Open for Innovation
KNIME

KNIME Big Data Extensions Admin Guide

KNIME AG, Zurich, Switzerland
Version 4.0 (last updated on 2020-09-02)

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 IT professionals who need to integrate KNIME Analytics Platform with an existing Hadoop/Spark environment.

The steps in this guide are required so that users of KNIME Analytics Platform run Spark workflows. Note that running Spark workflows on KNIME Server requires **additional** steps outlined in Secured Cluster Connection Guide for KNIME Server.

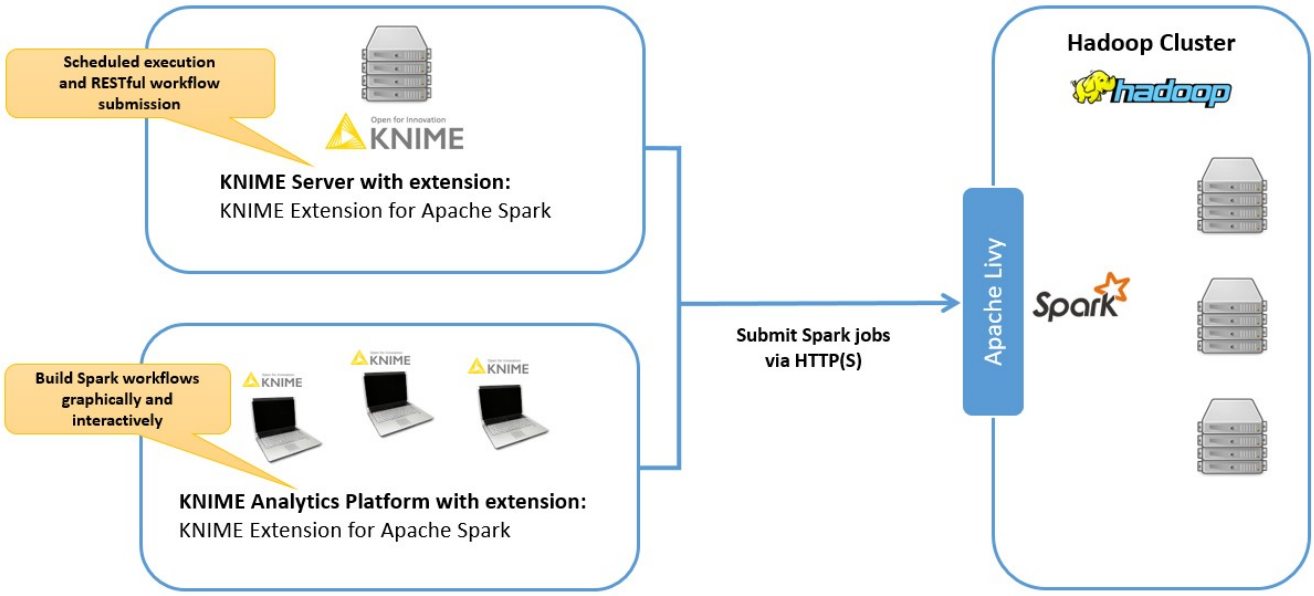

Figure 1. Overall architecture

KNIME Extension for Apache Spark requires a REST service to be installed on an edge/fronted node of the cluster. The REST service must be one of:

- Apache Livy (**recommended**, requires at least Spark 2.2)
- Spark Job Server (deprecated, still supported for Spark 2.1 and older)

Please follow the instructions of this guide to either install Livy (if necessary), or Spark Jobserver.

For new setups it is strongly recommended to use at least Spark 2.2 and Livy. Spark Job Server support is deprecated and will be discontinued in the near future. Spark Jobserver should only be chosen if support for Spark 2.1 or older is required.

Cloudera CDH Compatibility

KNIME Extension for Apache Spark is compatible with

- Spark 1.x as included in CDH 5
- Spark 2.x on CDH 5 as provided by Cloudera CDS
- Spark 2.x as included in CDH 6

Cloudera CDH 5/6 does not include Livy, therefore KNIME provides CSDs/parcels for Livy (see Cloudera CDH), as well as Spark Job Server for setups with Spark 2.1 or older.

Cloudera HDP Compatibility

KNIME Extension for Apache Spark is compatible with

- Spark 1.x and 2.x as included in HDP 2.2 - 2.6
- Spark 2.x as included in HDP 3.0 - 3.1

HDP 2.6.3 and later include compatible versions of Livy. On older versions of HDP, Spark Job Server needs to be installed.

Amazon EMR Compatibility

KNIME Extension for Apache Spark is compatible with all versions of Spark 2 and Livy included in Amazon EMR 5.9 - 5.23. EMR versions 5.24 and newer will be supported in future KNIME releases.

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`.

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.

Prerequisites

- A cluster with CDH 5.8 and newer, or CDH 6.0 and newer
  - *Only On CDH 5*: Spark 2.2 or higher as an add-on service (provided by Cloudera CDS)
- Root shell access (e.g. via SSH) on the machine where Cloudera Manager is installed.
- Full administrative access on the Cloudera Manager WebUI.

Installation steps

In a root shell on the machine where Cloudera Manager is installed:

1. Download a matching CSD from CSDs for Cloudera CDH to `/opt/cloudera/csd/` on the machine, where Cloudera Manager is installed.
2. **Only if Cloudera Manager cannot access the public internet:** Download/copy the matching `.parcel` and `.sha1` file from Parcels for Cloudera CDH to `/opt/cloudera/parcel-repo`.
3. Restart Cloudera Manager from the command line, for example with:

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

In 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.

Cloudera HDP

HDP already includes compatible versions of Apache Livy and Spark 2 (see Cloudera HDP Compatibility). 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 Amazon EMR Compatibility), 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. Spark Jobserver should only be chosen if support for Spark 2.1 or older is required. Using Spark Job Server has the following drawbacks:

- Complex installation procedure
- No more updates or bug fixes will be provided by KNIME
- A growing number of KNIME nodes relies on functionality not present in Spark Jobserver and Spark 2.1 or older

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. hese packages can be downloaded from Spark Jobserver downloads. 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 that matches your Hadoop distribution on the KNIME Extension for Apache Spark product website under **Installation Steps** > **KNIME Analytics Platform X.Y** > **Supplementary download links for the Spark Job Server**.

   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 [sjs_setup_webui]).

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 [sjs_setup_ldap] 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
  }
}
```

Downloads

Apache Livy downloads

CSDs for Cloudera CDH

- CSD for CDH 5
- CSD for CDH 6

Parcels for Cloudera CDH

Download links 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	

Download links 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	

Spark Jobserver downloads

Pick a version of Spark Job Server that matches your Hadoop environment:

- For Cloudera HDP and standard Apache Spark installations:
  - HDP 2.6.5 (Apache Spark 2.3)
  - HDP 2.6.4 (Apache Spark 2.2)
  - HDP 2.6.3 (Apache Spark 2.2)
  - HDP 2.6.x (Apache Spark 2.1)
  - HDP 2.5.x (Apache Spark 2.0)
  - HDP 2.6.x (Apache Spark 1.6)
  - HDP 2.5.x (Apache Spark 1.6)
  - HDP 2.4.x (Apache Spark 1.6)
  - HDP 2.3.4 (Apache Spark 1.5)
  - HDP 2.3.0 (Apache Spark 1.3)
  - HDP 2.2 (Apache Spark 1.2)
- For Cloudera CDH:
  - CDH 5.9 - 5.15 (Apache Spark 2.3)
  - CDH 5.8 - 5.15 (Apache Spark 2.2)
  - CDH 5.7 - 5.15 (Apache Spark 2.1)
  - CDH 5.7 - 5.11 (Apache Spark 2.0)
  - CDH 5.13 (Apache Spark 1.6)
  - CDH 5.12 (Apache Spark 1.6)
  - CDH 5.11 (Apache Spark 1.6)
  - CDH 5.10 (Apache Spark 1.6)
  - CDH 5.9 (Apache Spark 1.6)
  - CDH 5.8 (Apache Spark 1.6)
  - CDH 5.7 (Apache Spark 1.6)
  - CDH 5.6 (Apache Spark 1.5)
  - CDH 5.5 (Apache Spark 1.5)
  - CDH 5.4 (Apache Spark 1.3)
  - CDH 5.3 (Apache Spark 1.2)

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