



Flow Builder User Guide

Prepared For: Internal

Updated January 15, 2021

Latest Release: R4.5



Copyright © 2021 ZS Associates, Inc. All rights reserved.

Information in this document is subject to change without notice. These materials and the software described herein are solely for the use of licensed users. The software is subject to the license agreement that accompanies or is included with the software. No part of these materials may be reproduced, transcribed, stored in a retrieval system, circulated, quoted, or transmitted in any form or by any means, electronic or mechanical, recorded or otherwise, for any purpose, without express written consent of ZS Associates.

All information and images contained within the Flow Builder User Guide is © 2021 ZS Associates, Inc. which governs their use and reproduction. Illegal use or reproduction of any portion of this document is expressly forbidden without written consent.

ZS and accompanying logos are trademarks of ZS Associates.

Published by:

ZS Associates

1560 Sherman Avenue

Evanston, Illinois 60201 USA

www.zs.com

Table of Contents

Terms of Use	8
About this Guide	9
Notes, Warnings and Examples	9
Downloading Flow Builder Documentation	9
Introduction	10
What is Flow Builder?	10
Integration of Flow Builder with Other Applications.....	10
Flow Builder Components.....	10
Accounts	10
Branches.....	10
DAG	10
Environment	11
Flows	11
Job	11
Mapping	11
Node.....	11
Project	11
Flow Builder User Roles.....	12
Creating Accounts and Adding Account Administrators	14
Node Types	15
Source Nodes	15
Source Table.....	16
Source File.....	16
DB Source	18
Source Salesforce.....	18
Bulk Mode Support for SourceSalesforce Node	19
Spark Read	21

SQL Nodes	24
SQL	25
Unary SQL Template	25
Action Nodes	26
Spark Write	27
Write File	28
Write Table	30
Write Salesforce	30
Support for Batching in the Write Salesforce Node	31
Register Temp View	31
Flows Nodes	32
Action Flow	32
Multiple Flow	38
Excel Flow	38
Restricted File Upload for Excel Flow Node	38
Jar Action Flow	40
Remote Nodes	42
Remote Flow	42
Remote Transform	45
PySpark Nodes	46
Generate Base 64 Encoded Token	46
Repository Tab for a List of Deployed Artifacts	47
Maven Artifact Upload is a Review Feature for FlowBuilder Repository	47
Creating a Flow in Flow Builder	50
Adding a Project	50
Adding a Mapping	51
Updating a Mapping with Edit Mode	52
Adding, Editing and Saving Flows	52
Committing Changes	53

Provide a Commit or Draft Message	54
View Mappings with the Changelist	54
Discarding Changes	55
Discard Saved Changes.....	55
Discard Unsaved Changes.....	55
Renaming a Project/Mapping.....	56
Project Versioning	58
Manage Branching Tab.....	58
Commits Tab	59
Drafts Tab	60
Parameterizing Flows.....	61
Use the \${env} Template	61
Use the Environment Variable File	61
Override Environment Configuration at the Job Definition Level	62
Override Environment Variables at Job Execution level	62
Release Management	63
Creating Project Releases and Branches.....	63
Tagging Project Releases to an Execution Environment.....	64
Switching Between Releases and Branches	65
Creating a Hotfix Release	65
Remote Agent Process	66
Remote-Agent Strategies Supported.....	67
Cluster Backend Supported	67
Execution Types Supported	68
Labelled Remote Agent.....	68
Using the Remote-Agent Process Tabs	69
Shutting Down the Remote-Agent Manually	69
Setting the Maximum Runtime Duration for the Remote-Agent	70
Shutting Down the Remote-Agent via JarActionFlow	71

Execution Report.....	72
Jobs	73
Defining and Creating a Job Definition.....	73
Effective Remote Agent Configuration in Job Execution Logs	75
Additional Property Configuration	75
Environment Variable Support in Job Additional Props	77
Configuration Visibility at the Job-Definition Level	78
Configuration Visibility at the Job-Execution Level	79
Limiting the Job Execution Duration for a Job-Definition	80
Job Execution on a Labelled Remote-Agent	80
Chaining Jobs Together	81
Job Execution Chain Visualization	82
Chaining Job Execution Additional Properties.....	83
Job Chain Visualization	84
Terminating Remote Agent on Job Completion	85
Executing Excel Node as a Child Job.....	85
Sending Email Notifications on Job Failure	86
JSON for Email Configuration	86
JSON for Email Recipients	86
JSON for Target Group Email Configuration	87
Email Notification on Job Success and Remote Agent Shutdown Events.....	89
Email Configuration: On Job Success	89
Email Configuration: On Remote Agent Shutdown	90
Job State Notification Using Callback URL.....	91
httpHeaders	92
postUrl.....	92
Aborting Jobs.....	93
Asset Management	94
Creating and Subscribing Asset Releases	94

Subscribing Subsequent Asset Releases to a Client Project.....	94
Using User Interface to Subscribe Assets.....	94
Script-Runner.....	97
Scripts Supported	97
Using the Script-Runner	97
Frequently Asked Questions	99
CLI Commands.....	100
Submitting a Job	100
Automating Job Submission: End to End.....	101
Terminating Remote-Agent Cleanly	101
Asset Subscription	102
Copying Flows	104
Copy Entire Projects/Group of Mappings.....	107
Copy a Single Mapping Across/Within the Same Flow Builder Instance.....	107

Terms of Use

By accessing the content contained in this document (the "Content") you are consenting to be being bound by the terms of use set forth below.

The Content constitutes confidential and proprietary information belonging to ZS Associates, Inc. ("ZS").

Your use of the Content shall be for (a) your internal business purposes only, and (b) the limited purposes of (i) evaluating the compatibility of Flow Builder with your system infrastructure, and (ii) facilitating the actual use of Flow Builder within your system infrastructure (should you opt to license Flow Builder from ZS).

You may not use the Content to replicate, frame or mirror Flow Builder (or support a third party in doing the same).

Except for the purpose of exercising the rights granted by ZS herein, you may not edit, modify or revise the Content in any manner whatsoever without first obtaining ZS's prior written consent thereto. You may not delete or in any manner alter (i) any copyright, trademark, and other proprietary rights notices that appears on the Content as delivered to you.

Unless expressly authorized by ZS, you may not distribute, sublicense, lease, rent or re-syndicate the Content on a standalone basis.

You will not use the Content to:

- contain or promote sexually explicit, lewd and/or pornographic materials, or depictions of violent or sexual acts; promote violence, hate, discrimination based on race, sex, religion, nationality, disability, sexual orientation or age;
- be libelous, defamatory, knowingly false or misrepresent another person;
- harass, threaten, abuse or insult end users or any other person;
- be unlawful or facilitate the violation of any applicable law, regulation or governmental policy; or promote illegal activities;
- infringe upon the intellectual property rights of any third party, including the copyrights, trademarks, trade names, trade secrets or patents of such third party;
- offer or disseminates any fraudulent goods, services, schemes or promotions, including any make-money-fast schemes, chain letters, or pyramid (ponzi) schemes;
- violate the privacy, publicity, moral or any other right of any third party; or
- be harmful to ZS's or any end user's or other third party's systems and networks, including any transmissions which may damage, interfere with, surreptitiously intercept, or expropriate any system, program, data or personal information.

The Content is provided "AS IS" on an "AS AVAILABLE" basis without warranty or condition of any kind. ZS disclaims all warranties, express or implied, including implied warranties of merchantability, fitness for a particular purpose and non-infringement, and any warranties or conditions arising out of course of dealing or usage of trade. ZS is not responsible or liable (and makes no representation or warranty) for the accuracy, content, completeness, legality, reliability, or availability of the Content.

You will indemnify, defend (or settle) and hold ZS harmless from any and all claims, damages, liabilities, actions, judgments, costs and expenses (including reasonable attorneys' entitled to recover its reasonable fees) brought by a third party arising out of or in connection with your unauthorized use of the Content.

IN NO EVENT WILL ZS BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, OR SPECIAL DAMAGES WHATSOEVER (INCLUDING LOSS OF PROFITS, BUSINESS INTERRUPTION, OR USE) ARISING OUT OF THE USE OF OR INABILITY TO USE THE CONTENT, EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

About this Guide



Note: This *Flow Builder User Guide* is continuously being updated.

The *Flow Builder User Guide* provides a general overview of the functionality of the Flow Builder application. This guide can be used for training new users.

Notes, Warnings and Examples



Note is extra information needed to support a procedure or other content. Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a useful tip.



Warning is information used to avoid danger or another negative situation. Warnings should not be ignored as this may prevent you from understanding the content or cause other issues.



Example is specific information that illustrates something in more detail.

Downloading Flow Builder Documentation

Users can download the *Flow Builder User Guide* by clicking the **Help** button beside the Settings button on the top-right corner of the application.

Introduction

VERSO is the latest product family from ZS. It brings together sophisticated AI technology and consumer-grade user interfaces to enable intelligent commercial execution.

What is Flow Builder?

Flow Builder is a cloud-based data processing platform using Apache Spark as the processing engine and provides features to build, manage and execute data pipelines.

Integration of Flow Builder with Other Applications

Flow Builder can be accessed via the link in the top section of the VERSO Orchestration Center application.

Flow Builder Components

Accounts

Account represents an individual client. Account allows clients to create Projects which stores all the Flows. There are two types of accounts:

Client Account

Client Account (previously called Client Instance) is where users can build and execute their own flows. The flows created in a Client Account cannot be shared with other Accounts.

Asset Account

Asset Account (previously called Asset Instance) allows users to create shareable Flows. In an Asset Account, users must create an Asset Project Release. Projects in Client Accounts can then *subscribe* to these Asset Project Releases. After subscription these Asset Flows are accessible in read-only mode in the Client Project.

Branches

Branches are read/write copies of a master branch or a release.

DAG

A DAG is a Directed Acyclic Graph. Flows are represented by Directed Acyclic Graphs (DAGs) where the Nodes represent executable tasks and the edges are task dependencies. DAG is like a flow chart which tells which tasks are executed and in what order.

Environment

Environment (previously known as Consumer App) provides an interface for developing and executing Flows in a Project. When an Account is created in Flow Builder, a *Development Environment* gets created automatically. Environment can be linked to the VERSO Orchestration Engine application to access the respective Projects in the Account. The two types of Environments are:

Development Environment

A *Development Environment* is where the flows are developed and edited. Flows can be saved, committed and executed for testing purposes in this Development Environment. An Account can have only one Development Environment.

Execution Environment

Once a project has been tested and released in the Development Environment, the respective release is available in the *Execution Environment*. This environment should be used only for running the final flows. Editing of flows is not supported here. An Account can have multiple Execution Environments.

Flows

Flows are sequence of tasks through which a piece of work passes from initiation to completion. Flows are represented by DAGs where the Nodes represent executable tasks and the edges are task dependencies. DAG is like a flow chart which tells which tasks are executed and in what order.

Job

Job represents a Flow to be executed. It points to a node that needs to be executed. This node is also known as the *Trigger Node*. Jobs can be chained to execute other jobs on completion.

Mapping

Mapping is a container of Nodes. The DAG Flow is created in a Mapping.

Node

Node represents an executable task.

Project

Project is a container of Mappings. Multiple Projects can be created within an Account.

Flow Builder User Roles

Flow Builder has five native user roles listed. Non-Admin users have read-only access to the Settings page. The user role and the level of access in the Settings page is summarized in the following table.

User Role	Description and Permissions
Super Admin	This user role is only given to the SD Support Team and has Administrative access across the Flow Builder application.
Account Admin (also known as Instance Admin)	<p>The Account Administrator can see the <i>Account</i> tab in the Administration page and has the permission to:</p> <ul style="list-style-type: none"> • Create Execution Environments (Execution Consumer Apps). • Add Environment Users for any Environment/Consumer App present (both Development and Execution). • Define and edit the PER_INSTANCE and PER_CONSUMER_APP remote-agent property. • Have all Environment Administrator-level permissions • Able to view the Settings button. • View all tabs (Account, Environment, Strategy Configuration and Repositories). • Edit all properties.
Environment Admin (also known as Consumer App Administrator)	<p>The Environment Administrator can only see the <i>Environment</i> and the <i>Strategy Configuration</i> tab. This user role has the permission to:</p> <ul style="list-style-type: none"> • Add Environment Users (Environment Administrators, Environment Editors, Environment Viewers). • Define and edit PER_CONSUMER_APP remote-agent strategy. • Have all Environment Editor-level permissions. • Able to view the Settings button. • View all tabs (Account, Environment, Strategy Configuration and Repositories). • Cannot set instance level strategy, as before. • Cannot add a new Execution Environment in the Environment tab.
Environment Editor (also known as Consumer App Editor)	<p>This is a non-administrator user who is unable to navigate to the Flow Builder Administration page via the <i>Settings</i> button on the top right of the application. This user role is expected to focus primarily on Flow Development in the DAG Building area. This user role has the permission to:</p> <ul style="list-style-type: none"> • Create and rename projects and mappings.

User Role	Description and Permissions
	<ul style="list-style-type: none"> • Create, modify, edit, save, commit, discard, and delete flows. • Able to view Settings button. • View all tabs (Account, Environment, Strategy Configuration and Repositories). • Although the Account and Environment tabs are visible, the user cannot perform any action related to that level, such as Create an environment etc.
Environment Viewer (also known as Consumer App Viewer)	<p>This is a non-administrator user who is unable to access the Administration page of the Flow Builder application. This is a read-only user able to read/view the pre-existing flows.</p> <p>This user is unable to view the Settings button.</p>

Creating Accounts and Adding Account Administrators

At production level, an Account (Instance) is created for every client working on the Flow Builder Application. The Account is a workspace provided to clients to develop and execute flows.

Clients are expected to contact SD Support to create a new Account. Once a new Account has been created, the SD Support Team adds a user specified by the clients as an Account Administrator.

To create a new account:

1. Click the **Settings** button.
2. Click **Add New Account** under the Accounts tab (Figure 1 and Figure 2). This tab is only Visible to Flow Builder Super Admins.



Figure 1: Accounts Tab Used to Create a New Account

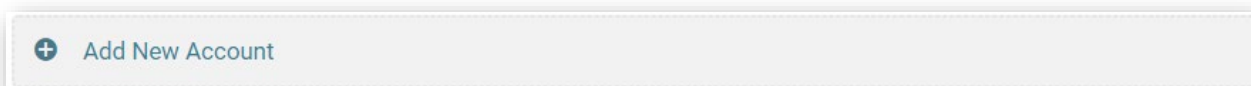


Figure 2: Option to Add a New Account

3. Fill in all the fields (Account Name, Account Type, Description and Account Code) (Figure 3).

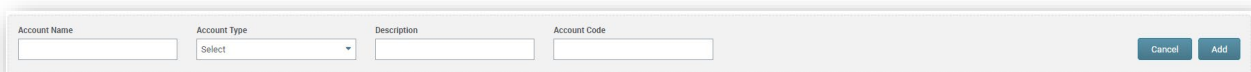


Figure 3: Fields Required to Create a New Account

4. Select the Account Type **Asset** or **Client**.
5. Click **Add** to create the account OR click **Cancel** to discard the account creation.

Node Types

Source Nodes

Source Nodes are used to read data from a source location onto which you can perform various operations (Figure 4).



Figure 4: List of Source Nodes

Source nodes can perform the following operations:

- **SourceTable node:** Read from a temporary view
- **SourceFile node:** Read from an S3 location
- **DBSource node:** Read from a database table
- **Source Salesforce node:** Read from Salesforce
- **Spark Read node:** Provide data and extract responses from Spark node (inline and rest respectively)

Source Table

A *SourceTable Node* reads from a temporary view. The properties for the Source Table node are as follows:

Node Name	Properties	Description
Source Table	Node Name	The user-specific node name
	Table Name	Name of the table registered with Register Temp Table Node
	Catalog Name	The schema into which a view is stored

Source File

A *SourceFile Node* reads from an S3 location. The properties for the Source File node are as follows:

Node Name	Properties	Description
Source Table	Node Name	The user-specific node name
	Table Name	Name of the table registered with Register Temp Table Node
	Catalog Name	The schema into which a view is stored
	Spark Source Name	The type of source node (currently supports REST and INLINE)
	Spark Source Options	Configuration JSON depending on the type of Spark Read node

Support for Additional Spark Options

The *SourceFile* node has support for the following optional parameters (Figure 5):

Option	Default	Description
Escape	\	Sets the single character used for escaping quoted values where the separator can be part of the value.
quote	"	Sets the single character used for escaping quoted values where the separator can be part of the value.
Encoding	UTF-8	Decodes the CSV files by the given encoding type.

src-utf16

Source File

dest-utf16

Write File

Node Properties

Node Name
src-utf16

FilePath
s3://aws-a0019-use1-00-q-s3b-data-lake-fb-00

Format
Csv

Header
True

InferSchema
True

Delimiter
,

Escape

Quote

Encoding
UTF-8

[View References](#)

Figure 5: New Properties for SourceFile: Escape, Quote, Encoding

DB Source

A *DBSource Node* reads from a database table. The properties for the DB Source node are as follows:

Node Name	Properties	Description
DB Source	Node Name	The user-specific node name
	Type	The type of database to be selected (Oracle or Postgres)
	Host	The host of the database
	Port	The port number of the database
	Service/Database	The service of the database
	Table Name	The table name from where data needs to be read
	User Name	The username to access the database
	Password	The password to access the database
	Spark Source Name	The type of source node (currently supports REST and INLINE)
	Spark Source Options	Configuration JSON depending on the type of Spark Read Node

Source Salesforce

A *Source Salesforce Node* reads from Salesforce. The properties for the Source Salesforce node are as follows:

Node Name	Properties	Description
Source Salesforce	Node Name	The user-specific node name
	Host	Salesforce API host address
	Username	Username for the account registered on the salesforce
	Password	Password for the account registered on the salesforce
	Version	Parameter of the API depicting the API version
	Page Size	Number of records to be fetched at a specified time
	SOQL	Salesforce Object Query Language

Bulk Mode Support for SourceSalesforce Node

Users have the option to either disable (Figure 6) or enable (Figure 7) bulk mode while using the SourceSalesforce node.

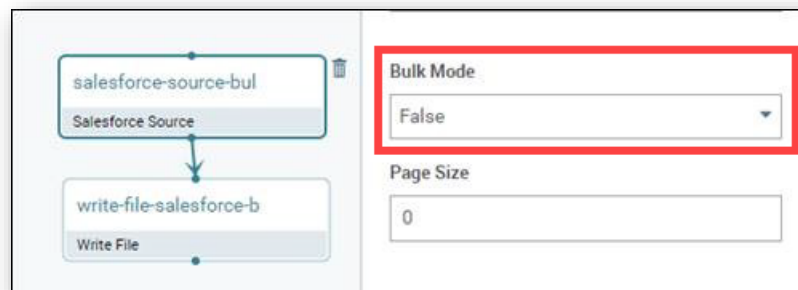


Figure 6: Bulk Mode is Disabled when BulkFlag is set to False

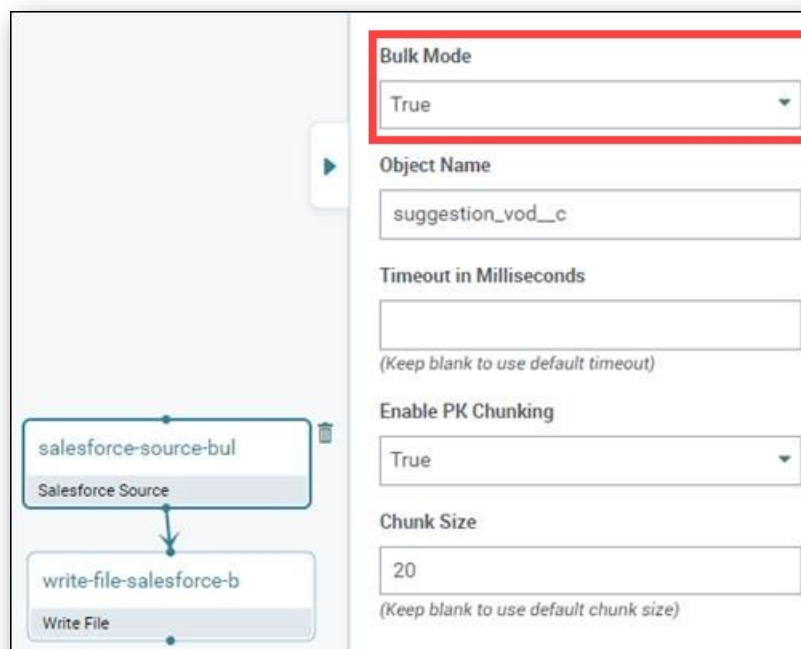


Figure 7: Bulk Mode is Enabled when BulkFlag is set to True

When BulkFlag is set to true, providing the Object Name is mandatory.

Bulk Flag = False	Bulk Flag = True
<ul style="list-style-type: none"> All existing SourceSalesforce nodes will continue working as is. When BulkFlag = false, the pageSize option can be used, though it is optional (default is 1000). 	<ul style="list-style-type: none"> Bulk Flag can be set to true from FlowBuilder UI. When BulkFlag = true, the pageSize option is not present. Use PK Chunking instead (optional). The Object Name is mandatory. Enabling PK Chunking is optional. Optional chunk size can be provided if PK Chunking is enabled.

What is PK Chunking?

PK chunking splits bulk queries on very large tables into chunks based on the record IDs or primary keys of the queried records.

What are the limitations of PK Chunking (from Salesforce)?

Salesforce recommends that you enable PK chunking when querying tables with more than 10 million records or when a bulk query consistently times out. However, the effectiveness of PK chunking depends on the specifics of the query and the queried data.

PK chunking works only on some objects. You can find the list of objects [here](#).

You can find the detailed working and limitations of PK chunking [here](#).

What are the default and maximum chunks that can be specified?

Default = 100,000 | Maximum = 250,000

What is the difference between Page Size and PK Chunking?

The main purpose of the Page Size option is to divide the result set into batches. While the restAPI uses the *pageSize* option for batching, the bulkAPI uses the *pkChunking* flag to allow the user to specify this chunk size to achieve a similar behavior.

The details on how bulk queries are processed by Salesforce can be found [here](#).

Spark Read

A *SparkRead Node* provides data and extract responses from Spark node (inline and rest respectively). The properties for a SparkRead node are as follows:

Node Name	Properties	Description
SparkRead	Node Name	The user-specific node name
	Spark Source Name	The type of source node (currently supports REST and INLINE)
	Spark Source Options	Configuration JSON depending on the type of Spark Read Node

SparkRead Node Options

Options	JSON Keys Description
INLINE creates a data frame given static data passed in a given JSON template	<p>rowDelimiter: Used when you want to provide a multi-row data to the schema</p> <p>schemaString: Used to specify the attribute name along with its datatype</p> <p>inferSchema: Set to false to restrict the application from interpreting the datatype of the schema</p> <p>header: Set to false to restrict the application from interpreting the header of the data</p> <p>input: Specifies the actual data passed to the schema</p> <p>delimiter: Separator between two columns while providing data</p> <p>Example: Sample JSON for INLINE option</p> <pre>{ "rowDelimiter": "\n", "schemaString": "`X-Actual-User` STRING, buIds STRING, `Content-Type` STRING, http_data STRING, labels STRING", "inferSchema": "false", "header": "true", "input": "`X-Actual-User`#buIds#`Content-Type`#http_data#labels\nzs\\pd99999#rpabuint1 #application/json#{`Name`: `Test24Mar27`}\n}#label1\nzs\\pd88888#rpabuint2#application/json#{`Name`: `Test24Mar28`}\n}#label2\n", "delimiter": "#" }</pre>
REST fetches a CSV or JSON formatted API response	<p>method: Accepts the value as GET/POST etc.</p> <p>httpParamTable: Takes the name of the temporary table registered using the RegisterTempTable</p>

Options	JSON Keys Description
	<p>url: Takes the API or the URL used to fetch the response of certain APIs. The extra query parameters that must be passed are supplied after the '?'. These variable names are included under the key.</p> <p>queryParamColumns: If there are more than one query params, they are separated by a comma.</p> <p>includeInputInOutput: Setting this property to true would reflect all the input parameters (builds, labels in this case) in the output.</p> <p>User credentials are passed in the keys 'userPassword' and 'userId'.</p> <p>Example: Sample JSON for REST option:</p> <pre>{ "method": "post", "httpParamTable": "input_param_table", "url": "<api_for_execution_revision>?buIds={buIds}&labels={labels}", "headerParamColumns": "X-Actual-User,Content-Type", "queryParamColumns": "buIds,labels", "includeInputInOutput": "false", "userPassword": "password", "userId": "user_id" }</pre>
<p>SparkRead REST datasource support for successHttpStatusCodes and retryHttpStatusCodes</p>	<p>successHttpStatusCodes:</p> <ul style="list-style-type: none"> This is a comma separated list of http status codes. When specified causes the http response status code to be compared with the success status codes and if it is not found an error is thrown. This is an optional parameter. <p>retryHttpStatusCodes:</p> <ul style="list-style-type: none"> This a comma separated list of http status codes which when specified causes the API call to be retried if the http response status code matches the one in that list. Maximum of three retries are made. This does not throw an error even if this is the third time you receive the same response code. This is an optional parameter and when not specified, the API call is made only once, which is the standard behavior. The user makes sure the API is safe to retry. <p>Sample Logs:</p> <pre>21:19:12.455 [Executor task launch worker for task 17] INFO com.zsassociates.spark.adapter.restv2.HttpRequestExtensions\$ - Attempt [1] failed. API call for URL: http://pu-sdkz13855- 02.zs.local:8080/spark-zs-web/test/fail/3 returned response with status code 522. Retry status codes: 504,522 21:19:12.669 [Executor task launch worker for task 17] INFO com.zsassociates.spark.adapter.restv2.HttpRequestExtensions\$ - Attempt [2] failed. API call for URL: http://pu-sdkz13855-</pre>

Options	JSON Keys Description
	<p>02.zs.local:8080/spark-zs-web/test/fail/3 returned response with status code 522. Retry status codes: 504,522 21:19:12.906 [Executor task launch worker for task 17] INFO com.zsassociates.spark.adapter.restv2.HttpRequestExtensions\$ - Attempt [3] was successful. API call for URL: http://pu-sdkz13855-02.zs.local:8080/spark-zs-web/test/fail/3 returned response with status code 200.</p> <p>In the above logs, since the API returned statusCode 522 for initial 2 attempts, and 522 was specified under the retryHttpStatusCodes, the code did retry. Once the API returned statusCode 200, which was specified under successHttpStatusCodes, the execution terminated.</p> <p>Example: Demonstration of successHttpStatusCodes and retryHttpStatusCodes</p> <pre>{ "url": "<api-url>", "method": "get", "httpParamTable": "input_param_table", "successHttpStatusCodes": "200", "retryHttpStatusCodes": "504,522" }</pre>

SQL Nodes

SQL Nodes are used to perform query operations on the source selected using the one of the source nodes (Figure 8).

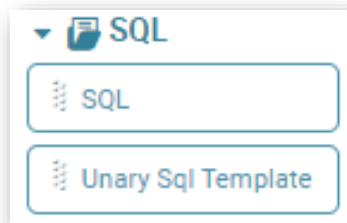


Figure 8: List of SQL Nodes

From R3 of Flow Builder, a SQL node can have other Source and SQL nodes as its parent other than just the SourceTable node. Possible parent nodes for the SQL node are:

- SourceTable
- SourceFile
- DBSource
- SQL
- Unary SQL Template
- Salesforce Source

This brings about three major changes:

1. Compact Flows (No extra PassThroughFlowFilter known as RegisterTempView node) would be needed to use SQL Nodes.
2. The SQL query structure changes. To select contents from a SourceFile parent named 'sf-1', the query should be 'select * from \$previousTable[sf-1]'.
3. In order to leverage the functionality of the Smart-SQL node, the following property should be added while creating the jobs definition.

The additional property to be used while creating a job-definition is as follows:

```
{
  "sparkzs": {
    "launcher": {
      "coreConf": {
        "isRegisterTempTableForTransform": true
      }
    }
  }
}
```

SQL

The properties for a SQL node are as follows:

Node Name	Properties	Description
SQL	Node Name	The user-specific node name
	SQL	The SQL query to fetch data from source nodes



Tip: For long SQL queries, you can press F11 to expand the query writing area to the complete screen.

Unary SQL Template

The properties for a Unary SQL Template node are as follows:

Node Name	Properties	Description
Unary SQL Template	Node Name	The user-specific node name
	SQL	The SQL query to fetch data from source nodes



Note: The Unary SQL Template node can only have a single parent whereas the SQL Node can have multiple parents.

Action Nodes

Action Nodes are used to perform various write operations (Figure 9). They also can register a temporary view and push data to S3.

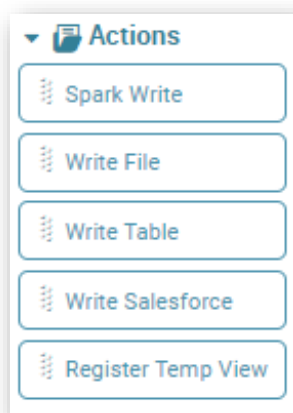


Figure 9: List of Action Nodes

Action Nodes can perform the following operations:

- Write to S3, hdfs, and dbfs using a Write File node
- Write into database table using a Write Table node
- Write into Salesforce using a Write Salesforce node
- Register a temporary view using a Register Temp View node
- Push data to S3 using a SparkWrite node

Spark Write

A *Spark Write Node* can push data to S3. The properties for this node are as follows:

Node Name	Properties	Description
SparkWrite	Node Name	The user-specific node name
	Spark Source Name	The type of source node (currently supports REST and INLINE)
	Spark Source Options	Configuration JSON depending on the type of Spark Read Node
	Partition By	To divide the result set into partitions in order to perform computation on each subset of partitioned data
	Options	JSON Keys Description
	csv, json, parquet	<pre>{ "path": "", "rowDelimiter": "\n", "nullValue": " ", "quote": "", "escape": "", "header": "false", "saveMode": "Overwrite", "delimiter": "#" }</pre>

Multi-Column Partition Support

While creating flows, users can partition their data on multiple columns. For example, if a user wants to partition the data based on *type*, *brand* and *value* simultaneously, the user can provide a comma-separated list of columns in the *Partition By* property of the WriteFile and SparkWrite Nodes (Figure 10).

Partition By

type,brand,value

Figure 10: Partition By Property of the WriteFile and SparkWrite Nodes

Write File

A *Write File Node* can write to S3, hdfs, dbfs. The properties for this node are as follows:

Node Name	Properties	Description
Write File	Node Name	The user-specific node name
	Format	The format into which the files must be written
	File Path	The location into which the file needs to be written
	Header	Whether the header needs be included in the file to be written
	Coalesce	Number of files into which output of parallel processes are written into
	Mode	One of the two (overwrite/append)
	Delimiter	Character to separate data in data stream
	Partition By	The partition to be used while writing into the file.

Support for Additional Spark Options

The *WriteFile* node has support for the following optional parameters (Figure 11):

Option	Default	Description
Escape	\	Sets the single character used for escaping quoted values where the separator can be part of the value.
quote	"	Sets the single character used for escaping quoted values where the separator can be part of the value.
Encoding	UTF-8	Decodes the CSV files by the given encoding type.

src-utf16

Source File

dest-utf16

Write File

Node Properties

Node Name

src-utf16

FilePath

s3://aws-a0019-use1-00-q-s3b-data-lake-fb-00

Format

Csv

Header

True

InferSchema

True

Delimiter

,

Escape

Quote

Encoding

UTF-8

View References

Figure 11: New Properties for WriteFile: Escape, Quote, Encoding

Write Table

A *Write Table Node* can write into a database table. The properties for this node are as follows:

Node Name	Properties	Description
Write Table	Node Name	The user-specific node name
	Type	The type of database that must be written to (Oracle/Postgres)
	Host	The host of the database
	Port	The port where the database is hosted
	Service/Database	The service of the database
	Table Name	The name of the table to be created
	User Name	The username to access the database
	Password	The password to access the database
	Mode	One of the two (overwrite/append)
	Spark Source Name	The type of source node (currently supports REST and INLINE)
	Spark Source Options	Configuration JSON depending on the type of Spark Read Node
	Partition By	To divide the result set into partitions in order to perform computation on each subset of partitioned data

Write Salesforce

A *Write Salesforce Node* can write into Salesforce. The properties for this node are as follows:

Node Name	Properties	Description
Write Salesforce	Node Name	The user-specific node name
	Host	Salesforce API Host Address
	User Name	Username for the account registered on the salesforce
	Password	Password for the account registered on the salesforce
	External ID Fieldname	External ID of the respective Salesforce object

Node Name	Properties	Description
	Version	The version of the salesforce application
	Page Size	Number of records to be fetched at a specified time
	Mode	One of the two (append/upsert)
	Object Name	The object of salesforce application to which content is to be written

Support for Batching in the Write Salesforce Node

The pageSize option for WriteSalesforce node is functional. Users can use pageSize option in the form of maxBundleSizeFactorInKB (this factor is NOT the size of the page and is used to calculate the page size).

If the pageSize option of the WriteSalesforce node is blank, then the default maxBundleSizeFactorInKB value will be used. This default value can be updated using the following configuration:

```
{
  "sparkzs": {
    "launcher": {
      "coreConf": {
        "isUseDefaultWriteSalesforceMaxBundleSizeFactor": false,
        "defaultWriteSalesforceMaxBundleSizeFactorInKB": <value-in-KB>
      }
    }
  }
}
```

Register Temp View

A *Register Temp View Node* can register a temporary view. The properties for this node are as follows:

Node Name	Properties	Description
Register Temp View	Node Name	The user-specific node name
	View Name	The name by which the view must be created

Flows Nodes

Flow Nodes are used to trigger entire flows (Figure 12).

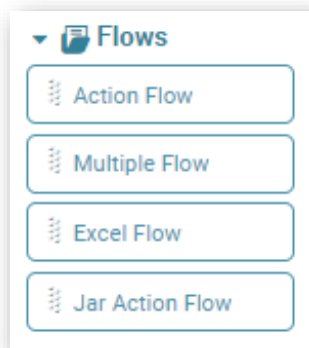


Figure 12: List of Flow Nodes

Flow Nodes can be used to do the following operations:

- Execute complete flows with an Action Flow.
- Allow execution of two or more flows simultaneously with a Multiple Flow.
- Allow flows with an ExcelNode using an Excel Flow.
- Allow flows with a JarActionFlow node using a Jar Action Flow.

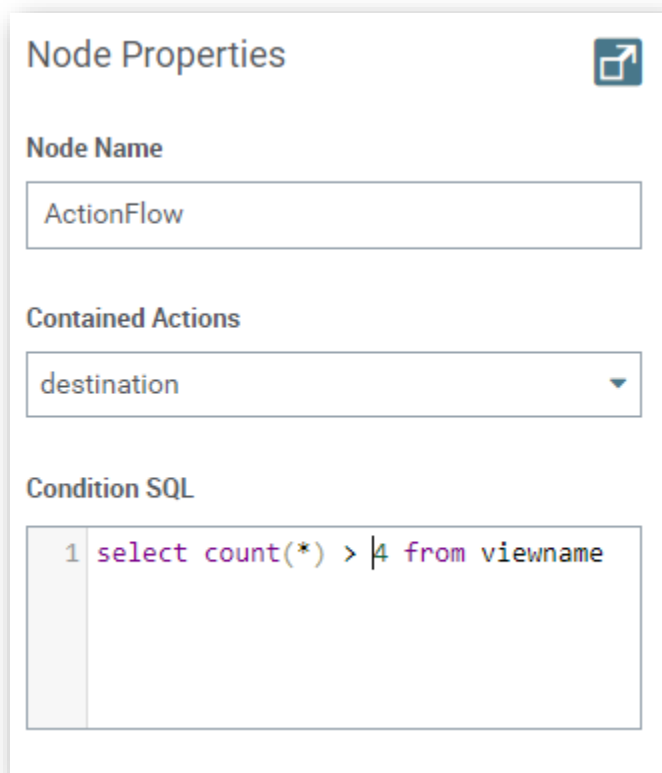
Action Flow

An *Action Flow* executes complete flows. The properties for this node are as follows:

Node Name	Properties	Description
Action Flow	Node Name	The user-specific node name
	Contained Actions	The list of Actions which must be triggered by the Action Flow Node

Action Flow Node as a Decision Node

A *Decision Node* is an Action Flow node that is executed based on the SQL that has been used in the Conditional node property of the Action Flow node. Figure 13 shows the ActionFlow node along with its Conditional SQL property.



Node Properties

Node Name

Contained Actions

destination ▾

Condition SQL

```
1 select count(*) > 4 from viewname
```

Figure 13: ActionFlow Node with Conditional SQL

Uses for a Decision Node

A Decision node (ActionFlow node with Conditional SQL) can be used to alter the execution flow of a DAG. The Conditional SQL must be written so that it outputs either 'true' or 'false', which is either '1' or '0' respectively.



Note: Consider the following SQL : `select count(*) > 4 from viewname`. Here, the ActionFlow node executes only when the count of rows in the view named 'viewname' is greater than 4. If the count is less than or equal to 4, the ActionFlow node, and by extension, the entire DAG that the ActionFlow node is pointing to, does not get executed.

DAG Example 1

Consider the following DAG example to get a better understanding of the Decision node (Figure 14).

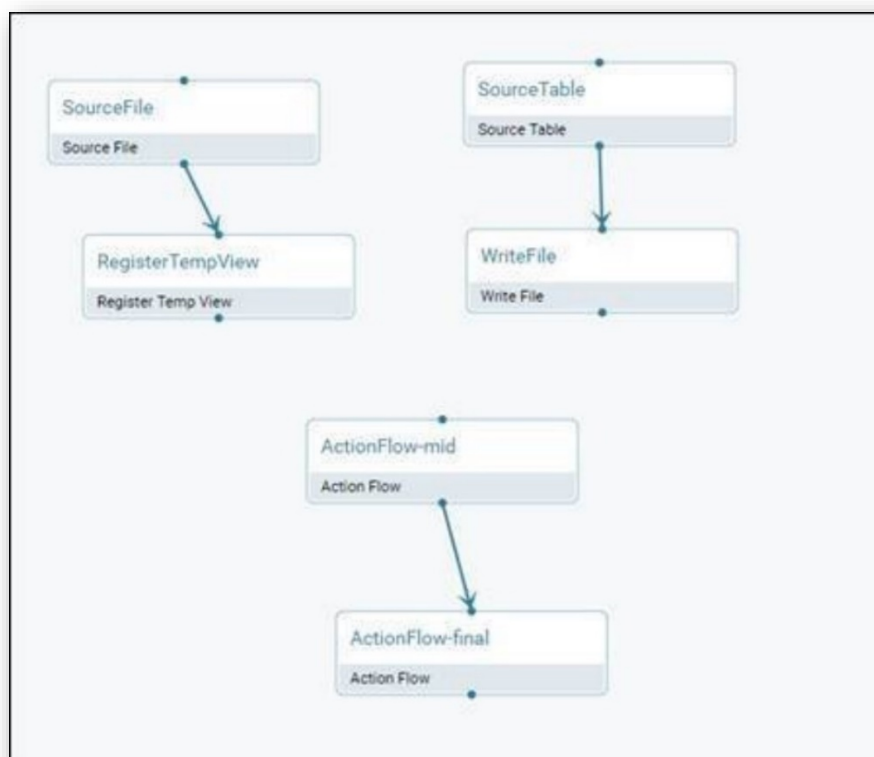


Figure 14: DAG Flow Using the Decision Node Example 1

Flow Description

In Figure 14, a SourceTable node is present which has been given the name of the view that is created using the first part of the flow, which is the one on the left with SourceFile node and RegisterTempView node.

Logically, to run these flows, there are two respective ActionFlows: ActionFlow-mid and ActionFlow-final, and both ActionFlows have conditional SQLs present.

Node Properties

The flow shown in Figure 14 has the following node properties:

Node Name	Description
SourceFile	Number of nodes is 4
RegisterTempView	Table Name is randomview
SourceTable	Table Name is randomview

Node Scenarios

Consider the following scenarios.

Node Name	Conditional SQL	Outcome
ActionFlow-mid	Select true	The flow FINISHED successfully, but no output file was generated.
ActionFlow-final	Select count(*) < 4 from randomview	
ActionFlow-mid	Select true	The flow FINISHED successfully, and the output file was generated as expected.
ActionFlow-final	Select count(*) >= 4 from randomview	
ActionFlow-mid	Select false	The flow FAILED (as was expected). The ActionFlow-mid was conditionally not executed. Hence, the temporary view was not created. As a result, when the SourceTable Node was trying to access the 'randomview', it could not find it and so the flow failed with the following exception. Error while running Json Runner The exception is : org.apache.spark.sql.AnalysisException: Table or view not found: randomview; line 1 pos 24.

DAG Example 2

Consider the following DAG example to get a better understanding of the Decision node (Figure 15).

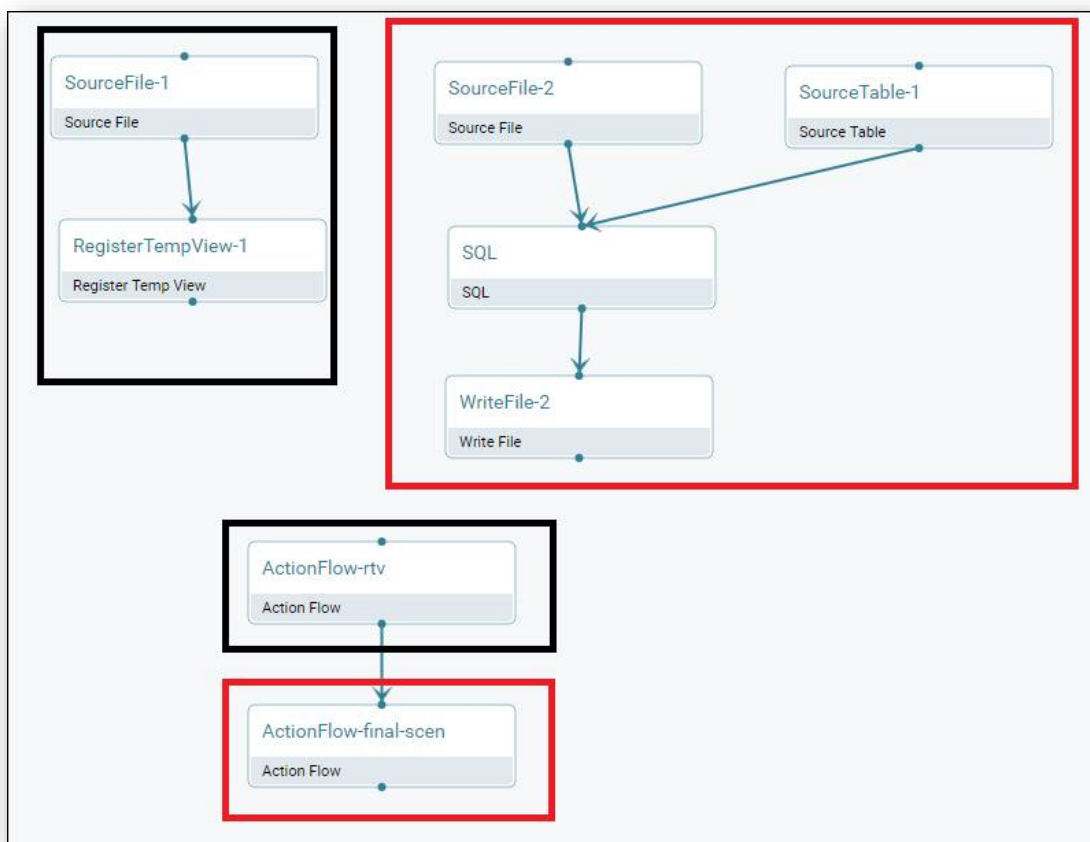


Figure 15: DAG Using the Decision Node Example 2



Note: The color of these ActionFlow nodes correspond to the flow they are executing.

Explanation and Expected Output

ActionFlow-RTV Is a simple ActionFlow node with no conditional SQL, so that it always executes and creates a temporary table (view) named testview.

Action-Flow-final-scenario-2 is an ActionFlow node WITH a conditional SQL. This points to the Write-File-2 node and encompasses the flow inside the RED box.

The Conditional SQL used is as follows:

Prerequisite	Conditional SQL	Outcome	Final Job State
<ul style="list-style-type: none"> The testview has 4 rows of data. The RED flow reads an entirely separate file which has approximately 1500 rows. The name of the file that has been read is input-long.csv 	Select count(*) >= 4 from testview	The output file for input-long.csv file was successfully written. Note: The Smart-SQL node was used.	Job FINISHED successfully
	Select count(*) < 4 from testview	The output file was NOT written.	Job FINISHED successfully

You could refer to the view that was created since you had referred it in the SourceTable named SourceTable-2. So, you could make use of the view in the Conditional SQL.

If this would not have been the case, the view could not have been used in the Conditional SQL for ActionFlow-final-scenario-2. A Decision node essentially decides whether a particular ActionFlow and its corresponding Transform Nodes have to be executed or not.

Multiple Flow

A *Multiple Flow* allows the execution of two or more flows simultaneously. The properties for this node are as follows:

Node Name	Properties	Description
Multiple Flow	Node Name	The user-specific node name

Excel Flow

An *Excel Flow* allows flows with an ExcelNode. The properties for this node are as follows:

Node Name	Properties	Description
Excel Flow	Node Name	The user-specific node name
	Action Flow Name	The name of the action-flow node coded in the excel
	Upload Excels	The excel files that must be uploaded for processing
	Properties	The properties necessary for excel node to execute. These can be: envCode, --tenantCode, --configBucket etc.

Restricted File Upload for Excel Flow Node

Only .xlsx and .xls files can be uploaded in the Excel Flow Node. A warning message displays when the user tries to upload other file formats than .xlsx or .xls as shown in Figure 16.

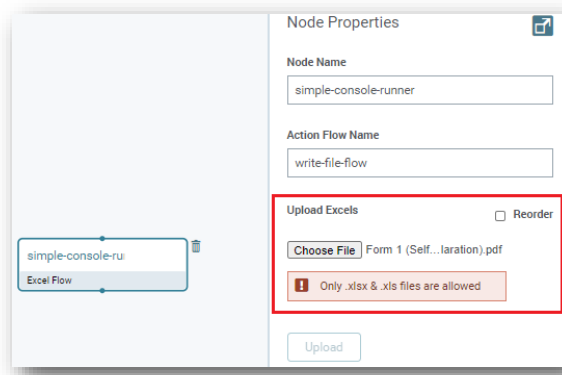


Figure 16: Warning Message

Condition SQL Support for Excel Flow Node

The Excel Flow node supports Condition SQL (Figure 17). The Condition SQL should evaluate to a Boolean value and this value is used to decide the execution of the target action or flow.

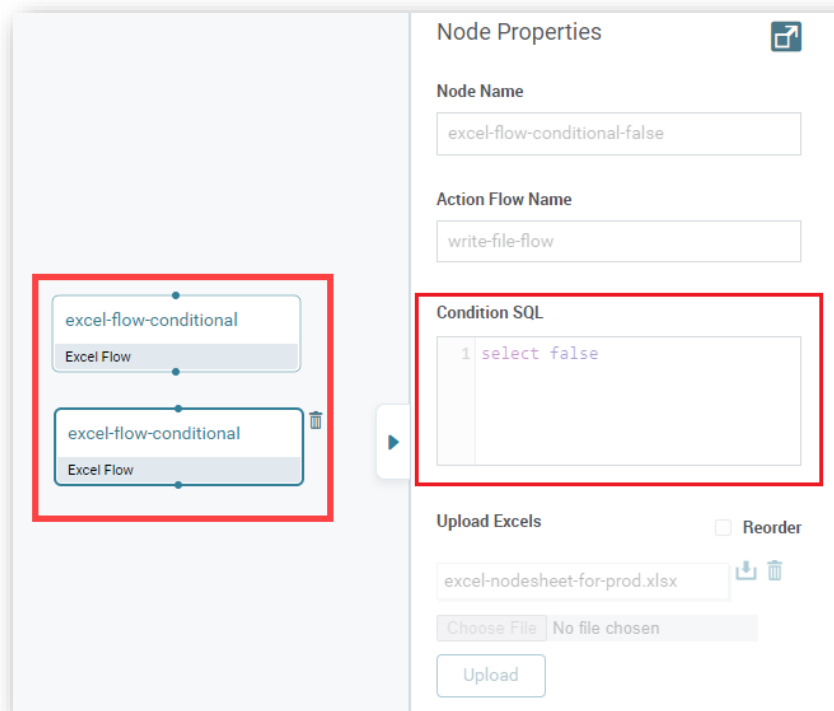


Figure 17: Condition SQL for Excel Node

Jar Action Flow

A *Jar Action Flow* allows flows with a JarActionFlow node. The properties for this node are as follows:

Node Name	Properties	Description
Jar Action Flow	Node Name	The user-specific node name
	Class name	The name of the class of which the JAR has been created
	Method	The name of the method you want to execute
	Arguments	A “string” parameter

Condition SQL Support for Jar Action Flow Node

The Jar Action Flow node supports Condition SQL (Figure 18). The Condition SQL should evaluate to a Boolean value and this value is used to decide the execution of the target action or flow.

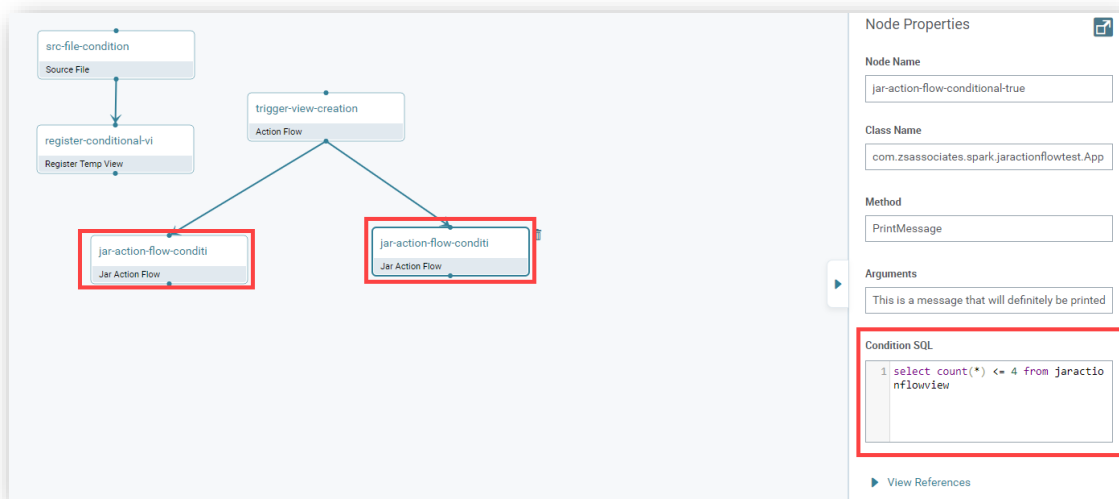


Figure 18: Condition SQL for JarActionFlow Node

Multiple Repository Support

Flow Builder jars can be resolved from multiple repositories.

You can specify multiple comma-separated repository paths using this configuration:

sparkzs.launcher.jars.repositories

The configuration example is as follows:

```
{
  "jars": {
    "excludes": "javax.ws.rs:javax.ws.rs-api,oracle:ojdbc6,jline:jline",
    "packages": "com.zsassociates.spark:spark-zs-weblauncher:#{releaseNumber}#",
    "spark": "spark-launcher",
    "repositories": "file:///localdev-AppManagementS3Bucket/releases/,file:///localdev-s3-bucket/releases/"
  }
}
```



Note: The highlighted repository paths are comma-separated.

Remote Nodes

Remote Nodes are connector nodes which can be used to stitch flows across multiple mappings (Figure 19).

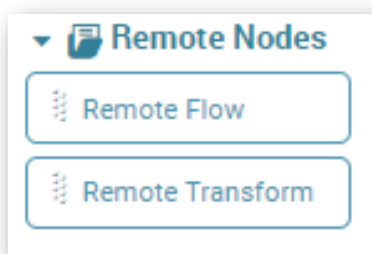


Figure 19: List of Remote Nodes

Remote Flow

A *Remote Flow Node* allows you to refer to any Flow Nodes present in different mappings. The properties for this node are as follows:

Node name	Properties	Description
Remote Flow	Node Name	The user-specific node name
	Contained Mappings	A name of mapping which needs to be referred
	Reference Flow	Action flow to which the node should refer in the above selected mapping

Condition SQL Support for Remote Flow Node

The Remote Flow node supports Condition SQL (Figure 20). The Condition SQL should evaluate to a Boolean value and this value is used to decide the execution of the target action or flow.

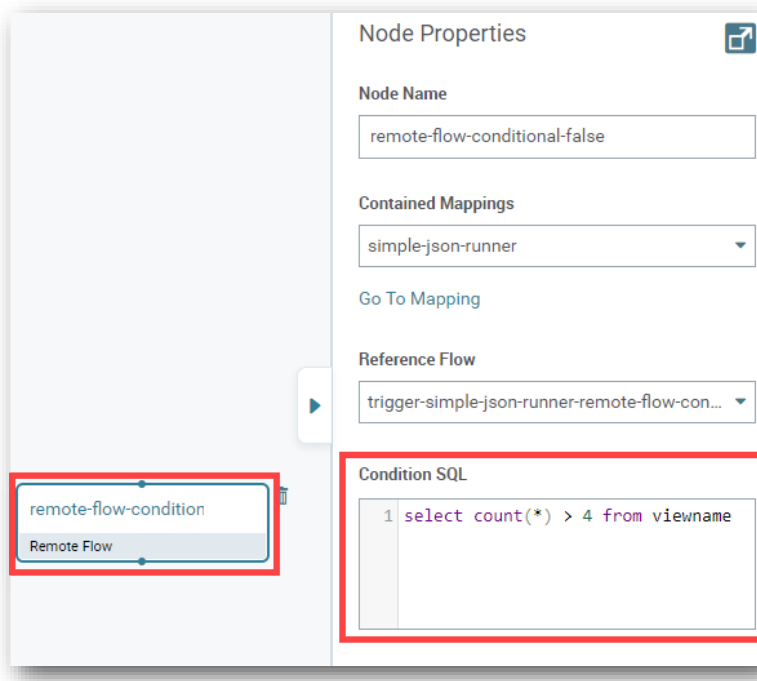


Figure 20: Condition SQL for Remote Flow Node

Remote-Flow Can Have Flow Nodes As Parent

The Remote Flow node can have multiple flow nodes as its parents as shown in Figure 21. A Parent can be any type of Flow node including the Remote Flow node itself.

The Remote Flow node shown in Figure 21 has an ActionFlow node, JarActionFlow node and the Remote Flow node itself as parent. When the MultipleFlow node is triggered, the entire flow is executed.

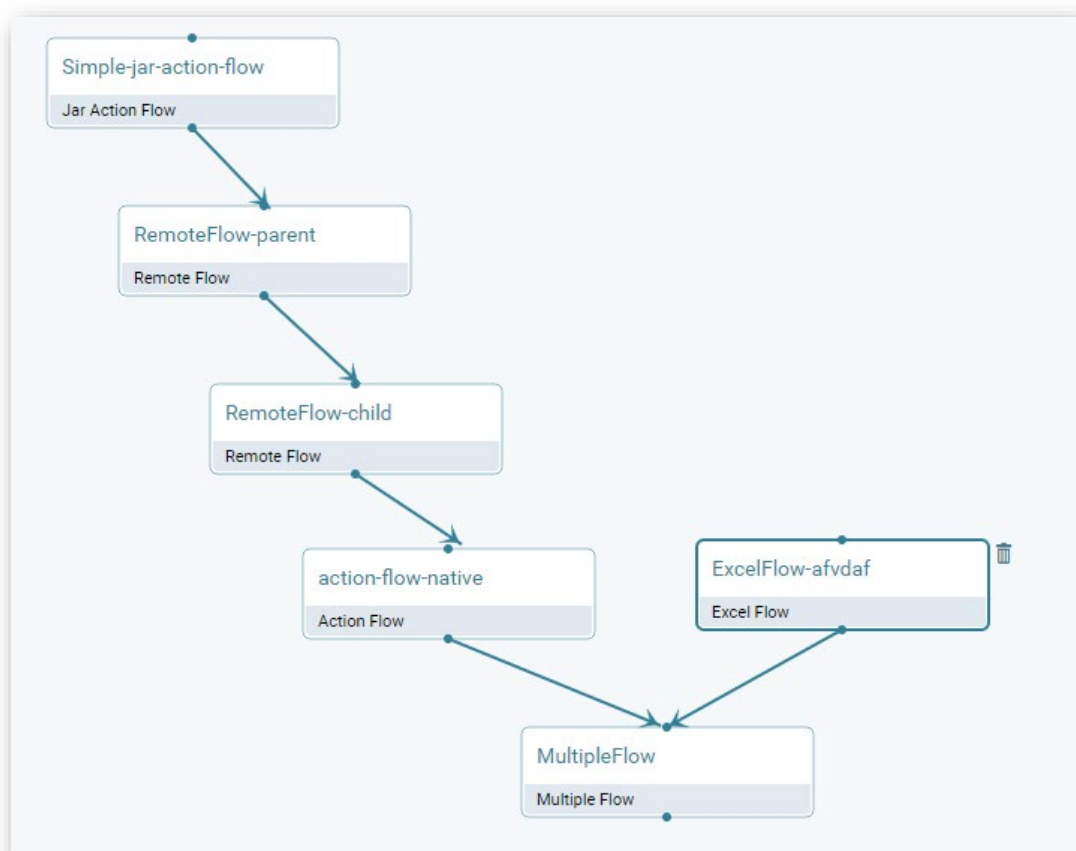


Figure 21: Remote Flow Node with Multiple Flow Node as Parent

Remote Transform

A *Remote Transform Node* allows you to refer to any Transform Nodes present in different mappings. The properties for this node are as follows:

Node name	Properties	Description
Remote Transform	Node Name	The user-specific node name
	Contained Mappings	A name of mapping which needs to be referred
	Reference Transform	Any category of node (from the Source, SQL and Action category of nodes) in the above selected mapping

For every Flow node or a Transform node, users can see where the concerned node has been referred. In Figure 22, the Action Flow named *trigger* has been pointed by a RemoteFlow node in a mapping named *sql-2*. The user can navigate to this mapping from the View References option.

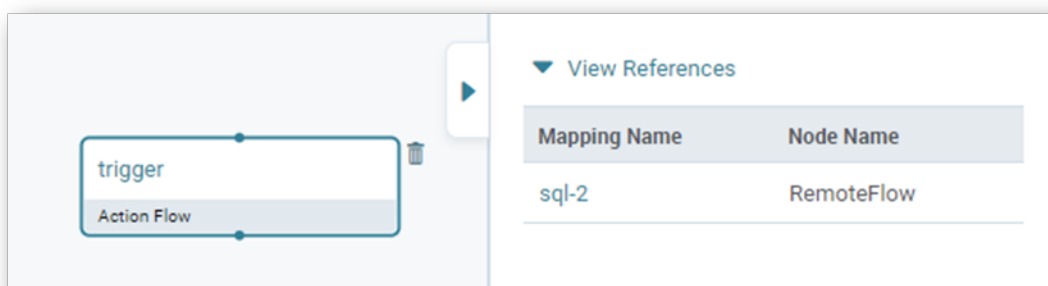


Figure 22: Remote Node References

When you try to delete a node used in a Remote node, a warning message displays stating the deleted node has dependencies on other nodes and urges users to discard changes to continue with the saving operation (Figure 23).

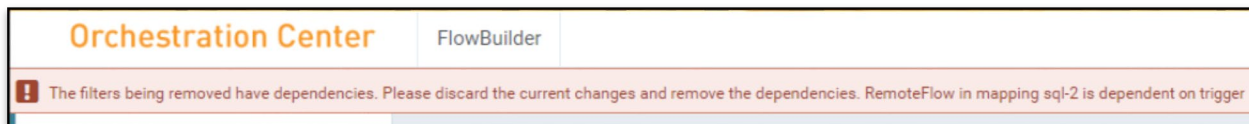


Figure 23: Warning on Deleting Nodes with Dependencies

PySpark Nodes

The following should be done on a local machine.

For any maven project structure, open the pom.xml and make changes in the following sections of the pom file:

```
<groupId>com.dqm.sample</groupId>
<version>0.0.1</version>
<artifactId>spark-zs-python-test-2</artifactId>
<distributionManagement>/ <repository>/<url> Enter-Base-64-Encoded-User-
Token </url></repository></distributionManagement>
```

Refer to [Generate Base 64 Encoded Token](#) for details.

1. Go to C:\go\to\project\project-name\src\main\python and create/edit the python file, say pySparkTestModule.py.
2. Once complete, open the command line at C:\go\to\project\project-name\and do a mvn deploy -DskipTests.
3. Upon a successful deployment, the changes made in the code are pushed to the Flow Builder repository.

Generate Base 64 Encoded Token

1. Obtain the user token. The user should have Administrator privileges.
2. Encode this token using a Base64 encoder (<https://www.base64encode.org/>).
3. Append the token here:
<https://mqnxany0xj.execute-api.us-east-1.amazonaws.com/spark-zs-web/maven-repository/<token-here>/releases>

Flow Builder Side

1. Create a project, mapping and a flow containing the Excel node. The Excel node has been repurposed as the Python node. The following properties need to be added:

```
{
  "__isPython": true,
  "pySparkFunctionName": "main_entry",
  "pySparkModuleName": "pySparkTestModule",
  "pySparkClassName": "PySparkTestClassName"
}
```

2. Once the flow has been created, create a job-definition. Pass the following JSON in the additional properties for the job-definition:

```
{
  "sparkzs": {
    "launcher": {
      "sparkConf": {
        "\"spark.pyspark.python\"": "python3.6",
        "\"spark.submit.deployMode\"": "cluster"
      },
      "sparkPythonModules": {
        "module2": {
          "artifactId": "spark-zs-python-test-2",
          "groupId": "com.dqm.sample",
          "version": "0.0.1"
        }
      }
    }
  }
}
```



Note: The artifactId, groupId and version should be the same as those specified in the pom.xml.

3. Run the job.

Repository Tab for a List of Deployed Artifacts

The *Repository* tab in the Settings page contains a list of all artifacts that have been uploaded to the currently logged in Account. For example, all the artifacts uploaded for the use of the PySpark node, is visible in the Repository tab. This list is sorted in descending order of the user's *Last Updated On* date.

Maven Artifact Upload is a Review Feature for FlowBuilder Repository

All the newly uploaded artifacts in the FlowBuilder repository will, by default, be disabled for usage in the Execution Environment. However, for Development Environment, they will continue to be available, irrespective of whether they are enabled/disabled for the Execution Environment.

In order for these artifacts to be enabled, they require a review process. The review process will be as follows:

1. An AccountAdmin or an EnvironmentAdmin uploads a maven artifact to the Development Environment (Figure 24).
2. This newly deployed artifact will be disabled by default for the Execution Environment. However, users will be able to use the artifact in the Development Environment.

Version : 11.0.8.0 | Current Role : Account Admin

Account Environment Strategy Configuration **Repository**

Artifact	Created On	Updated On	Enabled For Execution Env.	Review Comment
com.zsassociates.spark:spark-zs-pyspark-test-LOCAL-ASSET-1.0.0.3	Nov 5, 2020 10:12:57 AM	Nov 5, 2020 10:12:57 AM	<input type="checkbox"/> NO	

Figure 24: A New Maven Artifact Uploaded to the Development Environment

3. The client teams must submit a support ticket for the artifact to be reviewed.
4. Once the review is complete and the artifact is approved, the SuperAdmin/Support Team enables this artifact along with the appropriate review comments (Figure 25, Figure 26, Figure 27).

Version : 11.0.8.0 | Current Role : Super Admin

Account Environment Strategy Configuration Asset Subscription **Repository**

Artifact	Created On	Updated On	Enabled For Execution Env.	Review Comment	Actions
com.zsassociates.spark:spark-zs-pyspark-test-LOCAL-ASSET-1.0.0.3	Nov 5, 2020 10:12:57 AM	Nov 5, 2020 10:12:57 AM	<input type="checkbox"/> NO		<input type="button" value="Enable"/>

Figure 25: SuperAdmin Corresponds the Enable/Disable Button to Each Artifact

Repository

Created On	Updated On	Enabled For Execution Env.	Review Comment	Actions
Nov 5, 2020 10:12:57 AM	Nov 5, 2020 10:29:43 AM	<input checked="" type="checkbox"/> YES	Artifact reviewed and can be used in execution environment zs/vb20180	<input type="button" value="Disable"/>

Figure 26: SuperAdmin Enables the Artifact Along with a Review Comment

Version : 11.0.8.0 | Current Role : Account Admin

Account Environment Strategy Configuration **Repository**

Artifact	Created On	Updated On	Enabled For Execution Env.	Review Comment
com.zsassociates.spark:spark-zs-pyspark-test-LOCAL-ASSET-1.0.0.3	Nov 5, 2020 10:12:57 AM	Nov 5, 2020 10:29:43 AM	<input checked="" type="checkbox"/> YES	Artifact reviewed and can be used in execution environment zs/vb20180

Figure 27: Users See the Artifact Enabled for Usage in the Execution Environment



Notes:

- **Uploading a Maven Artifact:** A Maven artifact cannot be re-uploaded with the same version once it has been enabled. If a Maven artifact is uploaded in the asset account, and if a client subscribes to that asset, the client will have access to the asset's maven-artifacts by default.
- **For Development Environment:** Maven artifacts can be tested and developed irrespective of whether they are enabled/disabled for the Execution Environment.
- **For Execution Environment:** Maven artifacts cannot be executed in the Execution Environment unless they have been enabled. Maven artifacts cannot be uploaded by an Execution Environment user.

Creating a Flow in Flow Builder

Adding a Project

Navigate to the Flow Builder application. If this is the first project being created for an instance, you would see Figure 28:

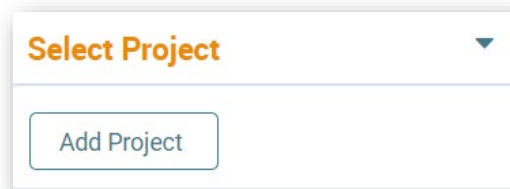


Figure 28: Project Drop-Down List When no Project Already Exists

However, if you want to add a new project to a pre-existing list of projects, then follow the following steps:

1. Click the project name.
2. Click the **Add Project** button at the bottom of the list (Figure 29).
3. In the dialog box, enter the name of the project (Figure 30).
4. Click **OK**. A new project with the specified name is created.

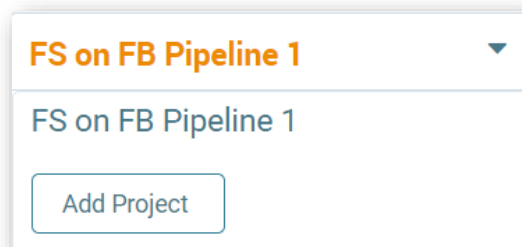
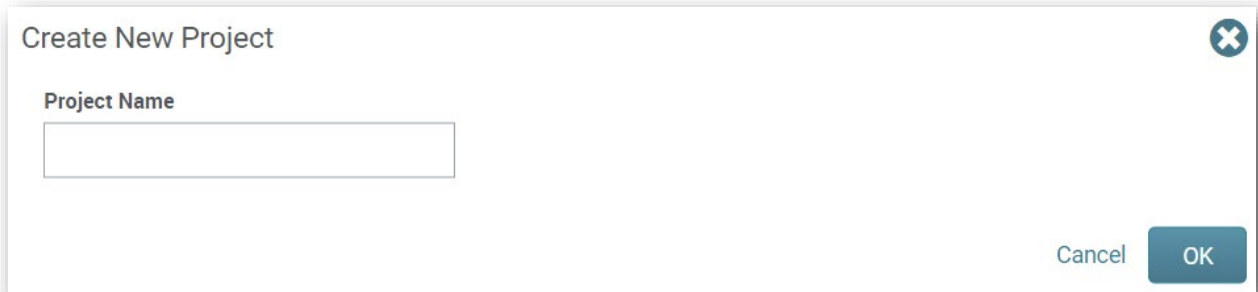


Figure 29: Project Drop-Down List When the Project Name is Clicked




The dialog box is titled "Create New Project" and has a close button (X) in the top right corner. It contains a label "Project Name" above a text input field. At the bottom right, there are two buttons: "Cancel" and "OK".

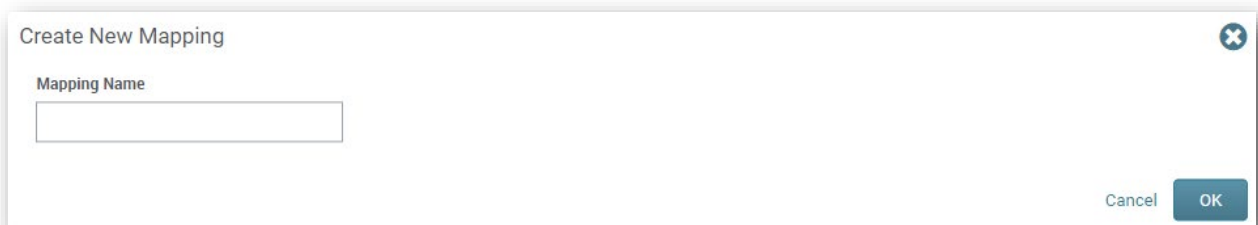
Figure 30: Create New Project Dialog Box

Adding a Mapping

In order to create a mapping for the project just created, perform the following steps:


1. Click the Create Mapping  icon.
2. In the dialog box, enter the name of the mapping (Figure 31).
3. Click *OK*.

A new mapping with the provided name is created.



The dialog box is titled "Create New Mapping" and has a close button (X) in the top right corner. It contains a label "Mapping Name" above a text input field. At the bottom right, there are two buttons: "Cancel" and "OK".

Figure 31: Create Mapping Dialog Box



Note: Flow Builder provides the functionality to create nested mappings which are mappings within mappings.

Updating a Mapping with Edit Mode

Once you create a mapping, a lock for the mapping is enabled (Figure 32). Once you have created and saved the flow, you can commit your changes and then the lock is disabled.

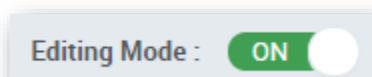


Figure 32: Editing Mode: ON

However, you need to take the lock manually if you make changes in any of the pre-existing mappings.



Warning: Flow Builder R4 does not support manually unlocking the flows.


Adding, Editing and Saving Flows

To add, edit and save a flow:

1. Drag and drop the nodes from the palette into the DAG Building area.
2. Double-click each node to change node properties. All the properties for a node are listed in the right property panel.
3. Once all the required nodes have been edited as needed, join the nodes to create a logical flow.
4. For a pre-existing mapping, go to the node you want to edit, double-click the node to open the right property panel, and obtain the lock to edit the nodes.
5. Once the changes are complete, click **Save** to create a draft version.

Committing Changes

Once you have added and edited the flows you have created, you can commit the changes by navigating to the Project Level and clicking the **Commit** button (Figure 33). A dialog box containing the changelist displays (Figure 34). The changelist includes the mappings you have edited/taken a lock on since the last successful commit. Clicking these mappings also navigates you to the Project Level.



Note: This changelist is visible even when you click **Discard** to delete your changes.

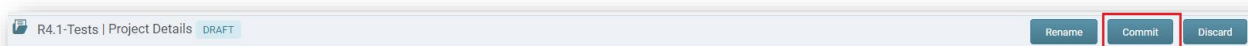


Figure 33: Click **Commit** Button to Commit the Changes

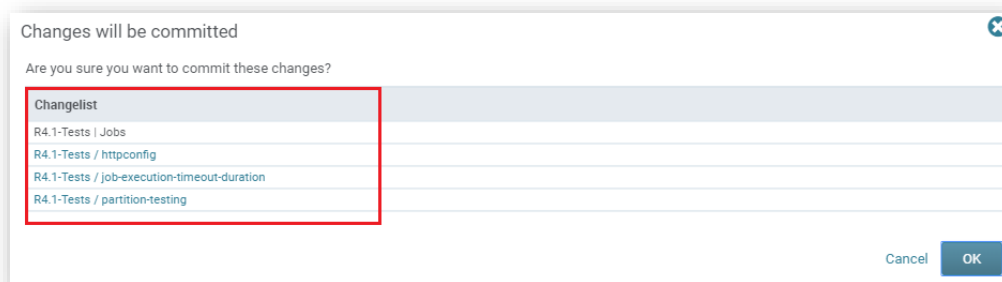



Figure 34: Changelist Visible before Performing Final Commit



Note: For the projects you have taken a lock or made any changes to but have not committed, a **DRAFT** tag is visible beside the name of the project (Figure 35).

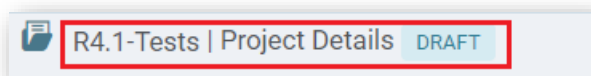
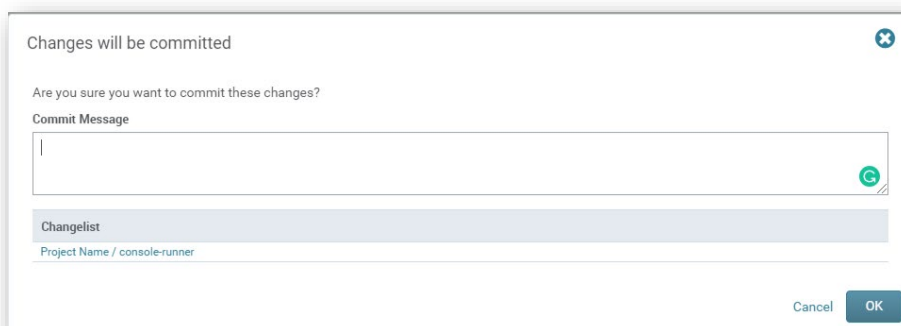


Figure 35: **DRAFT** Tag Beside Uncommitted Project

Provide a Commit or Draft Message

Users must provide a message specifying the changes that they have done by specifying a reason for the commit or draft (Figure 36). This message is mandatory so that the user is not allowed to commit or discard without it.

Review the list of mappings where the user has performed changes. The user can navigate to any mapping in this list by selecting it.



Changes will be committed

Are you sure you want to commit these changes?

Commit Message

Changelist

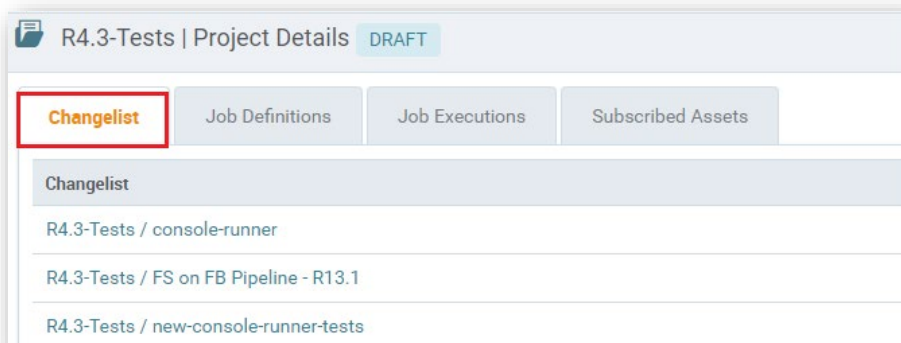
Project Name / console-runner

Cancel OK

Figure 36: Commit Message and Changelist when Committing/Discarding

View Mappings with the Changelist

Users can view the list of mappings they have committed through the Changelist tab which is at the Project level (Figure 37).



R4.3-Tests | Project Details DRAFT

Changelist Job Definitions Job Executions Subscribed Assets

Changelist

R4.3-Tests / console-runner

R4.3-Tests / FS on FB Pipeline - R13.1

R4.3-Tests / new-console-runner-tests

Figure 37: Changelist Tab at the Project Level

Discarding Changes

Discard Saved Changes

If you are unsatisfied with your changes, you can discard them to roll back to the last uncommitted version of your flow. Depending upon whether you have saved your changes or not, the discard operation can be bifurcated as:

If you have saved your changes and want to discard them, navigate to the Project level. Click the **Discard** button and then click **OK** (Figure 38).

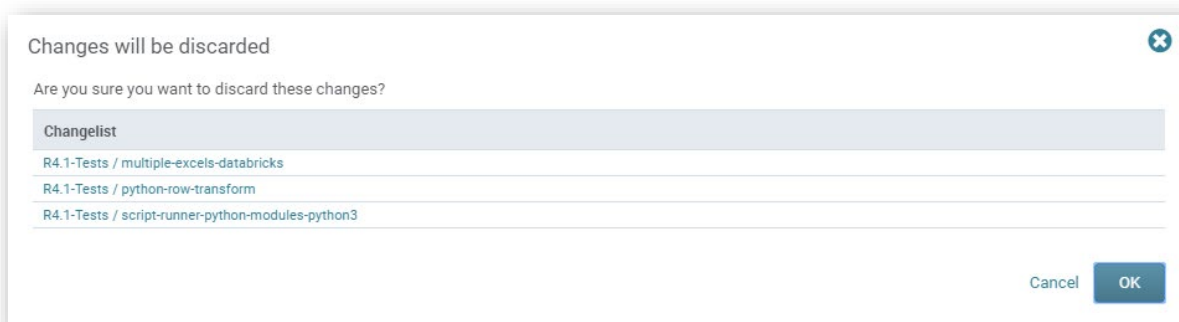


Figure 38: Dialog Box for Discarding Changes at the Project Level

Discard Unsaved Changes

If you have not saved your changes and try to navigate to another mapping, a dialog box displays giving you the option to **Cancel** your navigation action or **Discard Changes and Continue** (Figure 39). Choosing an option allows you to navigate to the other mapping.

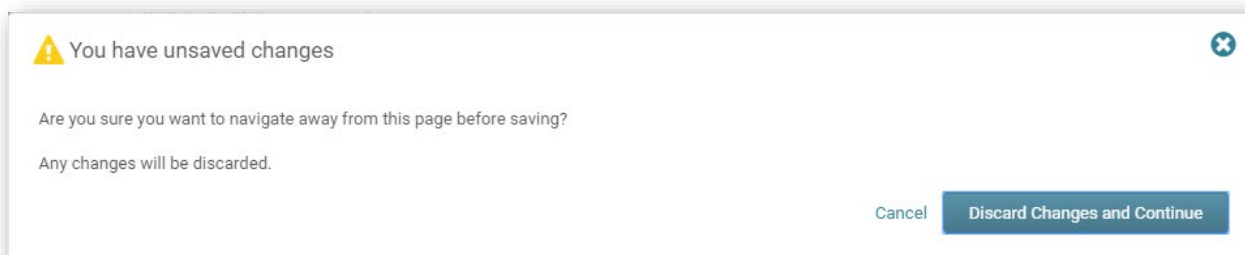


Figure 39: Dialog Box while Discarding Unsaved Changes

Renaming a Project/Mapping

To renaming a project:

1. Click the **Rename** button on the top right corner once you have navigated to the project-level (Figure 40).
2. Enter the new Project Name in the dialog box (Figure 41).
3. Click **OK**.

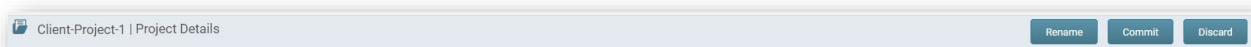


Figure 40: Rename Button for Renaming a Project

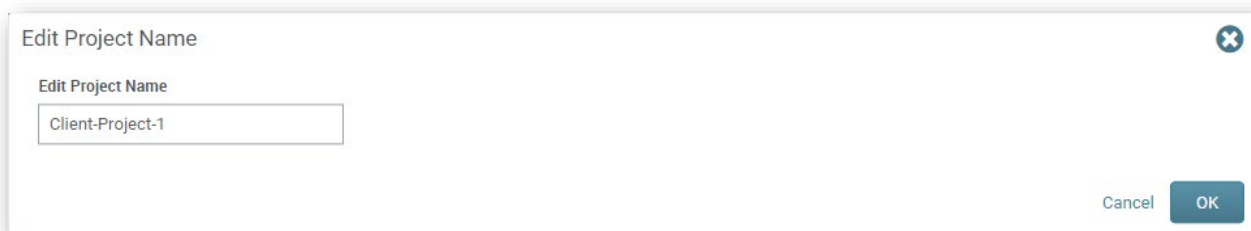



Figure 41: Dialog Box for Renaming a Project

To rename a mapping:

1. Select the mapping and click the *Edit* icon  next to the name of the mapping on the top panel.
2. Change the name of the mapping in the dialog box (Figure 42).
3. Click *OK*.

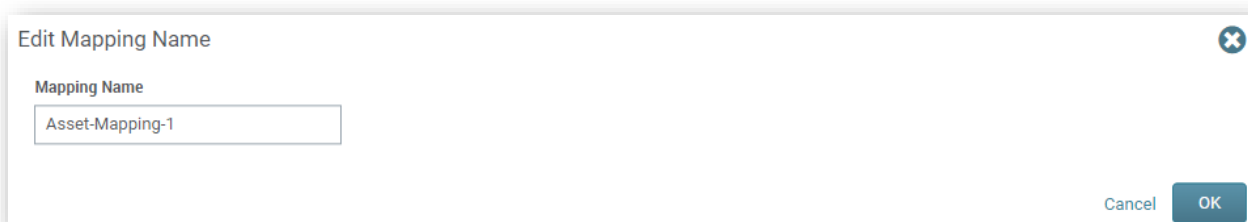


Figure 42: Dialog Box for Renaming a Mapping

Project Versioning

Within every client account or instance, users can create multiple projects. A project in Flow Builder is like having a folder in a conventional file system. Within a single project, users can create multiple mappings. When creating a flow, two versions of the flow get created.

- **Draft Version:** A *Draft Version* is created when the user takes a lock on a mapping. Another draft version is created when the user creates and saves a flow. For draft versions, the changes are only visible to the user who is creating and modifying the flows.
- **Main Version:** Once user saves a flow/mapping and commits the changes, a *Main Version* gets created. For main versions, the changes made by the user are visible to everyone.

The Project Versioning section includes the Commits, Drafts and Manage Branching tabs.

Manage Branching Tab

From the Manage Branching Tab, users can create branches and releases (Figure 43). The current branch/release on which the user is working on is highlighted and tagged *Current*.

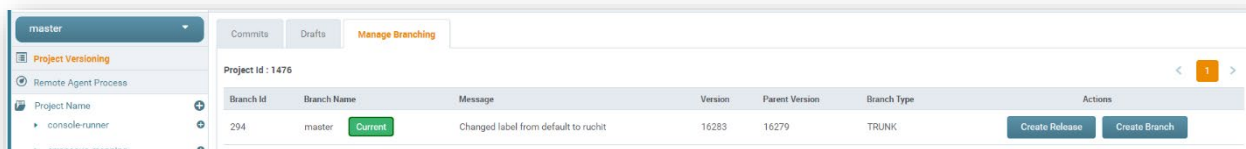


Figure 43: Project Versioning Section: The Manage Branching Tab

Commits Tab

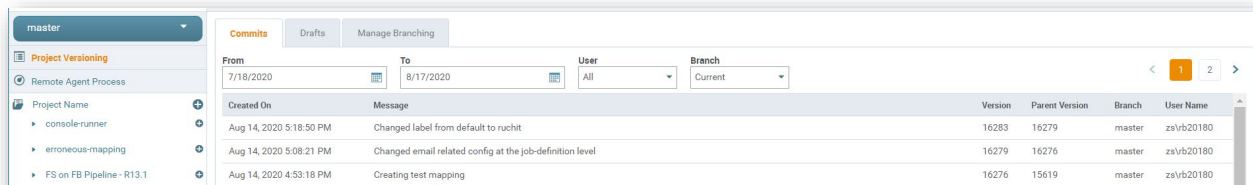
From the Commits Tab, users can view every commit completed in a project (Figure 44). While committing/ discarding a project change, the user must provide a reason for the change in a Commit or Discard Message to complete the commit/discard process.

From the Commits Tab, users can also:

- Filter the commit list to view commits within a given date range by selecting the *from* and *to* dates.
- View the commits of either themselves or all other users by selecting either *Me* or *All* in the User dropdown.
- View the current or all commits completed on a given branch by selecting either *Current* or *All* in the Branch dropdown.



Note: By default, the date range is *30 days*, User is *All* and Branch is *Current*. The commit list is paginated.



Project Versioning		Commits		Drafts		Manage Branching	
From	To	User	Branch	Created On	Message	Version	Parent Version
7/18/2020	8/17/2020	All	Current	Aug 14, 2020 5:18:50 PM	Changed label from default to ruchit	16283	16279
				Aug 14, 2020 5:08:21 PM	Changed email related config at the job-definition level	16279	16276
				Aug 14, 2020 4:53:18 PM	Creating test mapping	16276	15619

Figure 44: Project Versioning: Commits Tab

Drafts Tab

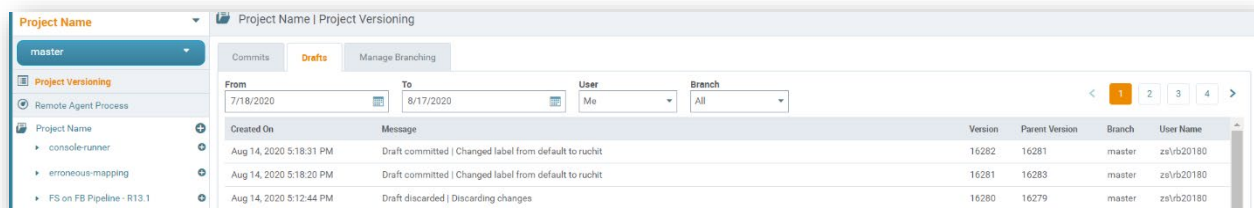
From the Drafts tab, users can take a lock, and create a DAG in the mapping. If there are uncommitted drafts, the Message column shows *DRAFT*. Once the draft has either been committed or discarded, the message changes to *Draft committed* or *Draft discarded* respectively along with the commit/discard message that the user enters as can be seen in Figure 45.

From the Drafts tab, users can also:

- Filter the drafts list to view commits within a given date range by selecting the *from* and *to* dates.
- View the drafts of either themselves or all other users by selecting either *Me* or *All* in the User dropdown.
- View the current or all drafts completed on a given branch by selecting either *Current* or *All* in the Branch dropdown.



Note: By default, the date range is *30 days*, User is *All* and Branch is *Current*. The commit list is paginated.



Created On	Message	Version	Parent Version	Branch	User Name
Aug 14, 2020 5:18:31 PM	Draft committed Changed label from default to ruchit	16282	16281	master	zs/vb20180
Aug 14, 2020 5:18:20 PM	Draft committed Changed label from default to ruchit	16281	16283	master	zs/vb20180
Aug 14, 2020 5:12:44 PM	Draft discarded Discarding changes	16280	16279	master	zs/vb20180

Figure 45: Project Versioning: Drafts Tab

Parameterizing Flows

Use the \${env} Template

In instances where you do not want to specify the value of a variable/property due to some privacy reason, such as your database password, you may use a placeholder variable instead of the actual password. The \${env} template for such a variable is used as follows:

If you specify your DB password with the value *abcd1234* using the \${env} template, then, for the property *Password*, the value is *\${env.pwd}*, where *pwd* is the name of the variable.

Use the Environment Variable File

Since you have specified just the name of the variable in the flow, the actual value of the variable is specified in the environment variable file. Present with a .conf extension, this environment variable configuration file is specified at the project level.



Example: If you use environment variables in a project named *DemoProject*. In the env.conf file, the configuration is specified as follows:

```
{
  "pwd": "abcd1234",
  "inputPath": "s3://path/to/input/file"
}
```

Save this configuration file to an S3 location.

Navigate to the project level by clicking **Settings** -> **ConsumerApp Tab** -> **DemoProject**. Inside the file option, enter the path where you have saved the env.conf file (Figure 46).

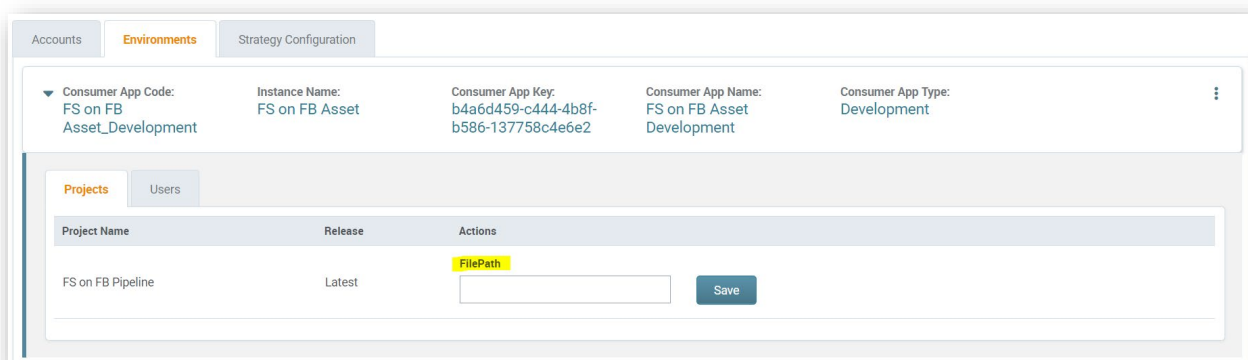


Figure 46: Providing the Path to the env.conf File under the Environments Tab



Warning: Though Flow Builder allows you to use environment variables using the placeholder `#{appconfig}`, new flows should not be created using Excel.

Override Environment Configuration at the Job Definition Level

Flow Builder provides support for using the `env.conf` at the *job definition* level. The configuration present at the job definition level takes precedence over the one present at the project level.

The following property can be passed at the job level to add/override the configuration that is present at the project level.

```
{
  "sparkzs": {
    "launcher": {
      "coreConf": {
        "envVariableConfigOverrideUrl": "s3://path-to-bucket/envOverride.conf"
      }
    }
  }
}
```

Override Environment Variables at Job Execution level

Environment variables can be overridden at the *job execution* level by providing the configuration JSON as the job execution additional properties.

The contents of the environment file at the project level is as follows:

```
{
  "srcFileBase": "s3://path/to/src/file/input-single-row.csv",
  "outputFileBase": "s3://path/to/dest/output-multi-row-base"
}
```

To override the `srcFileBase` variable for every execution, use the following configuration that is passed at the job-execution additional properties:

```
{
  "sparkzs": {
    "launcher": {
      "coreConf": {
        "envVariableConfigOverride": {
          "srcFileBase": "override-variable-value"
        }
      }
    }
  }
}
```

Release Management

Creating Project Releases and Branches

Once you have created the flows and respective jobs and are satisfied with the results, click the *Manage Versioning* icon [img alt="Manage Versioning icon" data-bbox="310 280 345 305]] next to the project name. The option to create releases and branches displays. If you want to create a release, start from the master branch and click the **Create Release** button (Figure 47) to enter a name of the new release (Figure 48). For example, you could use R1.

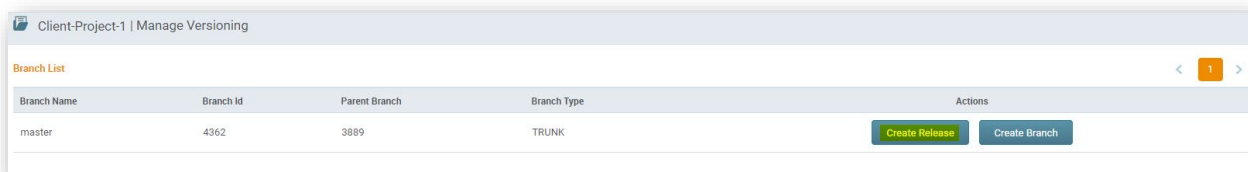


Figure 47: Create Release Button under the Manage Versioning Section

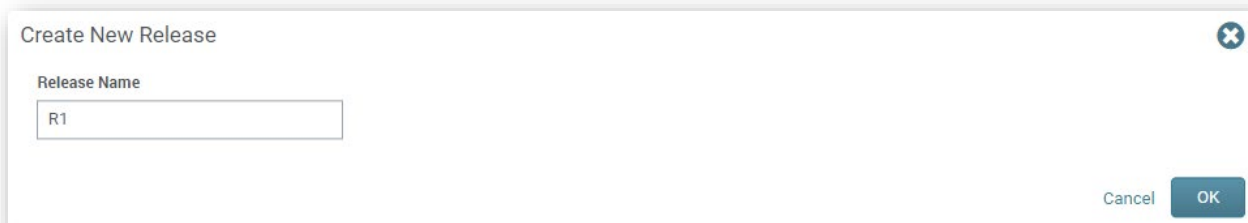


Figure 48: Create a New Release from a Master Branch

However, if you want to create a new branch from a pre-existing release, say R1, then click the **Create Branch** button to enter the name of a new branch, say R1-B1 (Figure 49 and Figure 50).

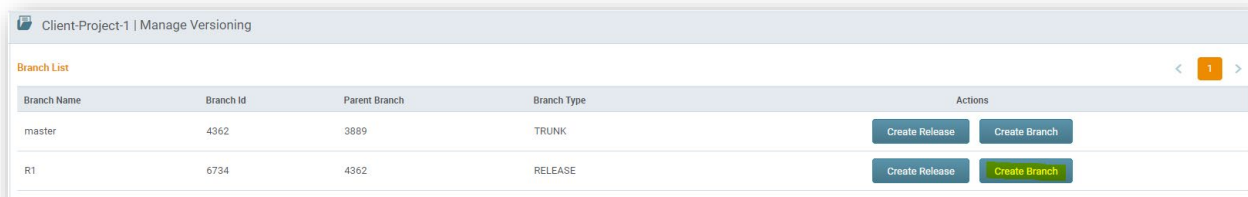
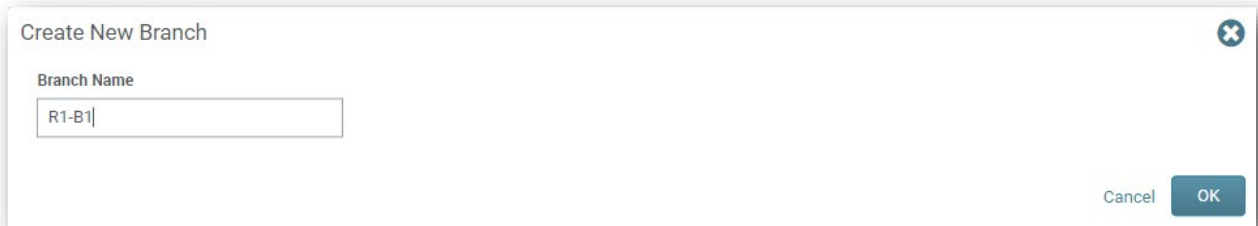


Figure 49: Create Branch Button under the Manage Versioning Section




Create New Branch

Branch Name

R1-B1

Cancel OK

Figure 50: Dialog Box to Create a New Branch (Branch R1-B1 from Release R1)



Note: A branch can be created only from a release.

Tagging Project Releases to an Execution Environment

To tag a created release to an Execution Environment:

1. Go to Settings and select the **Environments** Tab.
2. Expand the Execution Environment.
3. Beside the project, click the **Release** dropdown.
4. Select the release you want the Execution Environment to point at (Figure 51).
5. Click **Tag**. Now, this Execution Environment points to the tagged release.



Demo project

Select

Tag

FilePath

Save

Figure 51: Dropdown to Select a Release to be Tagged to the Execution Environment

Switching Between Releases and Branches

Once you have created the required releases and branches, navigate back to the project level.

To switch between various Branches and Releases:

1. Below the name of the project, the *master* branch is selected by default (Figure 52). Click the drop-down.
2. Select the *Trunk*, *Release* or *Branch* tab.
3. Select a specific Release or Branch to see the contents based on the selection.

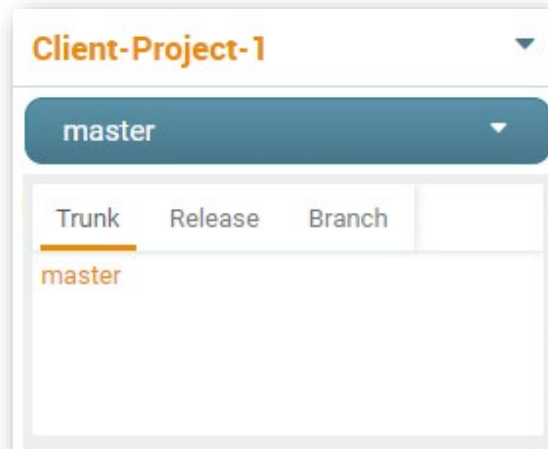


Figure 52: Menu to Toggle between Releases

Creating a Hotfix Release

If you have released a flow but want to perform some changes to the released version, you need to create a hotfix release.

To create a hotfix release:

1. Use a pre-existing release, say R1 and a pre-existing branch, say B1.
2. Create a new branch, say R1-B1.
3. Navigate to the branch R1-B1 and make the required changes.
4. Navigate to the Manage Versioning page.
5. From branch R1-B1, create another release R2. R2 is now the hotfix release.

Remote Agent Process

Users can access the Remote Agent from the left panel as shown in Figure 53.

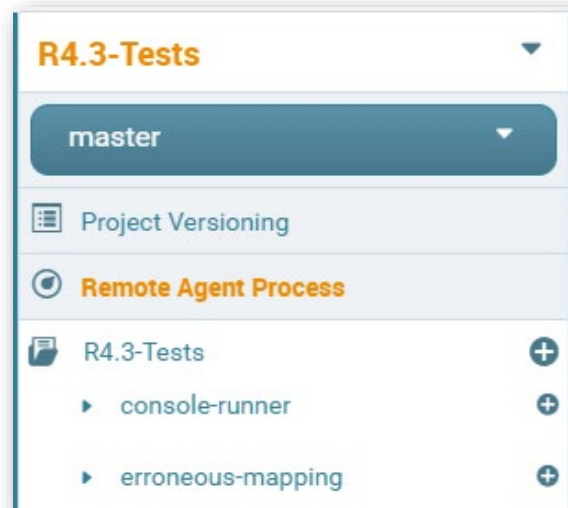


Figure 53: Accessing the Remote Agent

A *Remote-Agent* is a service in which Flow Builder jobs execute. A *Remote-Agent Strategy* is how the remote-agent is configured. This configuration includes how the user wants to configure the remote-agent and the EMR cluster where this remote-agent is hosted, and where the Flow Builder jobs are executed. Flow Builder allows users to configure different instances with different remote-agent strategies.



Note: The remote-agent configuration can be maintained and updated only by an Administrator. Currently, only the SD Support Team can edit and maintain this configuration.

Remote-Agent Strategies Supported

Remote-Agent Strategy Name	Description
Global Level (Global SaaS)	This is a master remote-agent strategy configuration applicable across all accounts and environments in the application that have chosen the Global SaaS Remote-Agent Strategy.
Account Level (PER_INSTANCE)	This remote-agent strategy configuration allows you to edit the settings. This strategy is applicable across all Environments (ConsumerApps) in the instance that have chosen the PER_INSTANCE Remote-Agent Strategy.
Environment Level (PER_CONSUMER_APP)	This remote-agent strategy is applicable at the Environment Level (Development and Execution) and is relevant for all the Environments/ConsumerApps that have chosen the PER_CONSUMER_APP Remote-Agent Strategy.

Cluster Backend Supported

Cluster Backend Name	Description
AWS_EMR_BACKED	Execution is done on the EMR Cluster.
DATABRICKS	Execution is done on the Databricks Cluster.
LOCAL_DEV	Execution is done on the user's local machine.



Note: Depending upon the type of Strategy/Cluster backend selected, the remote-agent configuration property changes. This is maintained by the SD Support Team.

Execution Types Supported

Execution Type Name	Description
POLL	The remote-agent continually polls Flow Builder at regular intervals to check whether there are any jobs waiting to be selected and executed.
TRIGGER*	Jobs only trigger when they are run manually, and the remote-agent does not poll for unexecuted jobs.
UNKNOWN*	Users configure how they want the job execution to occur.

* To be supported in a subsequent release. Not supported as of R4.1

Labelled Remote Agent

A *Labelled Remote-Agent* is a tag given to a remote-agent with a pre-defined set of configurations. For example, you might use this if you have two job-definitions and you want to run them on two differently configured clusters, each having different computing capabilities. For R3, Flow Builder supports job executions on a remote-agent running on just one EMR Cluster. For R4, Flow Builder supports running multiple remote-agents, each of a different label, with custom configurations.



Note: The custom configuration depends on your requirement and on the Administrator (currently SD Support Team) who creates the labelled remote-agents.




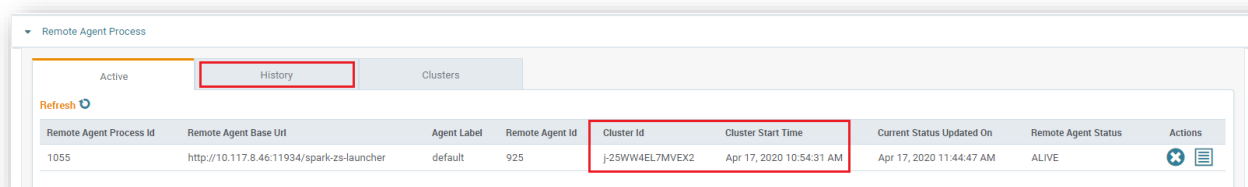
Example: You have two job-definitions JD1 and JD2, with JD1 being a resource intensive task requiring a cluster with a master node of m5.xlarge and 2 slave nodes of m5.4xlarge configuration. The SD Support Team Administrators create a labelled remote-agent with a client-specified name accommodating your configuration requirement. JD1 runs on the client-specified labelled remote-agent, and JD2 runs on a default remote-agent. This default remote-agent is a kind of labelled remote-agent with some default set configurations.

Using the Remote-Agent Process Tabs

As is visible in Figure 54, the remote-agent accordion houses the tabs: Active, History and Cluster.

- **Active Tab:** This tab contains the list of all distinct-labelled remote-agents. For R4.1, two new columns were added: Cluster ID and Cluster Start Time, thereby reducing the dependency of having access to the AWS Console. This helps provide relevant information during debugging processes, if needed.
- **History Tab:** Once a remote-agent is terminated (either automatically or manually), the entry goes into the History tab. A total of 50 legacy remote-agent entries are visible in the History tab.
- **Cluster Tab:** The active cluster's ID, State and Start-Time (only for the duration of the cluster's lifetime) are visible in this tab.

In the Remote-Agent accordion, only one entry for remote-agent per label is allowed. The logs for this entry can be visible by clicking the  icon.





Remote Agent Process Id	Remote Agent Base Uri	Agent Label	Remote Agent Id	Cluster Id	Cluster Start Time	Current Status Updated On	Remote Agent Status	Actions
1055	http://10.117.8.46:11934/spark-zs-launcher	default	925	j-25WW4EL7MVEX2	Apr 17, 2020 10:54:31 AM	Apr 17, 2020 11:44:47 AM	ALIVE	

Figure 54: Remote Agent Entries on the UI

Shutting Down the Remote-Agent Manually

Once all the jobs have run and there is no other job pending for execution, you can manually shutdown the remote-agent by clicking the  icon and then clicking **Yes**.

The remote-agent shuts down and the entry migrates to the *History* tab.



Auto Termination Feature: If the remote-agent is idle for a **specified** amount of time (by default, 60 minutes), it terminates automatically.



Note: In cases where the remote agent is shut down improperly, the remote-agent status is highlighted, and users are expected to contact the SD Support Team to terminate the remote-agent. This is done using the command:

```
swlh-force-mark-shutdown -label=<label_name> command.
```

For remote-agents terminated properly, the Super Admin can also trigger the `swlh-force-mark-shutdown` by clicking under the remote-agent accordion.

Setting the Maximum Runtime Duration for the Remote-Agent

Flow Builder R4.1 allows you to set the maximum runtime duration of the remote-agent. Setting the maximum runtime duration to a specified time, say 8 hours, automatically terminates the remote-agent irrespective of whether a job is running on the remote-agent or not. This can be set using the following property.

```
{
  "sparkzs": {
    "launcher": {
      "maxActiveDurationInMinutes": 5
    }
  }
}
```



Notes:

1. The time duration specified is in minutes.
2. If user does not want to put a cap on the remote-agent activity time, user can simply enter '0' as the value of the "maxActiveDurationInMinutes" property.
3. If the above property is not specified in the remote-agent json or at the tenantApp level, the default value is set to 24 * 60 minutes, i.e. 24 hours (1 day).
4. The `sparkzs.launcher.maxActiveDurationInMinutes` is applicable to the remote agent irrespective of EMR or Databricks backend.

Two scenarios arise if the remote-agent terminates while a job has been picked for execution or is already executing:

1. The job gets picked up seconds before the remote-agent is about to terminate. Then, the state of the job remains PICKED.
2. If the job is in the RUNNING state and the remote-agent then shuts down, the job state gets updated to the KILLED state.

Irrespective of the state of the job, the remote-agent shuts down, as this feature is a safeguard against runaway remote-agents and runaway clusters. It provides a second layer of protection to identify and terminate the remote-agents in case the auto-shutdown feature fails.

Shutting Down the Remote-Agent via JarActionFlow

Users can create a node that can shut down the remote-agent using a JarActionFlow in two ways:

1. Create a JarActionFlow and attach it to an already existing flow. This shuts down the remote-agent once the parent job has executed successfully.
2. Create a modular mapping with the single JarActionFlow node which implements the *swlh-force-shutdown* command (Figure 55). The user can then create a job out of it to shut down the remote-agent. This job can then be chained to any of the pre-existing jobs. After successful execution of all previous jobs in line, this *shutdown-remote-agent* job executes and terminates the remote-agent on which the job-chain was getting executed.

Node Name	shutdown-remote-agent
Class Name	com.zsassociates.spark.commandrunner.CommandRunner
Method	execute
Arguments	swlh-force-shutdown

Figure 55: Implementing a Jar Action Flow Node

The JarActionFlow node properties are as follows:

Class Name: com.zsassociates.spark.commandrunner.CommandRunner

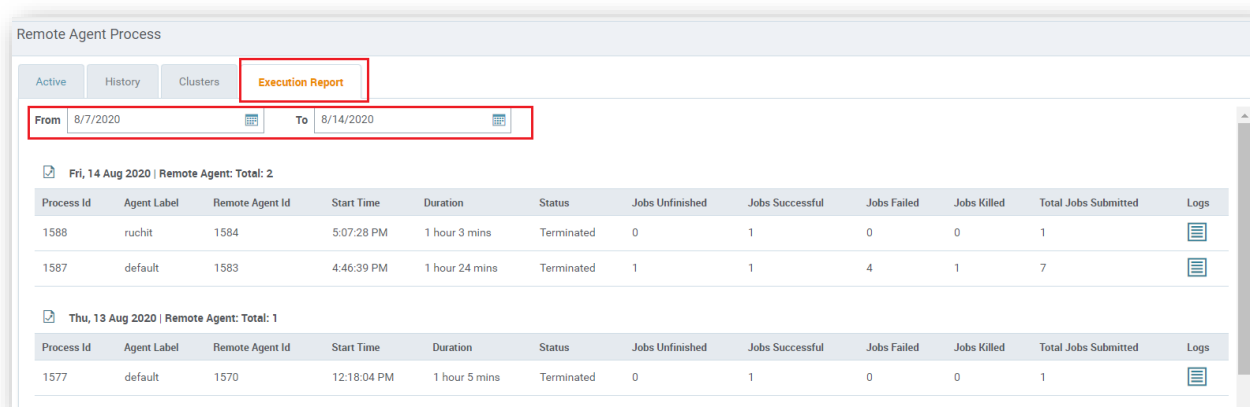
Method: execute

Arguments: swlh-force-shutdown

Execution Report

With FlowBuilder R4.3, users can see a consolidated Execution Report of their job-executions between 2 specified dates.

This report can be found in the Remote-Agent Process section in the Execution Report tab as shown in Figure 56.






Remote Agent Process											
<div>Active History Clusters Execution Report</div>											
<div>From: 8/7/2020 To: 8/14/2020</div>											
Fri, 14 Aug 2020 Remote Agent: Total: 2											
Process Id	Agent Label	Remote Agent Id	Start Time	Duration	Status	Jobs Unfinished	Jobs Successful	Jobs Failed	Jobs Killed	Total Jobs Submitted	Logs
1588	ruchit	1584	5:07:28 PM	1 hour 3 mins	Terminated	0	1	0	0	1	
1587	default	1583	4:46:39 PM	1 hour 24 mins	Terminated	1	1	4	1	7	
Thu, 13 Aug 2020 Remote Agent: Total: 1											
Process Id	Agent Label	Remote Agent Id	Start Time	Duration	Status	Jobs Unfinished	Jobs Successful	Jobs Failed	Jobs Killed	Total Jobs Submitted	Logs
1577	default	1570	12:18:04 PM	1 hour 5 mins	Terminated	0	1	0	0	1	

Figure 56: Remote Agent Process: Execution Report Tab

The Execution Reports page allows the user to:

- Select a specific date range to view the statistics.
- View the number of remote-agents that were provisioned, along with the job-execution stats for each remote-agent for a given day.
- View the logs for a given remote-agent.

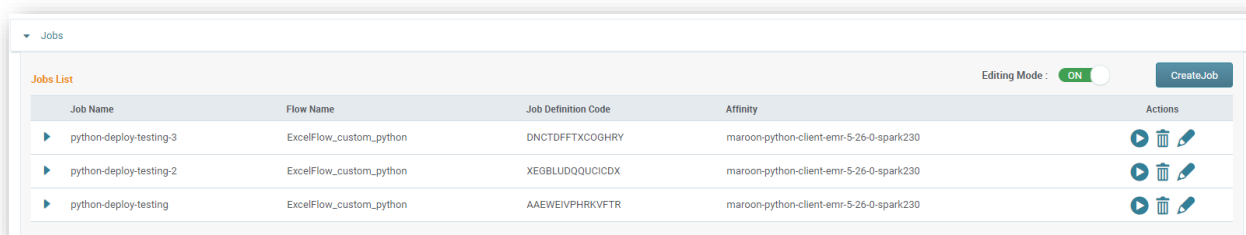
Jobs

Defining and Creating a Job Definition

A *Job-Definition* is a way of registering a Flow node in a mapping. Flow nodes trigger an action in a mapping. Job-Definition creation is a one-time process which can be executed by the user multiple times. Upon successful creation of a flow, you need to create a job-definition.

To create a job-definition:

1. Navigate to the Project Level.
2. Take the lock.
3. Click the **Create Job** button (Figure 57).
4. In the dialog box, select a mapping from the *Mapping List* drop-down (Figure 58).
5. From the *Flows* list drop-down, select the node you want to trigger.
6. Provide the name of the job in the *Job Name* field.
7. Enter the spark properties or override the default flag settings for Parallel Execution, Auto Transform Registration, and Hive in the [Additional Property](#) section.
8. Click **OK** to create the job-definition.



Jobs List				Editing Mode: ON	Create Job
Job Name	Flow Name	Job Definition Code	Affinity	Actions	
python-deploy-testing-3	ExcelFlow_custom_python	DNCTDFFTXXCOGHRY	maroon-python-client-emr-5-26-0-spark230		
python-deploy-testing-2	ExcelFlow_custom_python	XEQBLUDQQUICDX	maroon-python-client-emr-5-26-0-spark230		
python-deploy-testing	ExcelFlow_custom_python	AAEWEIFPHRKVFTR	maroon-python-client-emr-5-26-0-spark230		

Figure 57: The Create Job Button Activates after Taking a Lock in the Jobs Accordion

Create New Job Definition

Mapping List

Select

Flows List

Select

Job Name

Schedule


Additional Property

1 {}

Cancel



OK

Figure 58: Dialog Box for Job-Definition Creation.



Notes:

- If the Additional Property section is left empty, the default configuration is applicable for that job-definition. The default configuration remote-agent is triggered.
- Once created, a job-definition can be executed/submitted using the RUN button.
- A single job-definition can have multiple job-executions. For a job that contains the Excel node, a single execution comprises of a parent execution and a child-execution.
- For every job-execution, you can view the logs by clicking the logs icon under the Logs column.

Once the jobs have been triggered for executions, they are visible in two places. The Definitions tab contains all the definitions along with its executions. The *Executions* tab also contains all the executions for your current project (Figure 59). This allows you to view the status of all the job-definitions you triggered.

Jobs

Definitions

Executions

Refresh





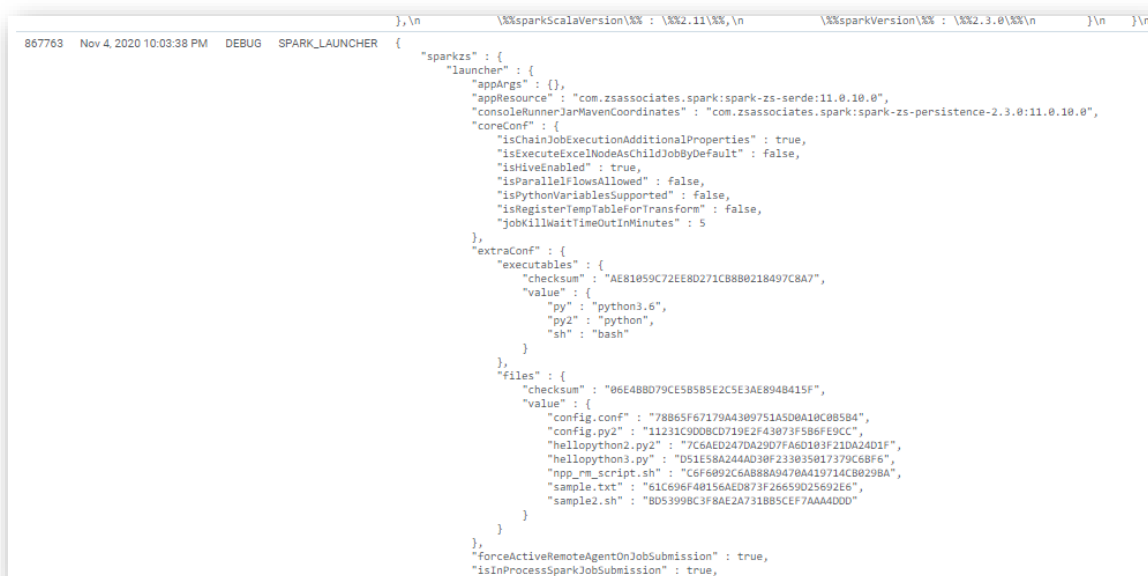
Create Time	Flow Name	Session State	Spark App Id	Agent Label	Update Time	Execution Status	Logs
Apr 16, 2020 12:21:02 AM	calculate-pi	FINISHED	application_1586973004507_0002	default	Apr 16, 2020 12:22:08 AM	Completed	
Apr 15, 2020 11:27:55 PM	calculate-pi	FINISHED	application_1586973004507_0001	default	Apr 15, 2020 11:29:17 PM	Completed	
Apr 9, 2020 6:00:38 PM	pyModules-python2	FINISHED	application_1586433062684_0001	default	Apr 9, 2020 6:02:51 PM	Completed	
Apr 2, 2020 5:11:18 PM	calculate-pi	FINISHED	application_1585824460859_0004	default	Apr 2, 2020 5:12:57 PM	Completed	

Figure 59: Definitions and Executions Tab inside Jobs Accordion

Effective Remote Agent Configuration in Job Execution Logs

The effective Remote Agent configuration is available in the execution log for each job execution. This allows users to identify the exact configuration that was used for that particular job execution (Figure 60).



```

867763 Nov 4, 2020 10:03:38 PM DEBUG SPARK_LAUNCHER {
  "sparkzs": {
    "launcher": {
      "appArgs": {},
      "appResource": "com.zsassociates.spark:spark-zs-sender:11.0.10.0",
      "consoleRunnerJarMavenCoordinates": "com.zsassociates.spark:spark-zs-persistence-2.3.0:11.0.10.0",
      "coreConf": {
        "isChainJobExecutionAdditionalProperties": true,
        "isExecuteExcelNodeAsChildJobByDefault": false,
        "isHiveEnabled": true,
        "isParallelFlowsAllowed": false,
        "isPythonVariablesSupported": false,
        "isRegisterTempTableForTransform": false,
        "jobKillWaitTimeOutInMinutes": 5
      },
      "extraConf": {
        "executables": {
          "checksum": "AE81059C72EED271CB880218497C8A7",
          "value": {
            "py": "python3.6",
            "py2": "python",
            "sh": "bash"
          }
        },
        "files": {
          "checksum": "06E48B079CE5B585E2C5E3AE8948415F",
          "value": {
            "config.conf": "78865F67179A4389751A5D0A18C0B584",
            "config.py2": "11231C9008CD719E2F43073F5B6F99C",
            "hellopython2.py2": "7C6AED247DA29D7FA6D183F21DA24D1F",
            "hellopython3.py": "D51E58A244AD30F233035017379C68F6",
            "npp_rm_script.sh": "C6F6892C6A888A9470A419714CB0298A",
            "sample.txt": "61C696F40156AED873F26659D25692E6",
            "sample2.sh": "BD53998C3F8AE2A731B85CEF7AAA40DD"
          }
        }
      },
      "forceActiveRemoteAgentOnJobSubmission": true,
      "isInProcessSparkJobSubmission": true,
    }
  }
}

```

Figure 60: Effective Configuration for Each Job Execution

Additional Property Configuration

Depending on the task you want to achieve, the configuration provided in the *Additional Property* section of the create job dialog box changes. Some of the operations that predominantly rely on this configuration are as follows:

Tasks	Corresponding Configuration Required
Job Chaining	<pre> { "sparkzs": { "launcher": { "jobEventHandlers": { "onSuccess": { "triggerJobs": { "job1": { "code": "<JD_CODE_FOR_NEXT_JOB_IN_CHAIN>", "waitForActiveAgent": true, "waitTimeOut": 1500 } } } } } } } </pre>

Tasks	Corresponding Configuration Required	
	<pre> { "sparkzs": { "launcher": { "affinity": { "label": "<LABEL_NAME>" } } } } </pre>	
Specify an Affinity-Based Remote-Agent See Job Execution on a Labelled Remote-Agent.	Label-Based Configuration: <pre> { "sparkzs": { "launcher": { "affinity": { "label": "<LABEL_NAME>" } } } } </pre>	Capability-Based Configuration: <pre> { "sparkzs": { "launcher": { "affinity": { "capabilities": { "capability1": "value1", "capability2": "value2" } } } } } </pre>
Override/Specify Spark Properties (overriding the deployMode in certain cases from cluster to client)	<pre> { "sparkzs": { "launcher": { "sparkConf": { "\"spark.submit.deployMode\"": "client" } } } } </pre>	
Override default flag settings for certain properties such as Parallel Execution, Auto Transform Registration, and Hive.	<pre> { "sparkzs": { "launcher": { "coreConf": { "isHiveEnabled": false, "isParallelFlowsAllowed": true, "isRegisterTempTableForTransform": true } } } } </pre>	
Limiting the Job Execution Duration for a Job-Definition	<pre> { "sparkzs": { "launcher": { "coreConf": { "jobExecutionTimeoutInMinutes": 2 } } } } </pre>	

Environment Variable Support in Job Additional Props

Environment variables can be used in the Job additional properties.

Consider the following example:

```
{
  "sparkzs": {
    "launcher": {
      "jobEventHandlers": {
        "onSuccess": {
          "callApi": {
            "authenticationHeader": "${env.authenticationHeader}",
            "postUrl": "${env.postUrl}"
          }
        }
      }
    }
  }
}
```

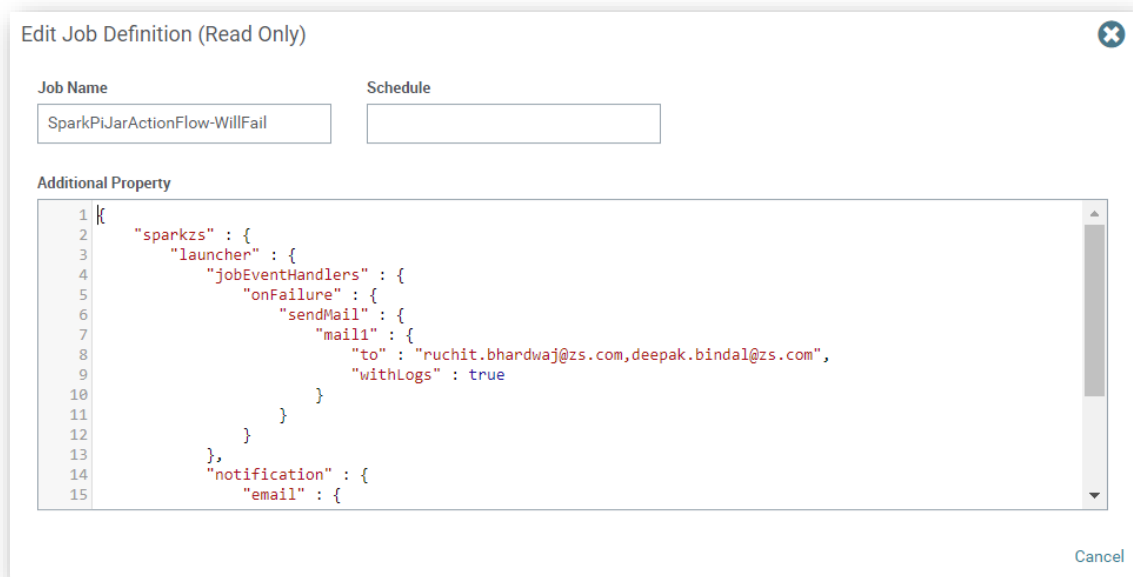
The two variables in the above example need to be defined as the environment variables. An Example of environment variables is as follows:

```
{
  "authenticationHeader": "Basic dXNlcm5hbWU6cGFzc3dvcmQ=",
  "postUrl": "http://localhost:8082/spark-zs-web/test/api-callback"
}
```

For more information about how to use environment variables, refer to the [Parameterizing Flows](#) section.

Configuration Visibility at the Job-Definition Level

Users can view the configuration that has been specified at the job-definition level without taking a lock. This can be done by clicking the Pencil icon to view the additional properties specified at the job-definition level. The Edit Job Definition dialog box appears in read-only mode (Figure 61).



Edit Job Definition (Read Only)

Job Name
SparkPiJarActionFlow-WillFail

Schedule

Additional Property

```

1 {
2   "sparkzs" : {
3     "launcher" : {
4       "jobEventHandlers" : {
5         "onFailure" : {
6           "sendMail" : {
7             "mail1" : {
8               "to" : "ruchit.bhardwaj@zs.com,deepak.bindal@zs.com",
9               "withLogs" : true
10            }
11          }
12        }
13      },
14      "notification" : {
15        "email" : {

```

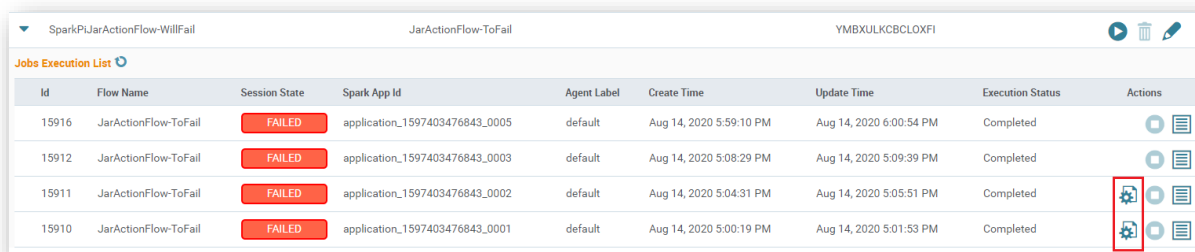
Cancel

Figure 61: Additional Property at the Job-Definition Level in Read-Only Mode

Configuration Visibility at the Job-Execution Level

Users can view the job execution additional properties configuration that was used for a given execution by clicking the configuration icon next to the job-execution run (Figure 62).

If no configuration was overridden for a given job-execution run, the icon is not visible. The presence of the configuration icon indicates the additional configuration has been overridden at the job-execution level (Figure 62).













Id	Flow Name	Session State	Spark App Id	Agent Label	Create Time	Update Time	Execution Status	Actions
15916	JarActionFlow-ToFail	FAILED	application_1597403476843_0005	default	Aug 14, 2020 5:59:10 PM	Aug 14, 2020 6:00:54 PM	Completed	 
15912	JarActionFlow-ToFail	FAILED	application_1597403476843_0003	default	Aug 14, 2020 5:08:29 PM	Aug 14, 2020 5:09:39 PM	Completed	 
15911	JarActionFlow-ToFail	FAILED	application_1597403476843_0002	default	Aug 14, 2020 5:04:31 PM	Aug 14, 2020 5:05:51 PM	Completed	  
15910	JarActionFlow-ToFail	FAILED	application_1597403476843_0001	default	Aug 14, 2020 5:00:19 PM	Aug 14, 2020 5:01:53 PM	Completed	  

Figure 62: Configuration Icon Indicates the Additional Configuration has been Overridden

When the user clicks the configuration icon, the configuration that is overridden at the job-execution level will be visible for every run (Figure 63).



Job Execution

Job Execution Id : 15911 Flow Name : JarActionFlow-ToFail

Additional Property

```

1 {
2   "sparkzs" : {
3     "launcher" : {
4       "jobEventHandlers" : {
5         "onFailure" : {
6           "sendMail" : {
7             "mail1" : {
8               "to" : "ruchit.bhardwaj@zs.com,deepak.bindal@zs.com",
9               "withLogs" : true
10            }
11          }
12        }
13      },
14      "notification" : {
15        "email" : {

```

Cancel

Figure 63: Job Execution

Limiting the Job Execution Duration for a Job-Definition


Flow Builder R4.1 allows you to limit the amount of time a job can run using the `jobExecutionTimeoutInMinutes` property. If the job is not executed within the stipulated time, execution is aborted, and the status is updated to KILLED.

The following log is visible in the job-execution logs:

```
Job: 8499 will be killed as it has exceeded the configured execution time out duration of
<jobExecutionTimeoutInMinutes> min.
```

Job Execution on a Labelled Remote-Agent

If your job needs to run on a custom provisioned cluster with a user-defined configuration, the SD Support Team creates a Labelled Remote-Agent that the user specifies while:

- Creating a new job-definition
- Editing a pre-existing job-definition using the  icon. This icon is clickable once the user takes the lock.

The configuration needed in the *Additional Property* field of the job-creation dialog box is:

```
{
  "sparkzs": {
    "launcher": {
      "affinity": {
        "label": "<LABEL_NAME>"
      }
    }
  }
}
```

Once the configuration has been set, the job can be executed.

Chaining Jobs Together

Flow Builder R4 provides a feature to chain jobs together so that after the successful execution of one job, the next job in the chain is triggered automatically. For example, you may have 3 job-definitions with their respective job-definition codes being *job-1-code*, *job-2-code*, *job-3-code* that you want to execute sequentially. You would use the job-definition code (JD Code) to chain them to execute sequentially.

For job-definition job-1, the Additional Property configuration would be:

```
{
  "sparkzs": {
    "launcher": {
      "jobEventHandlers": {
        "onSuccess": {
          "triggerJobs": {
            "job1": {
              "code": "job-2-code",
              "waitForActiveAgent": true,
              "waitTimeout": 1500
            }
          }
        }
      }
    }
  }
}
```

For job-definition job-2, the Additional Property configuration would be:

```
{
  "sparkzs": {
    "launcher": {
      "jobEventHandlers": {
        "onSuccess": {
          "triggerJobs": {
            "job1": {
              "code": "job-3-code",
              "waitForActiveAgent": true,
              "waitTimeout": 1500
            }
          }
        }
      }
    }
  }
}
```



Notes:

- For Flow Builder R4, conditional chaining is not supported.
- The subsequent job in the chain executes only if the preceding job executes successfully.
- As you can observe in the JSON, there is an option of *onSuccess*, however, there is no option of *onFailure*.

Job Execution Chain Visualization

Users can visualize the job execution chains from the Job Execution Tab (Figure 64).

Changelist

Job Definitions

Job Executions

Subscribed Assets

Refresh

Id	Create Time	Flow Name	Session State	Spark App Id	Agent Label	Update Time	Execution Status	Actions
16838	Dec 7, 2020 1:27:01 PM	trigger-api-callback	FINISHED	application_1607326784725_0001	default	Dec 7, 2020 1:30:45 PM	Completed	<div><div></div><div></div></div>

Figure 64: Job Execution Tab

A chain icon displays next to the first job execution in the chain (Figure 65).










16828	Dec 4, 2020 7:13:08 PM	jar-action-flow-conditional-true	FINISHED	application_1607088191223_0004	default	Dec 4, 2020 7:16:26 PM	Completed	 
16827	Dec 4, 2020 7:10:37 PM	jar-action-flow-print-message	FINISHED	application_1607088191223_0003	default	Dec 4, 2020 7:13:08 PM	Completed	 
16826	Dec 4, 2020 7:07:05 PM	trigger-salesforce-source-without-bulk	FINISHED	application_1607088191223_0002	default	Dec 4, 2020 7:10:37 PM	Completed	 
16825	Dec 4, 2020 7:03:27 PM	trigger-salesforce-bulk	FINISHED	application_1607088191223_0001	default	Dec 4, 2020 7:07:05 PM	Completed	  

Figure 65: Chain Icon Displays next to First Job Execution in Chain

On clicking the chain icon, a new dialog box appears showing the job execution chain (Figure 66).

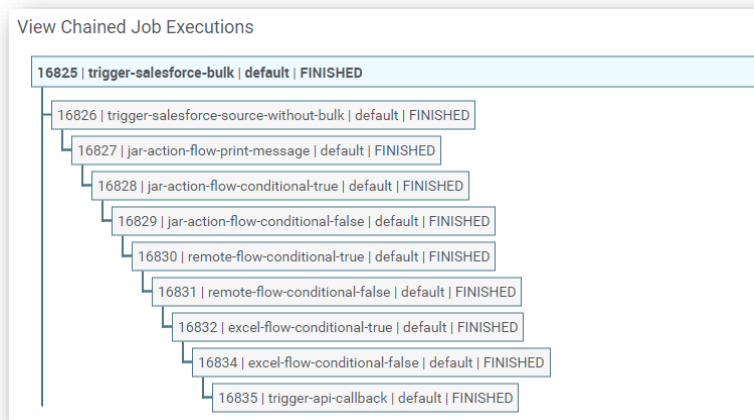


Figure 66: Job Execution Chain

Chaining Job Execution Additional Properties

When an additional property is passed at the job execution level, it is available only for that one job execution. If the jobs are chained, this job-execution level additional property can be passed to the other jobs in the chain by enabling it using the following configuration.

```
{
  "sparkzs": {
    "launcher": {
      "coreConf": {
        "isChainJobExecutionAdditionalProperties": true
      }
    }
  }
}
```



Note: By default, the value for *isChainJobExecutionAdditionalProperties* is set to false.

Job Chain Visualization

If a job-definition is a part of the chain, a chain icon displays beside the job-definition. Users can visualize the jobs they have chained together by clicking the chain icon (Figure 67).

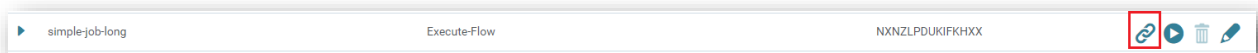


Figure 67: Job Chain Present Icon

Click the chain icon to view the entire chain starting from the job-definition selected. For example, if there are 5 jobs in the chain such as: Job1 – Job2 – Job3 – Job4 – Job5, and the user chooses to view the chain by selecting Job4, the visible chain is Job4 – Job5. Similarly, if a user selects Job2, the chain displays Job2 – Job3 – Job4 – Job5.

A sample job-chain shown in Figure 68 indicates the user has selected Job41 and is able to see the entire chain starting from Job41 until the end.

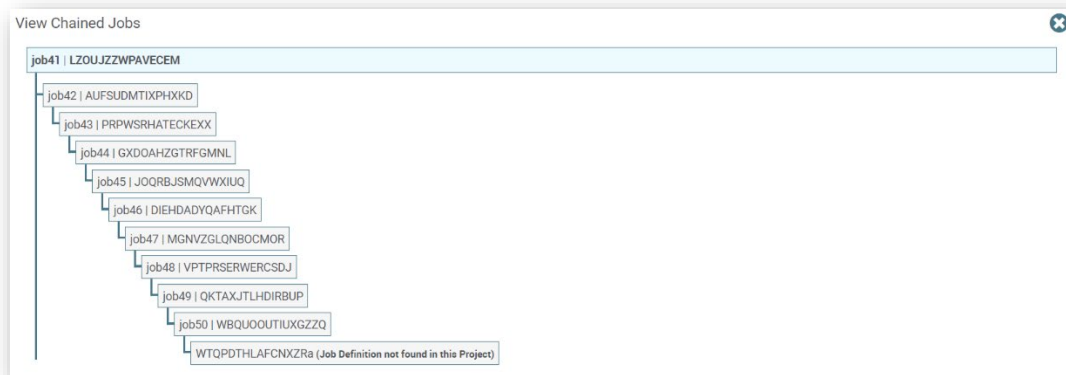


Figure 68: Sample Job Chain

Terminating Remote Agent on Job Completion

You can terminate the Remote Agent when a job has finished executing by providing following configuration in the job additional properties:

```
{
  "sparkzs": {
    "launcher": {
      "coreConf": {
        "terminateRemoteAgentOnJobCompletion": true
      }
    }
  }
}
```

Executing Excel Node as a Child Job

You can set the default execution method of Excel nodes by using the following configuration:

```
{
  "sparkzs": {
    "launcher": {
      "coreConf": {
        "isExecuteExcelNodeAsChildJobByDefault": false
      }
    }
  }
}
```

Setting the `isExecuteExcelNodeAsChildJobByDefault` to false causes the Excel flow not to be executed as child jobs. The value of this configuration is true by default.

You can also force a particular Excel node to be executed as a child job or not by providing the following property as a node option:

```
Key:
__runExcelFlowAsChildJob

Value:
true
```

The absence of this configuration causes the Excel flow node to be executed as per the `isExecuteExcelNodeAsChildJobByDefault` configuration.

Sending Email Notifications on Job Failure

FlowBuilder supports sending email notifications to a group of people if a job fails. This can be controlled through the following configuration. By default, the email notification feature is disabled.

JSON for Email Configuration

The email configuration is specified at the remote-agent level.

- email.enable: is used to enable or disable email sending feature.
- email.from: is a sender's email id.

```
{
  "sparkzs": {
    "launcher": {
      "notification": {
        "email": {
          "enabled": false,
          "from": "support@zs.com"
        }
      }
    }
  }
}
```

JSON for Email Recipients

The following configuration JSON can be used to specify the email recipients:

```
{
  "sparkzs": {
    "launcher": {
      "jobEventHandlers": {
        "onFailure": {
          "sendMail": {
            "mail1": {
              "to": "user1@zs.com, user2@zs.com",
              "withLogs": true
            },
            "mail2": {
              "to": "user1@zs.com, user2@zs.com",
              "withLogs": false
            },
            "mail3": {
              "to": "user1@zs.com"
            }
          }
        }
      }
    }
  }
}
```

```

    }
  }
}

```

JSON for Target Group Email Configuration

A *target group email configuration* is used to specify a group of recipients of the email as follows:


```

{
  "mail1": {
    "to": "user1@zs.com, user2@zs.com",
    "withLogs": true
  }
}

```

- **Target Group Name:** For the target group configuration, “*mail1*” identifies the name of the target group and can be any name.
- **Email Recipients:** The comma separated email ids can be provided in “*to*”.
- **With Logs:** Email are sent with logs for this target group if the “*withLogs*” configuration is set to true. By default, this value is false.

Some example emails are shown in Figure 69 and Figure 70.


ExecutionLogsForJobId-15829.log
16 KB

Log file as attachment

Job Failed

Flow Name	Session State	Spark App ID	Agent Label	Create Time	Update Time
JarActionFlow-ToFail	FAILED	application_1596822481261_0006	default	Fri Aug 07 18:12:38 UTC 2020	Fri Aug 07 18:12:38 UTC 2020

Detailed Exception Stacktrace

Error Details

Execution ID	Log Date	Message
799701	Fri Aug 07 18:14:14 UTC 2020	Error while running Json Runner The exception is : java.lang.ClassNotFoundException: com.zsassociates.spark.jaractionflowtest.SparkPi4 at java.net.URLClassLoader.findClass(URLClassLoader.java:381) at java.lang.ClassLoader.loadClass(ClassLoader.java:424) at java.lang.ClassLoader.loadClass(ClassLoader.java:357) at java.lang.Class.forName0(Native Method) at java.lang.Class.forName(Class.java:348) at com.zsassociates.spark.operators.JarActionFlowFilter.execute(JarActionFlowFilter.scala:35) at com.zsassociates.spark.persistence.JsonRunner\$.execute(JsonRunner.scala:82) at com.zsassociates.spark.persistence.JsonRunner\$.main(JsonRunner.scala:46)

Figure 69: Email Example 1

Job Failed					
Flow Name	Session State	Spark App ID	Agent Label	Create Time	Update Time
ActionFlowM1	FAILED	local-1597302054867	default	Thu Aug 13 12:30:26 IST 2020	Thu Aug 13 12:30:26 IST 2020
Error Details					
Error Details					
Job with ID 252 failed					

Figure 70: Email Example 2

Email Notification on Job Success and Remote Agent Shutdown Events

FlowBuilder supports sending email notifications to a group of people when the:

- Job finishes successfully, or
- The Remote agent is shutting down.

Email Configuration: On Job Success

The following configuration can be used to specify the email recipients that will be notified about a successful job execution:

```
{
  "sparkzs": {
    "launcher": {
      "jobEventHandlers": {
        "onSuccess": {
          "sendMail": {
            "mail1": {
              "to": "user1@zs.com,user2@zs.com",
              "withLogs": true
            },
            "mail2": {
              "to": "user1@zs.com,user2@zs.com",
              "withLogs": false
            },
            "mail3": {
              "to": "user1@zs.com"
            }
          }
        }
      }
    }
  }
}
```

Users can specify the recipients of the email by creating target groups. A target group configuration looks like the following:

```
{
  "mail1": {
    "to": "user1@zs.com,user2@zs.com",
    "withLogs": true
  }
}
```

- **Target Group Name:** Here mail1 is the name of the target group and can be any name. This name is used to visually identify the target group.
- **Email Recipients:** The comma separated email ids can be provided in to.
- **With Logs:** Email will be sent with logs for this target group if the *withLogs* configuration is set to true. By default, this value is false.

Email Configuration: On Remote Agent Shutdown

The following configuration is provided at the remote agent level:

```
{
  "notification": {
    "email": {
      "shutdownNotificationEmail": "user1@zs.com,user2@zs.com",
      "from": "support@zs.com",
      "enabled": true
    }
  }
}
```

If the *shutdownNotificationEmail* property is not present, the email notification upon remote agent shutdown or termination is disabled.

Job State Notification Using Callback URL

FlowBuilder supports a job state change notification for successful and failure final states. The user must provide a callback URL which supports HTTP POST method and accepts application/json type data as the request body. Whenever a job either fails or succeeds, the specified callback URL is invoked and a JSON is posted with the request that has the jobId and the state.

Example JSON that is posted to the callback URL:

```
{
  "jobId": 1,
  "state": "FINISHED"
}
```

The following configuration needs to be provided in the Job additional properties.

A callback notification configuration when the job is successful is as follows:

```
{
  "sparkzs": {
    "launcher": {
      "jobEventHandlers": {
        "onFailure": {
          "callApi": {
            "postUrl": "<post-url>",
            "httpHeaders": {
              "Authorization": "Basic ${env.config.basicAuth}",
              "x-actual-user": "DummyUser"
            }
          }
        }
      }
    }
  }
}
```

A callback notification configuration for job failure is as follows:

```
{
  "sparkzs": {
    "launcher": {
      "jobEventHandlers": {
        "onFailure": {
          "callApi": {
            "postUrl": "<post-url>",
            "httpHeaders": {
              "Authorization": "Basic ${env.config.basicAuth}",
              "x-actual-user": "DummyUser"
            }
          }
        }
      }
    }
  }
}
```

The following two configurations need to be provided in the **callApi** configuration:

httpHeaders

The HTTP Headers that need to be passed for calling the API can be passed as key-value pairs here as shown in the example above


For example, Authorization, x-actual-user can be passed as headers.

postUrl

The HTTP URL which supports POST method and accepts application/json.

For example: <http://localhost:8082/spark-zs-web/test/api-callback>

Aborting Jobs

A Job can be aborted by clicking on the Abort Jobs  button (Figure 71):

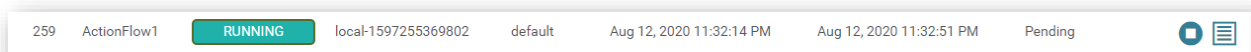


Figure 71: Abort Jobs Button

If the user clicks the ABORT Jobs button, the final state of the job is:

- **ABORTED:** if the job has not been picked for execution yet, or
- **KILLED:** if the job has been picked for execution.

Asset Management

Creating and Subscribing Asset Releases

Once a project has been created in an Asset Account, and the corresponding mappings and flows have been created, you can create a release of a project.



Note: Creation of a release of a project is essential as only a released version can be subscribed to a client account.

As of Flow Builder R4.1, the Asset Subscription is supported using the following two ways:

1. Using the command line. This can be done by all Administrators: SuperAdmin, AccountAdmin or EnvironmentAdmin.
2. From the UI. This can be done only by the SuperAdmin.

Subscribing Subsequent Asset Releases to a Client Project

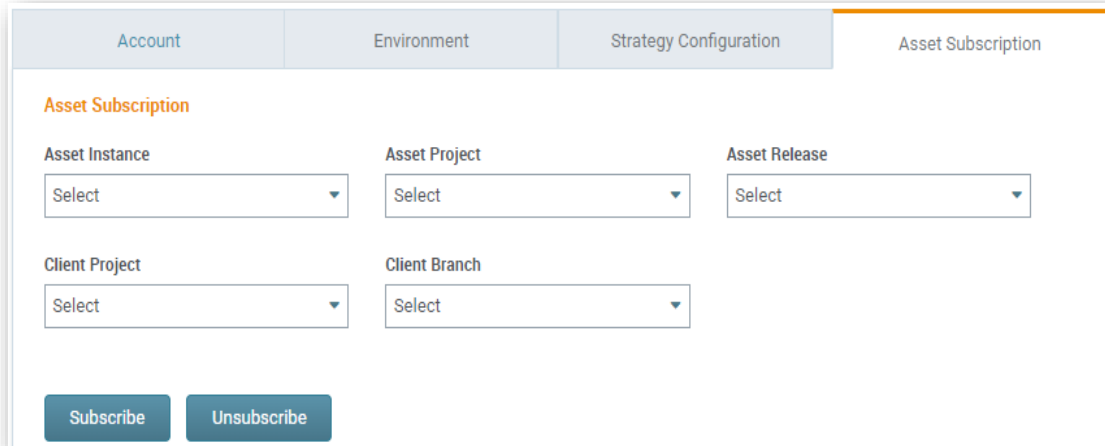
If you want to subscribe a newer release of the asset project to a client project, the same can be done by changing the commands by subscribing the release tag.

Using User Interface to Subscribe Assets

Flow Builder R4.1 supports subscribing to an asset project right from the User Interface. Only a SuperAdmin can perform this function.

To subscribe an asset from the UI:

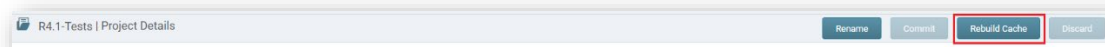
1. Click the ***Settings*** button on the top right-hand corner of the screen.
2. Click the ***Asset Subscription*** tab.
3. Select the ***Asset Instance Name***, ***Asset Project Name*** and the ***Release Number*** of the asset project from the drop-downs (Figure 72).
4. Select the Client Project and the Client Branch from the dropdowns.
5. Once all the parameters have been selected, you can subscribe and unsubscribe by clicking the respective buttons.



The image shows a web interface with four tabs: Account, Environment, Strategy Configuration, and Asset Subscription. The Asset Subscription tab is active. Below the tabs, there is a section titled "Asset Subscription" with five dropdown menus: Asset Instance, Asset Project, Asset Release, Client Project, and Client Branch. Each dropdown menu has a "Select" option. At the bottom of the section, there are two buttons: "Subscribe" and "Unsubscribe".

Figure 72: Asset Subscription Capability from the UI

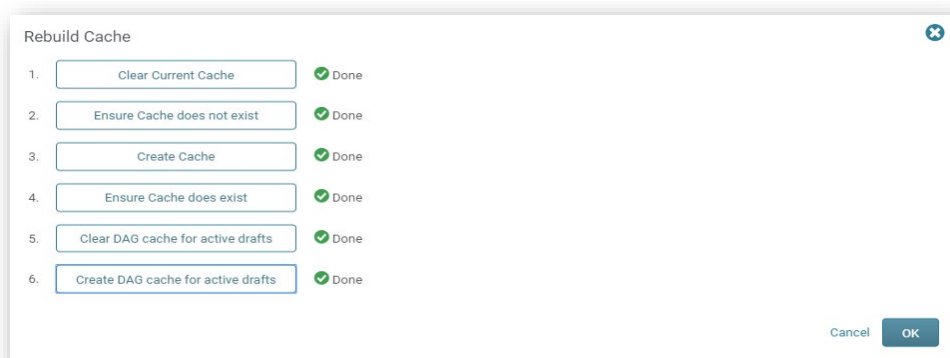
6. Navigate back to the project level and click **Commit**.
7. Once the commit has been successful, click **Rebuild Cache** (Figure 73).



The image shows a horizontal toolbar with four buttons: "Rename", "Commit", "Rebuild Cache", and "Discard". The "Rebuild Cache" button is highlighted with a red border.

Figure 73: Rebuild Cache Button

8. In the Rebuild Cache dialog box, click each button to complete the asset subscription procedure (Figure 74).



The image shows a "Rebuild Cache" dialog box with a close button (X) in the top right corner. It contains a list of six steps, each with a button and a "Done" status indicator (a green checkmark):

1. Clear Current Cache
2. Ensure Cache does not exist
3. Create Cache
4. Ensure Cache does exist
5. Clear DAG cache for active drafts
6. Create DAG cache for active drafts

At the bottom right of the dialog box, there are "Cancel" and "OK" buttons.

Figure 74: Rebuild Cache Dialog Box

After the asset has been subscribed properly, you can view the asset subscriptions and release tags for your project by expanding the *Subscribed Assets* accordion as shown in Figure 75 and Figure 76:

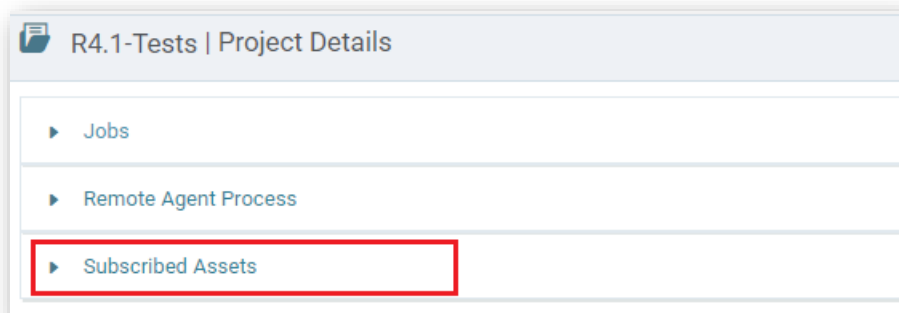
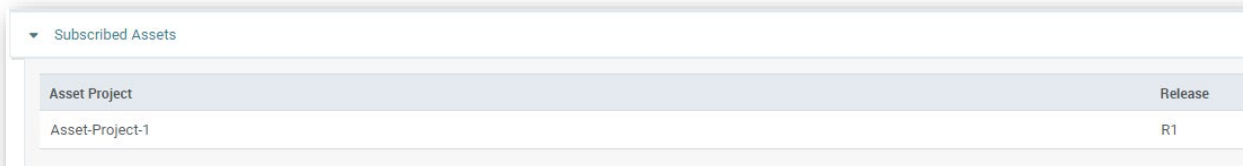


Figure 75: Subscribed Assets Accordion



Subscribed Assets	
Asset Project	Release
Asset-Project-1	R1

Figure 76: Expanded View of the Subscribed Assets Accordion

Script-Runner

Script-Runner is a way to access all the customization scripts client teams might have, which enables clients to easily migrate to Flow Builder. The script-runner replaces the need for the time-consuming JAR creation of each customization. Flow Builder's Script-Runner provides an efficient and highly modular solution to integrate and execute custom scripts.

Scripts Supported

The script-runner supports the execution of all existing customization scripts like Python2, Python3 and shell scripts. It also supports the installation of all the module dependencies for Python and shell scripts.



Note: These module dependencies are installed only on the master node of the EMR cluster on which the remote-agent runs.

Using the Script-Runner

To use the script-runner, follow these steps:

1. Identify the Python version required for your custom Python scripts to execute. Depending upon which Python version being used, the extension of the file needs to be py2 for Python2 and py for Python3. Refer the following table for an example:

Python Version	Filename Extension	Example
Python2	py2	filename1.py2
Python3	py	filename1.py

2. Once the above requirements have been identified, contact the SD Support with the following three things:
 - a. The list of executables. For example, py for python, sh for bash and so on.
 - b. The extra Python Modules used in their flows. For example, http lib2.
 - c. Python scripts with their proper extensions.

The SD Support Team generates an extraConfig incorporating these items and adding them to the remote-agent configuration.

3. Create a flow in Flow Builder using a JarActionFlow node with the properties listed in the following table to leverage the functionality of your custom scripts.

Property Name	Value
Name	Name of the node you want to specify.
ClassName	com.zsassociates.spark.commandrunner.CommandRunner
MethodName	execute
Arguments	script-runner -files <comma-separated-s3-locations-for-all-required-file> -conf <optional-conf-file-path> -scriptFileName <name-of-the-file-to-execute> -scriptArgs <optional-argument-list> For example: script-runner -files s3://s3-bucket/hello.py -conf s3://s3-bucket/hello.conf -scriptFileName hello.py -scriptArgs hello world

Since the python modules are installed on the master node of the EMR cluster, the execution of custom-scripts via Flow Builder is successful using the deployMode=client strategy at the job-definition level.

```
{
  "sparkzs": {
    "launcher": {
      "sparkConf": {
        "\"spark.submit.deployMode\"": "client"
      }
    }
  }
}
```



Note: Files specified in the `-conf` option can be modified. The URL to the configuration files can be provided here. Depending upon business requirements, this file can be changed accordingly, and the script-runner command uses this updated version of the `-conf` file. The following constraints are true for files passed in the `-conf` option:

- The file specified in `-conf` option cannot have same name found in the `extraConf.files` configuration.
- The file specified in `-conf` option cannot have same extension found in the `extraConf.executables` configuration.

Frequently Asked Questions

The following are the FAQs associated with the Script-Runner process.

Who creates the extraConf for supporting script execution?

The SD Support Team creates the extraConf for clients and adds it to their remote agent configuration. Client teams must notify the SD Support Team with any existing scripts/files that are used for customization purposes.

Where are the scripts/files used for customization located?

The customization scripts and files are located on S3.

Can client teams edit the remote agent configuration and add their own files?

The Client Team Administrator can only view the remote-agent configuration. If an Administrator edits it by adding a file to the list, for example, it causes an integrity violation, rendering the change to be useless.

Can client teams edit the scripts/files on S3 after they have been added to the remote agent configuration?

No, clients cannot edit the scripts/files on S3 after they have been added to the remote-agent. If the files are modified later by the client team, it causes an integrity violation.

CLI Commands



Note: Every command in this section requires the `spark-zs-cli-<version>.jar` to be installed on your local machine. Also, Java must be installed to use the commands.

For each of these commands, a set of pre-requisite commands must be run to reach the execution points. The list of these commands are:

Command	Description
<code>java -cp spark-zs-cli-8.0.0.66.jar com.zsassociates.spark.web.tools.SparkZSV2</code>	This command is used to enter the CLI jar.
<code>connect -url=<client-url> -userName=<username> -password=<password></code>	This command is used to log into Orchestration Engine.
<code>enter-project-tag</code>	This command lists all the projects in the environment/consumerapp. Each project corresponds to a unique project ID which needs to be selected to perform a project-specific operation. You are prompted to select a versionTagId or branch/release ID from a list of versions.

Submitting a Job

You can run a job identified by a unique Job-Definition-Code from a CLI command.

Command	Description
<code>job-execute - jobDefinitionCode=<JD-CODE> - waitForActiveAgent=true - waitTimeOut=3600</code>	<p>This command is used to execute an existing job-definition based on the JD-Code.</p> <ul style="list-style-type: none"> jobDefinitionCode: A new job created in the system automatically generates a job code. This jobDefinitionId remains consistent across environments/versions/apps. waitForActiveAgent: If an active agent does not exist, the command waits for the agent to start and then submit the job [Default: false]. waitTimeOut: The duration in seconds this command keeps checking the status and logs of the submitted job [Default: 1800].

Automating Job Submission: End to End

Flow Builder R4 supports grouping two or more CLI commands using the '-quiet' option. This can be used to automate a job-submission in an end-to-end manner. For example, a user can group the commands for logging into Orchestration Engine and submitting a job for execution.



Note: The two grouped commands should be separated by double quotes [" "].

Command	Description
<pre>java -cp spark-zs-cli-<version-number>.jar com.zsassociates.spark.web.tools.SparkZSV2 -quiet "connect -url=<enter-client-url> -userName=<enter- username> -password=<enter-password>" "job-execute -jobDefinitionCode=<enter-JD-code> - projectId=<enter-project-id> - waitForActiveAgent=true -waitTimeOut=<enter- timeout-time-in-seconds>"</pre>	<p>-quiet: This option allows you to group two or more CLI commands to execute them in a consecutive manner. The two grouped commands should be separated by double quotes [" "].</p>

Terminating Remote-Agent Cleanly

Apart from terminating the remote-agent from the UI in the Remote-Agent accordion, Flow Builder R4 provides a command to shut down a remote-agent manually based on the remote-agent label name.

Command	Description
<pre>swlh-force-shutdown -label=<label_name></pre>	<p>This command is used to shut down a remote-agent identified by the label_name.</p> <p>-label: label_name of the remote-agent you want to terminate.</p>

Also, if the remote-agent is shut down improperly, there might be a remote-agent entry persisting in the Active Tab of the Remote-Agent accordion. The remote-agent has stopped pinging when you see the warning icon beside the ALIVE state of the remote-agent as shown in Figure 77.

Remote Agent Process Id	Remote Agent Base Url	Agent Label	Remote Agent Id	Current Status Updated On	Remote Agent Status	Actions
2	http://192.168.43.109:11630/spark-zs-launcher	default	2	Dec 5, 2019 9:51:19 PM	ALIVE 	

Figure 77: Warning when the Remote-Agent has not Pinged for 15 Minutes


In such a scenario, clients are expected to contact the SD Support Team.

To remove the entry from the *Active* Tab of the remote-agent accordion, use the following command:

Command	Description
<code>swlh-force-mark-shutdown -label=<label_name></code>	<p>This command removes the entry for the remote-agent identified by the <code>label_name</code> that was shut down improperly.</p> <p>-label: <code>label_name</code> of the remote-agent you want to terminate.</p>

Asset Subscription

Flow Builder R4 supports using CLI commands to subscribe asset flows from an Asset Account/Instance to a project in the Client Account. The subscribed project appears as a child mapping at the client level.

	<p>Notes: Before you subscribe an asset-flow, keep the following points in mind:</p> <ul style="list-style-type: none"> • Ensure the user does not have a draft version at either the asset or the client instance level. • At the asset instance level, the user should have a released tag version to which the client is subscribing. • After the asset subscription is complete, the user should trigger the <code>commit-with-rebuild-cache</code> command from the CLI itself.
--	--

The sequence of commands used for asset subscription are:

Sr. No.	Command	Description
1	<code>java -cp spark-zs-cli-<version_number>.jar com.zsassociates.spark.web.tools.SparkZSV2</code>	This command is used to log into the CLI jar.

Sr. No.	Command	Description
2	<code>connect -url=<enter-url-here> -userName=<enter-username-here> -password=<enter-password></code>	This command connects to the OE where the Client Instance's environment has been pointed to– -url: Orchestration Engine home page url -userName: The user name used to access the application [Default: <empty string>] -password: The password to access the application [Default: <empty string>]
3	<code>enter-project-tag</code>	This command is used to select the project_id of the client project.
4	<code>connect -url=<enter-url-here> -userName=<enter-username-here> -password=<enter-password> -connectionName=asset</code>	This command connects to the Orchestration Engine Instance where the Asset Instance's ConsumerApp/Environment has been pointed: <ul style="list-style-type: none"> -url: Orchestration Engine home page url -userName: The user name used to access the application [Default: <empty string>] -password: The password to access the application [Default: <empty string>] -connectionName: The name of the connection on which this command should be executed. Its optional and the command gets executed on the default connection (the connection without any name) [Default: self]
5	<code>enter-project-tag -connectionName=asset</code>	This command is used to select the project_id of the asset project.
6	<code>subscribe-asset -assetConnectionName=asset</code>	This command subscribes the specified asset project to the said client project. -assetConnectionName: The connection to an asset instance/account that must be subscribed.
7	<code>commit-with-rebuild-cache</code>	This command commits the changes and creates cache for the entire subscribed asset project.

Copying Flows

Flow Builder R4 allows you to copy single mappings, multiple mappings, and by extension, the entire project. Commands used for copying are as follows:

Sr. No.	Command	Description
1	<code>java -cp spark-zs-cli- <version_number>.jar com.zsassociates.spark.web.tools .SparkZSV2</code>	This command is used to log into the CLI jar.
2	<code>connect -url=<enter-url-here> - userName=<enter-username-here> - password=<enter-password> - connectionName=src</code>	This command creates a named connection to Orchestration Engine for the specified OE Instance where the source project exists: <ul style="list-style-type: none"> • -url: Orchestration Engine home page url • -userName: The user name used to access the application [Default: <empty string>] • -password: The password to access the application [Default: <empty string>] • -connectionName: The name of the connection on which this command should be executed. It is optional and the command gets executed on the default connection (the connection without any name) [Default: self].
3	<code>enter-project-tag - connectionName=src</code>	This command is used to select the project_id of the source asset project.
4	<code>connect -url=<enter-url-here> - userName=<enter-username-here> - password=<enter-password> - connectionName=dest</code>	This command creates a named connection to Orchestration Engine for the specified OE Instance where the destination project is supposed to be: <ul style="list-style-type: none"> • -url: Orchestration Engine home page url • -userName: The user name used to access the application [Default: <empty string>] • -password: The password to access the application [Default: <empty string>] • -connectionName: The name of the connection on which this command should be executed. It is optional and the command gets executed on the default connection (the connection without any name) [Default: self]

Sr. No.	Command	Description
5	enter-project-tag - connectionName=dest	This command is used to select the project_id of the destination project.
6	connect -url=<enter-url-here> - userName=<enter-username-here> - password=<enter-password>	This command connects to the Orchestration Engine (Use the URL where the source is pointed)- <ul style="list-style-type: none"> • -url: Orchestration Engine home page url • -userName: The user name used to access the application [Default: <empty string>] • -password: The password to access the application [Default: <empty string>]
7	enter-project-tag	This command is used to select the project_id of the source project.
8	<p>If you want to copy a single mapping from Source to Destination</p> <pre>copy-flow -sourceFolderId=<source-mapping/project-id> -sourceConnectionName=src -destinationFolderId=<destination-mapping/project-id> -destinationConnectionName=dest</pre> <p>If you want to copy a multiple mappings/entire projects from Source to Destination</p> <pre>copy-flow -sourceFolderId=<source-mapping/project-id> -sourceConnectionName=src -destinationFolderId=<destination-mapping/project-id> -destinationConnectionName=dest -recursive</pre>	<p>This command is used to copy a single mapping/multiple mapping from source to destination.</p> <p>-sourceFolderId: The internal database Id of the source folder [Default: 0]</p> <p>-sourceConnectionName: By default, source is the same as the current connected instance. If the source is different please create a different connection using the connect command with the -connectionName parameter [Default: self]</p> <p>-destinationFolderId: The internal database Id of the destination folder [Default: 0]</p> <p>-destinationConnectionName: By default, destination is the same as the current connected instance. If the destination is different please create a different connection using the connect command with the -connectionName parameter [Default: self]</p> <p>-recursive: Pass the flag to copy all the child mappings are copied recursively [Default: false]</p>

Various copying scenarios are as follows:

Scenario	Pre-requisites	-sourceFolderId	-destinationFolderId
Copying entire project across different Flow Builder instances	A placeholder project should be present at the destination where the flows would be copied.	The ID of the source project	The ID of the destination project
Copying a hierarchy of mappings across different projects within the same Flow Builder instance	A parent mapping should already be present at the destination.	The ID of the parent mapping at source	The ID of the parent mapping at destination inside which the user wants the child mapping to be
Copying a single mapping within and across Flow Builder instances	A placeholder mapping should be already present at the destination. Only the contents of the mapping (nodes in mapping) are copied in this case.	The ID of the source mapping	The ID of the destination Placeholder mapping



Note: If you want to copy mappings across projects within the same Flow Builder instance, all commands remain the same except the URLs provided while creating the named connections. Both the URLs need to be the same in that case. All the remaining commands remain the same.

Copy Entire Projects/Group of Mappings

The expected hierarchy of mappings should be as follows for copying entire projects/copying a group of mappings as is:

```
Project-1
  Mapping-1
    Mapping-1.1
    Mapping 1.2
  Mapping-2
    Mapping-2.1
```

Here, the Mapping-1 and Mapping-2 are expected to be empty with all the contents being in the child mappings. So, when you want to copy:

- The entire Project-1 --> specify the project ID of 'Project-1' in the sourceFolderId option.
- A particular mapping hierarchy Mapping-1 --> specify the project ID of 'Mapping-1' in the sourceFolderId option.

Copy a Single Mapping Across/Within the Same Flow Builder Instance

The expected hierarchy of mappings should be as follows for copying a single mapping:

```
Project-1
  Mapping-1
  Mapping-2
```

Here, Mapping-1 and Mapping-2 are expected mappings to be copied with all the contents.

However, at the destination, a destination mapping should already exist since copying of a single mapping only copies the CONTENTS of the mapping from source to destination.