Open for Innovation
KNIME®

KNIME Extension for Apache Spark Installation Guide

KNIME AG, Zurich, Switzerland
Version 4.0 (last updated on 2019-06-29)

Table of Contents

Introduction

KNIME Extension for Apache Spark™ provides KNIME nodes to create workflows that offload computation to a Spark cluster.

This document describes steps to integrate KNIME Extension for Apache Spark™ with a Spark environment, which requires additional installation steps on the **cluster-side**.

This guide applies to KNIME Analytics Platform and KNIME Server. However, running Spark workflows on KNIME Server will require additional steps outlined in Secured Cluster Connection Guide for KNIME Server.

As depicted below, KNIME Extension for Apache Spark consists of:

- a client-side extension for KNIME Analytics Platform/KNIME Server
- a cluster-side REST service, that needs to be installed on an edge node of your Hadoop cluster or a node that can execute the `spark-submit` command. The service must be one of:
    - Apache Livy (**recommended**)
    - Spark Job Server (deprecated)

[sparkitechture] | *sparkitechture.png*

Supported Spark and Hadoop distributions

KNIME Extension for Apache Spark is compatible with Spark 1.2 and higher, as well as all versions of Spark 2.

The following Hadoop distributions include a compatible version of Spark:

- Hortonworks HDP 2.2 and higher
- Hortonworks HDP 3.0 and higher
- Cloudera CDH 5.3 and higher (with Spark 2 provided by Cloudera CDS).
- Cloudera CDH 6.0 and higher
- Amazon EMR 5.9 and higher

Supported Apache Livy versions

KNIME Extension for Apache Spark is compatible with Apache Livy 0.4.0 and higher.

Using KNIME with Livy requires at least Spark 2.2.

The following Hadoop distributions include compatible versions of Livy and Spark:

- Livy 0.4.0:
  - Hortonworks HDP 2.6.3 - 2.6.5
  - Amazon EMR 5.9 - 5.16
- Livy 0.5.0:
  - Hortonworks HDP 3.0 and higher
  - Amazon EMR 5.16 and higher

For Cloudera CDH 5/6, KNIME provides a CSD and parcel so that Livy can be installed as an add-on service (see Cloudera CDH).

Supported Spark Job Server versions

Spark Job Server support in KNIME is deprecated and will be discontinued in the near future. New installations should use Livy instead, if they use at least Spark 2.2.

KNIME Extension for Apache Spark is compatible with the following Spark Job Server versions (provided by KNIME):

- 0.6.2.3-KNIME (for Spark 1.x)
- 0.7.0.3-KNIME (for Spark 2.x).

Supported versions of KNIME software

KNIME Extension for Apache Spark is compatible with the following versions of KNIME software:

- KNIME Analytics Platform 3.7
- KNIME Server 4.8

Apache Livy setup

Cloudera CDH

For Cloudera CDH, KNIME provides a CSD and parcel so that Livy can be installed as an add-on service. The current version of Livy for CDH provided by KNIME is `0.5.0.knime3`.

Requirements

- CDH 5.8 and higher, or CDH 6.0 and higher
- *Only On CDH 5*: Spark 2.2 or higher as an add-on service (provided by Cloudera CDS)

Steps to install

The following steps require administrative shell (e.g. via SSH) access to the machine where Cloudera Manager is installed, as well as a user who has full administrative access on the Cloudera Manager WebUI.

The following steps describe how to install Livy as managed service through Cloudera Manager using a parcel. If in doubt, please also consider the official Cloudera documentation on handling parcels.

Steps to perform via shell on the machine where Cloudera Manager is installed:

1. Download/copy one of the following CSDs to `/opt/cloudera/csd/` on machine, where Cloudera Manager is installed:
   - CSD for CDH 5
   - CSD for CDH 6
2. *If Cloudera Manager cannot access the public internet:* Download/copy the matching `.parcel` and `.sha1` file to `/opt/cloudera/parcel-repo`:
   - For CDH 5:

RHEL/CentOS	RHEL 7: parcel / sha	RHEL 6: parcel / sha	RHEL 5: parcel / sha
SLES	SLES 12: parcel / sha	SLES 11: parcel / sha	
Ubuntu	Ubuntu 16 (Xenial): parcel / sha	Ubuntu 14 (Trusty): parcel / sha	Ubuntu 12 (Precise): parcel / sha
Debian	Debian 8 (Jessie): parcel / sha	Debian 7 (Wheezy): parcel / sha	

- For CDH 6:

RHEL/CentOS	RHEL 7: parcel / sha	RHEL 6: parcel / sha	RHEL 5: parcel / sha
SLES	SLES 12: parcel / sha	SLES 11: parcel / sha	
Ubuntu	Ubuntu 16 (Xenial): parcel / sha	Ubuntu 14 (Trusty): parcel / sha	Ubuntu 12 (Precise): parcel / sha
Debian	Debian 8 (Jessie): parcel / sha	Debian 7 (Wheezy): parcel / sha	

3. Restart Cloudera Manager from the command line, for example with:

```
systemctl restart cloudera-scm-server
```

Steps to perform on the Cloudera Manager WebUI:

1. Navigate to the Parcel manager and locate the `LIVY` parcel.
2. Download (unless already done manually), Distribute and Activate the `LIVY` parcel.
3. Add the Livy Service to your cluster (see the official Cloudera documentation on adding services).
4. Navigate to the HDFS service configuration and add the following settings to the *Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml*:
   - `hadoop.proxyuser.livy.hosts=*`
   - `hadoop.proxyuser.livy.groups=*`
5. *If your cluster is using HDFS Transparent Encryption:* Navigate to the KMS service configuration and add the following settings to the *Key Management Server Advanced Configuration Snippet (Safety Valve) for kms-site.xml*:
   - `hadoop.kms.proxyuser.livy.hosts=*`
   - `hadoop.kms.proxyuser.livy.groups=*`
6. *If you plan to run Spark workflows on KNIME Server:* Please consult the Secured Cluster Connection Guide for KNIME Server to allow KNIME Server to impersonate users.
7. Restart all services affected by your configuration changes.

Hortonworks HDP

Hortonworks HDP already includes compatible versions of Apache Livy and Spark 2 (see Supported Apache Livy versions). Please follow the respective Hortonworks documentation to install Spark with the *Livy for Spark2 Server* component:

- Installing Spark Using Ambari (HDP 2.6.5)
- Install Spark Using Ambari (HDP 3.1)

KNIME Extension for Apache Spark only supports *Livy for Spark2 Server* which uses Spark 2. The *Livy for Spark Server* component is not supported, since it is based on Spark 1.

Amazon EMR

Amazon EMR already includes compatible versions of Apache Livy and Spark 2 (see Supported Apache Livy versions), simply make sure to select *Livy* in the software configuration of your cluster.

Spark Job Server setup

Spark Job Server support in KNIME is deprecated and will be discontinued in the near future. New installations should use Livy instead, if they use at least Spark 2.2.

This section describes how to install Spark Job Server on a Linux machine. Spark Job Server provides a RESTful interface for submitting and managing Apache Spark jobs, jars, and job contexts. KNIME Extension for Apache Spark requires Spark Job Server to execute and manage Spark jobs.

Background

Spark Job Server was originally developed at Ooyala, but the main development repository is now on GitHub. For more information please consult the GitHub repository, including licensing conditions, contributors, mailing lists and additional documentation.

In particular, the README of the Job Server contains dedicated sections about **HTTPS / SSL Configuration and Authentication**. The GitHub repository also contains general Troubleshooting and Tips as well as YARN Tips section. All Spark Job Server documentation

is available in the doc folder of the GitHub repository.

KNIME packages the official Spark Job Server and — if necessary — adapts it for the supported Hadoop distributions. These packages can be downloaded from [download_sjs]. We **strongly recommend** using these packages as they contain additional bugfixes and improvements and have been tested and verified by KNIME.

Versions

The current versions of Spark Job Server provided by KNIME are:

- 0.6.2.3-KNIME (for Spark 1.x)
- 0.7.0.3-KNIME (for Spark 2.x).

Updating Spark Job Server

If you are updating an existing **0.6.2.1-KNIME** or older installation, there may be changes required to some of its configuration files. The `README` of the Spark Job Server packaged by KNIME highlights any necessary changes to those files.

If you are updating an existing **0.7.0.1-KNIME** or older installation, it is strongly recommended to make a fresh installation based on this guide. Due to the number of changes to the server's default configuration and directory layout, copying an old configuration will not work.

Due to different default installation locations between **0.6.2.x-KNIME** (for Spark 1.x) and **0.7.0.x-KNIME** (for Spark 2.x), it is possible to install both versions of Spark Job Server at the same time on the same machine.

Requirements

Spark Job Server uses the `spark-submit` command to run the Spark driver program and executes Spark jobs submitted via REST. Therefore, it must be installed on a machine that

- runs Linux (RHEL 6.x/7.x recommended, Ubuntu 14 .x is also supported)
- has full network connectivity to all your Hadoop cluster nodes,
- can be connected to via HTTP (default port TCP/8090) from KNIME Analytics Platform and/or KNIME Server,
- has all libraries and cluster-specific configuration for Spark, Hadoop and Hive libraries

set up.

This can for example be the Hadoop master node or a cluster edge node.

Installation

> The installation of Spark Job Server needs to be performed as the Linux `root` user. It is however not recommended to run Spark Job Server as `root`.

1. Locate the Spark Job Server package from [download_sjs] that matches your Hadoop distribution:

   Note: Due to different default installation locations between Spark Job Server versions **0.6.2.x-KNIME** (for Spark 1.x) and **0.7.0.x-KNIME** (for Spark 2.x), it is possible to install both versions of Spark Job Server at the same time on the same machine. If you choose to do this, walk through steps 2. — 7. once for each version.

2. Download the file on the machine where you want to install Spark Job Server.
3. Log in as `root` on that machine and install the Job Server as follows (replace xxx with the version of your download). First, define a shell variable which we will be using in the remainder of the installation:

   For 0.6.2.x-KNIME_xxx (Spark 1.x)

   ```
   LINKNAME=spark-job-server
   ```

   For 0.7.0.x-KNIME_xxx (Spark 2.x)

   ```
   LINKNAME=spark2-job-server
   ```

   For all versions of Spark Job Server proceed with

   ```
   useradd -d /opt/${LINKNAME}/ -M -r -s /bin/false spark-job-server
   su -l -c "hdfs dfs -mkdir -p /user/spark-job-server ; hdfs dfs -chown -R \
    spark-job-server /user/spark-job-server" hdfs
   cp /path/to/spark-job-server-xxx.tar.gz /opt
   cd /opt
   tar xzf spark-job-server-xxx.tar.gz
   ln -s spark-job-server-xxx ${LINKNAME}
   chown -R spark-job-server:spark-job-server ${LINKNAME} spark-job-server-xxx/
   ```

4. If you are installing on RedHat Enterprise Linux (RHEL), CentOS or Ubuntu 14.x then run the following commands to make Spark Job Server start during system boot:

On RHEL 6.x/CentOS 6.x

```
ln -s /opt/${LINKNAME}/spark-job-server-init.d /etc/init.d/${LINKNAME}
chkconfig --levels 2345 ${LINKNAME} on
```

On RHEL 7.x/CentOS 7.x

```
ln -s /opt/${LINKNAME}/spark-job-server-init.d /etc/init.d/${LINKNAME}
systemctl daemon-reload
systemctl enable ${LINKNAME}
```

On Ubuntu 14.x

```
ln -s /opt/${LINKNAME}/spark-job-server-init.d-ubuntu-sysv /etc/init.d/${LINKNAME}
update-rc.d ${LINKNAME} start 20 2 3 4 5 . stop 20 0 1 6 .
```

The boot script will run the Job Server as the `spark-job-server` system user. If you have installed Job Server to a different location, or wish to run it with a different user, you will have to change the `JSDIR` and `USER` variables in the boot script.

5. Edit `environment.conf` in the server's installation directory as appropriate. The most important settings are:
    - **master:**
        - Set `master = yarn-client` for running Spark in YARN-client mode
        - Set `master = spark://localhost:7077` for standalone mode
        - Set `master = local[4]` for local debugging.
        - **Note**: yarn-cluster mode is currently not supported by Spark Job Server.
    - **Settings for predefined contexts:** Under `context-settings`, you can predefine Spark settings for the default Spark context. Please note that these settings can be overwritten by the client-side configuration of the KNIME Extension for Apache Spark. Under `contexts` you can predefine Spark settings for non-default Spark contexts.
6. **Optional:** Edit `settings.sh` as appropriate:
    - `SPARK_HOME`, please change if Spark is not installed under the given location.
    - `LOG_DIR`, please change if you want to log to a non-default location.
7. **Optional:** Edit `log4j-server.properties` as appropriate. This should not be necessary unless you wish to change the defaults.

Starting Spark Job Server

Notes:

- In the following, replace `${LINKNAME}` with either `spark-job-server` or `spark2-job-server` depending on which value you have been using in the previous section.
- It is not recommended to start Spark Job Server with the `server_start.sh` in its installation directory.
- You can verify that Spark Job Server has correctly started via the WebUI (see Spark Job Server web UI).

On RHEL 6 and Ubuntu 14.x

```
/etc/init.d/${LINKNAME} start
```

On RHEL 7 and higher

```
systemctl start ${LINKNAME}
```

Stopping Spark Job Server

Notes:

- Replace `${LINKNAME}` with either `spark-job-server` or `spark2-job-server` depending on which value you have been using in the previous section.
- It is not recommended to stop the server with the `server_stop.sh` script.

On RHEL 6 and Ubuntu 14.x

```
/etc/init.d/${LINKNAME} stop
```

On RHEL 7 and higher

```
systemctl stop ${LINKNAME}
```

Installation on a Kerberos-secured cluster

In a Kerberos-secured cluster, Spark Job Server requires a Ticket Granting Ticket (TGT) to access Hadoop services and provide user impersonation. To set this up, please first follow the installation steps in the previous section. Then proceed with the following steps:

1. Create a service principal and a keytab file for the `spark-job-server` Linux user. By default this is assumed to be `spark-job-server/host@REALM`, where
   - `host` is the fully qualified hostname (FQDN) of the machine where you are installing Job Server,
   - `REALM` is the Kerberos realm of your cluster.
2. Upload the keytab file to the machine where Job Server is installed and limit its accessibility to only the spark-job-server system user:

   ```
   chown spark-job-server:spark-job-server /path/to/keytab
   chmod go= /path/to/keytab
   ```

3. Now you have to tell Job Server about the keytab file and, optionally, about the service principal you have created. In `/opt/spark-job-server-xxx/settings.sh` uncomment and edit the following lines:

   ```
   export JOBSERVER_KEYTAB=/path/to/keytab
   export JOBSERVER_PRINCIPAL=user/host@REALM
   ```

   Note: You only need to set the principal, if it is different from the assumed default principal `spark-job-server/$(hostname -f)/<default realm from /etc/krb5.conf>`

4. In `environment.conf` set the following properties:

   ```
   spark {
     jobserver {
       context-per-jvm = true
     }
   }
   shiro {
     authentication = on
     config.path = "shiro.ini"
     use-as-proxy-user = on
   }
   ```

   The effect of these settings is that Job Server will authenticate all of its users, and each user will have its own Spark context, that can access Hadoop resources in the name of this user.

5. Configure the authentication mechanism of Job Server in the `shiro.ini` file. Instructions for authenticating against LDAP or ActiveDirectory are covered in the Setting up LDAP authentication section of this guide. Some example templates can also be found in the Spark Job Server installation folder.

6. Add the following properties to the `core-site.xml` of your Hadoop cluster:

```
hadoop.proxyuser.spark-job-server.hosts = *
hadoop.proxyuser.spark-job-server.groups = *
```

   This must be done either via Ambari (on HDP) or Cloudera Manager (on CDH). A restart of the affected Hadoop services is required.

Setting up LDAP authentication

Spark Job Server uses the Apache Shiro™ framework to authenticate its users, which can delegate authentication to an LDAP server, e.g. OpenLDAP or Microsoft ActiveDirectory®.

Set up LDAP authentication as follows:

1. Activate shiro authentication in `environment.conf`:

```
shiro {
  authentication = on
  config.path = "shiro.ini"
  [...other settings may be here...]
}
```

2. Create an empty `shiro.ini`.
3. Configure `shiro.ini` using one of the templates from the following sections, depending on whether you want to authenticate against OpenLDAP or ActiveDirectory and with or without group membership checking.

   - Do not change the order of the lines in the templates.
   - Do not use single or double quotes unless you want them to be part of the configuration values. The `ini` file format does not support quoting.

4. Activate the shiro authentication cache by appending the following lines to `shiro.ini`:

```
cacheManager = org.apache.shiro.cache.MemoryConstrainedCacheManager
securityManager.cacheManager = $cacheManager
```

Template: OpenLDAP without group membership checking

```
myRealm = org.apache.shiro.realm.ldap.JndiLdapRealm
myRealm.contextFactory.url = ldap://ldapserver.company.com
myRealm.userDnTemplate = uid={0},ou=people,dc=company,dc=com
```

Notes:

- In `myRealm.userDnTemplate` the placeholder `{0}` is replaced with the login name the user enters.

Template: ActiveDirectory without group membership checking

```
myRealm = org.apache.shiro.realm.ldap.JndiLdapRealm
myRealm.contextFactory.url = ldap://ldapserver.company.com
myRealm.userDnTemplate = {0}@COMPANY.COM
```

Notes:

- In `myRealm.userDnTemplate` the placeholder `{0}` is replaced with the login name the user enters. ActiveDirectory then authenticates against the user record with a matching `sAMAccountName`.

Template: OpenLDAP with group membership checking

```
myRealm = spark.jobserver.auth.LdapGroupRealm
myRealm.contextFactory.url = ldap://ldapserver.company.com
myRealm.userDnTemplate = uid={0},ou=people,dc=company,dc=com
myRealm.contextFactory.systemUsername = uid=systemuser,dc=company,dc=com
myRealm.contextFactory.systemPassword = theSystemUserPassword
myRealm.userSearchFilter = (&(objectClass=inetOrgPerson)(uid={0}))
myRealm.contextFactory.environment[ldap.searchBase] = dc=company.com,dc=com
myRealm.contextFactory.environment[ldap.allowedGroups] = group1,group2
myRealm.groupSearchFilter = (&(member={2})(objectClass=posixGroup)(cn={0}))
```

Notes:

- In `myRealm.userDnTemplate` the placeholder `{0}` is replaced with the login name the user enters.
- `myRealm.contextFactory.systemUsername` is a technical user account that must be allowed to list all user DNs and determine their group membership.

- `myRealm.userSearchFilter` and `myRealm.groupSearchFilter` are only used to determine group membership, which takes place after successful user/password authentication.
- In `myRealm.contextFactory.environment[ldap.allowedGroups]` you list all group names separated by commas, without spaces. These group names will be put into the `{0}` placeholder of the `myRealm.groupSearchFilter` when trying to find the LDAP record of a group. The DN of the user, which is determined with `myRealm.userSearchFilter`, is put into the `{2}` placeholder.

Template: ActiveDirectory with group membership checking

```
myRealm = spark.jobserver.auth.LdapGroupRealm
myRealm.contextFactory.url = ldap://ldapserver.company.com
myRealm.userDnTemplate = {0}@COMPANY.COM
myRealm.contextFactory.systemUsername = systemuser@COMPANY.COM
myRealm.contextFactory.systemPassword = theSystemUserPassword
myRealm.userSearchFilter = (&(objectClass=person)(sAMAccountName={0}))
myRealm.contextFactory.environment[ldap.searchBase] = dc=company.com,dc=com
myRealm.contextFactory.environment[ldap.allowedGroups] = group1,group2
myRealm.groupSearchFilter = (&(member={2})(objectClass=group)(cn={0}))
```

Notes:

- In `myRealm.userDnTemplate` the placeholder `{0}` is replaced with the login name the user enters. ActiveDirectory then authenticates against the user record with a matching `sAMAccountName`.
- `myRealm.contextFactory.systemUsername` is a technical user account that must be allowed to list all user DNs and determine their group membership.
- `myRealm.userSearchFilter` and `myRealm.groupSearchFilter` are only used to determine group membership, which takes place after successful user/password authentication.
- In `myRealm.contextFactory.environment[ldap.allowedGroups]` you list all group names separated by commas, without spaces. These group names will be put into the `{0}` placeholder of the `myRealm.groupSearchFilter` when trying to find the LDAP record of a group. The DN of the user, which is determined with `myRealm.userSearchFilter`, is put into the `{2}` placeholder.

Maintenance

Clean up temporary files

It is advisable to restart Spark Job Server occasionally, and clean up its temporary files. Remove either the entire directory or only the jar files under `/tmp/spark-job-server`, or whichever file system locations you have set in `environment.conf`.

Spark Job Server web UI

Point your browser to `http://server:port` to check out the status of the Spark Job Server. The default port is `8090`. Three different tabs provide information about active and completed jobs, contexts and jars.

Retrieving Spark logs

By default, Spark Job Server logs to the following directories:

- `/var/log/spark-job-server/` (0.6.2.3-KNIME for Spark 1.x)
- `/var/log/spark2-job-server/` (0.7.0.3-KNIME for Spark 2.x)

This directory contains the following files:

- `spark-job-server.log` and `spark-job-server.out` which contain the logs of the part of Spark Job Server that runs the REST interface.
- Per created Spark context, a directory `jobserver-<user>~<ctxname><randomnumber>/` will be created. It contains a `spark-job-server.log` and `spark-job-server.out` that the respective `spark-submit` process logs to.

In some situations, it is helpful to obtain the full YARN container logs. These can be obtained using the `yarn logs` shell command or using the means of your Hadoop distribution:

- Cloudera CDH: Monitoring Spark Applications
- Hortonworks HDP: Using the YARN CLI to View Logs for Running Applications

Troubleshooting

Spark Job Server fails to restart

At times, Spark Job Server cannot be restarted when large tables were serialized from KNIME to Spark. It fails with a message similar to `java.io.UTFDataFormatException: encoded`

`string too long: 6653559 bytes`. In that case, it is advisable to delete `/tmp/spark-job-server`, or whichever file system locations you have set in `environment.conf`.

Spark Collaborative Filtering node fails

If the Spark Collaborative Filtering node fails with a `Job canceled because SparkContext was shutdown` exception the cause might be missing native libraries on the cluster. If you find the error message `java.lang.UnsatisfiedLinkError: org.jblas.NativeBlas.dposv` in your Job Server log the native JBlas library is missing on your cluster. To install the missing library execute the following command as root on all cluster nodes:

1. On RHEL-based systems

```
yum install libgfortran
```

1. On Debian-based systems

```
apt-get install libgfortran3
```

For detailed instructions on how to install the missing libraries go to the JBlas Github page. For information about the MLlib dependencies see the Dependencies section of the MLlib Guide.

The issue is described in SPARK-797.

Request to Spark Job Server failed, because the uploaded data exceeded that allowed by the Spark Job Server

Spark Job Server limits how much data can be submitted in a single REST request. For Spark nodes that submit large amounts of data to Spark Job Server, e.g. a large MLlib model, this can result in a request failure with an error as above. This problem can be addressed by adding and adjusting the following section in `environment.conf`:

```
spray.can.server {
      request-chunk-aggregation-limit = 200m
}

spray.can.server.parsing {
      max-content-length = 200m
}
```

Spark job execution failed because no free job slots were available on Spark Job Server

Spark Job Server limits how much jobs can run at the same time within the same context. This limit can be changed by adjusting the following setting in `environment.conf`:

```
spark {
  jobserver {
    max-jobs-per-context = 100
  }
}
```

KNIME extensions

This section describes how to install the client-side *KNIME Extension for Apache Spark* in KNIME Analytics Platform or KNIME Server. The extension provides all the necessary KNIME nodes to create workflows that execute on Apache Spark.

Additionally, you may wish to install the following extensions for added functionality:

- *KNIME Extension for Local Big Data Environments* provides the *Create Local Big Data Environment* node, which creates a fully functional local big data environment including Spark, Hive and HDFS. It allows you to try out the Spark nodes without the need for a cluster.
- *KNIME H2O Sparkling Water Integration* contains nodes to use H2O machine learning on Spark.
- *KNIME Extension for MOJO nodes on Spark* contains nodes for doing predictions with H2O MOJOs on Spark.

Requirements

Required Software: A compatible version of KNIME Analytics Platform or KNIME Server (see Supported versions of KNIME software).

Network Connectivity Requirements: KNIME Extension for Apache Spark (the client) needs to be able to make a HTTP(S) connection to the Livy/Spark Job Server service in the cluster. There are two ways to make this connection:

- **Direct connection (recommended):** Client → Livy/Spark Job Server (Livy default port: 8998 / Spark Job Server default port: 8090).

- Proxied connection: Client → TTP/HTTPS/SOCKS Proxy → Livy/Spark Job Server. Currently, only proxies that do not require any authentication are supported. Note that KNIME does not provide the proxy software itself.

Installation

The extension can be installed via the KNIME Update Manager:

1. Go to **File** > **Install KNIME Extensions**.
2. Open the category **KNIME Big Data Extensions**.
3. Select **KNIME Extension for Apache Spark**.
4. Click on **Next** and follow the subsequent dialog steps to install the extension.

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

Basic usage

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

[spark allnodes] | *spark_allnodes.png*

Figure 1. 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 ways to create a Spark context with KNIME nodes:

- Create Spark Context (Livy) (recommended)
- Create Spark Context (Jobserver) (deprecated)

Create Spark Context (Livy)

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

[create spark context livy] | *create_spark_context_livy.png*

Figure 2. Create Spark Context (Livy) node

Requirements

- **Apache Livy service** Livy needs to be installed as a service in your cluster. Please consult Apache Livy setup 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. Note that KNIME does not provide the proxy software itself.
- **Authentication**: If Livy requires Kerberos authentication, then KNIME Analytics Platform needs to be set up accordingly. Please follow the steps outlined in the Speaking Kerberos with KNIME Big Data Extensions blog post.
- **Remote file system**: 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 are:
    - HDFS, webHDFS and httpFS. 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 and Azure Blob Store, which is recommended when using Spark on Amazon EMR/Azure HDInsight. 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

[create spark context livy tab1] | *create_spark_context_livy_tab1.png*

Figure 3. 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.

[create spark context livy tab2] | *create_spark_context_livy_tab2.png*

Figure 4. 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.

Create Spark Context (Jobserver)

This node connects to Spark Job Server to create a new Spark context.

[create spark context] | *create_spark_context.png*

Its node dialog has two main tabs. The first tab is the Context Settings tab which allows you to specify the following Spark Context settings:

1. **Spark version:** Please choose the Spark version of the Hadoop cluster you are connecting to.
2. **Context name:** A unique name for the Spark context.
3. **Delete objects on dispose:** KNIME workflows with Spark nodes create objects such as DataFrames/RDDs during execution. This setting specifies whether those objects shall be deleted when closing a workflow.

4. **Override Spark settings:** Custom settings for the Spark context, e.g. the amount of memory to allocate per Spark executor. These settings override those from Job Server's `environment.conf`.
5. **Hide context exists warning:** If not enabled the node will show a warning if a Spark Context with the defined name already exists.

[create spark context tab1] | *create_spark_context_tab1.png*

Figure 5. Create Spark Context: Context Settings tab

The second tab is the **Connection Settings** tab which allows you to specify the following connection settings:

1. **Job server URL**: This is the HTTP/HTTPS URL under which the Spark Job Server WebUI can be reached. The default URL is `http://localhost:8090/`.
2. **Credentials:** If you have activated user authentication, you need to enter a username and password here.
3. **Job timeout in seconds/Job check frequency:** These settings specify how long a single Spark job can run before being considered failed, and, respectively, in which intervals the status of a job shall be polled.

[create spark context tab2] | *create_spark_context_tab2.png*

Figure 6. Create Spark Context: Connection Settings tab

Adapting default settings for the Create Spark Context node

The default settings of the Create Spark Context node can be specified via a preference page. The default settings are applied whenever the node is added to a KNIME workflow. To change the default settings, open **File** > **Preferences** > **KNIME** > **Big Data** > **Spark** and adapt them to your environment (see [knime_ext_create_spark_context]).

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.

[simple spark workflow example] | *simple_spark_workflow_example.png*

Figure 7. How to use the Destroy Spark Context node

Proxy settings

If your network requires you to connect to Livy or Spark Job Server 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

[spark workflow] | *spark_workflow.png*

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 example workflows, which are available in KNIME Analytics Platform in the KNIME Explorer View, when connecting to the EXAMPLES server. Under *01_Big_Data → 02_Spark_Executor* you will find a variety of example workflows that demonstrate how to use the Spark nodes.

KNIME AG
Technoparkstrasse 1
8005 Zurich, Switzerland
www.knime.com
info@knime.com