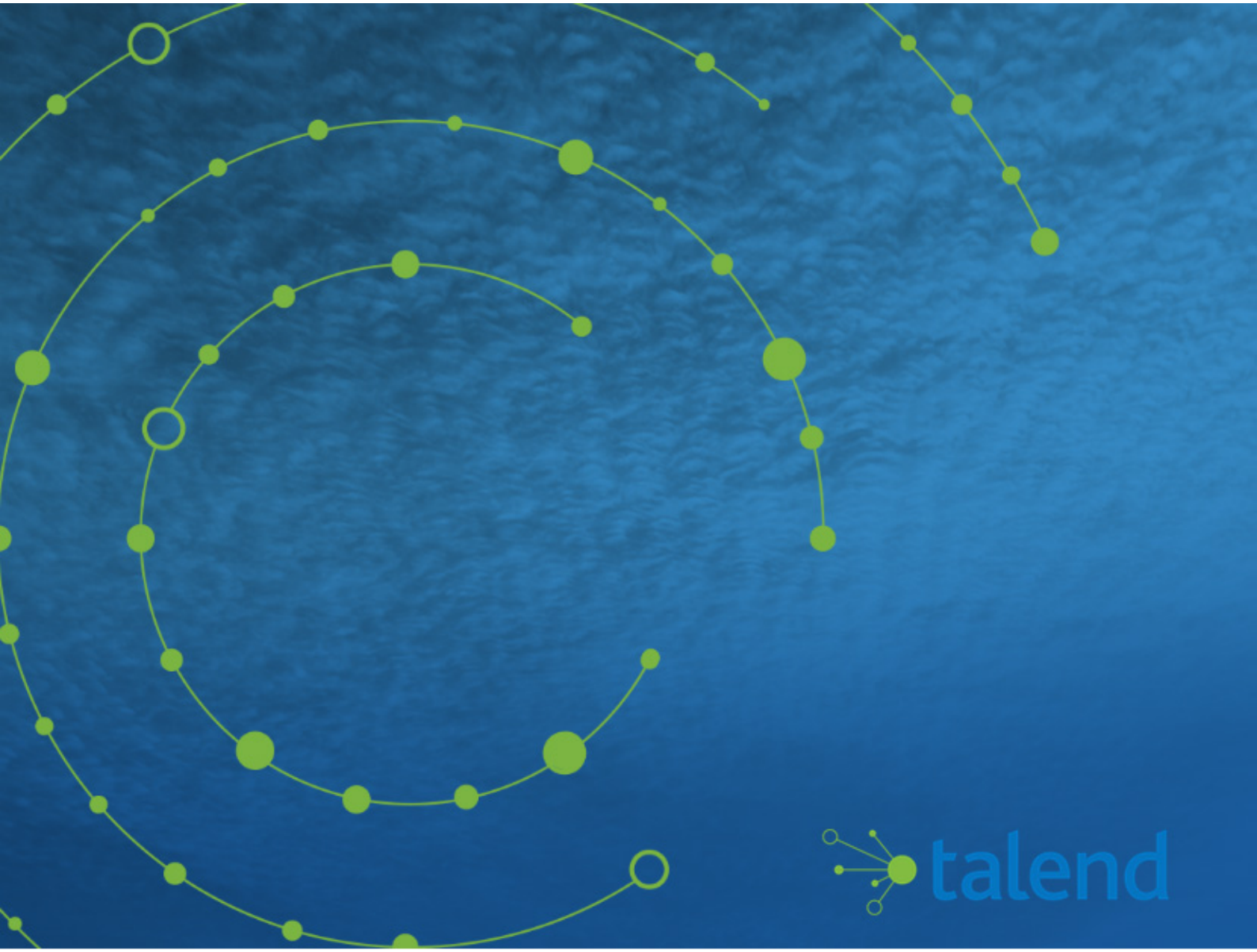


Big Data Job Examples

7.0.1

Contents

Copyleft

Adapted for 7.0.1. Supersedes previous releases.

Publication date: April 13, 2018

Notices

License Agreement

This product includes software developed at AOP Alliance (Java/J2EE AOP standards), ASM, Amazon, AntlR, Apache ActiveMQ, Apache Ant, Apache Avro, Apache Axiom, Apache Axis, Apache Axis 2, Apache Batik, Apache CXF, Apache Cassandra, Apache Chemistry, Apache Common Http Client, Apache Common Http Core, Apache Commons, Apache Commons Bcel, Apache Commons JxPath, Apache Commons Lang, Apache Datafu, Apache Derby Database Engine and Embedded JDBC Driver, Apache Geronimo, Apache HCatalog, Apache Hadoop, Apache Hbase, Apache Hive, Apache HttpClient, Apache HttpComponents Client, Apache JAMES, Apache Log4j, Apache Lucene Core, Apache Neethi, Apache Oozie, Apache POI, Apache Parquet, Apache Pig, Apache PiggyBank, Apache ServiceMix, Apache Sqoop, Apache Thrift, Apache Tomcat, Apache Velocity, Apache WSS4J, Apache WebServices Common Utilities, Apache Xml-RPC, Apache Zookeeper, Box Java SDK (V2), CSV Tools, Cloudera HTrace, ConcurrentLinkedHashMap for Java, Couchbase Client, DataNucleus, DataStax Java Driver for Apache Cassandra, Ehcache, Ezmorph, Ganymed SSH-2 for Java, Google APIs Client Library for Java, Google Gson, Groovy, Guava: Google Core Libraries for Java, H2 Embedded Database and JDBC Driver, Hector: A high level Java client for Apache Cassandra, Hibernate BeanValidation API, Hibernate Validator, HighScale Lib, HsqlDB, Ini4j, JClouds, JDO-API, JLine, JSON, JSR 305: Annotations for Software Defect Detection in Java, JUnit, Jackson Java JSON-processor, Java API for RESTful Services, Java Agent for Memory Measurements, Jaxb, Jaxen, JetS3T, Jettison, Jetty, Joda-Time, Json Simple, LZ4: Extremely Fast Compression algorithm, LightCouch, MetaStuff, Metrics API, Metrics Reporter Config, Microsoft Azure SDK for Java, Mondrian, MongoDB Java Driver, Netty, Ning Compression codec for LZF encoding, OpenSAML, Paraccel JDBC Driver, Parboiled, PostgreSQL JDBC Driver, Protocol Buffers - Google's data interchange format, Resty: A simple HTTP REST client for Java, Riak Client, Rocoto, SDSU Java Library, SL4J: Simple Logging Facade for Java, SQLite JDBC Driver, Scala Lang, Simple API for CSS, Snappy for Java a fast compressor/decompresser, SpyMemCached, SshJ, StAX API, StAXON - JSON via StAX, Super SCV, The Castor Project, The Legion of the Bouncy Castle, Twitter4J, Uuid, W3C, Windows Azure Storage libraries for Java, Woden, Woodstox: High-performance XML processor, Xalan-J, Xerces2, XmlBeans, XmlSchema Core, Xmlsec - Apache Santuario, YAML parser and emitter for Java, Zip4J,

atinject, dropbox-sdk-java: Java library for the Dropbox Core API, google-guice. Licensed under their respective license.

Gathering Web traffic information using Hadoop

To drive a focused marketing campaign based on habits or profiles of your customers or users, you need to be able to fetch data based on their habits or behavior on your website to be able to create user profiles and send them the right advertisements, for example.

The `ApacheWebLog` folder of the Big Data demo project that comes with your Talend Studio provides an example of finding out users having visited a website most often, by sorting out their IP addresses from a huge number of records in an access log file for an Apache HTTP server to enable further analysis on user behavior on the website. This section describes the procedures for creating and configuring Jobs that will implement this example. For more information about the Big Data demo project, see the Getting Started Guide.

Before discovering this example and creating the Jobs, you should have:

- Imported the demo project, and obtained the input access log file used in this example by executing the Job `GenerateWebLogFile` provided with the demo project.
- Installed and started Hortonworks Sandbox virtual appliance that the demo project is designed to work on, as described in the Getting Started Guide.
- An IP to host name mapping entry has been added in the `hosts` file to resolve the host name `sandbox`.

In this example, certain **Talend** Big Data components are used to leverage the advantage of the Hadoop open source platform for handling big data. In this scenario we use six Jobs:

- The first Job sets up an HCatalog database, table and partition in HDFS
- The second Job uploads the access log file to be analyzed to the HDFS file system.
- The third Job connects to the HCatalog database and displays the content of the uploaded file on the console.
- The fourth Job parses the uploaded access log file, including removing any records with a "404" error, counting the code occurrences in successful service calls to the website, sorting the result data and saving it in the HDFS file system.
- The fifth Jobs parse the uploaded access log file, including removing any records with a "404" error, counting the IP address occurrences in successful service calls to the website, sorting the result data and saving it in the HDFS file system.
- The last Job reads the result data from HDFS and displays the IP addresses with successful service calls and their number of visits to the website on the standard system console.

Translating the scenario into Jobs

This section describes how to set up connection metadata to be used in the example Jobs, and how to create, configure, and execute the Jobs to get the expected result of this scenario.

Setting up connection metadata to be used in the Jobs

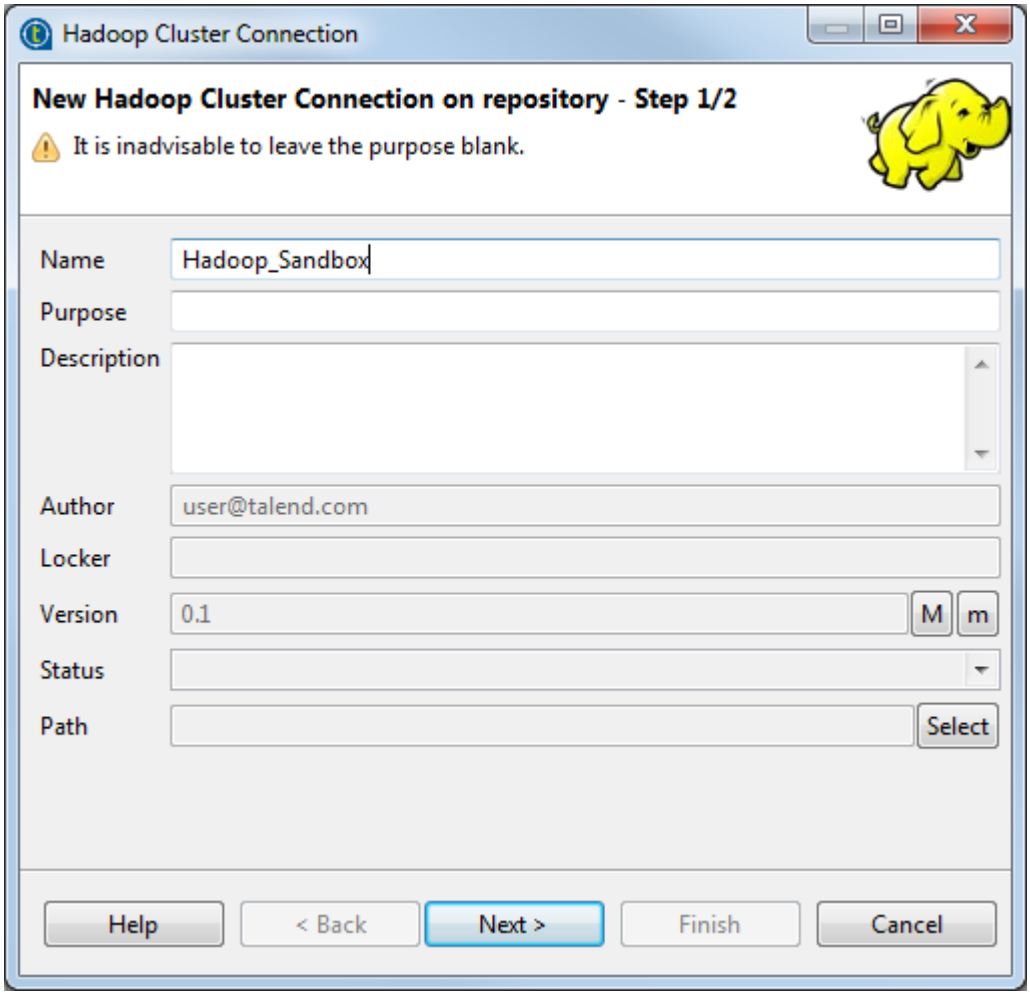
In this scenario, an HDFS connection and an HCatalog connection are repeatedly used in different Jobs. To simplify component configurations, we centralize those connections under a Hadoop cluster connection in the **Repository** view for easy reuse.

These centralized metadata items can be used to set up connection details in different components and Jobs. These connections do not have table schemas defined along with them; therefore, we will create generic schemas separately later on when configuring the example Jobs.

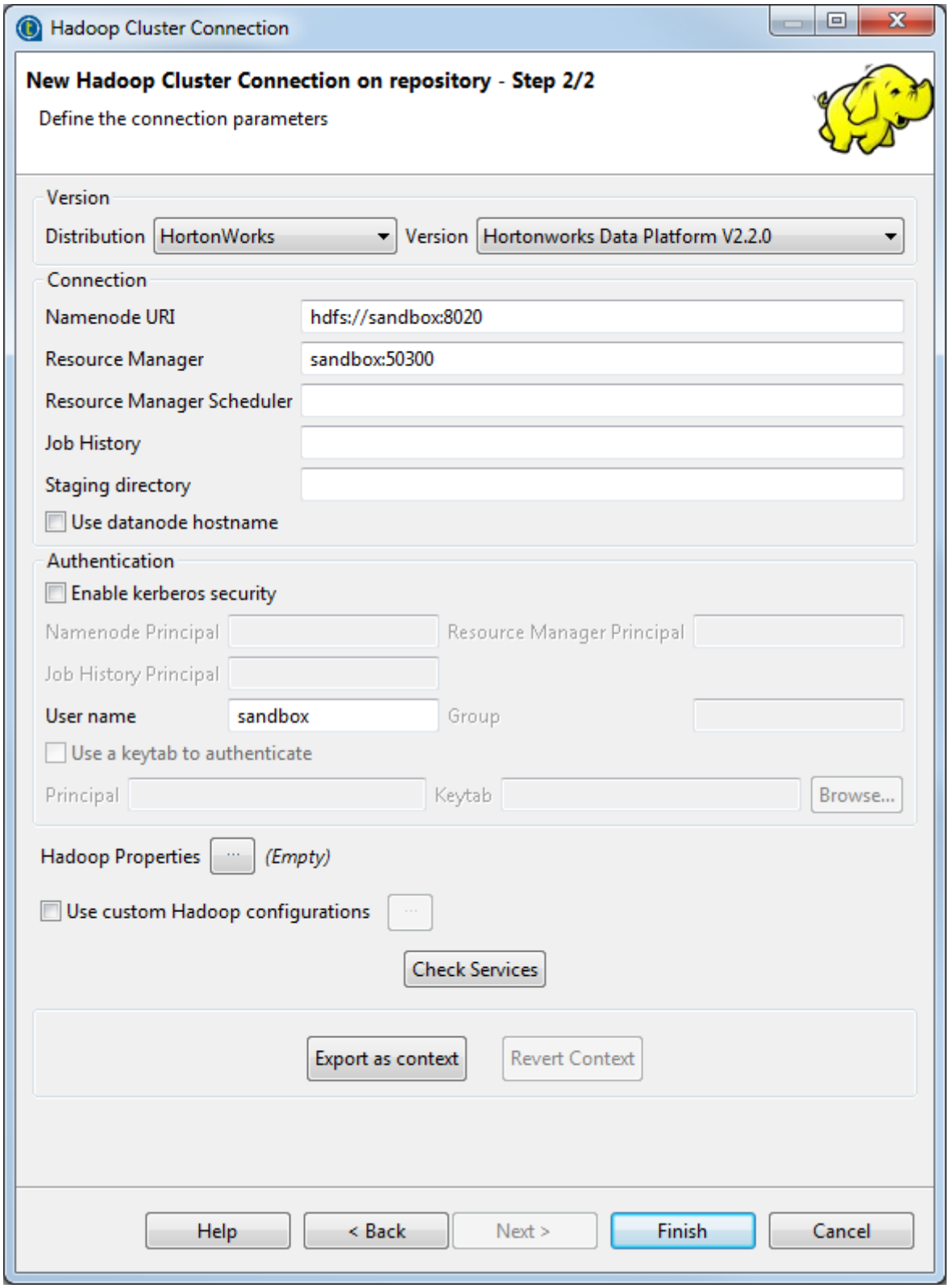
Setting up a Hadoop cluster connection

Procedure

1. Right-click **Hadoop cluster** under the **Metadata** node in the **Repository** tree view, and select **Create Hadoop** cluster from the contextual menu to open the connection setup wizard. Give the cluster connection a name, `Hadoop_Sandbox` in this example, and click **Next**.



2. Configure the Hadoop cluster connection:
   a) Select a Hadoop distribution and its version.
   b) Specify the NameNode URI and the Resource Manager. In this example, we use the host name `sandbox`, which is supposed to have been mapped to the IP address assigned to the Sandbox virtual machine, for both the NameNode and Resource Manager and the default ports, `8020` and `50300` respectively.
   c) Specify a user name for Hadoop authentication, `sandbox` in this example.



3. Click **Finish**.

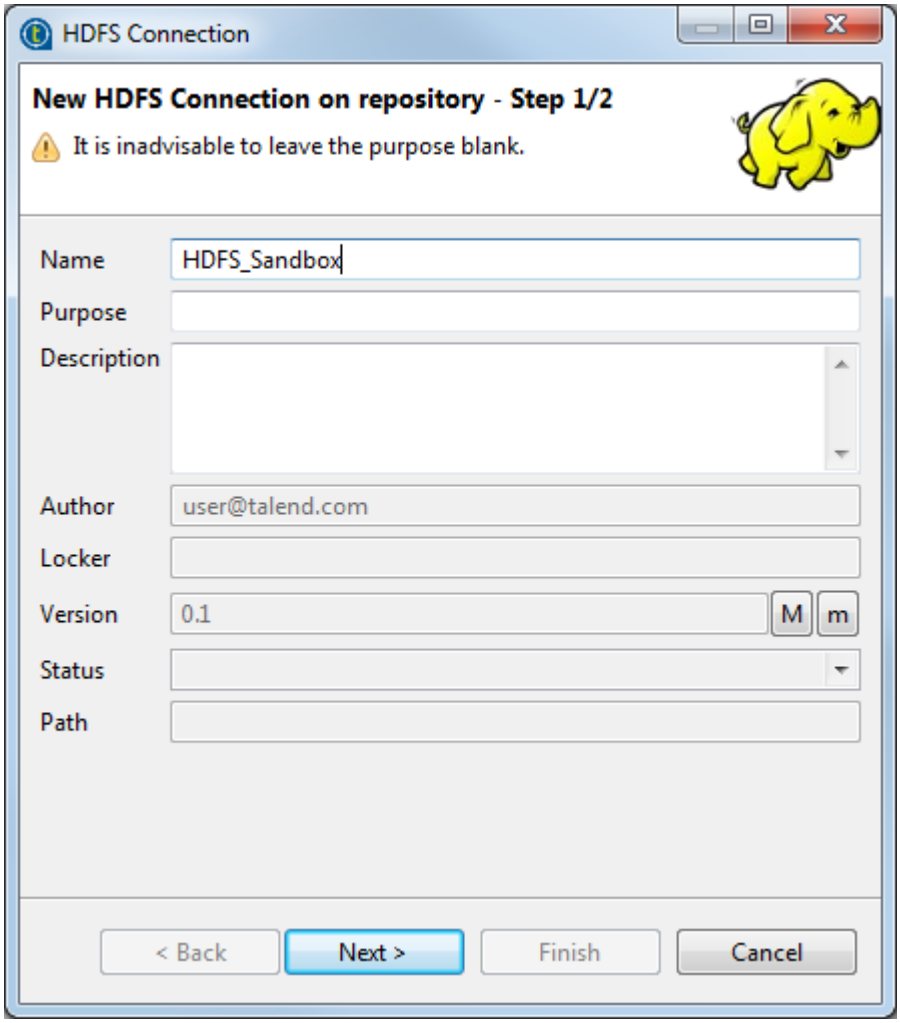
Results

The Hadoop cluster connection appears under the **Hadoop Cluster** node in the **Repository** view.

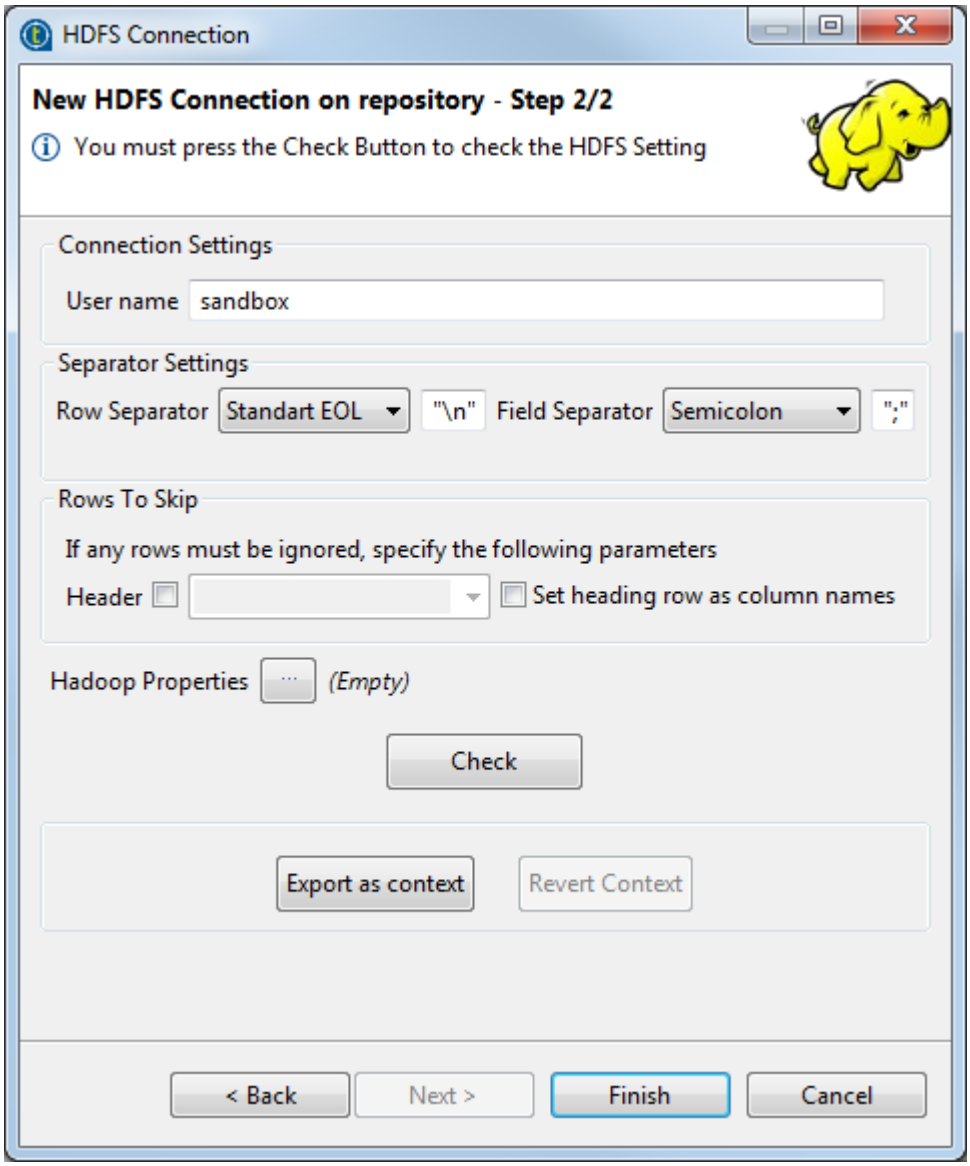
Setting up an HDFS connection

Procedure

1. Right-click the Hadoop cluster connection you just created, and select **Create HDFS** from the contextual menu to open the connection setup wizard. Give the HDFS connection a name, `HDFS_Sandbox` in this example, and click **Next**.



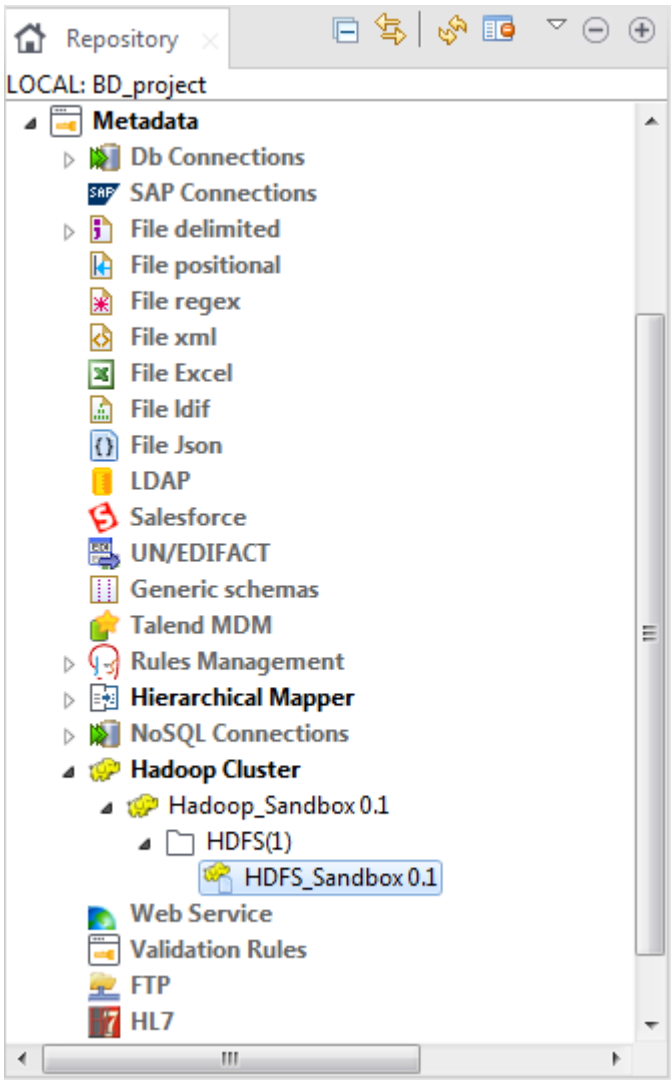
2. Customize the HDFS connection settings if needed and check the connection. As the example Jobs work with all the suggested settings, simply click **Check** to verify the connection.



3. Click **Finish**.

Results

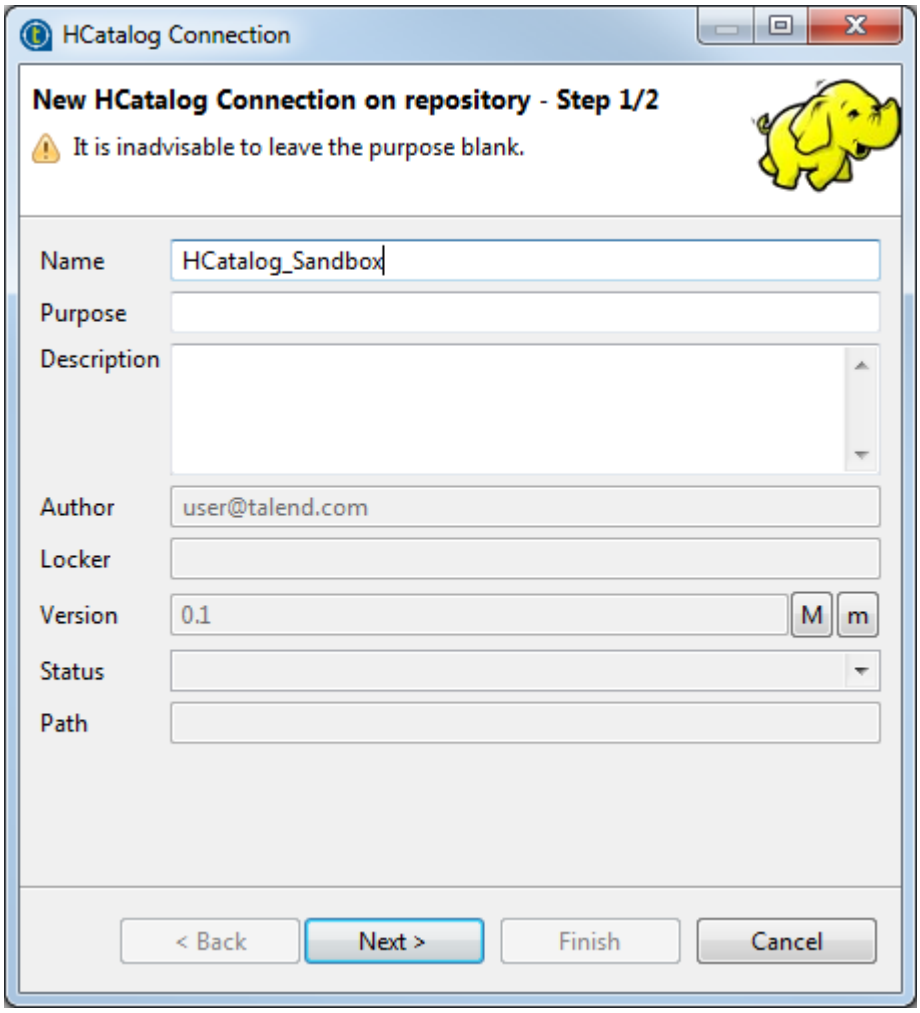
The HDFS connection appears under your Hadoop cluster connection.



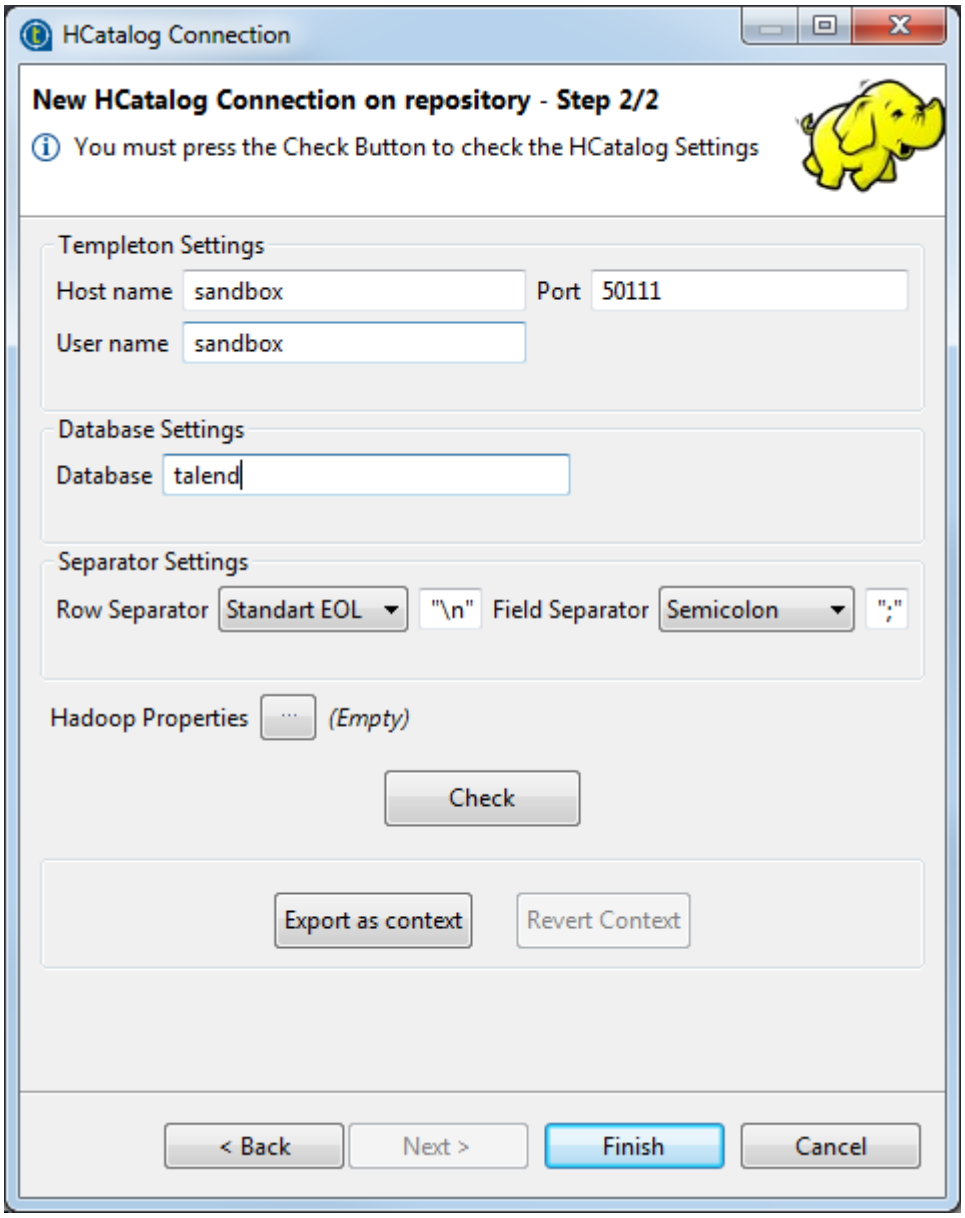
Setting up an HCatalog connection

Procedure

1. Right-click the Hadoop cluster connection you just created, and select **Create HCatalog** from the contextual menu to open the connection setup wizard. Give the HCatalog connection a name, `HCatalog_Sandbox` in this example, and click **Next**.



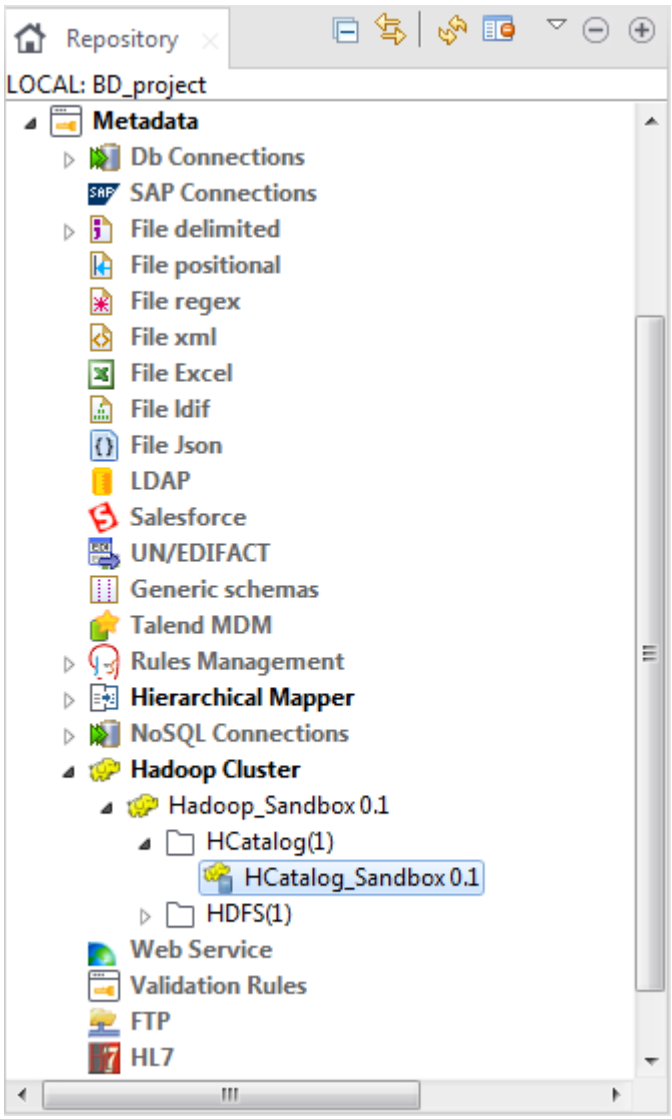
2. Enter the name of database you will use in the **Database** field, `talend` in this example, and click **Check** to verify the connection.



3. Click **Finish**.

Results

The HCatalog connection appears under your Hadoop cluster connection.



Creating the example Jobs

In this section, we will create six Jobs that will implement the `ApacheWebLog` example of the demo Job.

Create the first Job

Follow these steps to create the first Job, which will set up an HCatalog database to manage the access log file to be analyzed.

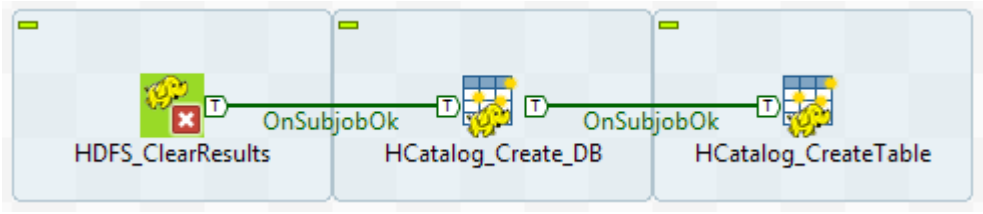
Procedure

1. In the **Repository** tree view, right-click **Job Designs** and select **Create folder** to create a new folder to group the Jobs that you will create.
2. Right-click the folder you just created, and select **Create job** to create your first Job. Name it `A_HCatalog_Create` to identify its role and execution order among the example Jobs.

   You can also provide a short description for your Job, which will appear as a tooltip when you move your mouse over the Job.
3. Drop a **tHDFSDelete** and two **tHCatalogOperation** components from the **Palette** onto the design workspace.

4. Connect the three components using **Trigger** > **On Subjob Ok** connections.

   The HDFS subjob will be used to remove any previous results of this demo example, if any, to prevent possible errors in Job execution, and the two HCatalog subjobs will be used to create an HCatalog database and set up an HCatalog table and partition in the created HCatalog table, respectively.
5. Label these components to better identify their functionality.



Create the second Job

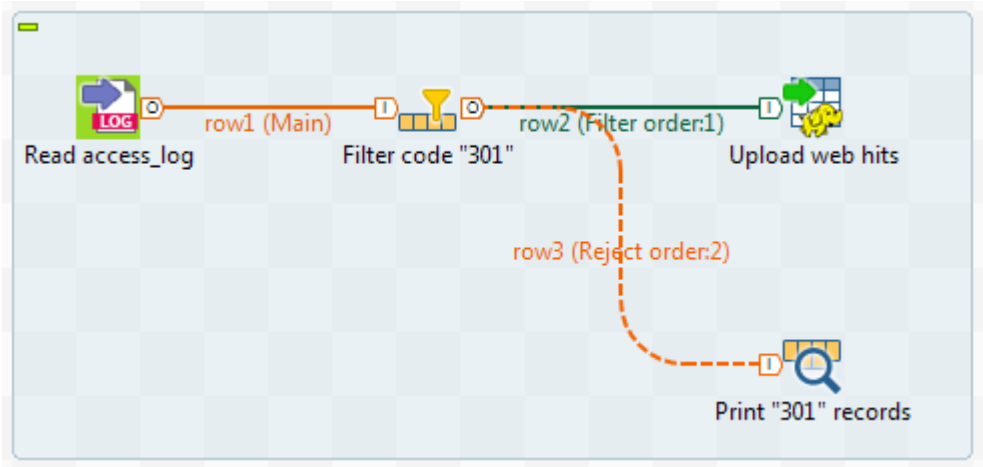
Follow these steps to create the second Job, which will upload the access log file to the HCatalog:

Procedure

1. Create a new Job and name it `B_HCatalog_Load` to identify its role and execution order among the example Jobs.
2. From the **Palette**, drop a **tApacheLogInput**, a **tFilterRow**, a **tHCatalogOutput**, and a **tLogRow** component onto the design workspace.
3. Connect the **tApacheLogInput** component to the **tFilterRow** component using a **Row** > **Main** connection, and then connect the **tFilterRow** component to the **tHCatalogOutput** component using a **Row** > **Filter** connection.

   This data flow will load the log file to be analyzed to the HCatalog database, with any records having the error code of "301" removed.
4. Connect the **tFilterRow** component to the **tLogRow** component using a **Row** > **Reject** connection.

   This flow will print the records with the error code of "301" on the console.
5. Label these components to better identify their functionality.

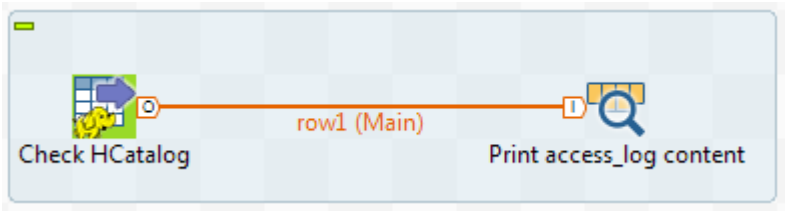


Create the third Job

Follow these steps to create the third Job, which will display the content of the uploaded file:

Procedure

1. Create a new Job and name it `C_HCatalog_Read` to identify its role and execution order among the example Jobs.
2. Drop a **tHCatalogInput** component and a **tLogRow** component from the **Palette** onto the design workspace, and link them using a **Row** > **Main** connection.
3. Label the components to better identify their functionality.

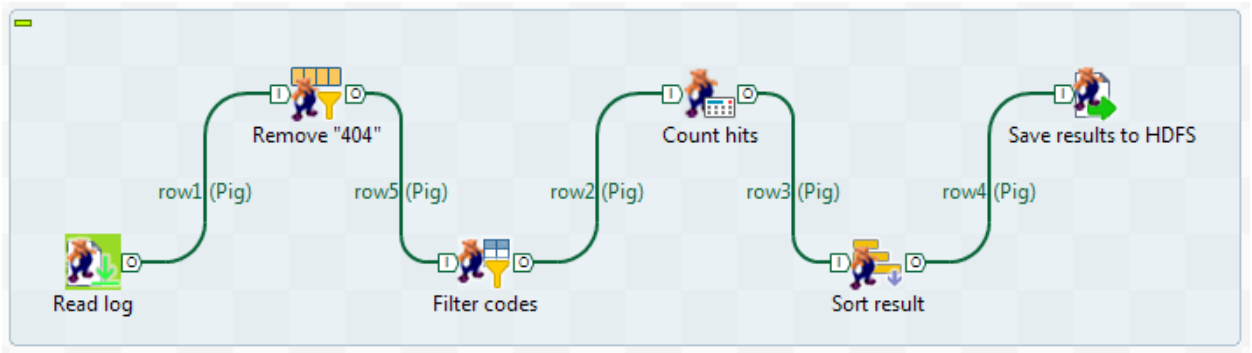


Create the fourth Job

Follow these steps to create the fourth Job, which will analyze the uploaded log file to get the code occurrences in successful calls to the website.

Procedure

1. Create a new Job and name it **D_Pig_Count_Codes** to identify its role and execution order among the example Jobs.
2. Drop the following components from the **Palette** to the design workspace:
   - a **tPigLoad**, to load the data to be analyzed,
   - a **tPigFilterRow**, to remove records with the '404' error from the input flow,
   - a **tPigFilterColumns**, to select the columns you want to include in the result data,
   - a **tPigAggregate**, to count the number of visits to the website,
   - a **tPigSort**, to sort the result data, and
   - a **tPigStoreResult**, to save the result to HDFS.
3. Connect these components using **Row** > **Pig Combine** connections to form a Pig chain, and label them to better identify their functionality.

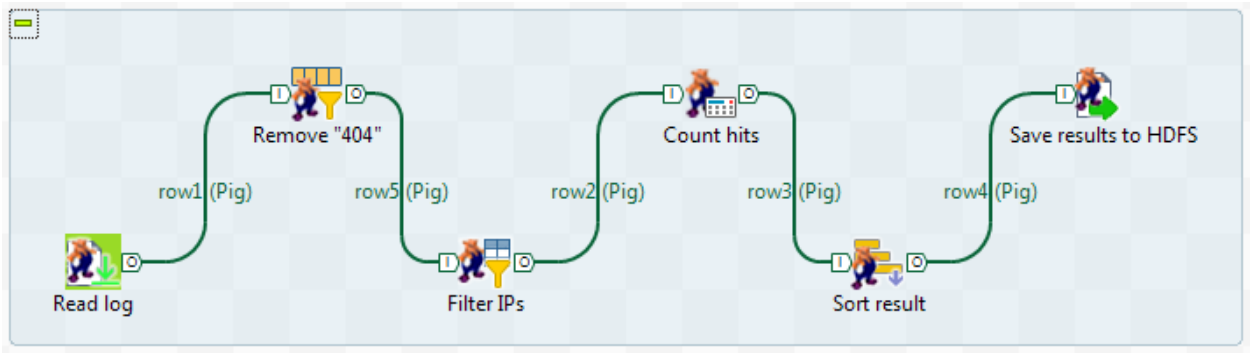


Create the fifth Job

Follow these steps to create the fift Job, which will analyze the uploaded log file to get the IP occurrences of successful service calls to the website.

Procedure

1. Right-click the previous Job in the **Repository** tree view and select **Duplicate**.
2. In the dialog box that appears, name the Job `E_Pig_Count_IPs` to identify its role and execution order among the example Jobs.
3. Change the label of the **tPigFilterColumns** component to identify its role in the Job.

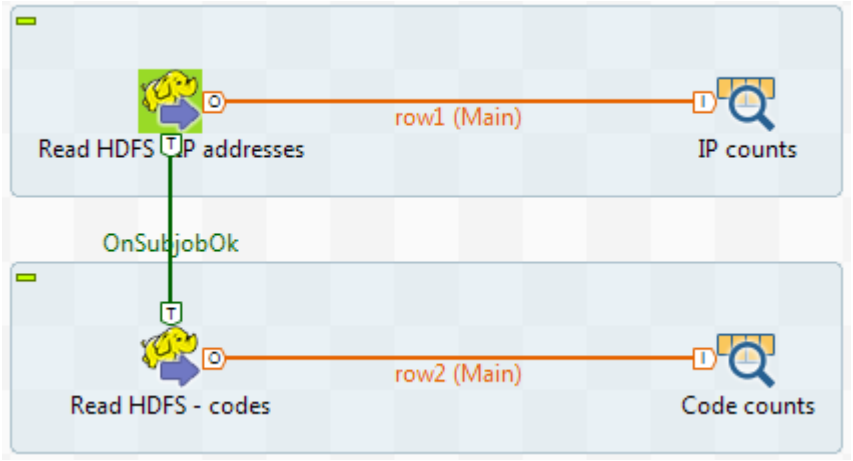


Create the sixth Job

Follow these steps to create the last Job, which will display the results of access log analysis.

Procedure

1. Create a new Job and name it `F_Read_Results` to identify its role and execution order among the example Jobs.
2. From the **Palette**, drop two **tHDFSInput** components and two **tLogRow** components onto the design workspace.
3. Link the first **tHDFSInput** to the first **tLogRow**, and the second **tHDFSInput** to the second **tLogRow** using **Row** > **Main** connections.
4. Link the first **tHDFSInput** to the second **tHDFSInput** using a **Trigger** > **OnSubjobOk** connection.
5. Label the components to better identify their functionality.



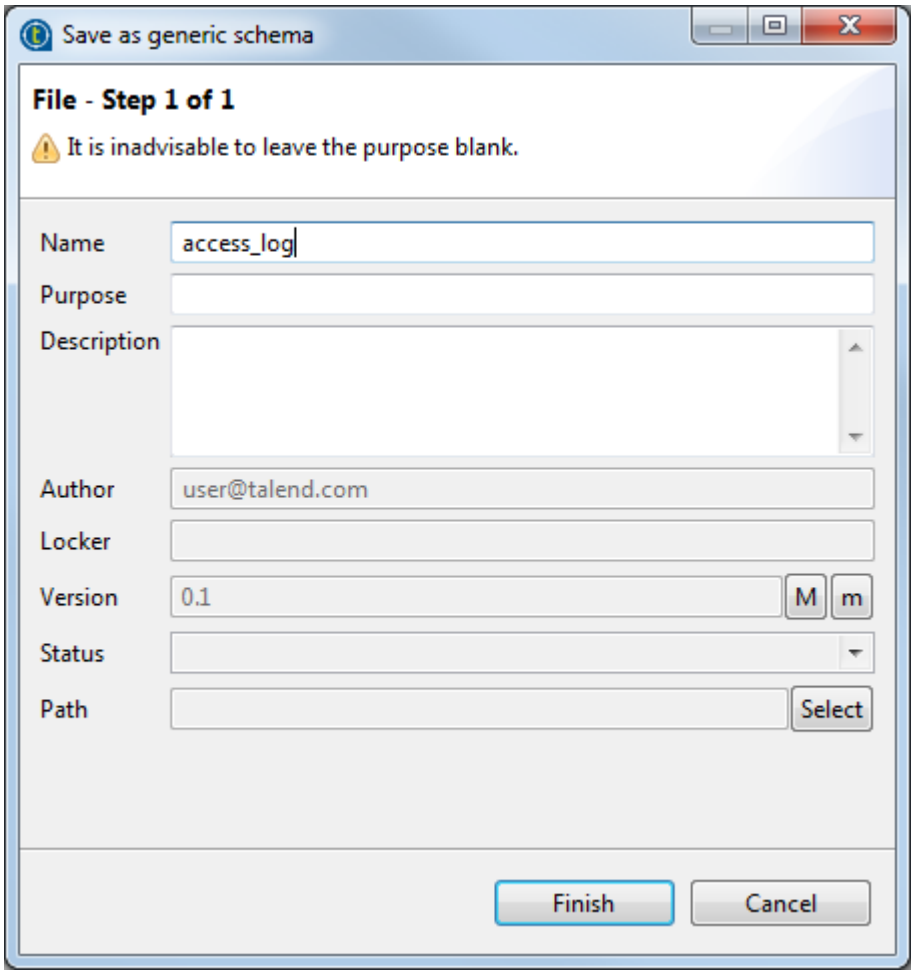
Centralize the schema for the access log file for reuse in Job configurations

To handle the access log file to be analyzed on the Hadoop system, you needed to define an appropriate schema in the relevant components.

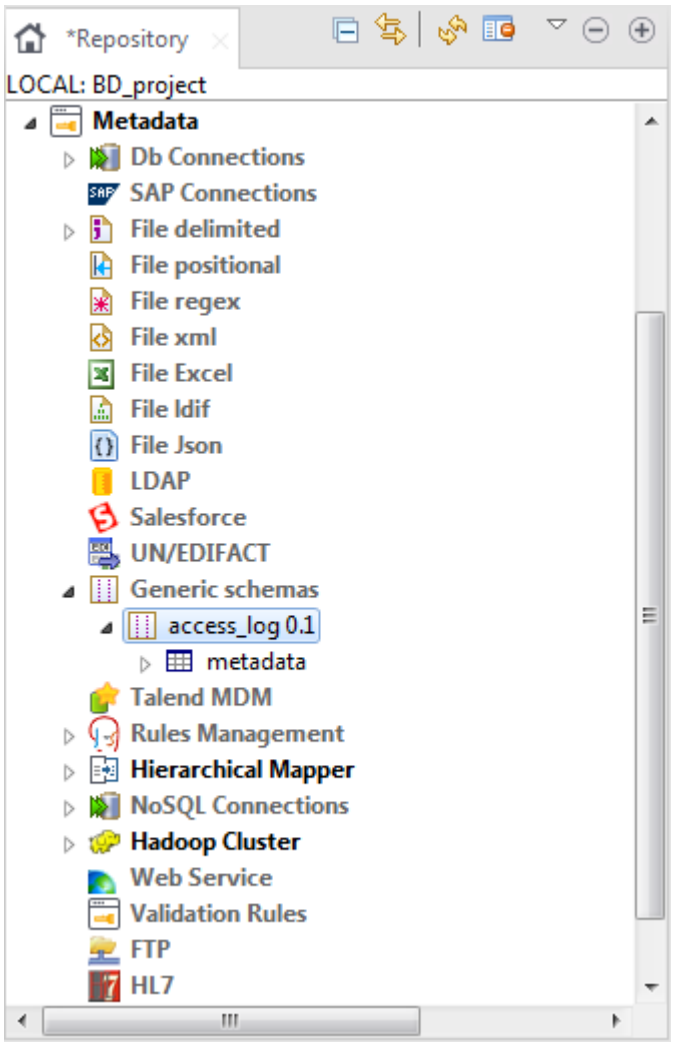
To simplify the configuration, before we start to configure the Jobs, we can save the read-only schema of the **tApacheLogInput** component as a generic schema that can be reused across Jobs.

Procedure

1. In the Job `B_HCatalog_Read`, double-click the **tApacheLogInput** component to open its **Basic settings** view.
2. Click the **[...]** button next to the **Edit schema** to open the **[Schema]** dialog box.
3. Click the  button to open the **[Select folder]** dialog box.
4. In this example we have not created any folder under the **Generic schemas** node, so simply click **OK** to close the dialog box and open the generic schema setup wizard.
5. Give your generic schema a name, `access_log` in this example, and click **Finish** to close the wizard and save the schema.



6. Click **OK** to close the **[Schema]** dialog box. Now the generic schema appears under the **Generic schemas** node of the **Repository** view and is ready for use where it is needed in your Job configurations.



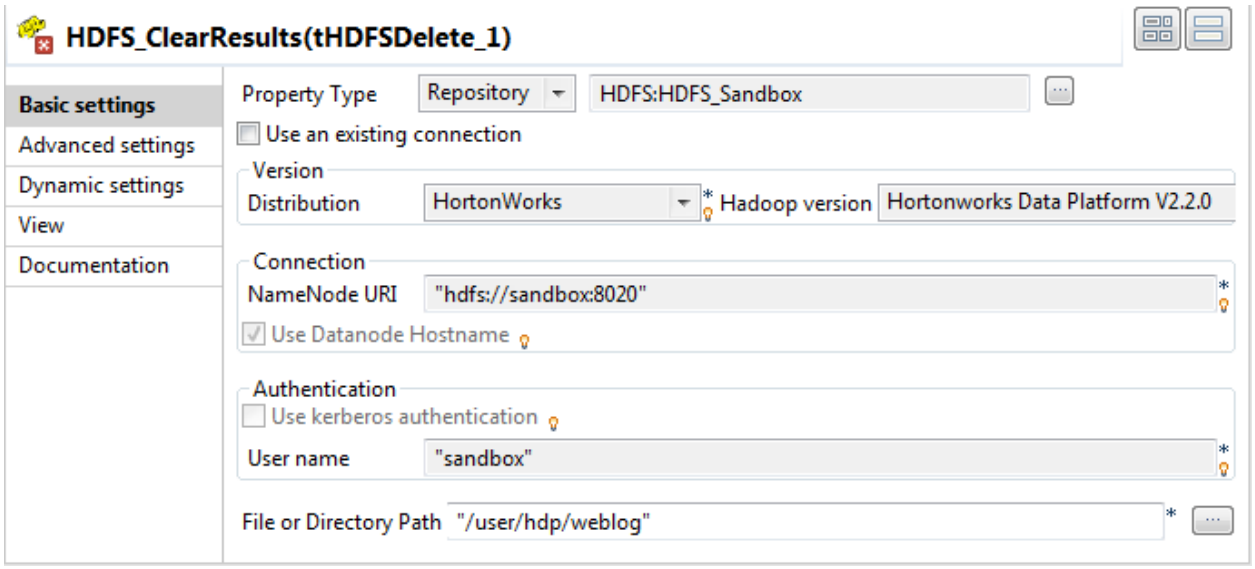
Configuring the first Job

In this step, we will configure the first Job, `A_HCatalog_Create`, to set up the HCatalog system for processing the access log file.

Set up an HCatalog database

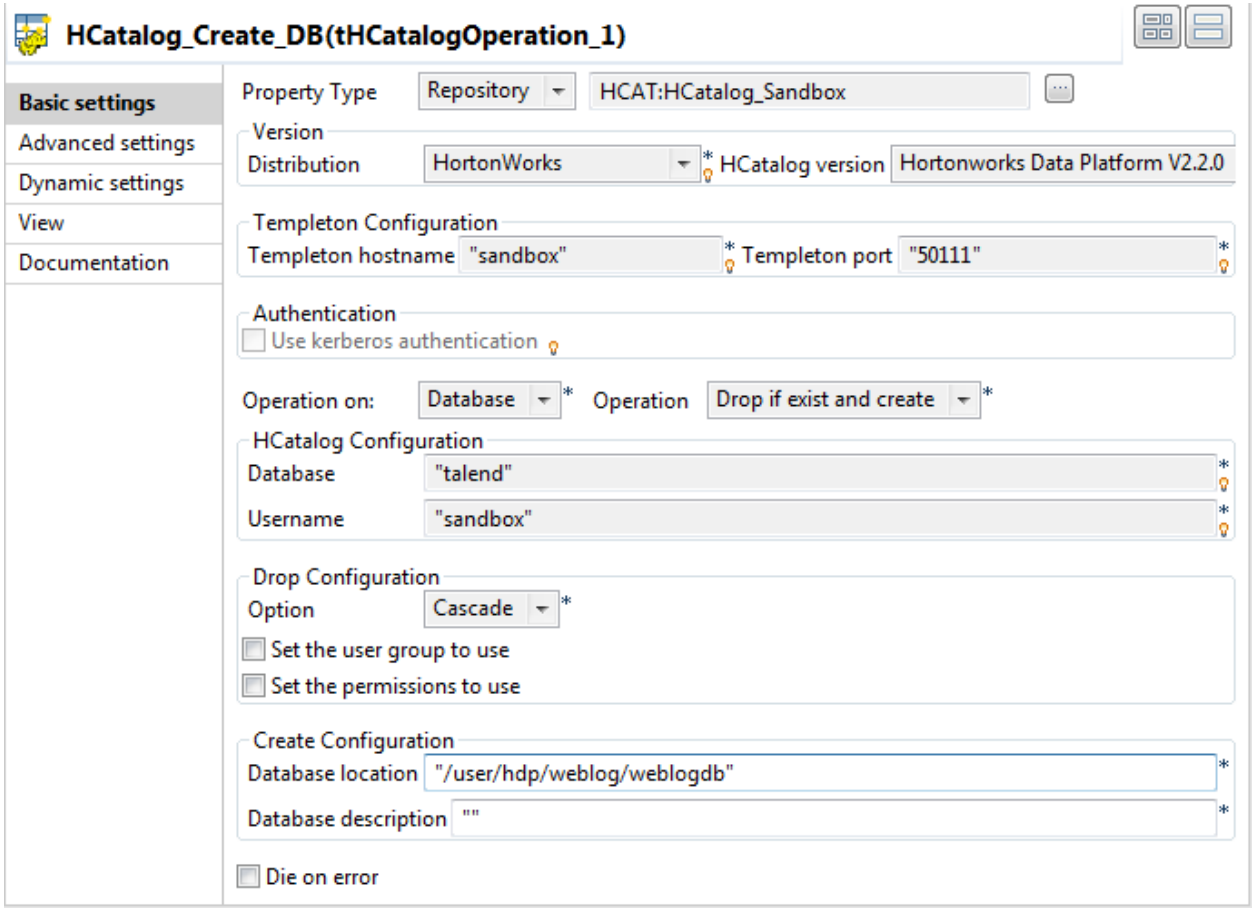
Procedure

1. Double-click the **tHDFSDelete** component, which is labelled `HDFS_ClearResults` in this example, to open its **Basic settings** view on the **Component** tab.



2. Click the **Property Type** list box and select **Repository**, and then click the **[...]** button to open the **[Repository Content]** dialog box to use a centralized HDFS connection.
3. Select the HDFS connection defined for connecting to the HDFS system and click **OK**.

   All the connection details are automatically filled in the respective fields.
4. In the **File or Directory Path** field, specify the directory where the access log file will be stored on the HDFS, `/user/hdp/weblog` in this example.
5. Double-click the first **tHCatalogOperation** component, which is labelled `HCatalog_Create_DB` in this example, to open its **Basic settings** view on the **Component** tab.

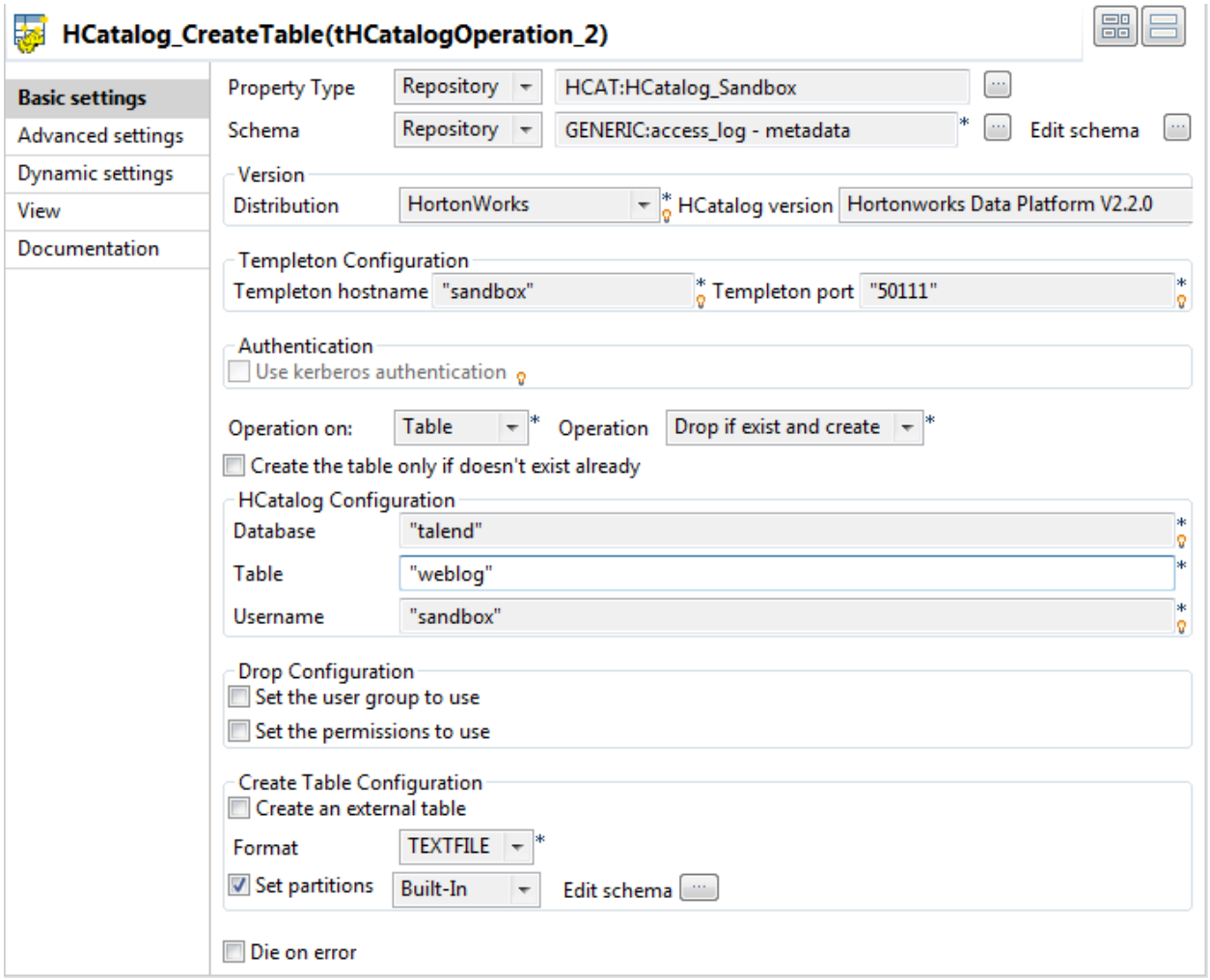


6. Click the **Property Type** list box and select **Repository**, and then click the **[...]** button to open the **[Repository Content]** dialog box to use a centralized HCatalog connection.
7. Select the HCatalog connection defined for connecting to the HCatalog database and click **OK**. All the connection details are automatically filled in the respective fields.
8. From the **Operation on** list, select **Database**; from the **Operation** list, select **Drop if exist and create**.
9. In the **Option** list of the **Drop configuration** area, select **Cascade**.
10. In the **Database location** field, enter the location for the database file is to be created in HDFS, `/user/hdp/weblog/weblogdb` in this example.

Set up an HCatalog table and partition

Procedure

1. Double-click the second **tHCatalogOperation** component, labelled `HCatalog_CreateTable` in this example, to open its **Basic settings** view on the **Component** tab.



2. Define the same HCatalog connection details using the same procedure as for the first **tHCatalogOperation** component.
3. Click the **Schema** list box and select **Repository**, then click the **[...]** button next to the field that appears to open the **[Repository Content]** dialog box, expand **Metadata** > **Generic schemas** > **access_log** and select schema. Click **OK** to confirm your choice and close the dialog box. The generic schema of `access_log` is automatically applied to the component.

Alternatively, you can directly select the generic schema of `access_log` from the **Repository** tree view and then drag and drop it onto this component to apply the schema.

4. From the **Operation on** list, select **Table**; from the **Operation** list, select **Drop if exist and create**.
5. In the **Table** field, enter a name for the table to be created, `weblog` in this example.
6. Select the **Set partitions** check box and click the **[...]** button next to **Edit schema** to set a partition and partition schema.

   The partition schema must not contain any column name defined in the table schema. In this example, the partition schema column is named `ipaddresses`.
7. Upon completion of the component settings, press **Ctrl+S** to save your Job configurations.

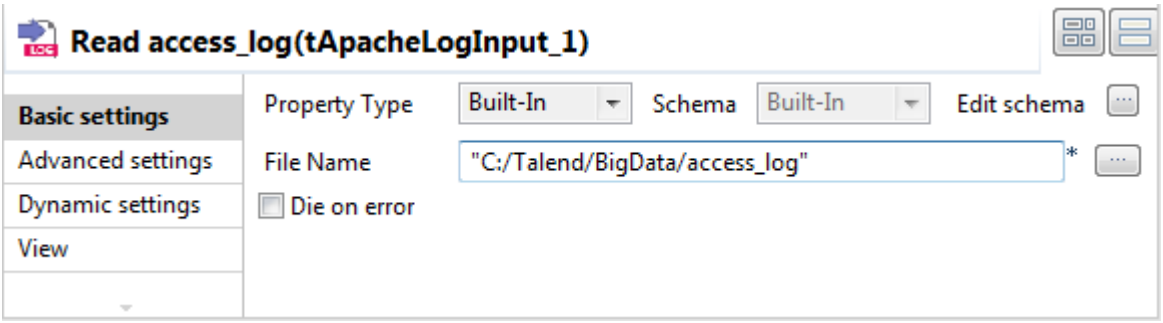
Upload the access log file to HCatalog

In this step, we will configure the second Job, `B_HCatalog_Load`, to upload the access log file to the Hadoop system.

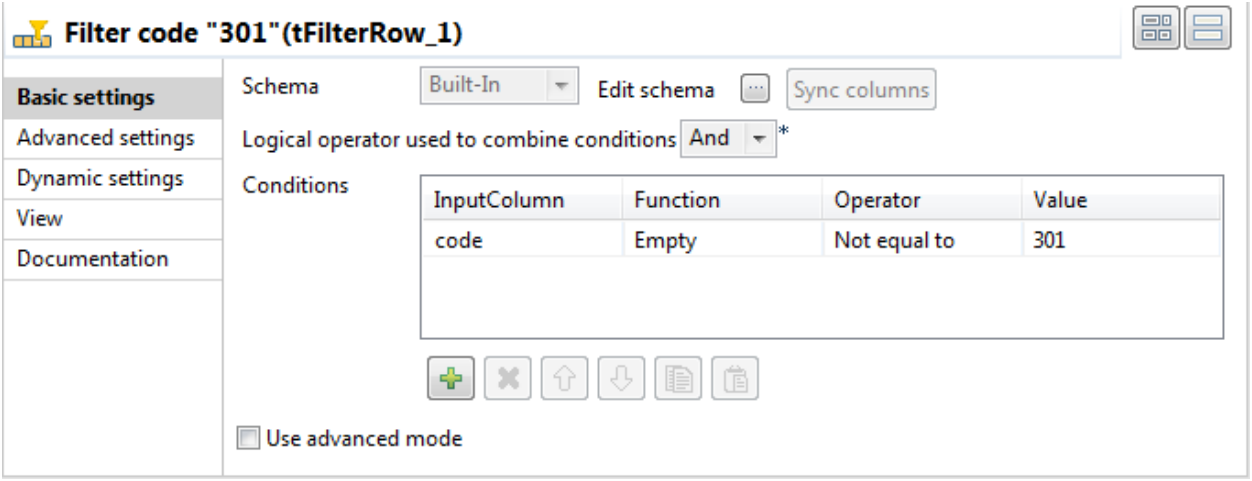
Procedure

1. Double-click the **tApacheLogInput** component to open its **Basic settings** view, and specify the path to the access log file to be uploaded in the **File Name** field.

   In this example, we store the log file `access_log` in the directory `C:/Talend/BigData`.



2. Double-click the **tFilterRow** component to open its **Basic settings** view.



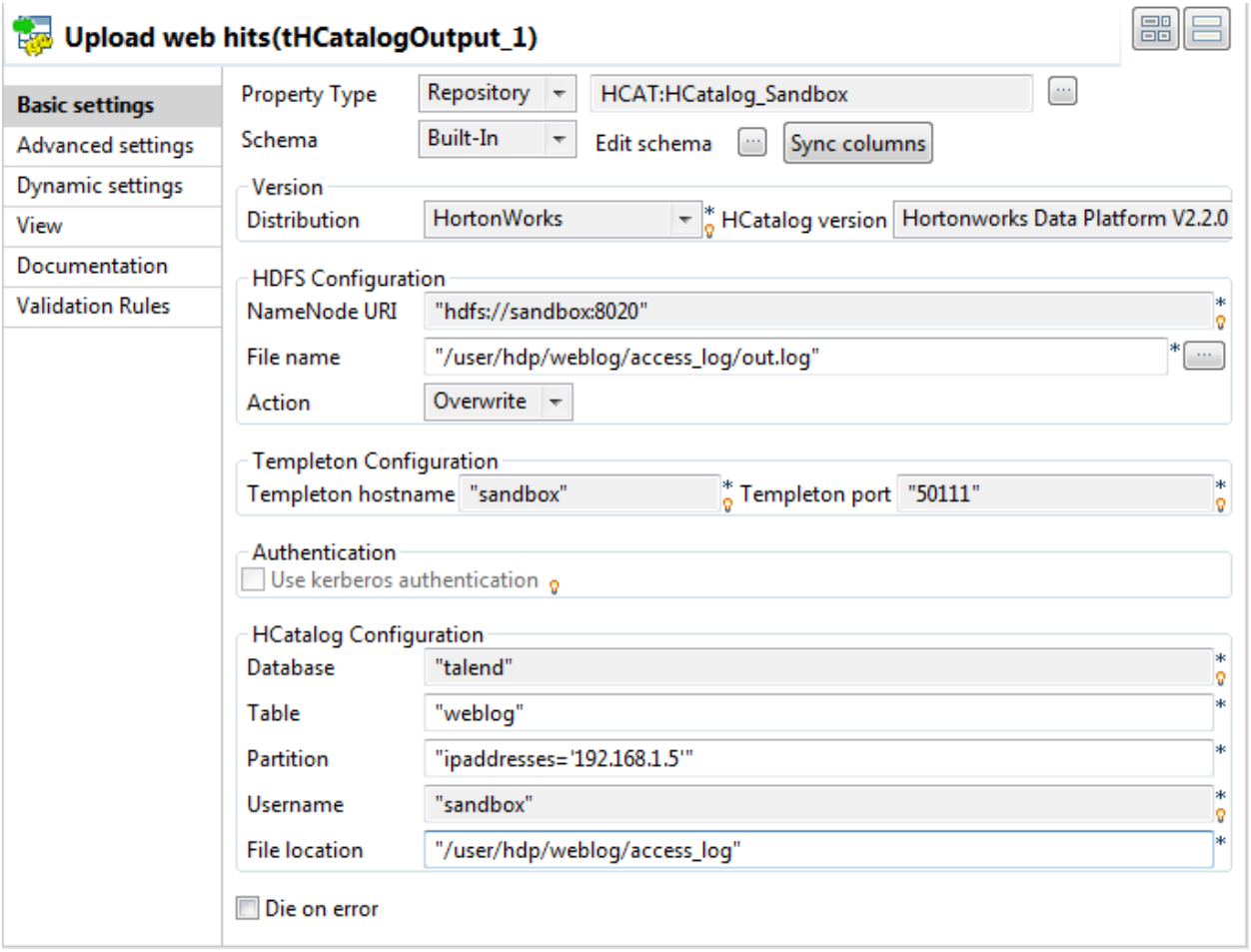
3. From the **Logical operator used to combine conditions** list box, select **AND**.
4. Click the **[+]** button to add a line in the **Filter configuration** table, and set filter parameters to send records that contain the code of "301" to the Reject flow and pass the rest records on to the Filter flow:

   a) In the **InputColumn** field, select the `code` column of the schema.

b) In the **Operator** field, select **Not equal to**.

c) In the **Value** field, enter `301`.

5. Double-click the **tHCatalogOutput** component to open its **Basic settings** view.



6. Click the **Property Type** list box and select **Repository**, and then click the **[...]** button to open the **[Repository Content]** dialog box to use a centralized HCatalog connection.
7. Select the HCatalog connection defined for connecting to the HCatalog database and click **OK**.

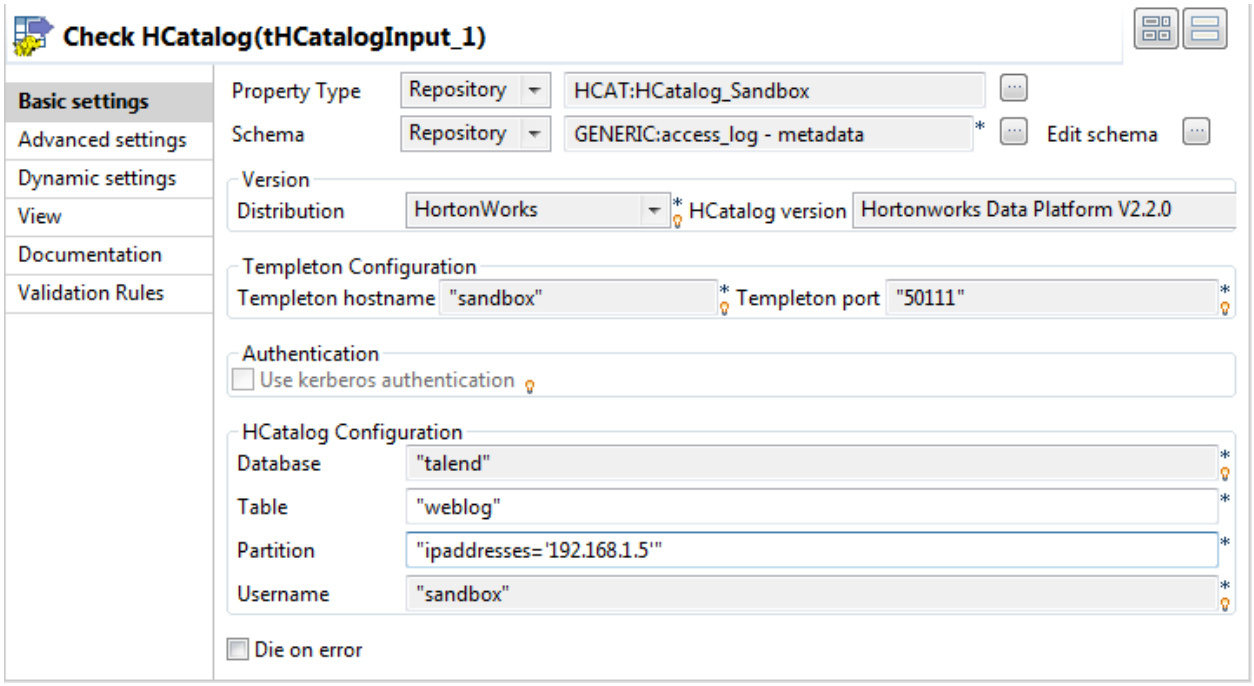
   All the connection details are automatically filled in the respective fields.
8. Click the **[...]** button to verify that the schema has been properly propagated from the preceding component. If needed, click **Sync columns** to retrieve the schema.
9. From the **Action** list, select **Create** to create the file or **Overwrite** if the file already exists.
10. In the **Partition** field, enter the partition name-value pair between double quotation marks, `ipaddresses='192.168.1.15'` in this example.
11. In the **File location** field, enter the path where the data will be save, `/user/hdp/weblog/access_log` in this example.
12. Double-click the **tLogRow** component to open its **Basic settings** view, and select the **Vertical** option to display each row of the output content in a list for better readability.
13. Upon completion of the component settings, press **Ctrl+S** to save your Job configurations.

Configuring the third Job

In this step, we will configure the third Job, `C_HCatalog_Read`, to check the content of the log uploaded to the HCatalog.

Procedure

1. Double-click the **tHCatalogInput** component to open its **Basic settings** view in the **Component** tab.



2. Click the **Property Type** list box and select **Repository**, and then click the **[...]** button to open the **[Repository Content]** dialog box to use a centralized HCatalog connection.
3. Select the HCatalog connection defined for connecting to the HCatalog database and click **OK**.

   All the connection details are automatically filled in the respective fields.
4. Click the **Schema** list box and select **Repository**, then click the **[...]** button next to the field that appears to open the **[Repository Content]** dialog box, expand **Metadata** > **Generic schemas** > **access_log** and select schema. Click **OK** to confirm your select and close the dialog box. The generic schema of `access_log` is automatically applied to the component.

   Alternatively, you can directly select the generic schema of `access_log` from the **Repository** tree view and then drag and drop it onto this component to apply the schema.
5. In the **Basic settings** view of the **tLogRow** component, select the **Vertical** mode to display the each row in a key-value manner when the Job is executed.
6. Upon completion of the component settings, press **Ctrl+S** to save your Job configurations.

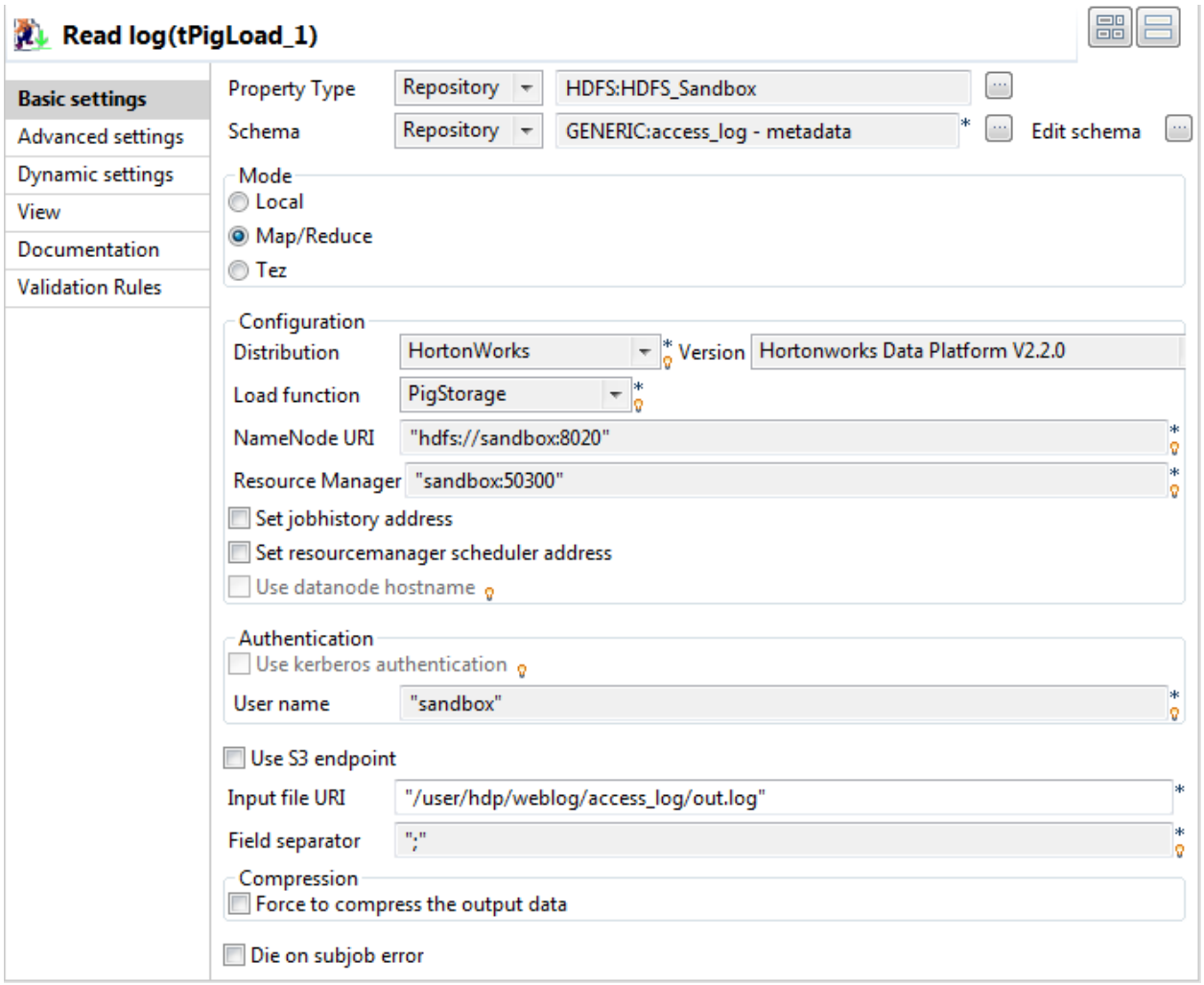
Configuring the fourth Job

In this step, we will configure the fourth Job, `D_Pig_Count_Codes`, to analyze the uploaded access log file using a Pig chain to get the codes of successful service calls and their number of visits to the website.

Read the log file to be analyzed through the Pig chain

Procedure

1. Double-click the **tPigLoad** component to open its **Basic settings** view.



2. Click the **Property Type** list box and select **Repository**, and then click the **[...]** button to open the **[Repository Content]** dialog box to use a centralized HDFS connection.
3. Select the HDFS connection defined for connecting to the HDFS system and click **OK**.

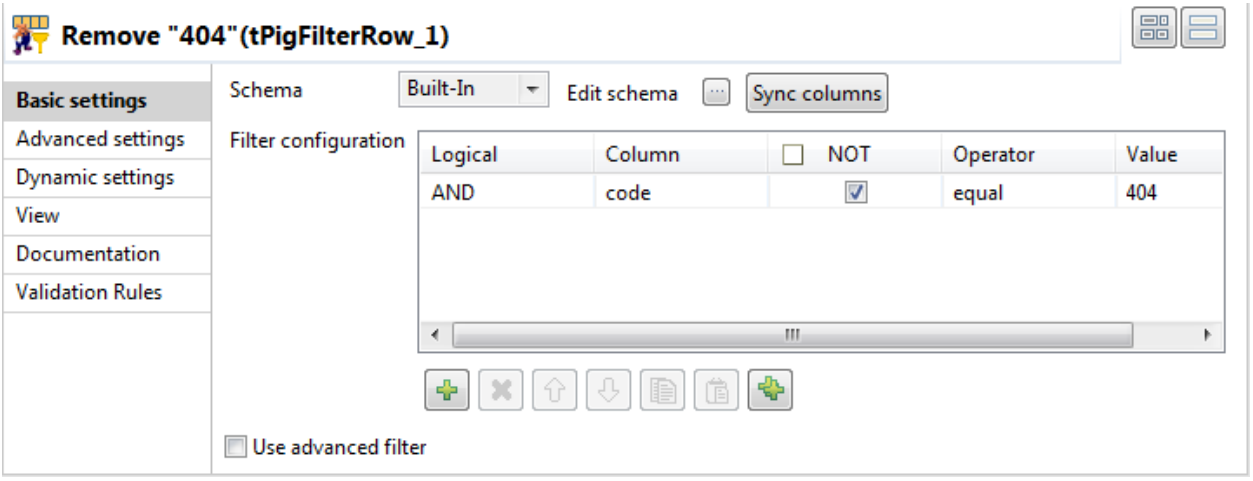
   All the connection details are automatically filled in the respective fields.
4. Select the generic schema of `access_log` from the Repository tree view and then drag and drop it onto this component to apply the schema.
5. From the **Load function** list, select **PigStorage**, and fill the **Input file URI** field with the file path de fined in the previous Job, `/user/hdp/weblog/access_log/out.log` in this example.

Analyze the log file and save the result

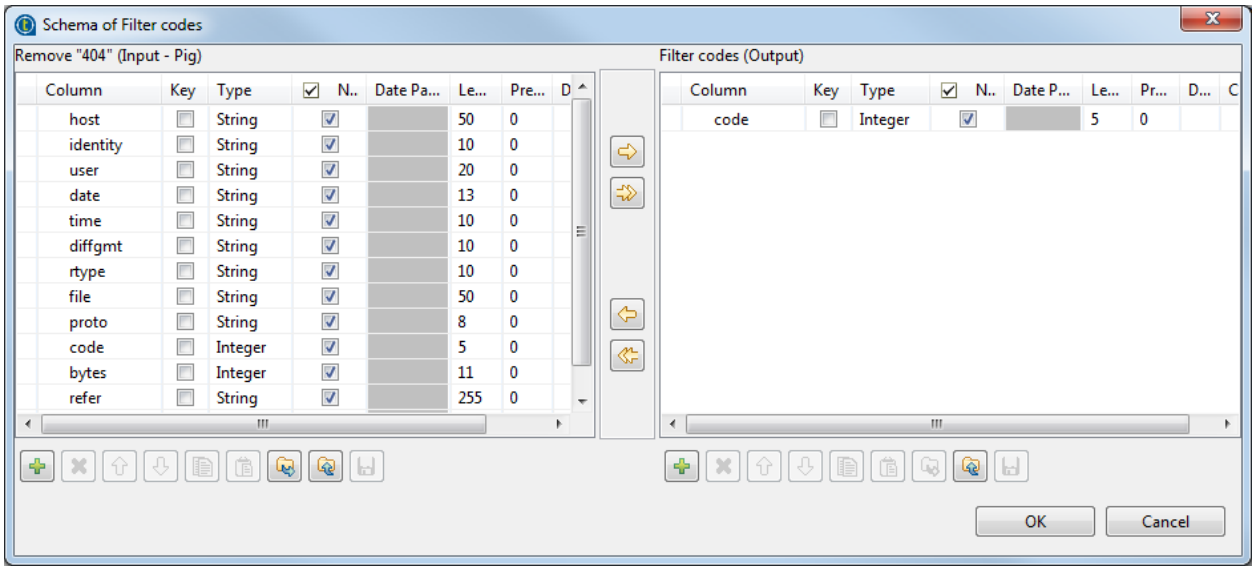
Procedure

1. In the **Basic settings** view of the **tPigFilterRow** component, click the **[+]** button to add a line in the **Filter configuration** table, and set filter parameters to remove records that contain the code of 404 and pass the rest records on to the output flow:
   a) In the **Logical** field, select **AND**.
   b) In the **Column** field, select the `code` column of the schema.
   c) Select the **NOT** check box.
   d) In the **Operator** field, select **equal**.

e) In the **Value** field, enter `404`.



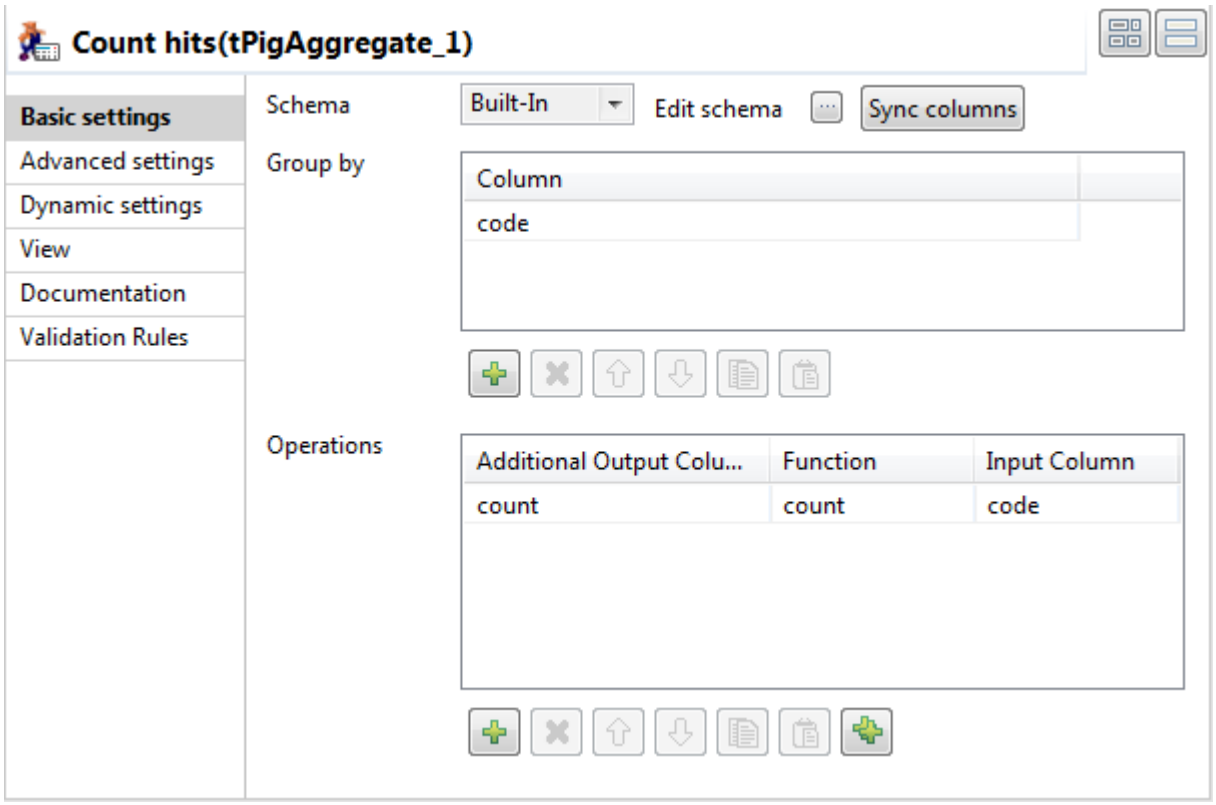
2. In the **Basic settings** view of the **tPigFilterColumns** component, click the **[...]** button to open the **[Schema]** dialog box. Select the column `code` in the **Input** panel and click the single-arrow button to copy the column to the **Output** panel to pass the information of the `code` column to the output flow. Click **OK** to confirm the output schema settings and close the dialog box.



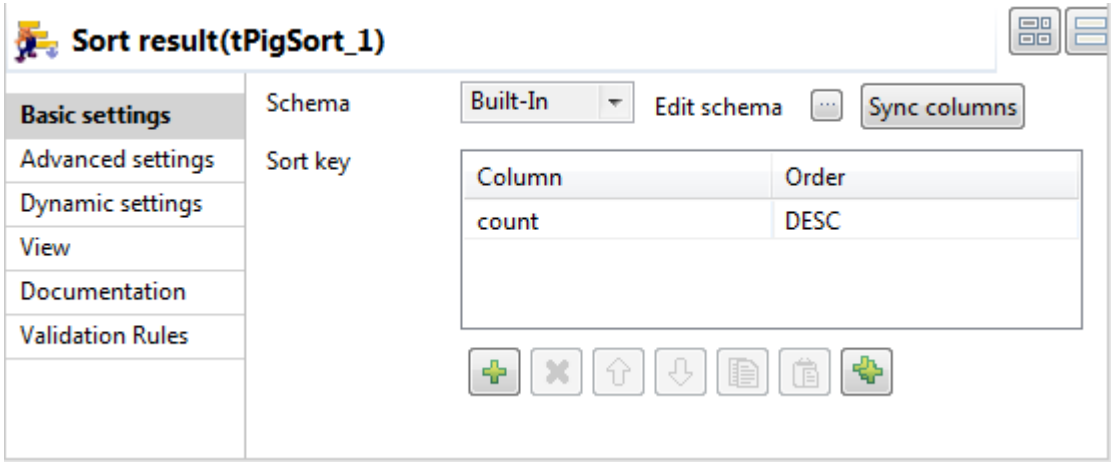
3. In the **Basic settings** view of the **tPigAggregate** component, click **Sync columns** to retrieve the schema from the preceding component, and permit the schema to be propagated to the next component.
4. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box, and add a new column: `count`.

   This column will store the number of occurrences of each code of successful service calls.
5. Configure the following parameters to count the number of occurrences of each code:
   a) In the **Group by** area, click the **[+]** button to add a line in the table, and select the column `count` in the **Column** field.

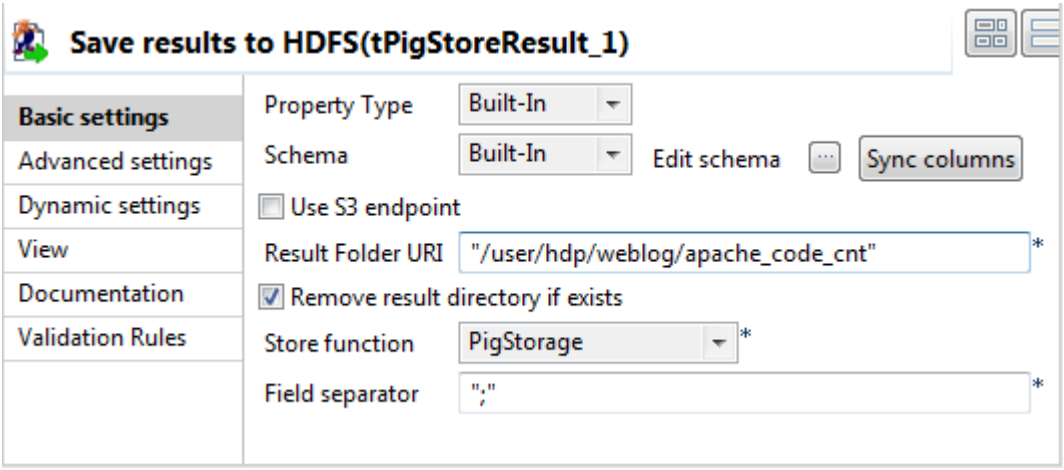
   b) In the **Operations** area, click the **[+]** button to add a line in the table, and select the column `count` in the **Additional Output Column** field, select **count** in the **Function** field, and select the column `code` in the **Input Column** field.



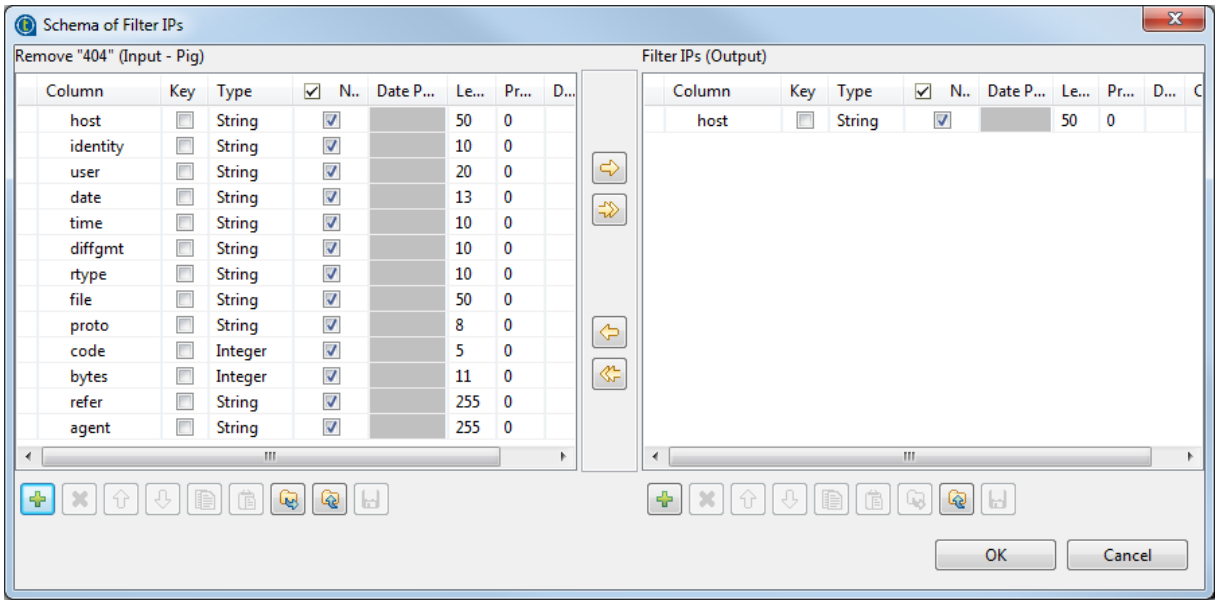
6. In the **Basic settings** view of the **tPigSort** component, configure the sorting parameters to sort the data to be passed on:
   a) Click the **[+]** button to add a line in the **Sort key** table.
   b) In the **Column** field, select **count** to set the column `count` as the key.
   c) In the **Order** field, select **DESC** to sort data in the descendent order.



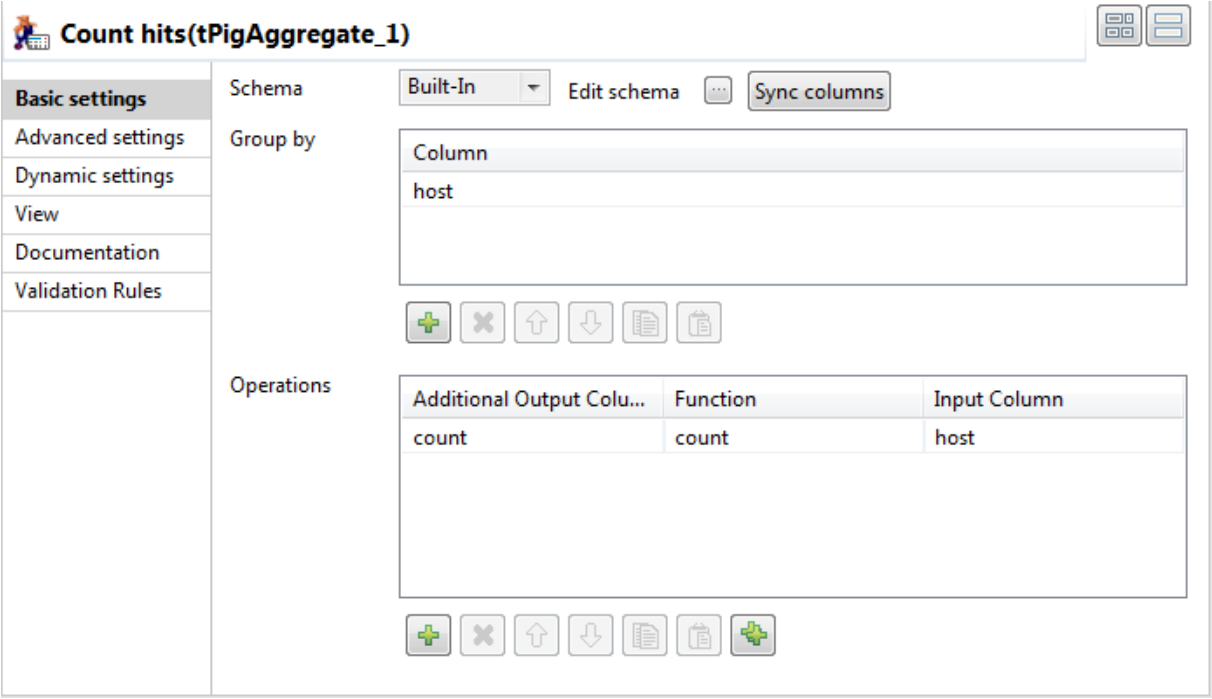
7. In the **Basic settings** view of the **tPigStoreResult** component, configure the component properties to upload the result data to the specified location on the Hadoop system:
   a) Click **Sync columns** to retrieve the schema from the preceding component.
   b) In the **Result file URI** field, enter the path to the result file, `/user/hdp/weblog/apache_code_cnt` in this example.
   c) From the **Store function** list, select **PigStorage**.
   d) If needed, select the **Remove result directory if exists** check box.



8. Save the schema of this component as a generic schema in the **Repository** for convenient reuse in the last Job, like what we did in Centralize the schema for the access log file for reuse in Job configurations on page 16. Name this generic schema `code_count`.
9. In this step, we will configure the fifth Job, `E_Pig_Count_IPs`, to analyze the uploaded access log file using a similar Pig chain as in the previous Job to get the IP addresses of successful service calls and their number of visits to the website. We can use the component settings in the previous Job, with the following differences:
   a) In the **[Schema]** dialog box of the **tPigFilterColumns** component, copy the column `host`, instead of `code`, from the **Input** panel to the **Output** panel.



   b) In the **tPigAggregate** component, select the column `host` in the **Column** field of the **Group by** table and in the **Input Column** field of the **Operations** table.



c) In the **tPigStoreResult** component, fill the **Result file URI** field with `/user/hdp/weblog/apache_ip_cnt`.

d) Save a generic schema named `ip_count` in the **Repository** from the schema of the **tPigStoreResult** component for convenient reuse in the last Job.

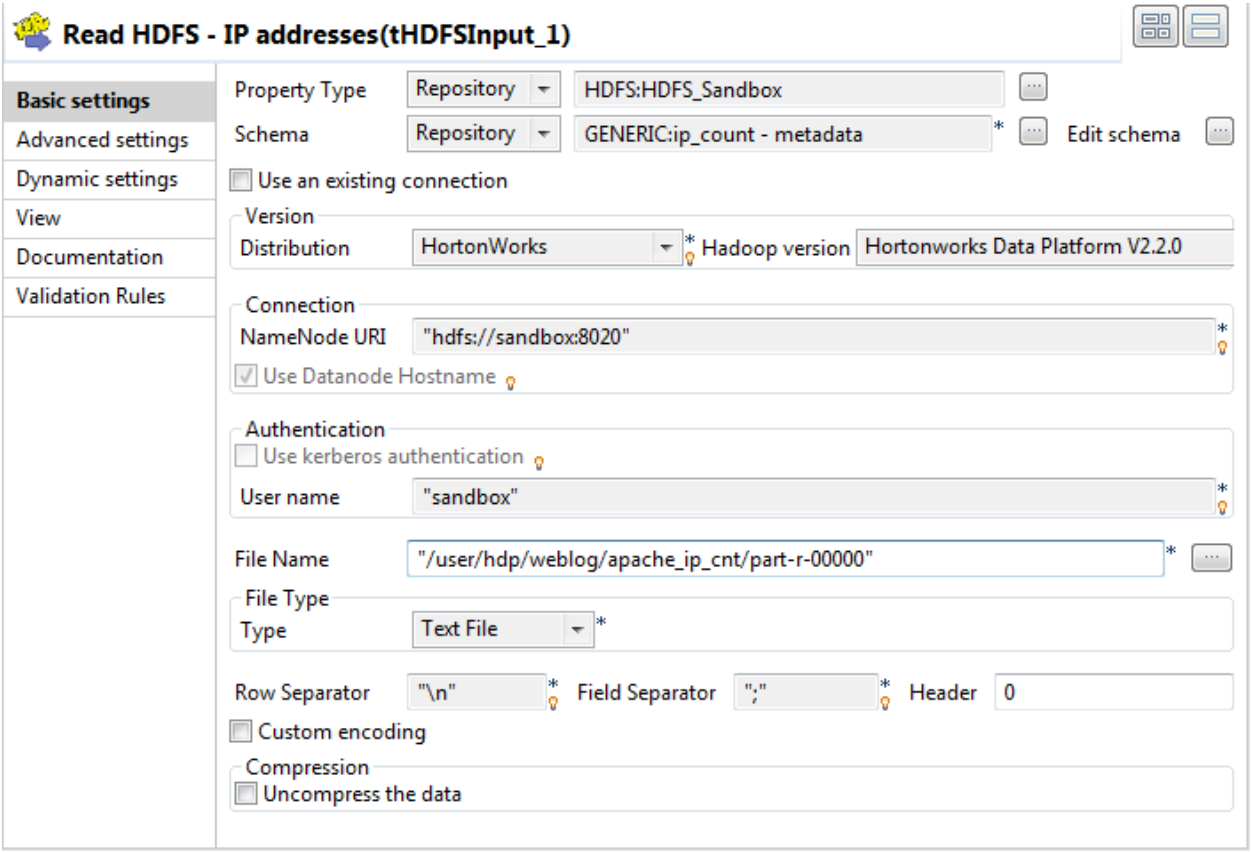
e) Upon completion of the component settings, press **Ctrl+S** to save your Job configurations.

Configuring the last Job

In this step, we will configure the last Job, `F_Read_Results`, to read the results data from Hadoop and display them on the standard system console.

Procedure

1. Double-click the first **tHDFSInput** component to open its **Basic settings** view.



2. Click the **Property Type** list box and select **Repository**, and then click the **[...]** button to open the **[Repository Content]** dialog box to use a centralized HDFS connection.
3. Select the HDFS connection defined for connecting to the HDFS system and click **OK**.

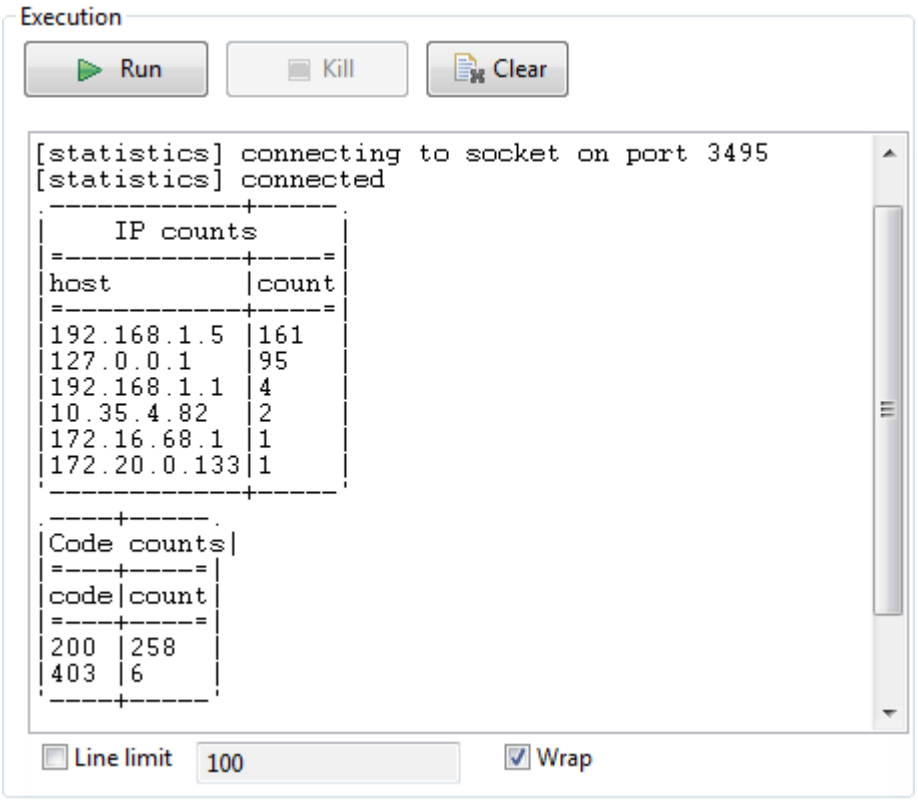
   All the connection details are automatically filled in the respective fields.
4. Apply the generic schema of `ip_count` to this component. The schema should contain two columns, `host` (string, 50 characters) and `count` (integer, 5 characters),
5. In the **File Name** field, enter the path to the result file in HDFS, `/user/hdp/weblog/apache_ip_cnt/part-r-00000` in this example.
6. From the **Type** list, select the type of the file to read, **Text File** in this example.
7. In the **Basic settings** view of the **tLogRow** component, select the **Table** option for better readability.
8. Configure the other subjob in the same way, but in the second **tHDFSInput** component:
   a) Apply the generic schema of `code_count`, or configure the schema of this component manually so that it contains two columns: `code` (integer, 5 characters) and `count` (integer, 5 characters).
   b) Fill the **File Name** field with `/user/hdp/weblog/apache_code_cnt/part-r-00000`.
9. Upon completion of the component settings, press **Ctrl+S** to save your Job configurations.

Running the Jobs at one click

After the six Jobs are properly set up and configured, click the **Run** button on the **Run** tab or press **F6** to run them one by one in the alphabetic order of the Job names.

You can view the execution results on the console of each Job.

Upon successful execution of the last Job, the system console displays IP addresses and codes of successful service calls and their number of occurrences.

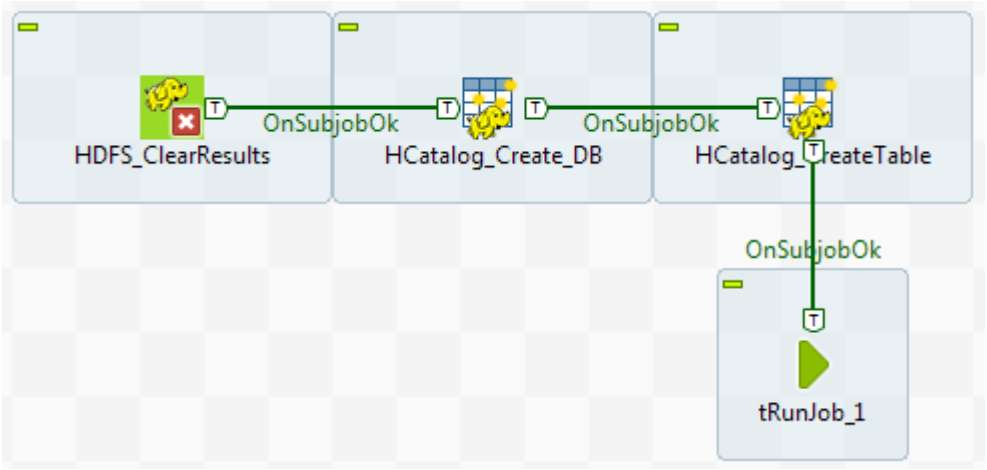


Running the Jobs at one click

It is possible to run all the Jobs in the required order at one click.

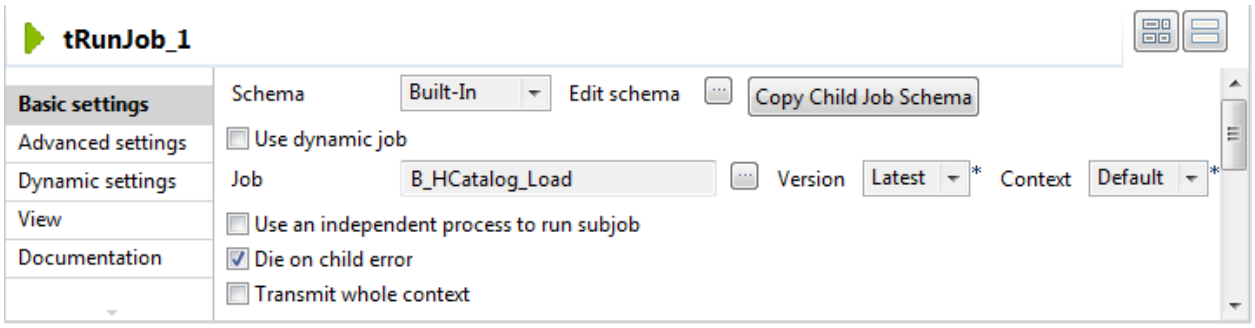
Procedure

1. Drop a **tRunJob** component onto the design workspace of the first Job, `A_HCatalog_Create` in this example. This component appears as a subjob.
2. Link the preceding subjob to the **tRunJob** component using a **Trigger** > **On Subjob Ok** connection.



3. Double-click the **tRunJob** component to open its **Basic settings** view.
4. Click the **[...]** button next to the **Job** field to open the **[Repository Content]** dialog box. Select the Job that should be triggered after successful execution of the current Job, and click **OK** to close the dialog box.

   The next Job to run appears in the **Job** field.



5. Double-click the **tRunJob** component again to open the next Job. Repeat the steps above until a **tRunJob** is configured in the Job `E_Pig_Count_IPs` to trigger the last Job, `F_Read_Results`.
6. Run the first Job.

   The successful execution of each Job triggers the next Job, until all the Jobs are executed, and the execution results are displayed in the console of the first Job.