KNIME Big Data Extensions User Guide

KNIME AG, Zurich, Switzerland
Version 5.1 (last updated on 2023-09-14)

Table of Contents

Overview

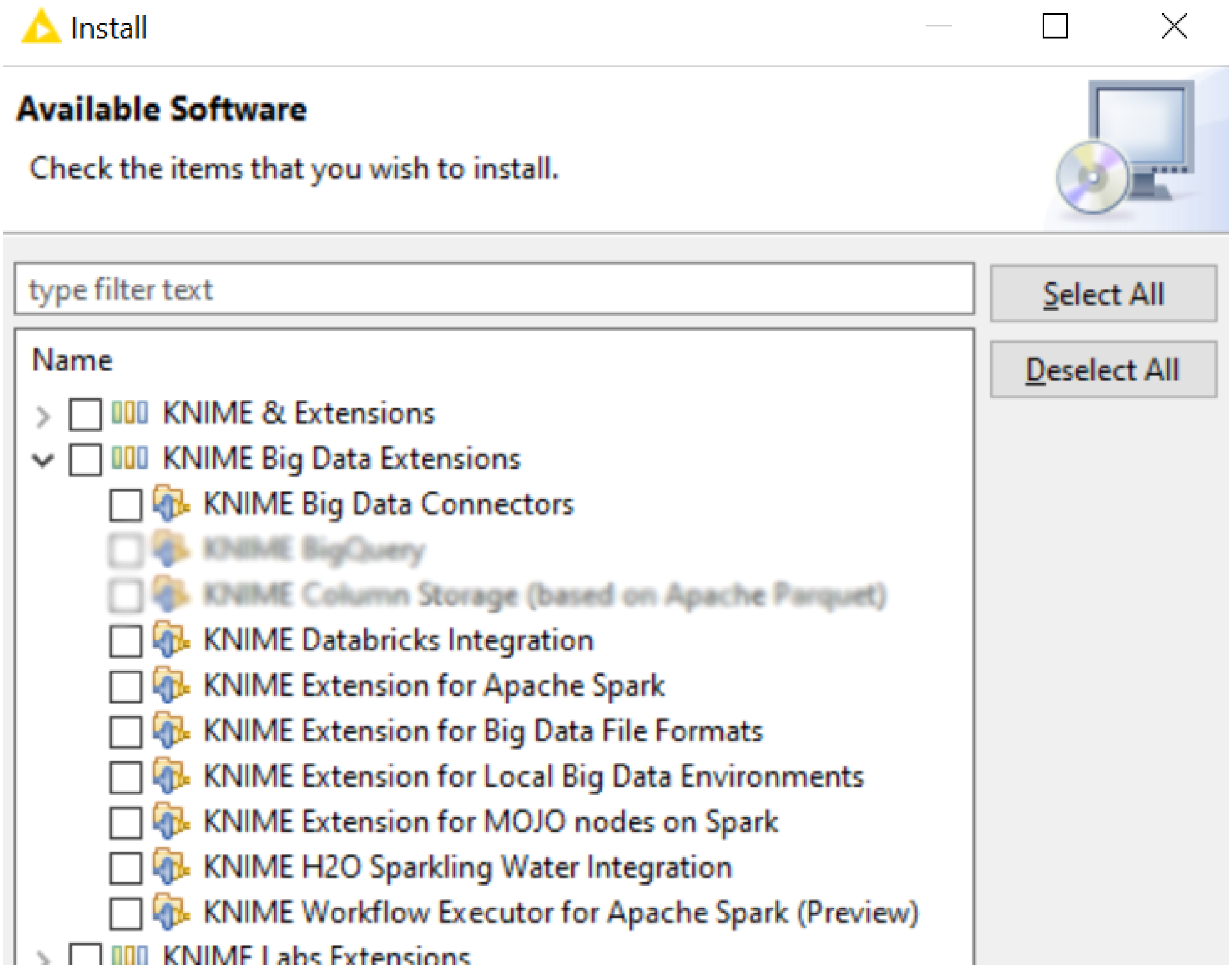
KNIME Big Data Extensions integrate Apache Spark and the Apache Hadoop ecosystem with KNIME Analytics Platform.

This guide is aimed at users of KNIME Analytics Platform who want to build workflows that need to access, process and analyze large amounts of data in a big data environment.

Note that additional installation and configuration steps may be necessary in your big data environment. Please consult the Big Data Extensions Admin Guide for details.

Installation

Navigate to *File* → *Install KNIME Extensions* and open the *KNIME Big Data Extensions* category. Check the boxes of those extensions that you wish to install.



Install

Available Software

Check the items that you wish to install.

type filter text

Select All

Deselect All

Name

- › □ KNIME & Extensions
- ⌄ □ KNIME Big Data Extensions
  - □ KNIME Big Data Connectors
  - □ KNIME BigQuery
  - □ KNIME Column Storage (based on Apache Parquet)
  - □ KNIME Databricks Integration
  - □ KNIME Extension for Apache Spark
  - □ KNIME Extension for Big Data File Formats
  - □ KNIME Extension for Local Big Data Environments
  - □ KNIME Extension for MOJO nodes on Spark
  - □ KNIME H2O Sparkling Water Integration
  - □ KNIME Workflow Executor for Apache Spark (Preview)
- › □ KNIME Labs Extensions

KNIME Big Data Extensions are a set of several extensions that build on each other:

- *KNIME Big Data Connectors* provide connector nodes to read/write files in HDFS and query Hive and Impala with SQL.
- *KNIME Extension for Big Data File Formats* allows to read/write popular file formats such as Parquet and ORC.
- *KNIME Extension for Apache Spark* provides over 60 nodes for data access and wrangling, as well as predictive analytics in Spark. The following extensions add even more functionality around Spark:
  - *KNIME Databricks Integration* integrates Databricks with KNIME. Please refer to the KNIME Databricks Integration User Guide.
  - *KNIME Extension for Local Big Data Environments* provides a node to create a completely local big data environment with Spark and Hive, without any additional configuration or software installation.
  - *KNIME Extension for MOJO nodes on Spark* provides nodes to do prediction with H2O MOJOs in Spark.
  - *KNIME H2O Sparkling Water Integration* integrates the KNIME H2O nodes with Spark to learn H2O models on data in Spark.
  - *KNIME Workflow Executor for Apache Spark (Preview)* allows to execute non-Spark KNIME nodes on Apache Spark.

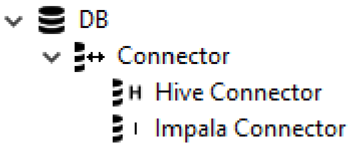
Once you have installed the extension(s), restart KNIME Analytics Platform.

> If you don’t have direct internet access you can also install the extensions from a zipped update site. Follow the steps outlined in Adding Local Update Sites.

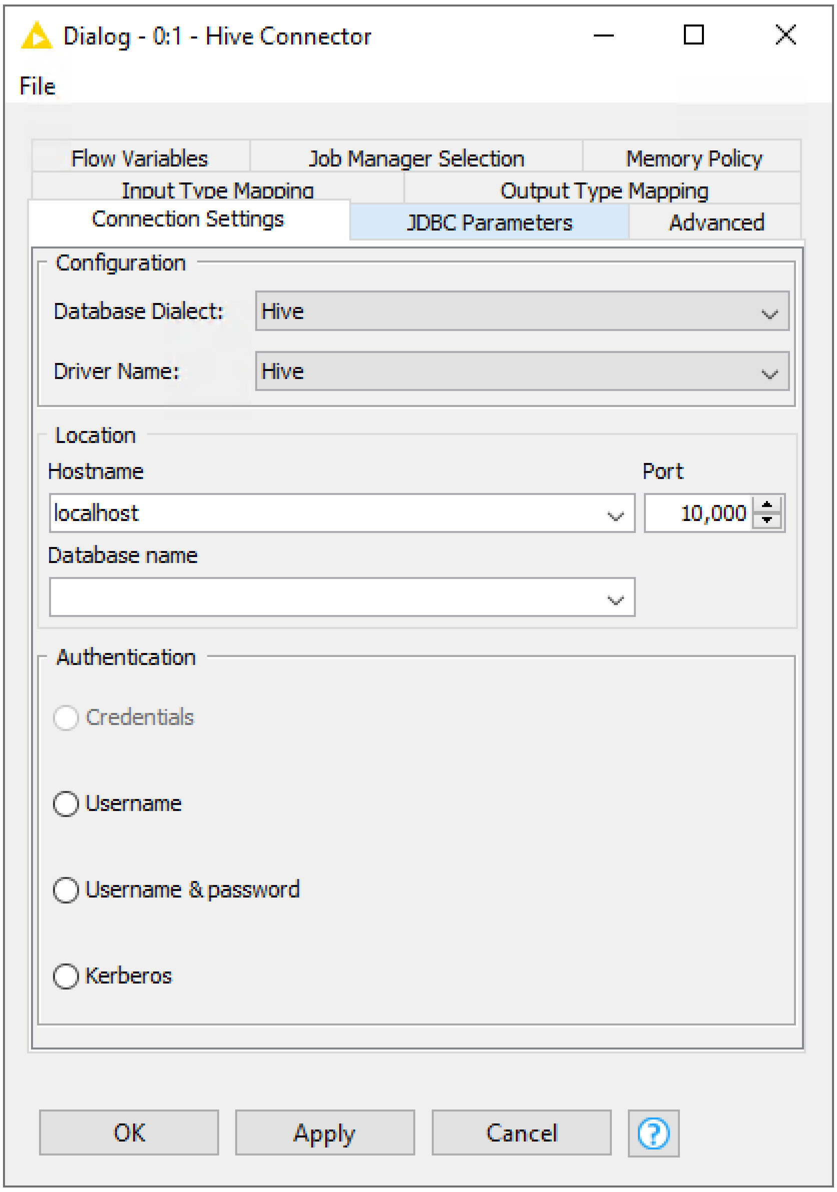
> The Spark extensions only support Apache Spark versions 3.x and 2.4. Support for older Spark versions (Spark 1.x and 2.0 - 2.3) has been moved to separate extensions. To install them, navigate to *File* → *Install KNIME Extensions* and uncheck the *Group items by category* box. Now search for "Spark". The resulting extensions which are suffixed with *(legacy)* provide support for older Spark versions.

Hive and Impala

The *KNIME Big Data Connectors* extension provides nodes to connect to Hive and Impala.



Hive Connector



Database Dialect: Hive

Driver Name: Hive

Hostname: localhost

Port: 10,000

Database name:

Figure 1. Hive Connector configuration dialog

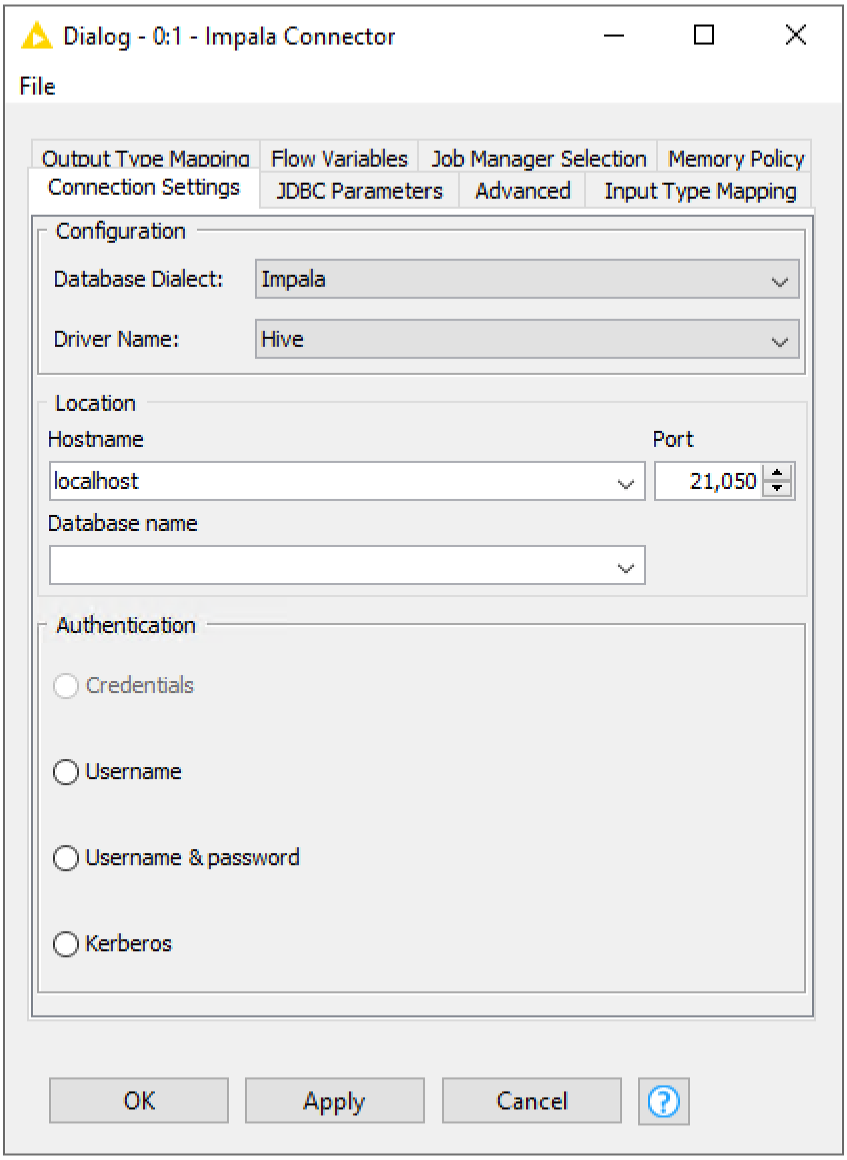
The *Hive Connector* node creates a connection to Hive via JDBC. You need to provide the following information:

- the hostname (or IP address) of the server
- the port
- a database name.
- An authentication method (as required by Hive):
  - *Credentials* where username and password are be supplied via credentials flow variable (see *Credentials Input* node).
  - *Username* where the username is supplied in the dialog.

- *Username & Password* where username and password are supplied in the dialog.
- Kerberos where authentication is performed via Kerberos.

> **When using Kerberos authentication:** Additional parameters need to be provided in the JDBC Parameters tab. The exact parameters depend on the JDBC driver selected in the *Driver Name* setting.
>
> The built-in driver with name "Hive" requires the following parameters for Kerberos:
>
> - `kerberosAuthType=fromSubject`
> - `principal=hive/<hostqdn>@<REALM>`, where
>   - `<hostqdn>` is the fully qualified hostname of the Hive service
>   - `<REALM>` is the Kerberos realm of the Hive service
>
> Proprietary Hive drivers (e.g. provided by Cloudera) require the following parameters:
>
> - `AuthMech=1`
> - `KrbServiceName=hive`
> - `KrbHostFQDN=<hostqdn>`, where `<hostqdn>` is the fully qualified hostname of the Hive service
> - `KrbRealm=<REALM>`, where `<REALM>` is the Kerberos realm of the Hive service
>
> Please note the proprietary drivers need to registered first as described in Register your own JDBC drivers (KNIME Database Extension Guide).

Impala Connector



Database Dialect: Impala

Driver Name: Hive

Hostname: localhost

Port: 21,050

Database name:

Figure 2. Impala Connector configuration dialog

The *Impala Connector* node creates a connection to Impala via JDBC. You need to provide the following information:

- the hostname (or IP address) of the Impala service
- the port
- a database name.
- An authentication method (as required by Impala):
  - *Credentials* where username and password are be supplied via credentials flow variable (see *Credentials Input* node).
  - *Username* where the username is supplied in the dialog.
  - *Username & Password* where username and password are supplied in the dialog.
  - Kerberos where authentication is performed via Kerberos.

ℹ **When using Kerberos authentication:** Additional parameters need to be provided in the JDBC Parameters tab. The exact parameters depend on the JDBC driver selected in the *Driver Name* setting.

The built-in driver with name "Hive" requires the following parameters for Kerberos:

- `kerberosAuthType=fromSubject`
- `principal=impala/<hostqdn>@<REALM>`, where
  - `<hostqdn>` is the fully qualified hostname of the Hive service
  - `<REALM>` is the Kerberos realm of the Hive service

Proprietary Hive drivers (e.g. provided by Cloudera) require the following parameters:

- `AuthMech=1`
- `KrbServiceName=impala`
- `KrbHostFQDN=<hostqdn>`, where `<hostqdn>` is the fully qualified hostname of the Hive service
- `KrbRealm=<REALM>`, where `<REALM>` is the Kerberos realm of the Hive service

Bulk data loading

The *DB Loader* node supports bulk loading of data from KNIME Analytics Platform into a Hive or Impala table. Note that the database table needs to exist prior to executing the *DB Loader* node. The example below uses the *DB Table Creator* node to create the table prior to loading the data into the table, but this is not necessary if the table does already exist.

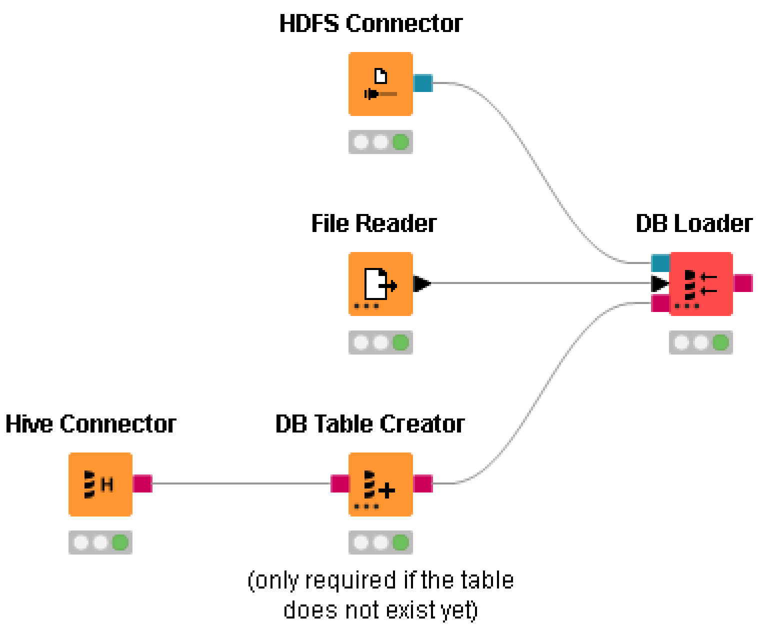
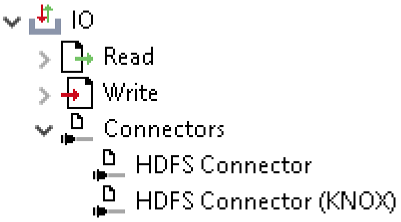


Figure 3. Workflow that creates a Hive table and then loads data into it.

HDFS

The *KNIME Big Data Connectors* extension provides several nodes to connect to HDFS



- *HDFS Connector* combines different HDFS compatible protocols in one node:
  - *HDFS* directly communicates with HDFS during data transfer, i.e. the NameNode and all DataNodes. This node requires **direct network connectivity (no proxies, no firewall)** between KNIME Analytics Platform/KNIME Server and the Hadoop cluster, which is often not the case. Restricted network connectivity typically results in timout errors during data transfer.
  - *WebHDFS* uses HTTP to directly connect to HDFS, i.e. the NameNode and all DataNodes. It is possible to connect through a HTTP proxy, but still all cluster nodes need to be reachable through the proxy.
  - *WebHDFS with SSL* uses HTTPS (SSL encrypted) to directly connect to HDFS.
  - *HttpFS* (**recommended**) uses HTTP to connect to a httpFS service on a cluster frontend/edge node. The httpFS service serves as an intermediary between the internal cluster network and KNIME Analytics Platform/KNIME Server.

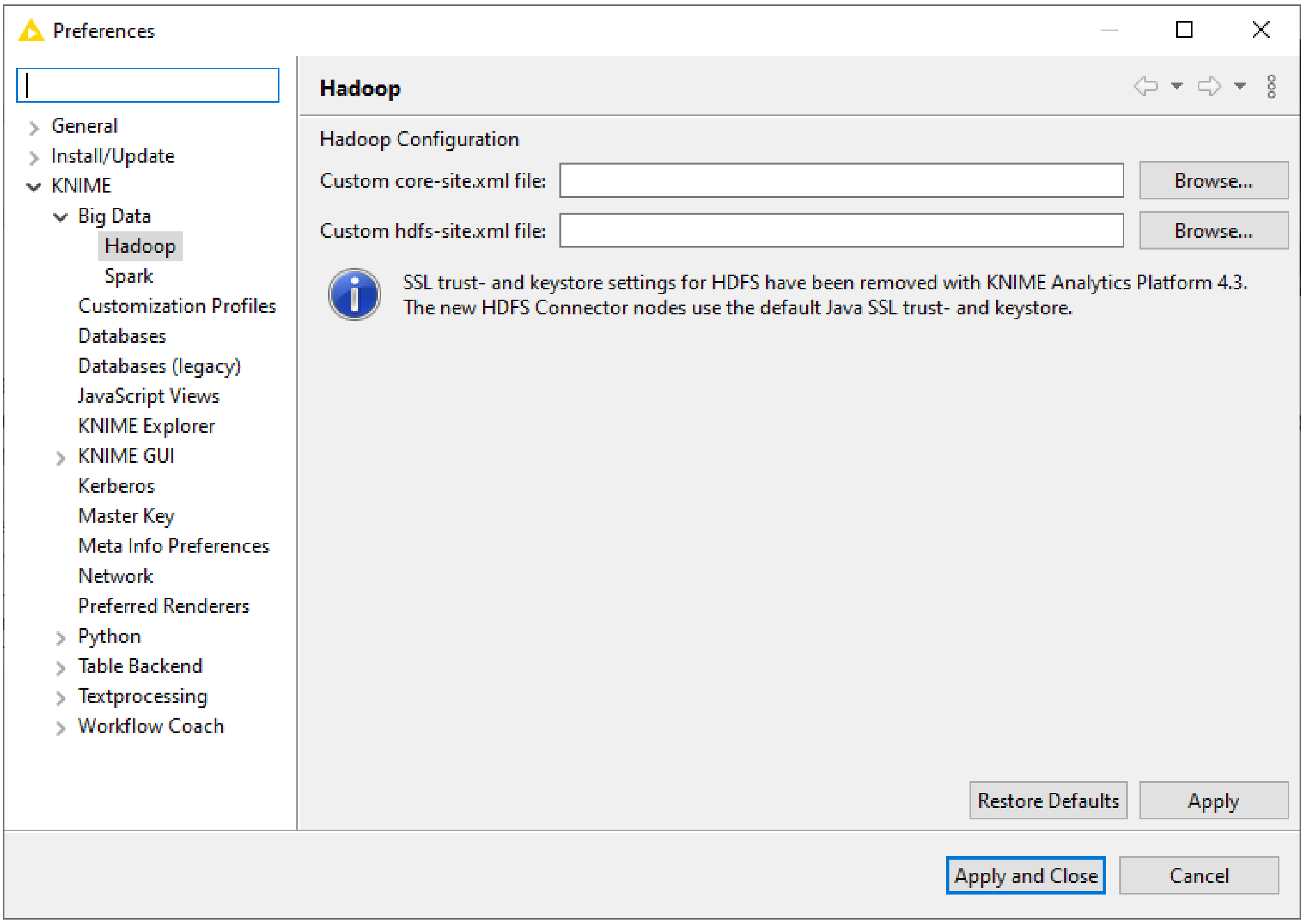
  - *HttpFS with SSL* (**recommended**) uses HTTPS (SSL encrypted) to connect to a httpFS service on a cluster frontend/edge node.

- *HDFS Connector (KNOX)* uses HTTP to connect to a Apache KNOX service on a cluster frontend/edge node. The KNOX service serves as an intermediary between the internal cluster network and KNIME Analytics Platform/KNIME Server.

The above connector nodes can be used together with the KNIME File Handling nodes to upload, download or list files and perform other file operations.

Please consult the example workflows, which are available on KNIME Hub in the KNIME Examples space.

Preferences



Hadoop Configuration

Custom core-site.xml file:

Custom hdfs-site.xml file:

SSL trust- and keystore settings for HDFS have been removed with KNIME Analytics Platform 4.3. The new HDFS Connector nodes use the default Java SSL trust- and keystore.

The Hadoop preferences allow to specify typical Hadoop configuration files (`core-site.xml` and `hdfs-site.xml`) if necessary.

More information about SSL encryption with corporate or self-signed SSL certificates can be found in the KNIME Server Administration Guide.

Spark

KNIME Extension for Apache Spark provides a set of over 60 nodes to create and execute Apache Spark applications.

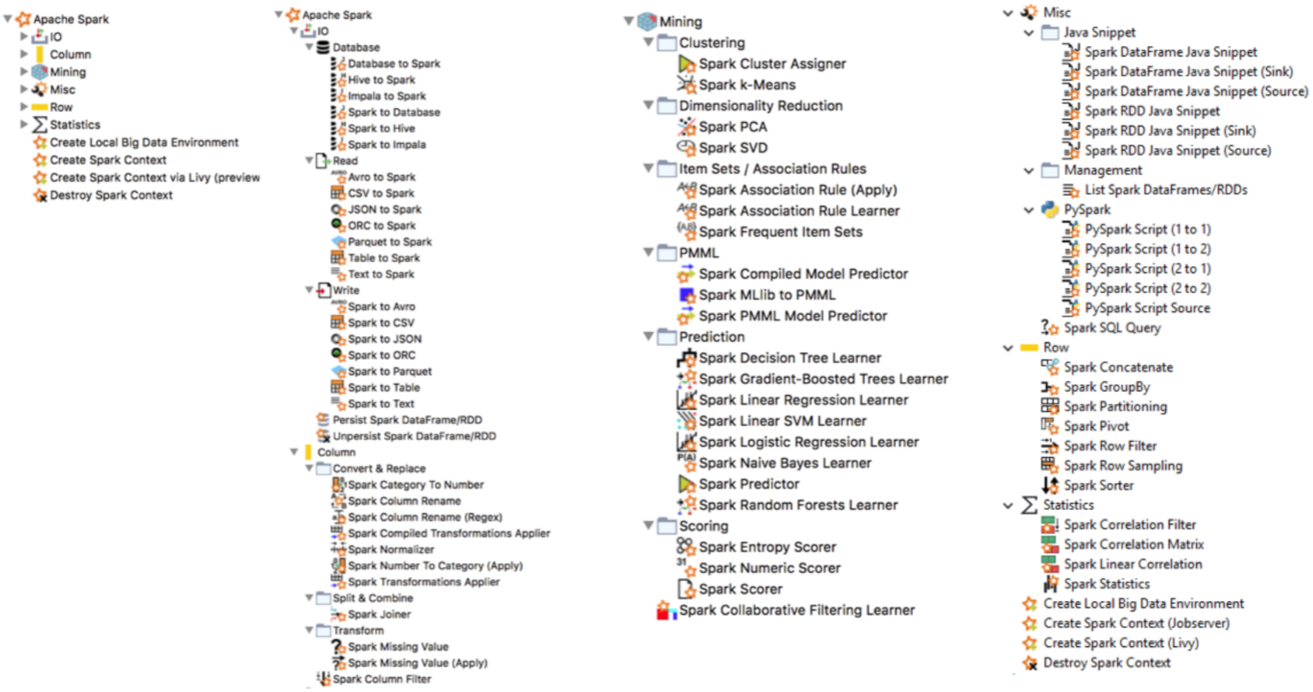


Figure 4. All Spark nodes

The first step in any Spark workflow is to create a Spark context, which represents the connection to a Spark cluster. The Spark context also reserves resources (CPU cores and memory) in your cluster to be exclusively used by your workflow. Hence, a Spark context should be created at the beginning of a workflow and destroyed at the end, in order to release the resources.

There are several nodes to create a Spark context:

- Create Spark Context (Livy) (recommended)
- *Create Local Big Data Environment* (requires *KNIME Extension for Local Big Data Environments*)
- Create Databricks Environment (requires *KNIME Databricks integration*)

Create Spark Context (Livy)

The *Create Spark Context (Livy)* node connects to an Apache Livy server to create a new Spark context.

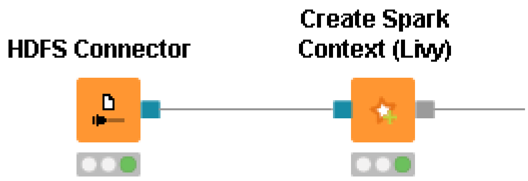
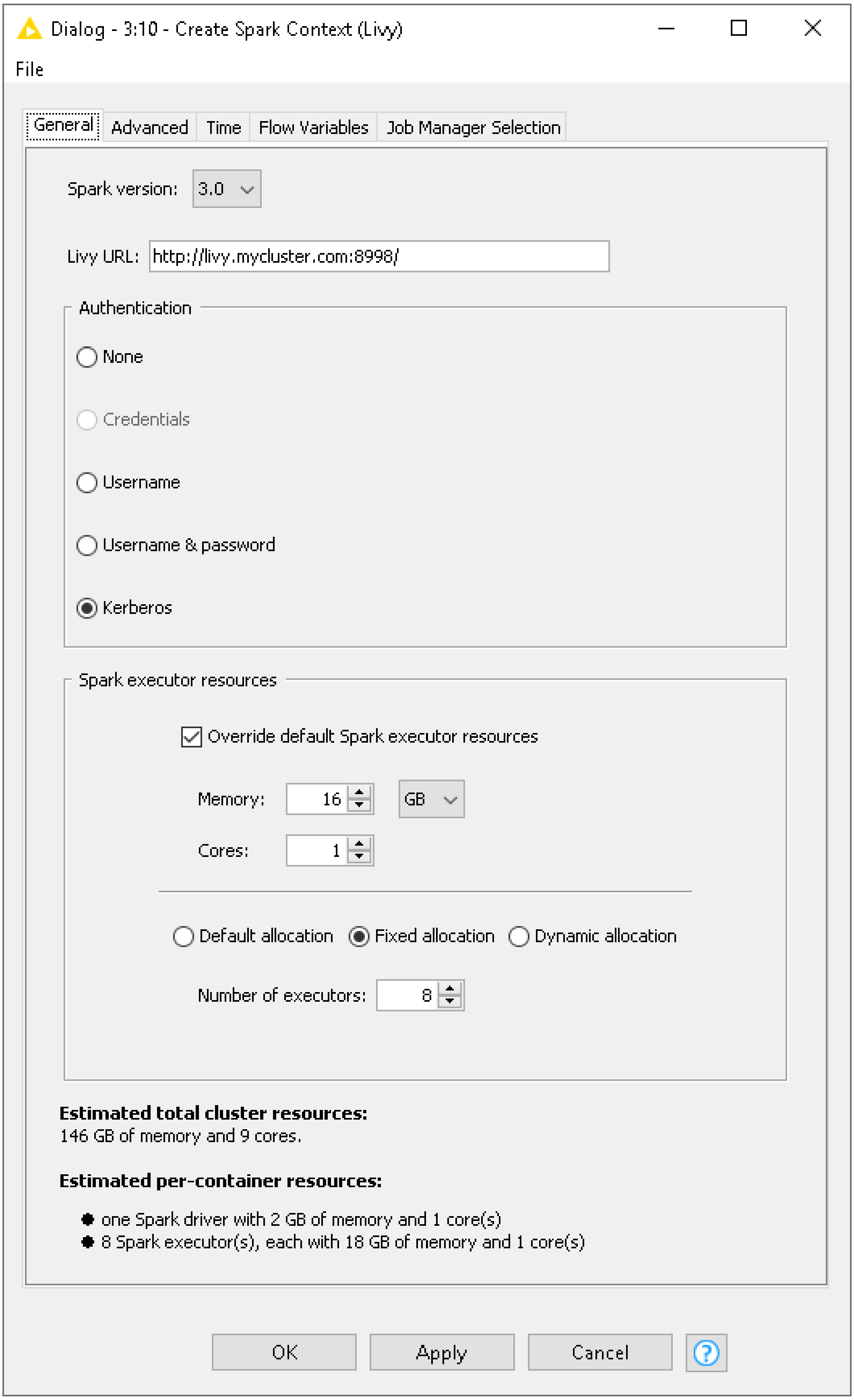


Figure 5. Create Spark Context (Livy) node

Requirements

- **Apache Livy service** Livy needs to be installed as a service in your cluster. Please consult the Apache Livy setup section for more details.
- **Network Connectivity:** The node initiates a HTTP(S) connection to Livy (default port TCP/8998). Currently, only HTTP(S) proxies that do not require any authentication are supported.
- **Authentication**: If Livy requires Kerberos authentication, then KNIME Analytics Platform needs to be set up accordingly.
- **Remote file system connection**: The node requires access to a remote file system to exchange temporary files between KNIME Analytics Platform and the Spark context (running on the cluster). Supported file systems connectors are:
  - HDFS and HDFS (KNOX). Note that the node must access the remote file system with the same user as the Spark context. When authenticating with Kerberos against both HDFS/WebHDFS/HttpFS and Livy, then the same user will be used. Otherwise, this must be ensured manually.
  - Amazon S3, Azure Blob Store and Google Cloud Storage, which is recommended when using Spark on Amazon EMR/Azure HDInsight/Google Datapoc. Note that for these file systems a staging area must be specified in the **Advanced** tab of the *Create Spark Context (Livy)* node.

Node dialog



Spark version: 3.0

Livy URL: http://livy.mycluster.com:8998/

Authentication

○ None

○ Credentials

○ Username

○ Username & password

● Kerberos

Spark executor resources

☑ Override default Spark executor resources

Memory: 16 GB

Cores: 1

○ Default allocation ● Fixed allocation ○ Dynamic allocation

Number of executors: 8

Estimated total cluster resources:
146 GB of memory and 9 cores.

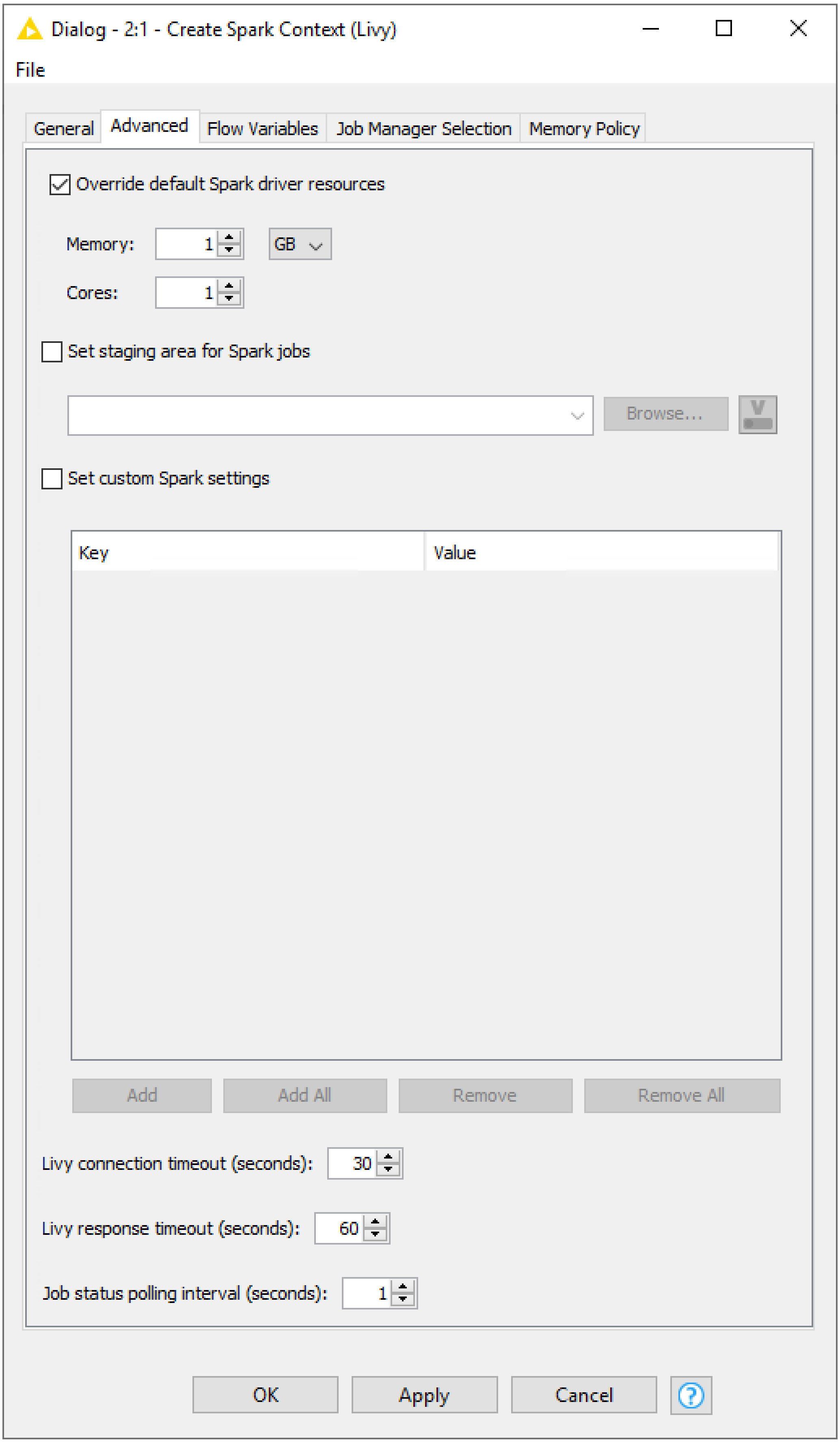
Estimated per-container resources:

- one Spark driver with 2 GB of memory and 1 core(s)
- 8 Spark executor(s), each with 18 GB of memory and 1 core(s)

Figure 6. Create Spark Context (Livy): General settings tab

The node dialog has two tabs. The first tab provides the most commonly used settings when working with Spark:

1. **Spark version:** Please choose the Spark version of the Hadoop cluster you are connecting to.
2. **Livy URL:** The URL of Livy including protocol and port e.g. `http://localhost:8998/`.
3. **Authentication:** How to authenticate against Livy. Supported mechanism are Kerberos and None.
4. **Spark Executor resources:** Sets the resources to be request for the Spark executors. If enabled, you can specify the amount of memory and the number of cores for each executor. In addition you can specify the Spark executor allocation strategy.
5. **Estimated resources:** An estimation of the resources that are allocated in your cluster by the Spark context. The calculation uses default settings for memory overheads etc. and is thus only an estimate. The exact resources might be different depending on your specific cluster settings.



Dialog - 2:1 - Create Spark Context (Livy)

☑ Override default Spark driver resources

Memory: 1 GB

Cores: 1

□ Set staging area for Spark jobs

□ Set custom Spark settings

Key	Value

Livy connection timeout (seconds): 30

Livy response timeout (seconds): 60

Job status polling interval (seconds): 1

Figure 7. Create Spark Context (Livy): Advanced settings tab

The second tab provides the advanced settings that are sometimes useful when working with Spark:

1. **Override default Spark driver resources:** If enabled, you can specify the amount of memory and number of cores to be allocated for the Spark driver process.
2. **Set staging area for Spark jobs:** If enabled, you can specify a directory in the connected remote file system, that will be used to transfer temporary files between KNIME and the Spark context. If no directory is set, then a default directory will be chosen, e.g. the HDFS user home directory. However, if the remote file system is Amazon S3 or Azure Blob Store, then a staging directory must be provided.
3. **Set custom Spark settings:** If enabled, you can specify additional Spark settings. A tooltip is provided for the keys if available. For further information about the Spark settings refer to the Spark documentation.

Destroy Spark Context node

Once you have finished your Spark job, you should destroy the created context to free up the resources your Spark Context has allocated on the cluster. To do so you can use the **Destroy Spark Context** node.

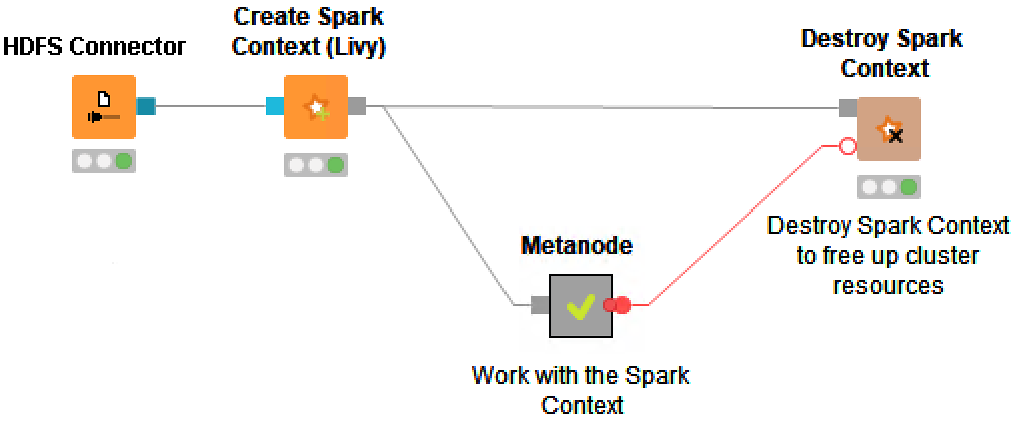
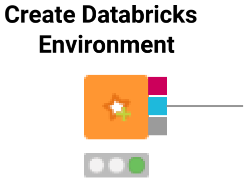

HDFS Connector
Create Spark Context (Livy)
Destroy Spark Context
Metanode
Destroy Spark Context to free up cluster resources
Work with the Spark Context

Figure 8. How to use the Destroy Spark Context node

Create Databricks Environment

This node connects to a Databricks cluster from within KNIME Analytics Platform.



For a more detailed guide on how to configure this node and how to create a Databricks cluster please refer to the KNIME Databricks Integration User Guide.

Create H2O Sparkling Water Context

The *Create H2O Sparkling Water Context* node creates a H2O context inside a given Spark context. This allows you use the KNIME H2O nodes on data in Spark. This node requires the *KNIME H2O Sparkling Water Integration*. More information about installing the integration can be found above.

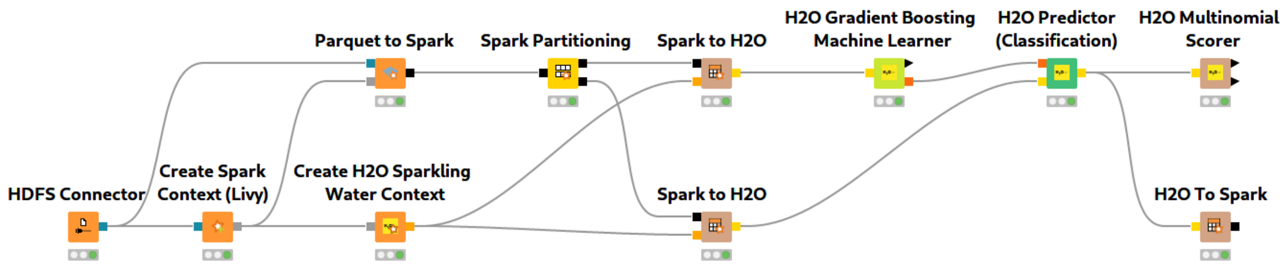

Parquet to Spark
Spark Partitioning
Spark to H2O
H2O Gradient Boosting Machine Learner
H2O Predictor (Classification)
H2O Multinomial Scorer
HDFS Connector
Create Spark Context (Livy)
Create H2O Sparkling Water Context
Spark to H2O
H2O To Spark

Figure 9. Create H2O Sparkling Water Context node

> KNIME supports only a limited set of Spark and H2O versions. The next section contains a table with all compatible versions. Open *File* > *Preferences* > *KNIME* > *H2O-3* and select an H2O version that works with the Spark version you are using.

The *Create H2O Sparkling Water Context* node detects if the required H2O libraries are already present on the cluster. If not, it tries to upload them, which means uploading ~20MB of libraries every time the node is executed. For better performance, the libraries can be directly installed from maven when the Spark context is created. To do so, specify the correct maven coordinate in the Spark setting `spark.jars.packages` (see the *Set custom Spark settings* option here). The table below specifies the combinations of Spark and H2O versions that KNIME supports, as well as the respective Sparkling Water maven coordinate.

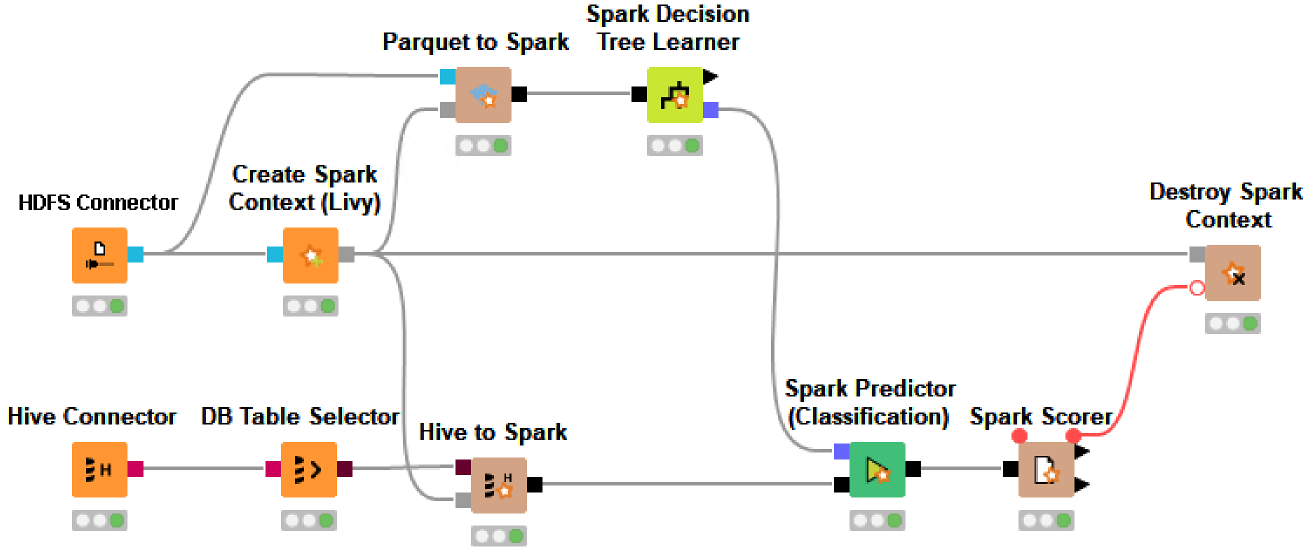
Spark Version	H2O Version	Sparkling Water Maven Coordinate
3.3	3.36.1.5	`ai.h2o:sparkling-water-package_2.12:3.36.1.5-1-3.3`
3.2	3.36.1.2	`ai.h2o:sparkling-water-package_2.12:3.36.1.2-1-3.2`
3.1	3.32.1.2	`ai.h2o:sparkling-water-package_2.12:3.32.1.2-1-3.1`
3.0	3.32.1.2	`ai.h2o:sparkling-water-package_2.12:3.32.1.2-1-3.0`

2.4	3.30.0.4	`ai.h2o:sparkling-water-package_2.11:3.30.0.4-1-2.4`
2.4	3.24.0.4	`ai.h2o:sparkling-water-package_2.11:2.4.12`
2.4	3.22.0.2	`ai.h2o:sparkling-water-package_2.11:2.4.1`
2.3	3.24.0.4	`ai.h2o:sparkling-water-package_2.11:2.3.30`
2.3	3.22.0.2	`ai.h2o:sparkling-water-package_2.11:2.3.18`
2.2	3.22.0.2	`ai.h2o:sparkling-water-package_2.11:2.2.29`

Proxy settings

If your network requires you to connect to Livy Server or Databricks via a proxy, please open *File* > *Preferences* > *Network Connections*. Here you can configure the details of your HTTP/HTTPS/SOCKS proxies. Please consult the official Eclipse documentation on how to configure proxies.

Example workflow



The above example workflow first creates a Spark context (*Create Spark Context (Livy)*) and then reads training data from a Parquet file stored in HDFS (*Parquet to Spark*). It then trains a decision tree model (*Spark Decision Tree Learner*) on that data. Additionally, it reads a Hive table with test data into Spark (*Hive to Spark*), uses the previously learned model to perform predictions (*Spark Predictor*), and determines the accuracy of the model given the test data (*Spark Scorer*).

For more examples consult the KNIME Examples space on KNIME Hub where you will find a variety of example workflows that demonstrate how to use the Spark nodes.

Open for Innovation
KNIME

KNIME AG
Talacker 50
8001 Zurich, Switzerland
www.knime.com
info@knime.com