

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

> https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges and invoke one of the following commands:

- On RHEL or CentOS 7:

```
yum -y install https://yum.enterprisedb.com/edb-repo-rpms/edb-repo-latest.noarch.rpm
```

- On RHEL or CentOS 8:

```
dnf -y install https://yum.enterprisedb.com/edb-repo-rpms/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EnterpriseDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:<password>@yum.enterprisedb.com/edb/redhat/rhel-$releasever-$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

After saving your changes to the configuration file, you can use the `yum install` command to install the foreign data wrapper on RHEL or CentOS 7:

- On RHEL or CentOS 7:

```
yum install edb-as<xx>-hdfs_fdw
```

- On RHEL or CentOS 8:

```
dnf install edb-as<xx>-hdfs_fdw
```

where xx is the server version number.

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter a `y`, and press `Return` to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

Installing the Hadoop Foreign Data Wrapper on a Debian or Ubuntu Host

To install the Hadoop Foreign Data Wrapper on a Debian or Ubuntu host, you must have credentials that allow access to the EDB repository. To request credentials for the repository, visit the EDB website.

The following steps will walk you through on using the EDB apt repository to install a DEB package. When using the commands, replace the `username` and `password` with the credentials provided by EDB.

1. Assume superuser privileges:

```
sudo su –
```

1. Configure the EnterpriseDB repository:

```
sh -c 'echo "deb
https://username:password@apt.enterprisedb.com/$(lsb_release
-cs)-edb/ $(lsb_release -cs) main" >
/etc/apt/sources.list.d/edb-$(lsb_release -cs).list'
```

1. Add support to your system for secure APT repositories:

```
apt-get install apt-transport-https
```

1. Add the EBD signing key:

```
wget -q -O - https://username:password
@apt.enterprisedb.com/edb-deb.gpg.key \| apt-key add -
```

1. Update the repository metadata:

```
apt-get update
```

1. Install DEB package:

```
apt-get install edb-as<xx>-hdfs_fdw
```

where xx is the server version number.

---

5 Configuring the Hadoop Foreign Data Wrapper

Before creating the extension and the database objects that use the extension, you must modify the Postgres host, providing the location of the supporting libraries.

1. After installing Postgres, modify the `postgresql.conf` located in:

   ```
   /var/lib/edb/as_version/data
   ```

2. Modify the configuration file with your editor of choice, adding the `hdfs_fdw.jvmpath` parameter to the end of the configuration file, and setting the value to specify the location of the Java virtual machine (`libjvm.so`). Set the value of `hdfs_fdw.classpath` to indicate the location of the java class files used by the adapter; use a colon (:) as a delimiter between each path. For example:

```
hdfs_fdw.classpath=
'/usr/edb/as12/lib/HiveJdbcClient-1.0.jar:
/home/edb/Projects/hadoop_fdw/hadoop/share/hadoop/common/hadoop-
common-2.6.4.jar:
/home/edb/Projects/hadoop_fdw/apache-hive-1.0.1-bin/lib/hive-jdbc-1.0.1-
standalone.jar'
```

Note

If you are using EDB Advanced Server and have a `DATE` column in your database, you must set `edb_redwood_date = OFF` in the `postgresql.conf` file.

1. After setting the parameter values, restart the Postgres server. For detailed information about controlling the service on an Advanced Server host, see the EDB Postgres Advanced Server Installation Guide, available at:

> https://www.enterprisedb.com/resources/product-documentation

Before using the Hadoop Foreign Data Wrapper, you must:

1. Use the CREATE EXTENSION command to create the extension on the Postgres host.
2. Use the CREATE SERVER command to define a connection to the Hadoop file system.
3. Use the CREATE USER MAPPING command to define a mapping that associates a Postgres role with the server.
4. Use the CREATE FOREIGN TABLE command to define a table in the Advanced Server database that corresponds to a database that resides on the Hadoop cluster.

CREATE EXTENSION

Use the `CREATE EXTENSION` command to create the `hdfs_fdw` extension. To invoke the command, use your client of choice (for example, psql) to connect to the Postgres database from which you will be querying the Hive or Spark server, and invoke the command:

```
CREATE EXTENSION [IF NOT EXISTS] hdfs_fdw [WITH] [SCHEMA
schema_name];
```

Parameters

`IF NOT EXISTS`

> Include the `IF NOT EXISTS` clause to instruct the server to issue a notice instead of throwing an error if an extension with the same name already exists.

`schema_name`

> Optionally specify the name of the schema in which to install the extension's objects.

Example

The following command installs the `hdfs_fdw` hadoop foreign data wrapper:

```
CREATE EXTENSION hdfs_fdw;
```

For more information about using the foreign data wrapper `CREATE EXTENSION` command, see:

> https://www.postgresql.org/docs/current/static/sql-createextension.html.

CREATE SERVER

Use the `CREATE SERVER` command to define a connection to a foreign server. The syntax is:

```
CREATE SERVER server_name FOREIGN DATA WRAPPER hdfs_fdw
  [OPTIONS (option 'value' [, ...])]
```

The role that defines the server is the owner of the server; use the `ALTER SERVER` command to reassign ownership of a foreign server. To create a foreign server, you must have `USAGE` privilege on the foreign-data wrapper specified in the `CREATE SERVER` command.

Parameters

`server_name`

> Use `server_name` to specify a name for the foreign server. The server name must be unique within the database.

`FOREIGN_DATA_WRAPPER`

> Include the `FOREIGN_DATA_WRAPPER` clause to specify that the server should use the `hdfs_fdw` foreign data wrapper when connecting to the cluster.

`OPTIONS`

> Use the `OPTIONS` clause of the `CREATE SERVER` command to specify connection information for the foreign server. You can include:

Option	Description
host	The address or hostname of the Hadoop cluster. The default value is localhost.
port	The port number of the Hive Thrift Server or Spark Thrift Server. The default is 10000.

Option	Description
client_type	Specify hiveserver2 or spark as the client type. To use the ANALYZE statement on Spark, you must specify a value of spark; if you do not specify a value for client_type, the default value is hiveserver2.
auth_type	The authentication type of the client; specify LDAP or NOSASL. If you do not specify an auth_type, the data wrapper will decide the auth_type value on the basis of the user mapping: • If the user mapping includes a user name and password, the data wrapper will use LDAP authentication. • If the user mapping does not include a user name and password, the data wrapper will use NOSASL authentication.
connect_timeout	The length of time before a connection attempt times out. The default value is 300 seconds.
fetch_size	A user-specified value that is provided as a parameter to the JDBC API setFetchSize. The default value is 10,000.
log_remote_sql	If true, logging will include SQL commands executed on the remote hive server and the number of times that a scan is repeated. The default is false.
query_timeout	Use query_timeout to provide the number of seconds after which a request will timeout if it is not satisfied by the Hive server. Query timeout is not supported by the Hive JDBC driver.
use_remote_estimate	Include the use_remote_estimate to instruct the server to use EXPLAIN commands on the remote server when estimating processing costs. By default, use_remote_estimate is false, and remote tables are assumed to have 1000 rows.

Example

The following command creates a foreign server named `hdfs_server` that uses the `hdfs_fdw` foreign data wrapper to connect to a host with an IP address of `170.11.2.148`:

```
CREATE SERVER hdfs_server FOREIGN DATA WRAPPER hdfs_fdw OPTIONS
```

```
(host '170.11.2.148', port '10000', client_type 'hiveserver2', auth_type 'LDAP',
connect_timeout '10000', query_timeout '10000');
```

The foreign server uses the default port (`10000`) for the connection to the client on the Hadoop cluster; the connection uses an LDAP server.

For more information about using the `CREATE SERVER` command, see:

> https://www.postgresql.org/docs/current/static/sql-createserver.html

CREATE USER MAPPING

Use the `CREATE USER MAPPING` command to define a mapping that associates a Postgres role with a foreign server:

```
CREATE USER MAPPING FOR role_name SERVER server_name
    [OPTIONS (option 'value' [, ...])];
```

You must be the owner of the foreign server to create a user mapping for that server.

Please note: the Hadoop Foreign Data Wrapper supports NOSASL and LDAP authentication. If you are creating a user mapping for a server that uses LDAP authentication, use the `OPTIONS` clause to provide the connection credentials (the username and password) for an existing LDAP user. If the server uses NOSASL authentication, omit the OPTIONS clause when creating the user mapping.

Parameters

`role_name`

> Use `role_name` to specify the role that will be associated with the foreign server.

`server_name`

> Use `server_name` to specify the name of the server that defines a connection to the Hadoop cluster.

`OPTIONS`

> Use the `OPTIONS` clause to specify connection information for the foreign server. If you are using LDAP authentication, provide a:

> `username`: the name of the user on the LDAP server.
>
> `password`: the password associated with the username.
>
> If you do not provide a user name and password, the data wrapper will use NOSASL authentication.

Example

The following command creates a user mapping for a role named `enterprisedb`; the mapping is associated with a server named `hdfs_server`:

```
CREATE USER MAPPING FOR enterprisedb SERVER hdfs_server;
```

If the database host uses LDAP authentication, provide connection credentials when creating the user mapping:

```
CREATE USER MAPPING FOR enterprisedb SERVER hdfs_server OPTIONS
(username 'alice', password '1safepwd');
```

The command creates a user mapping for a role named `enterprisedb` that is associated with a server named `hdfs_server`. When connecting to the LDAP server, the Hive or Spark server will authenticate as `alice`, and provide a password of `1safepwd`.

For detailed information about the `CREATE USER MAPPING` command, see:

> https://www.postgresql.org/docs/current/static/sql-createusermapping.html

CREATE FOREIGN TABLE

A foreign table is a pointer to a table that resides on the Hadoop host. Before creating a foreign table definition on the Postgres server, connect to the Hive or Spark server and create a table; the columns in the table will map to to columns in a table on the Postgres server. Then, use the `CREATE FOREIGN TABLE` command to define a table on the Postgres server with columns that correspond to the table that resides on the Hadoop host. The syntax is:

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE collation ]
[ column_constraint [ ... ] ]
    | table_constraint }
```

```
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
  SERVER server_name [ OPTIONS ( option 'value' [, ... ] ) ]
```

where `column_constraint` is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expr) [ NO INHERIT ] | DEFAULT default_expr }
```

and `table_constraint` is:

```
[ CONSTRAINT constraint_name ] CHECK (expr) [ NO INHERIT ]
```

Parameters

`table_name`

> Specifies the name of the foreign table; include a schema name to specify the schema in which the foreign table should reside.

`IF NOT EXISTS`

> Include the `IF NOT EXISTS` clause to instruct the server to not throw an error if a table with the same name already exists; if a table with the same name exists, the server will issue a notice.

`column_name`

> Specifies the name of a column in the new table; each column should correspond to a column described on the Hive or Spark server.

`data_type`

> Specifies the data type of the column; when possible, specify the same data type for each column on the Postgres server and the Hive or Spark server. If a data type with the same name is not available, the Postgres server will attempt to cast the data type to a type compatible with the Hive or Spark server. If the server cannot identify a compatible data type, it will return an error.

`COLLATE collation`

> Include the `COLLATE` clause to assign a collation to the column; if not specified, the column data type's default collation is used.

`INHERITS (parent_table [, ... ])`

> Include the `INHERITS` clause to specify a list of tables from which the new foreign table automatically inherits all columns. Parent tables can be plain tables or foreign tables.

`CONSTRAINT constraint_name`

> Specify an optional name for a column or table constraint; if not specified, the server will generate a constraint name.

`NOT NULL`

> Include the `NOT NULL` keywords to indicate that the column is not allowed to contain null values.

`NULL`

> Include the `NULL` keywords to indicate that the column is allowed to contain null values. This is the default.

`CHECK (expr) [NO INHERIT]`

> Use the `CHECK` clause to specify an expression that produces a Boolean result that each row in the table must satisfy. A check constraint specified as a column constraint should reference that column's value only, while an expression appearing in a table constraint can reference multiple columns.
>
> A `CHECK` expression cannot contain subqueries or refer to variables other than columns of the current row.
>
> Include the `NO INHERIT` keywords to specify that a constraint should not propagate to child tables.

`DEFAULT default_expr`

> Include the `DEFAULT` clause to specify a default data value for the column whose column definition it appears within. The data type of the default expression must match the data type of the column.

`SERVER server_name [OPTIONS (option 'value' [, ... ] ) ]`

> To create a foreign table that will allow you to query a table that resides on a Hadoop file system, include the `SERVER` clause and specify the `server_name` of the foreign server that uses the Hadoop data adapter.

> Use the OPTIONS clause to specify the following options and their corresponding values:

option	value
dbname	The name of the database on the Hive server; the database name is required.
table_name	The name of the table on the Hive server; the default is the name of the foreign table.

Example

To use data that is stored on a distributed file system, you must create a table on the Postgres host that maps the columns of a Hadoop table to the columns of a Postgres table. For example, for a Hadoop table with the following definition:

```
CREATE TABLE weblogs (
 client_ip           STRING,
 full_request_date   STRING,
 day                 STRING,
 month               STRING,
 month_num           INT,
 year                STRING,
 hour                STRING,
 minute              STRING,
 second              STRING,
 timezone            STRING,
 http_verb           STRING,
 uri                 STRING,
 http_status_code    STRING,
 bytes_returned      STRING,
 referrer            STRING,
 user_agent          STRING)
row format delimited
fields terminated by '\t';
```

You should execute a command on the Postgres server that creates a comparable table on the Postgres server:

```
CREATE FOREIGN TABLE weblogs
```

```
(
 client_ip              TEXT,
 full_request_date      TEXT,
 day                    TEXT,
 Month                  TEXT,
 month_num              INTEGER,
 year                   TEXT,
 hour                   TEXT,
 minute                 TEXT,
 second                 TEXT,
 timezone               TEXT,
 http_verb              TEXT,
 uri                    TEXT,
 http_status_code       TEXT,
 bytes_returned         TEXT,
 referrer               TEXT,
 user_agent             TEXT
)
SERVER hdfs_server
        OPTIONS (dbname 'webdata', table_name 'weblogs');
```

Include the `SERVER` clause to specify the name of the database stored on the Hadoop file system (`webdata`) and the name of the table (`weblogs`) that corresponds to the table on the Postgres server.

For more information about using the `CREATE FOREIGN TABLE` command, see:

> https://www.postgresql.org/docs/current/static/sql-createforeigntable.html

Data Type Mappings

When using the foreign data wrapper, you must create a table on the Postgres server that mirrors the table that resides on the Hive server. The Hadoop data wrapper will automatically convert the following Hive data types to the target Postgres type:

Hive	Postgres
BIGINT	BIGINT/INT8
BOOLEAN	BOOL/BOOLEAN

Hive	Postgres
BINARY	BYTEA
CHAR	CHAR
DATE	DATE
DOUBLE	FLOAT8
FLOAT	FLOAT/FLOAT4
INT/INTEGER	INT/INTEGER/INT4
SMALLINT	SMALLINT/INT2
STRING	TEXT
TIMESTAMP	TIMESTAMP
TINYINT	INT2
VARCHAR	VARCHAR

6 Using the Hadoop Foreign Data Wrapper

You can use the Hadoop Foreign Data Wrapper either through the Apache Hive or the Apache Spark. Both Hive and Spark store metadata in the configured metastore, where databases and tables are created using HiveQL.

Using HDFS FDW with Apache Hive on Top of Hadoop

Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time, this language allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

There are two versions of Hive - HiveServer1 and HiveServer2 which can be downloaded from the Apache Hive website.

Note

The Hadoop Foreign Data Wrapper supports only HiveServer2.

To use HDFS FDW with Apache Hive on top of Hadoop:

Step 1: Download weblogs_parse and follow instructions from Wiki Pentaho website.

Step 2: Upload weblog_parse.txt file using these commands:

```
hadoop fs -mkdir /weblogs
hadoop fs -mkdir /weblogs/parse
hadoop fs -put weblogs_parse.txt /weblogs/parse/part-00000
```

Step 3: Start HiveServer, if not already running, using following command:

```
$HIVE_HOME/bin/hiveserver2
```

or

```
$HIVE_HOME/bin/hive --service hiveserver2
```

Step 4: Connect to HiveServer2 using hive beeline client. For example:

```
$ beeline
Beeline version 1.0.1 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000/default;auth=noSasl
```

Step 5: Create Table in Hive.

```
CREATE TABLE weblogs (
    client_ip           STRING,
    full_request_date   STRING,
    day                 STRING,
    month               STRING,
    month_num           INT,
    year                STRING,
    hour                STRING,
    minute              STRING,
    second              STRING,
    timezone            STRING,
    http_verb           STRING,
    uri                 STRING,
```

```
    http_status_code    STRING,
    bytes_returned      STRING,
    referrer            STRING,
    user_agent          STRING)
row format delimited
fields terminated by '\t';
```

Step 6: Load data in weblogs table.

```
hadoop fs -cp /weblogs/parse/part-00000 /user/hive/warehouse/weblogs/
```

Step 7: Access data from PostgreSQL. You can now use the the weblog table in PostgreSQL. Once you are connected using psql, follow the below steps:

```
-- set the GUC variables appropriately, e.g. :
hdfs_fdw.jvmpath='/home/edb/Projects/hadoop_fdw/jdk1.8.0_111/jre/lib/amd64/server/
'
hdfs_fdw.classpath='/usr/local/edbas/lib/postgresql/HiveJdbcClient-1.0.jar:

/home/edb/Projects/hadoop_fdw/hadoop/share/hadoop/common/hadoop-common-
2.6.4.jar:
                /home/edb/Projects/hadoop_fdw/apache-hive-1.0.1-bin/lib/hive-jdbc-
1.0.1-standalone.jar'

-- load extension first time after install
CREATE EXTENSION hdfs_fdw;

-- create server object
CREATE SERVER hdfs_server
        FOREIGN DATA WRAPPER hdfs_fdw
        OPTIONS (host '127.0.0.1');

-- create user mapping
CREATE USER MAPPING FOR postgres
    SERVER hdfs_server OPTIONS (username 'hive_username', password
'hive_password');

-- create foreign table
CREATE FOREIGN TABLE weblogs
```

```
(
 client_ip              TEXT,
 full_request_date      TEXT,
 day                    TEXT,
 Month                  TEXT,
 month_num              INTEGER,
 year                   TEXT,
 hour                   TEXT,
 minute                 TEXT,
 second                 TEXT,
 timezone               TEXT,
 http_verb              TEXT,
 uri                    TEXT,
 http_status_code       TEXT,
 bytes_returned         TEXT,
 referrer               TEXT,
 user_agent             TEXT
)
SERVER hdfs_server
        OPTIONS (dbname 'default', table_name 'weblogs');


-- select from table
postgres=# SELECT DISTINCT client_ip IP, count(*)
          FROM weblogs GROUP BY IP HAVING count(*) > 5000 ORDER BY 1;
       ip        | count
-----------------+-------
 13.53.52.13     |  5494
 14.323.74.653   | 16194
 322.6.648.325   | 13242
 325.87.75.336   |  6500
 325.87.75.36    |  6498
 361.631.17.30   | 64979
 363.652.18.65   | 10561
 683.615.622.618 | 13505
(8 rows)

-- EXPLAIN output showing WHERE clause being pushed down to remote server.
EXPLAIN (VERBOSE, COSTS OFF) SELECT client_ip, full_request_date, uri FROM
```

```
weblogs WHERE http_status_code = 200;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Foreign Scan on public.weblogs
   Output: client_ip, full_request_date, uri
   Remote SQL: SELECT client_ip, full_request_date, uri FROM default.weblogs
WHERE ((http_status_code = '200'))
(3 rows)
```

Using HDFS FDW with Apache Spark on Top of Hadoop

Apache Spark is a general purpose distributed computing framework which supports a wide variety of use cases. It provides real time stream as well as batch processing with speed, ease of use and sophisticated analytics. Spark does not provide storage layer as it relies on third party storage providers like Hadoop, HBASE, Cassandra, S3 etc. Spark integrates seamlessly with Hadoop and can process existing data. Spark SQL is 100% compatible with HiveQL and can be used as a replacement of Hiveserver2, using Spark Thrift Server.

To use HDFS FDW with Apache Spark on top of Hadoop:

Step 1: Download and install the Apache Spark in local mode.

Step 2: In the folder $SPARK_HOME/conf create a file spark-defaults.conf containing the following line:

```
spark.sql.warehouse.dir hdfs://localhost:9000/user/hive/warehouse
```

By default spark uses derby for both meta data and the data itself (called warehouse in spark). In order to have spark use hadoop as warehouse, you should add this property.

Step 3: Start Spark Thrift Server.

```
./start-thriftserver.sh
```

Step 4: Make sure Spark thrift server is running using log file.

Step 5: Create a local file names.txt with below data:

```
$ cat /tmp/names.txt
```

```
1,abcd
2,pqrs
3,wxyz
4,a_b_c
5,p_q_r
,
```

Step 6: Connect to Spark Thrift Server2 using spark beeline client. For example:

```
 $ beeline
Beeline version 1.2.1.spark2 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000/default;auth=noSasl
org.apache.hive.jdbc.HiveDriver
```

Step 7: Get the sample data ready on spark. Run the following commands in beeline command line tool:

```
 ./beeline
Beeline version 1.2.1.spark2 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000/default;auth=noSasl
org.apache.hive.jdbc.HiveDriver
Connecting to jdbc:hive2://localhost:10000/default;auth=noSasl
Enter password for jdbc:hive2://localhost:10000/default;auth=noSasl:
Connected to: Spark SQL (version 2.1.1)
Driver: Hive JDBC (version 1.2.1.spark2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000> create database my_test_db;
+---------+--+
| Result  |
+---------+--+
+---------+--+
No rows selected (0.379 seconds)
0: jdbc:hive2://localhost:10000> use my_test_db;
+---------+--+
| Result  |
+---------+--+
+---------+--+
No rows selected (0.03 seconds)
0: jdbc:hive2://localhost:10000> create table my_names_tab(a int, name string)
```

```
                        row format delimited fields terminated by ' ';
+---------+--+
| Result  |
+---------+--+
+---------+--+
No rows selected (0.11 seconds)
0: jdbc:hive2://localhost:10000>

0: jdbc:hive2://localhost:10000> load data local inpath '/tmp/names.txt'
                        into table my_names_tab;
+---------+--+
| Result  |
+---------+--+
+---------+--+
No rows selected (0.33 seconds)
0: jdbc:hive2://localhost:10000> select * from my_names_tab;
+-------+---------+--+
|   a   |  name   |
+-------+---------+--+
| 1     | abcd    |
| 2     | pqrs    |
| 3     | wxyz    |
| 4     | a_b_c   |
| 5     | p_q_r   |
| NULL  | NULL    |
+-------+---------+--+
```

Here are the corresponding files in Hadoop:

```
 $ hadoop fs -ls /user/hive/warehouse/
Found 1 items
drwxrwxrwx   - org.apache.hive.jdbc.HiveDriver supergroup 0 2020-06-12 17:03
/user/hive/warehouse/my_test_db.db

$ hadoop fs -ls /user/hive/warehouse/my_test_db.db/
Found 1 items
drwxrwxrwx   - org.apache.hive.jdbc.HiveDriver supergroup 0 2020-06-12 17:03
/user/hive/warehouse/my_test_db.db/my_names_tab
```

Step 8: Access data from PostgreSQL. Connect to Postgres using psql:

```
-- set the GUC variables appropriately, e.g. :
hdfs_fdw.jvmpath='/home/edb/Projects/hadoop_fdw/jdk1.8.0_111/jre/lib/amd64/server/
'
hdfs_fdw.classpath='/usr/local/edbas/lib/postgresql/HiveJdbcClient-1.0.jar:

/home/edb/Projects/hadoop_fdw/hadoop/share/hadoop/common/hadoop-common-
2.6.4.jar:
                /home/edb/Projects/hadoop_fdw/apache-hive-1.0.1-bin/lib/hive-jdbc-
1.0.1-standalone.jar'

-- load extension first time after install
CREATE EXTENSION hdfs_fdw;

-- create server object
CREATE SERVER hdfs_server
  FOREIGN DATA WRAPPER hdfs_fdw
  OPTIONS (host '127.0.0.1', port '10000', client_type 'spark', auth_type 'NOSASL');

-- create user mapping
CREATE USER MAPPING FOR postgres
  SERVER hdfs_server OPTIONS (username 'spark_username', password
'spark_password');

-- create foreign table
CREATE FOREIGN TABLE f_names_tab( a int, name varchar(255)) SERVER
hdfs_svr
  OPTIONS (dbname 'testdb', table_name 'my_names_tab');

-- select the data from foreign server
select * from f_names_tab;
 a |  name
---+--------
 1 | abcd
 2 | pqrs
 3 | wxyz
 4 | a_b_c
 5 | p_q_r
```

```
 0 |
(6 rows)

-- EXPLAIN output showing WHERE clause being pushed down to remote server.
EXPLAIN (verbose, costs off) SELECT name FROM f_names_tab WHERE a > 3;
                           QUERY PLAN
-----------------------------------------------------------------------
 Foreign Scan on public.f_names_tab
   Output: name
   Remote SQL: SELECT name FROM my_test_db.my_names_tab WHERE ((a >
'3'))
(3 rows)
```

Note

The same port was being used while creating foreign server because Spark Thrift Server is compatible with Hive Thrift Server. Applications using Hiveserver2 would work with Spark except for the behaviour of `ANALYZE` command and the connection string in case of `NOSASL`. It is suggested to use `ALTER SERVER` and change the `client_type` option if Hive is to be replaced with Spark.

---

7 Identifying the Hadoop Foreign Data Wrapper Version

The Hadoop Foreign Data Wrapper includes a function that you can use to identify the currently installed version of the `.so` file for the data wrapper. To use the function, connect to the Postgres server, and enter:

```
SELECT hdfs_fdw_version();
```

The function returns the version number:

```
 hdfs_fdw_version
-----------------
20005
```