Hadoop Foreign Data Wrapper

Version 2

Built at 2024-04-25T23:25:02

1 Hadoop Foreign Data Wrapper

The Hadoop Foreign Data Wrapper (`hdfs_fdw`) is a Postgres extension that allows you to access data that resides on a Hadoop file system from EDB Postgres Advanced Server. The foreign data wrapper makes the Hadoop file system a read-only data source that you can use with Postgres functions and utilities or with other data that resides on a Postgres host.

2 Release Notes

The Hadoop Foreign Data Wrapper documentation describes the latest version including minor releases and patches. The release notes in this section provide information on what was new in each release. For new functionality introduced in a minor or patch release, there are also indicators within the content about what release introduced the feature.

Version	Release Date
2.3.1	20 Jul 2023
2.3.0	06 Jan 2023
2.2.0	26 May 2022
2.1.0	02 Dec 2021
2.0.8	24 Jun 2021
2.0.7	23 Nov 2020
2.0.5	10 Dec 2019
2.0.4	28 Nov 2018

2.1 Hadoop Foreign Data Wrapper 2.3.1 release notes

Released: 20 Jul 2023

Enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.3.1 include:

Type	Description
Enhancement	Added support for PostgreSQL 16.
Feature	Added `enable_join_pushdown` and `enable_aggregate_pushdown` GUCs to support JOIN and AGGREGATE pushdowns, respectively.
Feature	Added table and server level option to `enable_order_by_pushdown` and `enable_aggregate_pushdown` GUCs.

2.2 Hadoop Foreign Data Wrapper 2.3.0 release notes

Released: 06 Jan 2023

Enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.3 include:

Type	Description
Feature	When possible, we push the `ORDER BY` clause to the remote server. This approach provides the ordered result set from the foreign server, which can help to enable an efficient merge join.
Feature	Push down `LIMIT` / `OFFSET` to remote HDFS servers: Where possible, perform `LIMIT` and `OFFSET` operations on the remote server. This reduces network traffic between local Postgres and remote HDFS/Hive servers.

2.3 Hadoop Foreign Data Wrapper 2.2.0 release notes

Released: 26 May 2022

Enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.2 include:

Type	Description
Feature	Push down aggregates to remote HDFS servers: Push aggregates to the remote HDFS server instead of fetching all of the rows and aggregating them locally. This gives a performance boost for cases where aggregates can be pushed down.
Enhancement	Add support for NUMERIC data type.
Bug fix	Do not push expressions evaluating as non-builtin data types.

2.4 Hadoop Foreign Data Wrapper 2.1.0 release notes

Released: 02 Dec 2021

Enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.1.0 include:

Type	Description
Enhancement	Support for EDB Postgres Advanced Server 14.
Enhancement	Join Pushdown: If a query has a join between two foreign tables from the same remote server, you can now push that join down to the remote server instead of fetching all the rows for both the tables and performing a join locally.

2.5 Hadoop Foreign Data Wrapper 2.0.8 release notes

Released: 24 Jun 2021

New features, enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.0.8 include the following:

Type	Description
Enhancement	Support for Hadoop version 3.2.x
Enhancement	Support for Hive version 3.1.x
Enhancement	Support for Spark version 3.0.x
Bug Fix	Fixed building a SELECT query having a whole-row reference to avoid an error.
Bug Fix	Fixed crash with the queries involving LEFT JOIN LATERAL.
Bug Fix	Use proper hive-SQL quoting of table or column names containing special characters.

2.6 Hadoop Foreign Data Wrapper 2.0.7 release notes

Released: 23 Nov 2020

New features, enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.0.7 include the following:

Type	Description
Enhancement	Support for EDB Postgres Advanced Server 13.
Enhancement	Support for Ubuntu 20.04 LTS platform.
Enhancement	Updated LICENSE file.

2.7 Hadoop Foreign Data Wrapper 2.0.5 release notes

Released: 10 Dec 2019

Enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.0.5 include:

Type	Description
Enhancement	Support for EDB Postgres Advanced Server 12.

2.8 Hadoop Foreign Data Wrapper 2.0.4 release notes

Released: 28 Nov 2018

Enhancements, bug fixes, and other changes in Hadoop Foreign Data Wrapper 2.0.4 include:

Type	Description
Enhancement	Support for EDB Postgres Advanced Server 11.

3 Supported platforms

The Hadoop Foreign Data Wrapper is supported on the same Linux platforms as EDB Postgres Advanced Server. To determine the platform support for the Hadoop Foreign Data Wrapper, see thePlatform Compatibility page on the EDB website or Installing Hadoop Foreign Data Wrapper.

Supported database versions

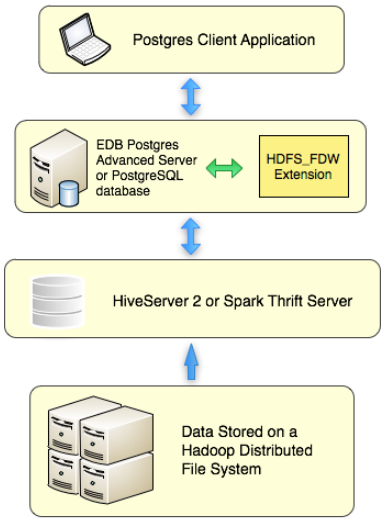
This table lists the latest Hadoop Foreign Data Wrapper versions and their supported corresponding EDB Postgres Advanced Server (EPAS) versions.

Hadoop Foreign Data Wrapper	EPAS 16	EPAS 15	EPAS 14	EPAS 13	EPAS 12
2.3.1	Y	Y	Y	Y	Y
2.3.0	N	Y	Y	Y	Y
2.2.0	N	Y	Y	Y	Y
2.1.0	N	N	Y	Y	Y
2.0.9	N	N	Y	Y	Y
2.0.8	N	N	N	Y	Y
2.0.7	N	N	N	Y	Y
2.0.6	N	N	N	Y	Y

4 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.

5 Key features

These are the key features of the Hadoop Foreign Data Wrapper.

WHERE clause pushdown

Hadoop Foreign Data Wrappper allows the pushdown of `WHERE` clauses to the foreign server for execution. This feature optimizes remote queries to reduce the number of rows transferred from foreign servers.

Column pushdown

Hadoop Foreign Data Wrapper supports column pushdown. As a result, the query brings back only those columns that are a part of the select target list.

Join pushdown

Hadoop Foreign Data Wrapper supports join pushdown. It pushes the joins between the foreign tables of the same remote Hive or Spark server to the remote Hive or Spark server, enhancing the performance.

From version 2.3.0 and later, you can enable the join pushdown at session and query level using the `enable_join_pushdown` GUC variable.

For more information, see Example: Join pushdown.

Aggregate pushdown

Hadoop Foreign Data Wrapper supports aggregate pushdown. It pushes the aggregates to the remote Hive or Spark server instead of fetching all of the rows and aggregating them locally. This gives a very good performance boost for the cases where aggregates can be pushed down. The pushdown is currently limited to aggregate functions min, max, sum, avg, and count, to avoid pushing down the functions that are not present on the Hadoop server. Also, aggregate filters and orders are not pushed down.

For more information, see Example: Aggregate pushdown.

ORDER BY pushdown

Hadoop Foreign Data Wrapper supports order by pushdown. If possible, push the `ORDER BY` clause to the remote server. This approach provides the ordered result set from the foreign server, which can help to enable an efficient merge join.

For more information, see Example: ORDER BY pushdown

LIMIT OFFSET pushdown

Hadoop Foreign Data Wrapper supports limit offset pushdown. Wherever possible, perform LIMIT and OFFSET operations on the remote server. This reduces network traffic between local Postgres and remote HDFS/Hive servers. ALL/NULL options aren't supported on the Hive server, which means they are not pushed down. Also, OFFSET without LIMIT isn't supported on the remote server. Queries having that construct are not pushed.

For more information, see Example: LIMIT OFFSET pushdown

Automated cleanup

Hadoop Foreign Data Wrappper allows the cleanup of foreign tables in a single operation using the `DROP EXTENSION` command. This feature is specifically useful when a foreign table is set for a temporary purpose. The syntax is:

```
DROP EXTENSION hdfs_fdw CASCADE;
```

For more information, see DROP EXTENSION.

6 Supported authentication methods

The Hadoop Foreign Data Wrapper supports NOSASL and LDAP authentication modes. To use NOSASL, don't specify any `OPTIONS` values 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, 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
```

<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, seeCreate 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
```

7 Installing Hadoop Foreign Data Wrapper on Linux

Select a link to access the applicable installation instructions:

Linux x86-64 (amd64)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8, RHEL 7
- Oracle Linux (OL) 9, Oracle Linux (OL) 8, Oracle Linux (OL) 7
- Rocky Linux 9, Rocky Linux 8
- AlmaLinux 9, AlmaLinux 8
- CentOS 7

SUSE Linux Enterprise (SLES)

- SLES 15, SLES 12

Debian and derivatives

- Ubuntu 22.04, Ubuntu 20.04
- Debian 11, Debian 10

Linux IBM Power (ppc64le)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15, SLES 12

7.1 Installing Hadoop Foreign Data Wrapper on Linux x86 (amd64)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9
- RHEL 8
- RHEL 7
- Oracle Linux (OL) 9
- Oracle Linux (OL) 8
- Oracle Linux (OL) 7
- Rocky Linux 9
- Rocky Linux 8
- AlmaLinux 9
- AlmaLinux 8
- CentOS 7

SUSE Linux Enterprise (SLES)

- SLES 15
- SLES 12

Debian and derivatives

- Ubuntu 22.04
- Ubuntu 20.04
- Debian 11
- Debian 10

After you complete the installation, see Initial configuration.

7.1.1 Installing Hadoop Foreign Data Wrapper on RHEL 9 or OL 9 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
```

- Enable additional repositories to resolve dependencies:

```
sudo dnf config-manager --set-enabled PowerTools
```

Install the package

```
sudo dnf -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.2 Installing Hadoop Foreign Data Wrapper on RHEL 8 or OL 8 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

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

- Enable additional repositories to resolve dependencies:

```
sudo dnf config-manager --set-enabled PowerTools
```

Install the package

```
sudo dnf -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.3 Installing Hadoop Foreign Data Wrapper on AlmaLinux 9 or Rocky Linux 9 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install epel-release
  ```
- Enable additional repositories to resolve dependencies:

  ```
  sudo dnf config-manager --set-enabled crb
  ```

Install the package

```
sudo dnf -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.4 Installing Hadoop Foreign Data Wrapper on AlmaLinux 8 or Rocky Linux 8 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter this command:

```
dnf repolist | grep enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo dnf -y install epel-release
```

- Enable additional repositories to resolve dependencies:

```
sudo dnf config-manager --set-enabled powertools
```

Install the package

```
sudo dnf -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.5 Installing Hadoop Foreign Data Wrapper on RHEL 7 or OL 7 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

- Enable additional repositories to resolve dependencies:

```
subscription-manager repos --enable "rhel-*-optional-rpms" --enable "rhel-*-extras-rpms"  --enable "rhel-ha-for-rhel-*-server-rpms"
```

Install the package

```
sudo yum -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.6 Installing Hadoop Foreign Data Wrapper on CentOS 7 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
  ```

Install the package

```
sudo yum -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.7 Installing Hadoop Foreign Data Wrapper on SLES 15 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

```
zypper lr -E | grep enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

```
sudo SUSEConnect -p PackageHub/15.4/x86_64
```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

```
sudo zypper -n install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.8 Installing Hadoop Foreign Data Wrapper on SLES 12 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

```
sudo SUSEConnect -p PackageHub/12.5/x86_64
sudo SUSEConnect -p sle-sdk/12.5/x86_64
```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

```
sudo zypper -n install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.9 Installing Hadoop Foreign Data Wrapper on Ubuntu 22.04 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.10 Installing Hadoop Foreign Data Wrapper on Ubuntu 20.04 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.11 Installing Hadoop Foreign Data Wrapper on Debian 11 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.1.12 Installing Hadoop Foreign Data Wrapper on Debian 10 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server

- Installing PostgreSQL

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

```
sudo apt-get -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.2 Installing Hadoop Foreign Data Wrapper on Linux IBM Power (ppc64le)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL)

- RHEL 9
- RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15
- SLES 12

After you complete the installation, see Initial configuration.

7.2.1 Installing Hadoop Foreign Data Wrapper on RHEL 9 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
```

- Refresh the cache:

```
sudo dnf makecache
```

Install the package

```
sudo dnf -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.2.2 Installing Hadoop Foreign Data Wrapper on RHEL 8 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

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

- Refresh the cache:

```
sudo dnf makecache
```

Install the package

```
sudo dnf -y install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.2.3 Installing Hadoop Foreign Data Wrapper on SLES 15 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/15.4/ppc64le
  ```
- Refresh the metadata:

  ```
  sudo zypper refresh
  ```

Install the package

```
sudo zypper -n install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

7.2.4 Installing Hadoop Foreign Data Wrapper on SLES 12 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter this command:

```
zypper lr -E | grep enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

```
sudo SUSEConnect -p PackageHub/12.5/ppc64le
sudo SUSEConnect -p sle-sdk/12.5/ppc64le
```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

```
sudo zypper -n install edb-as15-hdfs_fdw
```

Where `15` is the version of EDB Postgres Advanced Server. Replace `15` with the version of EDB Postgres Advanced Server you are using.

8 Initial configuration

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.

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

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

Modify the configuration file, 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**
>
> Copy the jar files (`hive-jdbc-1.0.1-standalone.jar` and `hadoop-common-2.6.4.jar`) from the respective Hive and Hadoop sources or website to the PostgreSQL instance where Hadoop Foreign Data Wrapper is installed.
>
> If you're using EDB Postgres Advanced Server and have a `DATE` column in your database, you must set `edb_redwood_date = OFF` in the `postgresql.conf` file.

After setting the parameter values, restart the Postgres server. For detailed information about controlling the service on an EDB Postgres Advanced Server host, see the EDB Postgres Advanced Server documentation.

Before using the Hadoop Foreign Data Wrapper:

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 EDB Postgres 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 returning 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 the PostgreSQL documentation.

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 in the database.

`FOREIGN_DATA_WRAPPER`

Include the `FOREIGN_DATA_WRAPPER` clause to specify for the server to 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`.
client_type	Specify `hiveserver2` or `spark` as the client type. To use the `ANALYZE` statement on Spark, you must specify a value of `spark`. The default value is `hiveserver2`.
auth_type	The authentication type of the client. Specify `LDAP` or `NOSASL`. If you don't specify an `auth_type`, the data wrapper decides the `auth_type` value on the basis of the user mapping. If the user mapping includes a user name and password, the data wrapper uses LDAP authentication. If the user mapping doesn't include a user name and password, the data wrapper uses NOSASL authentication.
connect_timeout	The length of time before a connection attempt times out. The default value is 300 seconds
enable_aggregate_pushdown	Similar to the table-level option but configured at the server level. If `true`, pushes the aggregate operations to the foreign server instead of fetching rows from the foreign server and performing the operations locally. You can also set this option for an individual table. The table-level value of the option takes precedence over the server-level option value. Default is `true`.
enable_join_pushdown	Similar to the table-level option but configured at the server level. If `true`, pushes the join between two foreign tables from the same foreign server instead of fetching all the rows for both the tables and performing a join locally. You can also set this option for an individual table and, if any of the tables involved in the join has set the option to `false`, then the join isn't pushed down. The table-level value of the option takes precedence over the server-level option value. Default is `true`.
enable_order_by_pushdown	Similar to the table-level option but configured at the server level. If `true`, pushes the order-by operation to the foreign server instead of fetching rows from the foreign server and performing the sort locally. You can also set this option for an individual table. The table-level value of the option takes precedence over the server-level option value. Default is `true`.
fetch_size	Provided as a parameter to the JDBC API `setFetchSize`. The default value is `10,000`.
log_remote_sql	If `true`, logging includes 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 times out if it isn't satisfied by the Hive server. Query timeout is not supported by the Hive JDBC driver.
use_remote_estimate	Include `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 the PostgreSQL documentation.

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.

> **Note**
>
> The Hadoop Foreign Data Wrapper supports NOSASL and LDAP authentication. If you're creating a user mapping for a server that uses LDAP authentication, use the `OPTIONS` clause to provide the connection credentials (the user name 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 to associate 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're using LDAP authentication, provide:

`username` — The name of the user on the LDAP server.

`password` — the password associated with the username.

If you don't provide a user name and password, the data wrapper uses 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 authenticates as `alice`, and provides a password of `1safepwd`.

For detailed information about the `CREATE USER MAPPING` command, see the PostgreSQL documentation.

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 map 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' [, ... ] )
]
```

`column_constraint` is:

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

`table_constraint` is:

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

Parameters

`table_name`

Specify the name of the foreign table. Include a schema name to specify the schema in which the foreign table resides.

`IF NOT EXISTS`

Include the `IF NOT EXISTS` clause to instruct the server not to return an error if a table with the same name already exists. If a table with the same name exists, the server issues a notice.

`column_name`

Specifies the name of a column in the new table. Each column corresponds to a column described on the Hive or Spark server.

`data_type`

Specify 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 isn't available, the Postgres server attempts to cast the data type to a type compatible with the Hive or Spark server. If the server can't identify a compatible data type, it returns an error.

`COLLATE collation`

Include the `COLLATE` clause to assign a collation to the column. The column data type's default collation is used by default.

`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 generates a constraint name.

`NOT NULL`

Include the `NOT NULL` keywords to indicate that the column isn't 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 can reference that column's value only, while an expression appearing in a table constraint can reference multiple columns.

A `CHECK` expression can't contain subqueries or refer to variables other than columns of the current row.

Include the `NO INHERIT` keywords to specify that a constraint can't 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 in. 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 allows you to query a table that resides on a Hadoop file system, include the `SERVER` clause and specify the `server_name` value 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.
enable_aggregate_pushdown	Similar to the server-level option but configured at the table level. If `true`, pushes the aggregate operations to the foreign server instead of fetching rows from the foreign server and performing the operations locally. You can also set this option for an individual table. The table-level value of the option takes precedence over the server-level option value. Default is `true`.
enable_join_pushdown	Similar to the server-level option but configured at the table level. If `true`, pushes the join between two foreign tables from the same foreign server instead of fetching all the rows for both the tables and performing a join locally. You can also set this option for an individual table. If any of the tables involved in the join has set the option to `false`, then the join isn't pushed down. The table-level value of the option takes precedence over the server-level option value. Default is `true`.
enable_order_by_pushdown	Similar to the server-level option but configured at the table level. If `true`, pushes the order-by operation to the foreign server instead of fetching rows from the foreign server and performing the sort locally. You can also set this option for an individual table. The table-level value of the option takes precedence over the server-level option value. Default is `true`.

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';
```

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 the PostgreSQL documentation.

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 automatically converts the following Hive data types to the target Postgres type:

Hive	Postgres
BIGINT	BIGINT/INT8
BOOLEAN	BOOL/BOOLEAN
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

DROP EXTENSION

Use the `DROP EXTENSION` command to remove an extension. To invoke the command, use your client of choice (for example, psql) to connect to the Postgres database from which you're dropping the Hadoop server, and run the command:

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT
];
```

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of returning an error if an extension with the specified name doesn't exist.

`name`

Optionally, specify the name of the installed extension.

`CASCADE`

Automatically drop objects that depend on the extension. It drops all the other dependent objects too.

`RESTRICT`

Don't allow to drop extension if any objects, other than its member objects and extensions listed in the same `DROP` command, depend on it.

Example

The following command removes the extension from the existing database:

```
DROP EXTENSION hdfs_fdw;
```

For more information about using the foreign data wrapper `DROP EXTENSION` command, see the PostgreSQL documentation.

DROP SERVER

Use the `DROP SERVER` command to remove a connection to a foreign server. The syntax is:

```
DROP SERVER [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT
]
```

The role that drops the server is the owner of the server. Use the `ALTER SERVER` command to reassign ownership of a foreign server. To drop a foreign server, you must have `USAGE` privilege on the foreign-data wrapper specified in the `DROP SERVER` command.

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of returning an error if a server with the specified name doesn't exist.

`name`

Optionally, specify the name of the installed server.

`CASCADE`

Automatically drop objects that depend on the server. It drops all the other dependent objects too.

`RESTRICT`

Don't allow to drop the server if any objects depend on it.

Example

The following command removes a foreign server named `hdfs_server`:

```
DROP SERVER hdfs_server;
```

For more information about using the `DROP SERVER` command, see the PostgreSQL documentation.

DROP USER MAPPING

Use the `DROP USER MAPPING` command to remove a mapping that associates a Postgres role with a foreign server. You must be the owner of the foreign server to remove a user mapping for that server.

```
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_USER | PUBLIC } SERVER
server_name;
```

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of returning an error if the user mapping doesn't exist.

`user_name`

Specify the user name of the mapping.

`server_name`

Specify the name of the server that defines a connection to the Hadoop cluster.

Example

The following command drops a user mapping for a role named `enterprisedb`. The mapping is associated with a server named `hdfs_server`:

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

For detailed information about the `DROP USER MAPPING` command, see the PostgreSQL documentation.

DROP FOREIGN TABLE

A foreign table is a pointer to a table that resides on the Hadoop host. Use the `DROP FOREIGN TABLE` command to remove a foreign table. Only the owner of the foreign table can drop it.

```
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT
]
```

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of returning an error if the foreign table with the specified name doesn't exist.

`name`

Specify the name of the foreign table.

`CASCADE`

Automatically drop objects that depend on the foreign table. It drops all the other dependent objects too.

`RESTRICT`

Don't allow to drop foreign table if any objects depend on it.

Example

```
DROP FOREIGN TABLE warehouse;
```

For more information about using the `DROP FOREIGN TABLE` command, see the PostgreSQL documentation.

9 Upgrading

If you have an existing installation of Hadoop Foreign Data Wrapper that you installed using the EDB repository, you can update your repository configuration file and then upgrade Hadoop to a more recent product version.

To perform the process, open a terminal window and enter the commands that apply to the operating system and package manager used for the installation:

To update your repository configuration file:

```
sudo <package-manager>
```

Where `<package-manager>` is the package manager used with your operating system:

Package manager	Operating system
dnf	RHEL 8/9 and derivatives
yum	RHEL 7 and derivatives, CentOS 7
zypper	SLES
apt-get	Debian and Ubuntu

To upgrade to the latest product version, enter one of the following commands:

Operating system	Upgrade command
RHEL 8/9 and derivatives	`sudo dnf upgrade edb-as<xx>-hdfs_fdw`
RHEL 7 and derivatives, CentOS 7	`sudo yum upgrade edb-as<xx>-hdfs_fdw edb-hdfs-libs`
SLES	`sudo zypper upgrade edb-as<xx>-hdfs_fdw`
Debian and Ubuntu	`sudo apt-get --only-upgrade install edb-as<xx>-hdfs-fdw edb-hdfs`

Where

- `<package-manager>` is the package manager used with your operating system.
- `<xx>` is the EDB Postgres Advanced Server version number.

10 Uninstalling

You use the `remove` command to uninstall Hadoop Foreign Data Wrapper packages. To uninstall, open a terminal window, assume superuser privileges, and enter the command that applies to the operating system and package manager used for the installation:

- On RHEL or CentOS 7:

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

- On RHEL or Rocky Linux or AlmaLinux 8:

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

- On SLES:

```
zypper remove edb-as<xx>-hdfs_fdw
```

- On Debian or Ubuntu

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

11 Using Apache Hive or Apache Spark

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 helps with 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's inconvenient or inefficient to express this logic in HiveQL.

You can download the two versions of Hive—HiveServer1 and HiveServer2—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:

1. Download weblogs_parse and follow the instructions at the Wiki Pentaho website.
2. Upload the `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
```

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

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

or

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

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

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

5. Create a table in Hive. The example creates a table named `weblogs`:

```
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';
```

6. Load data into the table.

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

7. Access your data from Postgres. You can now use the `weblog` table. Once you're connected using psql, follow these 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 that supports a wide variety of use cases. It provides real-time streaming as well as batch processing with speed, ease-of-use, and sophisticated analytics. Spark doesn't provide a storage layer, as it relies on third-party storage providers like Hadoop, HBASE, Cassandra, S3, and so on. Spark integrates seamlessly with Hadoop and can process existing data. Spark SQL is 100% compatible with HiveQL. You can use it to replace Hiveserver2, using Spark Thrift Server.

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

1. Download and install Apache Spark in local mode.
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 the meta data and the data itself (called a warehouse in Spark). To have Spark use Hadoop as a warehouse, add this property.
3. Start Spark Thrift Server.

   ```
   ./start-thriftserver.sh
   ```
4. Make sure Spark Thrift Server is running and writing to a log file.
5. Create a local file (`names.txt`) that contains the following entries:

   ```
   $ cat /tmp/names.txt
   1,abcd
   2,pqrs
   3,wxyz
   4,a_b_c
   5,p_q_r
   ,
   ```
6. Connect to Spark Thrift Server2 using the 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
   ```
7. Prepare the sample data on Spark. Run the following commands in the 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    |
+-------+---------+--+
```

The following commands list 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
```

8. Access your data from 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

> This example uses the same port while creating the foreign server because Spark Thrift Server is compatible with Hive Thrift Server. Applications using Hiveserver2 work with Spark except for the behavior of the `ANALYZE` command and the connection string in the case of `NOSASL`. We recommend using `ALTER SERVER` and changing the `client_type` option if you replace Hive with Spark.

12 Example: Join pushdown

This example shows join pushdown between the foreign tables of the same remote HIVE/SPARK server to that remote HIVE/SPARK server:

Tables on HIVE/SPARK server:

```
0: jdbc:hive2://localhost:10000> describe
emp;
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| empno     | int        | NULL
|
| ename     | string     | NULL
|
| job       | string     | NULL
|
| mgr       | int        | NULL
|
| hiredate  | date       | NULL
|
| sal       | int        | NULL
|
| comm      | int        | NULL
|
| deptno    | int        | NULL
|
+-----------+------------+----------+--+
8 rows selected (0.747
seconds)
0: jdbc:hive2://localhost:10000> describe
dept;
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| deptno    | int        | NULL
|
| dname     | string     | NULL
|
| loc       | string     | NULL
|
+-----------+------------+----------+--+
3 rows selected (0.067
seconds)
```

Tables on Postgres server:

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

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

-- create user
mapping
CREATE USER MAPPING FOR public SERVER hdfs_server OPTIONS (username 'user1', password
'pwd123');

-- create foreign
table
CREATE FOREIGN TABLE dept
(
    deptno
INTEGER,
    dname           VARCHAR(14),
    loc
VARCHAR(13)
)
SERVER hdfs_server OPTIONS (dbname 'fdw_db', table_name
'dept');

-- create foreign
table
CREATE FOREIGN TABLE emp
(
    empno           INTEGER,
    ename           VARCHAR(10),
    job
VARCHAR(9),
    mgr
INTEGER,
    hiredate
DATE,
```

```
    sal
INTEGER,
    comm            INTEGER,
    deptno
INTEGER
)
SERVER hdfs_server OPTIONS (dbname 'fdw_db', table_name
'emp');
```

Queries with join pushdown:

```
 --inner join
edb=# EXPLAIN VERBOSE SELECT t1.ename, t2.dname FROM emp t1 INNER JOIN dept t2 ON ( t1.deptno = t2.deptno
);
```

```
                                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=15.00..35.00 rows=5000 width=84)
   Output: t1.ename, t2.dname
   Relations: (fdw_db.emp t1) INNER JOIN (fdw_db.dept t2)
   Remote SQL: SELECT r1.`ename`, r2.`dname` FROM (`fdw_db`.`emp` r1 INNER JOIN `fdw_db`.`dept` r2 ON (((r1.`deptno` = r2.`deptno`))))
 (4 rows)
```

```
 --left join
edb=# EXPLAIN VERBOSE SELECT t1.ename, t2.dname FROM emp t1 LEFT JOIN dept t2 ON ( t1.deptno = t2.deptno
);
```

```
                                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=15.00..35.00 rows=5000 width=84)
   Output: t1.ename, t2.dname
   Relations: (fdw_db.emp t1) LEFT JOIN (fdw_db.dept t2)
   Remote SQL: SELECT r1.`ename`, r2.`dname` FROM (`fdw_db`.`emp` r1 LEFT JOIN `fdw_db`.`dept` r2 ON (((r1.`deptno` = r2.`deptno`))))
(4 rows)
```

```
 --right join
edb=# EXPLAIN VERBOSE SELECT t1.ename, t2.dname FROM emp t1 RIGHT JOIN dept t2 ON ( t1.deptno = t2.deptno
);
```

```
                                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=15.00..35.00 rows=5000 width=84)
   Output: t1.ename, t2.dname
   Relations: (fdw_db.dept t2) LEFT JOIN (fdw_db.emp t1)
   Remote SQL: SELECT r1.`ename`, r2.`dname` FROM (`fdw_db`.`dept` r2 LEFT JOIN `fdw_db`.`emp` r1 ON (((r1.`deptno` = r2.`deptno`))))
(4 rows)
```

```
 --full join
edb=# EXPLAIN VERBOSE SELECT t1.ename, t2.dname FROM emp t1 FULL JOIN dept t2 ON ( t1.deptno = t2.deptno
);
```

```
                                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=15.00..35.00 rows=5000 width=84)
   Output: t1.ename, t2.dname
   Relations: (fdw_db.emp t1) FULL JOIN (fdw_db.dept t2)
   Remote SQL: SELECT r1.`ename`, r2.`dname` FROM (`fdw_db`.`emp` r1 FULL JOIN `fdw_db`.`dept` r2 ON (((r1.`deptno` = r2.`deptno`))))
(4 rows)
```

```
 --cross join
edb=# EXPLAIN VERBOSE SELECT t1.ename, t2.dname FROM emp t1 CROSS JOIN dept
t2;
```

```
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=15.00..35.00 rows=1000000 width=84)
   Output: t1.ename, t2.dname
   Relations: (fdw_db.emp t1) INNER JOIN (fdw_db.dept t2)
   Remote SQL: SELECT r1.`ename`, r2.`dname` FROM (`fdw_db`.`emp` r1 INNER JOIN `fdw_db`.`dept` r2 ON (TRUE))
(4 rows)
```

Enable/disable GUC for join pushdown queries at the table level:

```
 -- enable join pushdown at the table
 level
ALTER FOREIGN TABLE emp OPTIONS (SET enable_join_pushdown
'true');
EXPLAIN (VERBOSE, COSTS OFF)
SELECT e.empno, e.ename,
d.dname
  FROM emp e JOIN dept d ON (e.deptno =
d.deptno)
  ORDER BY
e.empno;
```

```
                                                                           QUERY PLAN
```



```
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort
   Output: e.empno, e.ename, d.dname
   Sort Key: e.empno
   ->  Foreign Scan
         Output: e.empno, e.ename, d.dname
         Relations: (fdw_db.emp e) INNER JOIN (fdw_db.dept d)
         Remote SQL: SELECT r1.`empno`, r1.`ename`, r2.`dname` FROM (`fdw_db`.`emp` r1 INNER JOIN `fdw_db`.`dept` r2 ON (((r1.`deptno` = r2.`deptno`))))
(7 rows)
```

```
--Disable the GUC
enable_join_pushdown.
SET hdfs_fdw.enable_join_pushdown to
false;
-- Pushdown shouldn't happen as enable_join_pushdown is
false.
EXPLAIN (VERBOSE, COSTS OFF)
SELECT e.empno, e.ename,
d.dname
  FROM emp e JOIN dept d ON (e.deptno =
d.deptno)
  ORDER BY
e.empno;
```

```
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Sort
   Output: e.empno, e.ename, d.dname
   Sort Key: e.empno
   ->  Nested Loop
         Output: e.empno, e.ename, d.dname
         Join Filter: (e.deptno = d.deptno)
         ->  Foreign Scan on public.emp e
               Output: e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm, e.deptno
               Remote SQL: SELECT `empno`, `ename`, `deptno` FROM `fdw_db`.`emp`
         ->  Materialize
               Output: d.dname, d.deptno
               ->  Foreign Scan on public.dept d
                     Output: d.dname, d.deptno
                     Remote SQL: SELECT `deptno`, `dname` FROM `fdw_db`.`dept`
```

Enable/disable GUC for join pushdown queries at the session level:

```
SET hdfs_fdw.enable_join_pushdown to
true;
EXPLAIN (VERBOSE, COSTS OFF)
SELECT e.empno, e.ename,
d.dname
  FROM emp e JOIN dept d ON (e.deptno =
d.deptno)
  ORDER BY
e.empno;
```

```
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Sort
   Output: e.empno, e.ename, d.dname
   Sort Key: e.empno
   ->  Nested Loop
         Output: e.empno, e.ename, d.dname
         Join Filter: (e.deptno = d.deptno)
         ->  Foreign Scan on public.emp e
               Output: e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm, e.deptno
               Remote SQL: SELECT `empno`, `ename`, `deptno` FROM `fdw_db`.`emp`
         ->  Materialize
               Output: d.dname, d.deptno
               ->  Foreign Scan on public.dept d
                     Output: d.dname, d.deptno
                     Remote SQL: SELECT `deptno`, `dname` FROM `fdw_db`.`dept`
(14 rows)
```

> **Note**
>
> You can also enable/disable join pushdown at the server level using a GUC. For more information, seeCREATE SERVER.

13 Example: Aggregate pushdown

This example shows aggregate pushdown between the foreign tables of the same remote HIVE/SPARK server to that remote HIVE/SPARK server.

Tables on HIVE/SPARK server:

```
0: jdbc:hive2://localhost:10000> describe
emp;
```

```
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| empno     | int        | NULL
|
| ename     | string     | NULL
|
| job       | string     | NULL
|
| mgr       | int        | NULL
|
| hiredate  | date       | NULL
|
| sal       | int        | NULL
|
| comm      | int        | NULL
|
| deptno    | int        | NULL
|
+-----------+------------+----------+--+
8 rows selected (0.747
seconds)
0: jdbc:hive2://localhost:10000> describe
dept;
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| deptno    | int        | NULL
|
| dname     | string     | NULL
|
| loc       | string     | NULL
|
+-----------+------------+----------+--+
3 rows selected (0.067
seconds)
```

Tables on Postgres server:

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

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

-- create user
mapping
CREATE USER MAPPING FOR public SERVER hdfs_server OPTIONS (username 'user1', password
'pwd123');

-- create foreign
table
CREATE FOREIGN TABLE dept
(
    deptno
INTEGER,
    dname           VARCHAR(14),
    loc
VARCHAR(13)
)
SERVER hdfs_server OPTIONS (dbname 'fdw_db', table_name
'dept');

CREATE FOREIGN TABLE emp
(
    empno           INTEGER,
    ename           VARCHAR(10),
    job
VARCHAR(9),
    mgr
INTEGER,
    hiredate
DATE,
    sal
INTEGER,
    comm            INTEGER,
    deptno
INTEGER
)
SERVER hdfs_server OPTIONS (dbname 'fdw_db', table_name
'emp');
```

Enable/disable GUC for AGGREGATE pushdown queries at the session level:

```
 SET hdfs_fdw.enable_aggregate_pushdown to
 true;
-- aggregate functions
EXPLAIN (VERBOSE, COSTS OFF)
SELECT deptno, COUNT(*),SUM(sal),MAX(sal),MIN(sal),AVG(sal) FROM
emp
```

```
GROUP BY
deptno
HAVING deptno IN
(10,20)
ORDER BY deptno;
```

```
 QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--
 Sort
   Output: deptno, (count(*)), (sum(sal)), (max(sal)), (min(sal)), (avg(sal))
   Sort Key: emp.deptno
   ->  Foreign Scan
         Output: deptno, (count(*)), (sum(sal)), (max(sal)), (min(sal)), (avg(sal))
         Relations: Aggregate on (k_test.emp)
         Remote SQL: SELECT `deptno`, count(*), sum(`sal`), max(`sal`), min(`sal`), avg(`sal`) FROM `k_test`.`emp` WHERE (`deptno`  IN (10,20)) GROUP BY
`deptno`
(7 rows)
```

Note

You can also enable/disable aggregate pushdown at the server/table level using a GUC. For more information, see CREATE SERVER and CREATE FOREIGN TABLE.

14 Example: ORDER BY pushdown

This example shows ORDER BY pushdown between the foreign tables of the same remote HIVE/SPARK server as the remote HIVE/SPARK server.

Tables on HIVE/SPARK server:

```
0: jdbc:hive2://localhost:10000> describe
emp;
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| empno     | int        | NULL
|
| ename     | string     | NULL
|
| job       | string     | NULL
|
| mgr       | int        | NULL
|
| hiredate  | date       | NULL
|
| sal       | int        | NULL
|
| comm      | int        | NULL
|
| deptno    | int        | NULL
|
+-----------+------------+----------+--+
8 rows selected (0.747
seconds)
0: jdbc:hive2://localhost:10000> describe
dept;
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| deptno    | int        | NULL
|
| dname     | string     | NULL
|
| loc       | string     | NULL
|
+-----------+------------+----------+--+
3 rows selected (0.067
seconds)
```

Tables on Postgres server:

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

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

-- create user
mapping
CREATE USER MAPPING FOR public SERVER hdfs_server OPTIONS (username 'user1', password
'pwd123');

-- create foreign
table
```

```
CREATE FOREIGN TABLE emp
(
    empno           INTEGER,
    ename           VARCHAR(10),
    job
VARCHAR(9),
    mgr
INTEGER,
    hiredate
DATE,
    sal
INTEGER,
    comm            INTEGER,
    deptno
INTEGER
)
SERVER hdfs_server OPTIONS (dbname 'fdw_db', table_name
'emp');
```

Enable/disable GUC for ORDER BY pushdown queries at the session level:

```
edb=# SET hdfs_fdw.enable_order_by_pushdown TO ON;
SET
edb=# EXPLAIN (COSTS OFF) SELECT * FROM emp order by
deptno;
__OUTPUT__
     QUERY PLAN
---------------------
 Foreign Scan on
emp
(1 row)

edb=# SET hdfs_fdw.enable_order_by_pushdown TO OFF;
SET
edb=# EXPLAIN (COSTS OFF) SELECT * FROM emp order by
deptno;
```

```
        QUERY PLAN
---------------------------
 Sort
   Sort Key: deptno
   ->  Foreign Scan on emp
(3 rows)
```

> **Note**
>
> You can also enable/disable ORDER BY pushdown at the server/table level using a GUC. For more information, seecreate server and create foreign table.

15 Example: LIMIT OFFSET pushdown

This example shows LIMIT OFFSET pushdown between the foreign tables of the same remote HIVE/SPARK server as the remote HIVE/SPARK server:

Tables on HIVE/SPARK server:

```
0: jdbc:hive2://localhost:10000> describe
emp;
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| empno     | int        | NULL
|
| ename     | string     | NULL
|
| job       | string     | NULL
|
| mgr       | int        | NULL
|
| hiredate  | date       | NULL
|
| sal       | int        | NULL
|
| comm      | int        | NULL
|
| deptno    | int        | NULL
|
+-----------+------------+----------+--+
8 rows selected (0.747
seconds)
0: jdbc:hive2://localhost:10000> describe
dept;
+-----------+------------+----------+--+
| col_name  | data_type  | comment
|
+-----------+------------+----------+--+
| deptno    | int        | NULL
|
```

```
| dname      | string     | NULL
|
| loc        | string     | NULL
|
+------------+------------+----------+--+
3 rows selected (0.067
seconds)
```

Tables on Postgres server:

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

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

-- create user
mapping
CREATE USER MAPPING FOR public SERVER hdfs_server OPTIONS (username 'user1', password
'pwd123');

-- create foreign
table
CREATE FOREIGN TABLE emp
(
    empno           INTEGER,
    ename           VARCHAR(10),
    job
VARCHAR(9),
    mgr
INTEGER,
    hiredate
DATE,
    sal
INTEGER,
    comm            INTEGER,
    deptno
INTEGER
)
SERVER hdfs_server OPTIONS (dbname 'fdw_db', table_name
'emp');
```

Query with LIMIT OFFSET pushdown:

```
 -- LIMIT
 OFFSET
EXPLAIN (VERBOSE, COSTS OFF)
SELECT empno FROM emp e ORDER BY empno LIMIT 5 OFFSET
2;
__OUTPUT__
                                      QUERY PLAN
-------------------------------------------------------------------------------------------
 Foreign Scan on public.emp
e
   Output: empno
   Remote SQL: SELECT `empno` FROM `fdw_db`.`emp` ORDER BY `empno` ASC NULLS LAST LIMIT 2,
5
(3 rows)

SELECT empno FROM emp e ORDER BY empno LIMIT 5 OFFSET
2;
```

```
 empno
-------
  7521
  7566
  7654
  7698
  7782
(5 rows)
```

16 Identifying the 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
-----------------
<xxxxx>
```