

# Visual Flow User Guide

*September, 2023*  
*Version 1.4.1*

## Document Revisions

Date	Version	Document Changes
12/08/2020	0.9.2	Initial Draft
22/04/2021	0.9.3	Pipeline Operations
26/04/2021	0.9.4	Job Operations
07/05/2021	0.9.5	Project Name, Project Operations
25/05/2021	0.9.6	Project Name in document
09/07/2021	0.9.7	Pipeline Operators, Job Operations, Storages
24/10/2021	0.9.8	Jobs and Pipelines statuses, Custom container, Storages
13/11/2021	0.9.9	New Data Storages
22/11/2021	0.9.10	New Data Storage
15/12/2021	0.9.11	Logs levels
29/12/2021	0.9.12	Cache stage, Job Operations
21/01/2022	0.9.13	New Data Storage, Stage descriptions
04/02/2022	0.9.14	Description of STDOUT storage
18/02/2022	0.9.15	Truncate mode
04/03/2022	0.9.16	Full query mode in Transformer stage
18/03/2022	0.9.17	Sort stage
01/04/2022	0.9.18	Password component
08/08/2022	0.9.19	Style and verification updates to all chapters
19/08/2022	0.9.20	Added Slice Stage and updates to 4.2.1 and 5.2
27/09/2022	1.0.0	Updates for release 19.09.2022 and new design
11/11/2022	1.1.0	Updates for release 11.11.2022
05/02/2023	1.2.0	Updates for release 01.01.2023
20/04/2023	1.3.0	Updates for release 03.04.2023
18/09/2023	1.4.1	Updates for releases 1.4.0 and 1.4.1

## Revision Details

Version	Updates
0.9.19	Chapter 3.1 split into “3.1. Getting Started” and “3.2. Create Project”
0.9.19	“4.2. Create a job” updated to “4.2. Create a job.Available Stages” and broken down to multiple sections per each stage
0.9.19	Added section 3 for Connections to “3.3. Manage Project Settings”
0.9.19	Updated screenshots for Jobs list and Pipelines list
0.9.19	Style and verification updates to all chapters

0.9.20	Added Slice Stage and updates to “4.2.1. Read Stage” and to “5.2. Create a Pipeline ”
1.0.0	Added User profile menu to “3.1. Getting started”
1.0.0	Updated all screens as per new design
1.0.0	Added Job History to jobs overview actions
1.0.0	Added chapter “5.5. Scheduling a pipeline”
1.0.0	Added about Job History to “4.4. Job Execution”
1.0.0	Added about copying stages on canvas to “4.3. Job Designer Functions Overview”
1.0.0	Updated “4.2.1. Read Stage”. Added statement about selecting existing connection
1.1.0	Added information about ClickHouse, Dataframe and Local file storages, added Wait stage to “5.2. Create a Pipeline”.
1.1.0	Updated all screenshots and applied format and style corrections to all chapters.
1.2.0	Added chapters about Job Designer stages: Pivot, Add/Update Column, String Functions, Date/Time, Drop/Fill Nulls
1.2.0.	All screens updated due to new functionality (new stages, new action buttons, language switcher, pipeline statuses and color updates)
1.2.0.	Added statements about Suspend/Resume functionality to chapter “5.4. Pipeline Execution”
1.2.0.	Updated chapters about Jobs Overview page and Parameters screen due to redesign
1.3.0	All Job Designer and Pipeline Designer screens updated due to new Auto Refresh button
1.3.0	Updated screen on Pipeline creation due to updates on notifications
1.3.0	Added chapter for Validate stage.
1.3.0	Settings→Basic screen updated
1.3.0	Updated Job Designer and Pipeline Designer functions overview
1.4.1	Added API storage into list of available storages on “4.2.1. Read Stage” chapter
1.4.1	Updates to chapters “3.1. Getting Started” , “3.4. Manage Project Settings” and “4.2.14. Join Stage”

## Table of Contents

1	<a href="#">Introduction.....</a>	6
	<a href="#">1.1 ...Terminology .....</a>	6
	<a href="#">1.2...Scope and Purpose .....</a>	7
	<a href="#">1.3 ...Process Overview .....</a>	7
2	<a href="#">Roles and Authorizations.....</a>	8
3	<a href="#">Project Operations.....</a>	9
	<a href="#">3.1 ...Getting started .....</a>	9
	<a href="#">3.2 ...Create Project.....</a>	10
	<a href="#">3.3 ...Project Overview .....</a>	12
	<a href="#">3.4 ...Manage Project Settings. ....</a>	13
4	<a href="#">Job Operations .....</a>	15
	<a href="#">4.1 ...Jobs Overview .....</a>	15
	<a href="#">4.2 ...Create a Job. Available Stages.....</a>	16
	<a href="#">4.2.1..Read Stage .....</a>	17
	<a href="#">4.2.2. Write Stage .....</a>	23
	<a href="#">4.2.3. Group By Stage .....</a>	26
	<a href="#">4.2.4. Remove Duplicates Stage .....</a>	27
	<a href="#">4.2.5. Filter Stage .....</a>	27
	<a href="#">4.2.6. Transformer Stage .....</a>	27
	<a href="#">4.2.7. Sort Stage.....</a>	28
	<a href="#">4.2.8. Slice Stage.....</a>	28
	<a href="#">4.2.9. Pivot Stage.....</a>	28
	<a href="#">4.2.10. Add/Update Column Stage.....</a>	29
	<a href="#">4.2.11. String Functions Stage .....</a>	30
	<a href="#">4.2.12. Date/Time Stage.....</a>	30
	<a href="#">4.2.13. Drop/Fill Nulls Stage.....</a>	30
	<a href="#">4.2.14. Join Stage.....</a>	31
	<a href="#">4.2.15. Change Data Capture Stage .....</a>	32
	<a href="#">4.2.16. Union Stage .....</a>	33
	<a href="#">4.2.17. Cache Stage .....</a>	33
	<a href="#">4.2.18. Validate Stage.....</a>	33
	<a href="#">4.3 ...Job Designer Functions Overview .....</a>	36
	<a href="#">4.4 ...Job Execution .....</a>	36

5	<u>Pipeline Operations...</u>	38
5.1	<u>...Pipelines Overview</u>	38
5.2	<u>...Create a Pipeline</u>	39
5.3	<u>...Pipeline Designer Functions Overview</u>	45
5.4	<u>...Pipeline Execution</u>	45
5.5	<u>...Scheduling a pipeline</u>	48

# 1. Introduction

## 1.1. Terminology

**ETL** is an abbreviation for *extract, transform, load* — three database functions combined into onetool to pull data out of one database, transform it and place into another.

- ✓ **Extract** is the process of *reading data* from a database. In this stage, the data is collected, often from multiple and different types of sources.
- ✓ **Transform** is the process of *converting the extracted data* from its previous form into the form needed to place it into another database.
- ✓ **Load** is the process of *writing the data* into the target database.

A **job** is a chain of individual stages linked together. It describes the flow of data from its source to target. Usually, a job has a minimum of one data input and output. However, some jobs can accept more than one data input and output it to more than one target.

In Visual Flow, the available stages are:

- ✓ Read
- ✓ Write
- ✓ Group By
- ✓ Remove duplicates
- ✓ Filter
- ✓ Transformer
- ✓ Sort
- ✓ Slice
- ✓ Pivot
- ✓ Add/Update Column
- ✓ String Functions
- ✓ Date/Time
- ✓ Drop/Fill Nulls
- ✓ Join
- ✓ Union
- ✓ Change Data Capture
- ✓ Cache
- ✓ Validate

A **pipeline** is a compound of multiple jobs and can be run. In Visual Flow, the user can use such stages as:

- ✓ Job
- ✓ Container
- ✓ Notification
- ✓ Wait

## 1.2. Scope and Purpose

The Visual Flow web application is the ETL tool designed for effective data manipulation via a convenient and user-friendly interface. The tool has the following capabilities:

- Can integrate data from heterogeneous sources:
  - ✓ API
  - ✓ AWS S3
  - ✓ Cassandra
  - ✓ ClickHouse
  - ✓ Dataframe
  - ✓ DB2
  - ✓ Elasticsearch
  - ✓ IBM COS
  - ✓ Local File
  - ✓ Mongo
  - ✓ MSSQL
  - ✓ MySQL
  - ✓ Oracle
  - ✓ PostgreSQL
  - ✓ Redis
  - ✓ Redshift
  - ✓ Redshift-jdbc
- Perform data processing and transformation
- Leverage metadata for analysis and maintenance

## 1.3. Process Overview

Visual Flow jobs and pipelines exist within a certain namespace (project), so the first step in the application would be to create a project or enter an existing one. Then you need to enter Job Designer to create a job.

*Job Designer* is a graphical user interface used to create, maintain, execute and analyze jobs. Each job determines the data sources, the required transformations and the destination of data.

*Pipeline designer* is a graphical user interface aimed at managing pipelines. Designing a pipeline is similar to doing so on a job.

Important: When editing stages in the Configuration Panel, to save data a user must click the *Confirm* button, otherwise the data will be lost.

Visual Flow key functions include but not limited to:

- ✓ Create a project which serves as a namespace for jobs and/or pipelines
- ✓ Manage project settings
- ✓ User access management
- ✓ Create/maintain a job in Job Designer
- ✓ Job execution and logs analysis
- ✓ Create/maintain a pipeline in Pipeline Designer
- ✓ Pipeline execution
- ✓ Import/Export jobs and pipelines

## 2. Roles and authorizations

The following roles are available in the application:

- ✓ Viewer
- ✓ Operator
- ✓ Editor
- ✓ Administrator

They enable one to perform the below operations within the namespaces they contain access to. Only a *Super-admin* user can create and delete a workspace (project) and grant initial access to this project.

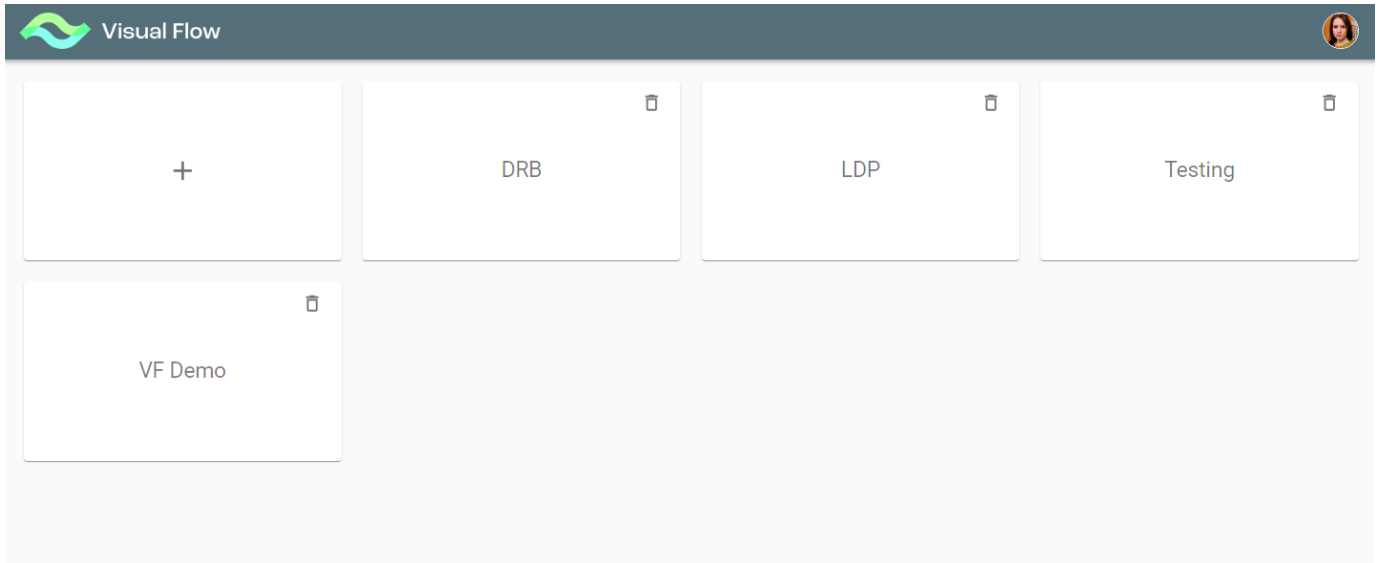
Role	Actions		
	Project Settings	Jobs	Pipelines
Viewer	View All	View All	View All
Operator	View All	View All / execute jobs	View All / execute pipelines
Editor	Edit All but Users/Roles	Edit / execute jobs	Edit / execute pipelines
Admin	Edit All	Edit / execute jobs	Edit / execute pipelines



### 3. Project Operations.

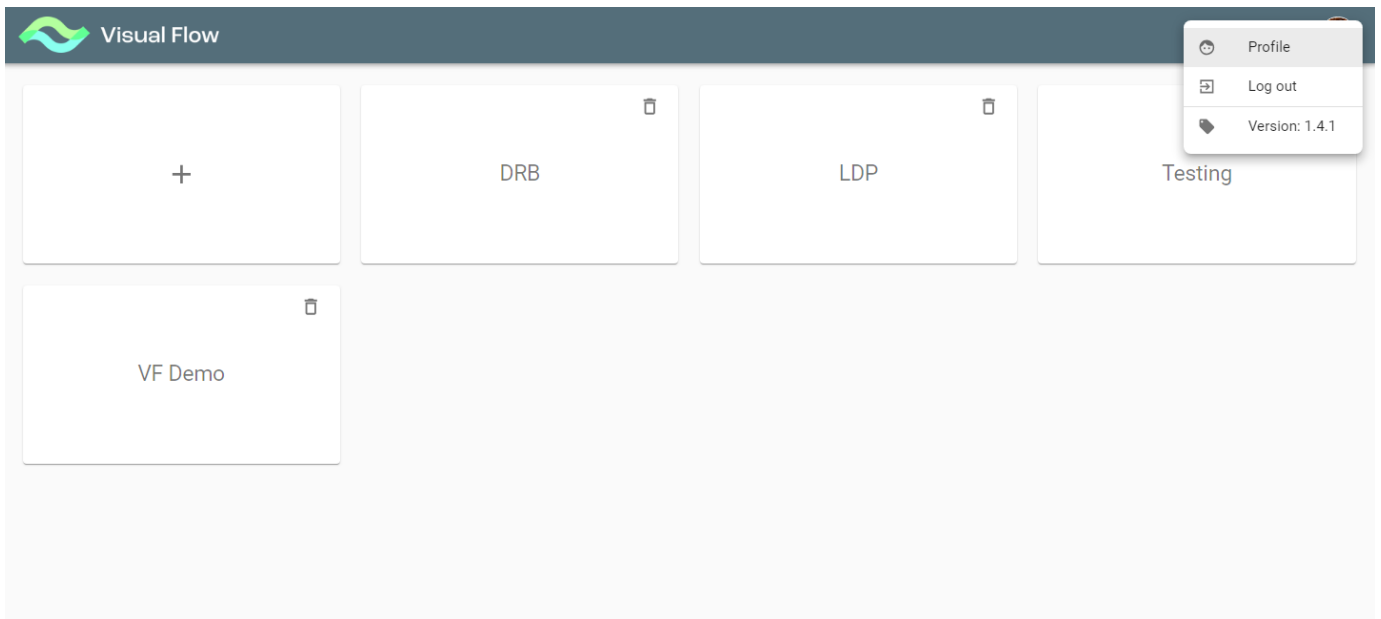
#### 3.1. Getting started

Once you first log on to the application, you see the initial screen with all existing projects:



If you are not authorized for a particular project, it is locked for you, so you see the lock icon on its tile. Please contact project owners to get access to their projects.

If you click on the user icon in the top right corner, you get to user profile menu:

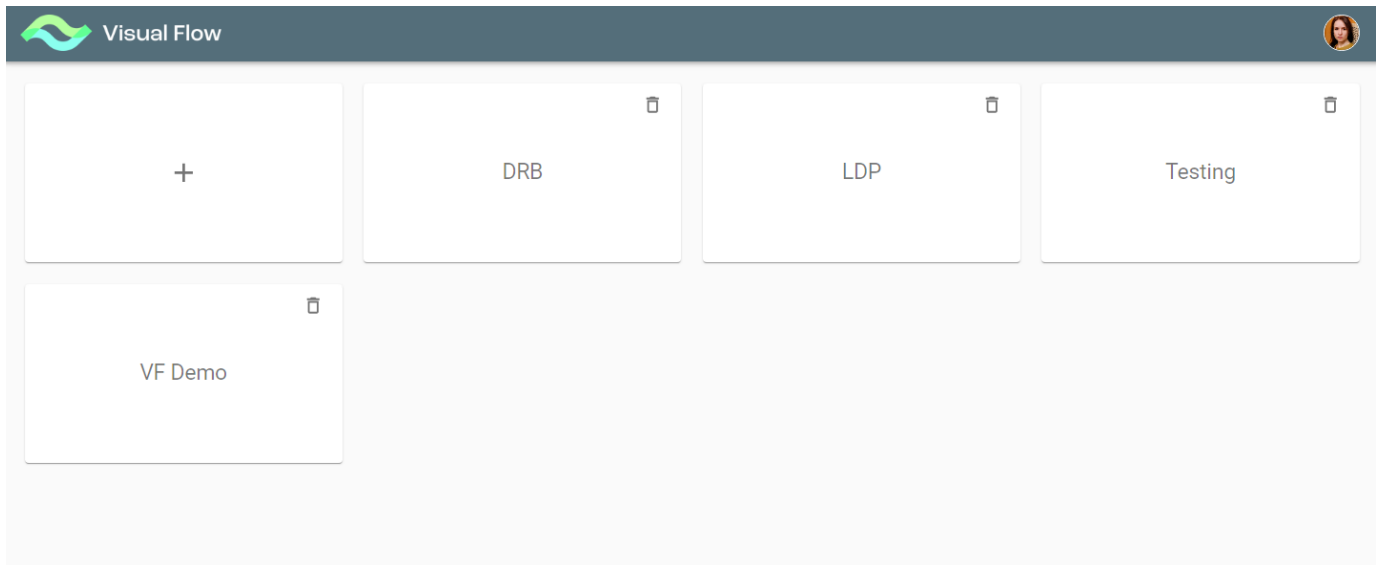


Here you can view your user profile or log out. Also it displays the application current version number.

### 3.2. Create a Project

To create a project, you need to push the “+” button.

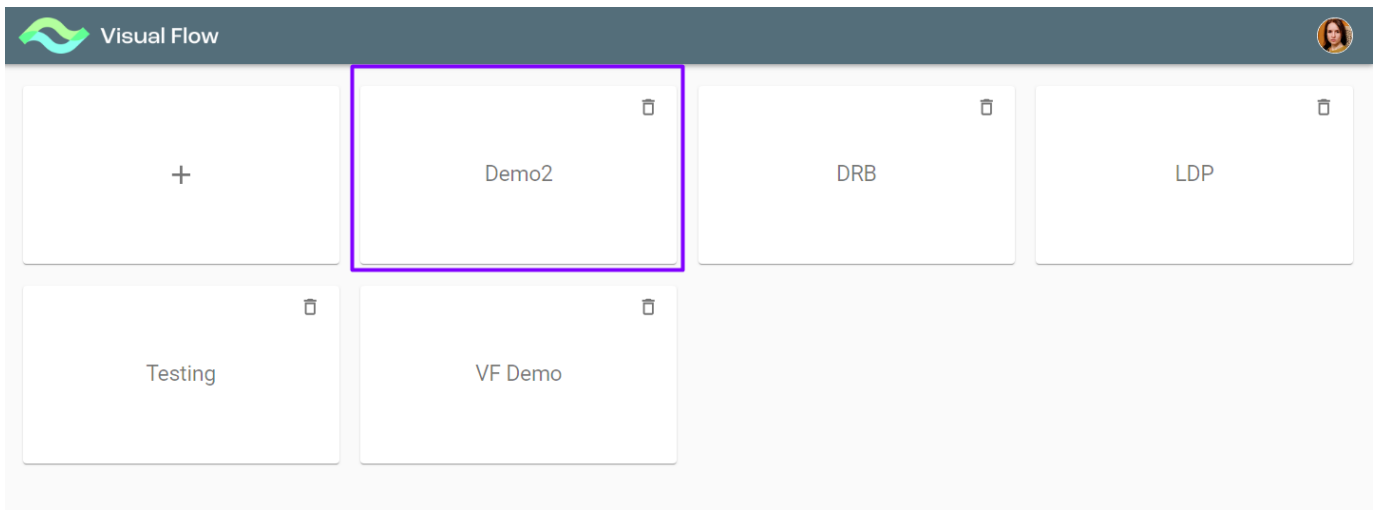
Note: this is the action of the Super-admin user only. The button is not visible for the application roles (Viewer, Operator, Editor, Admin).



With the “+” button pushed, you get to *Create Project Form* to enter basic project settings:

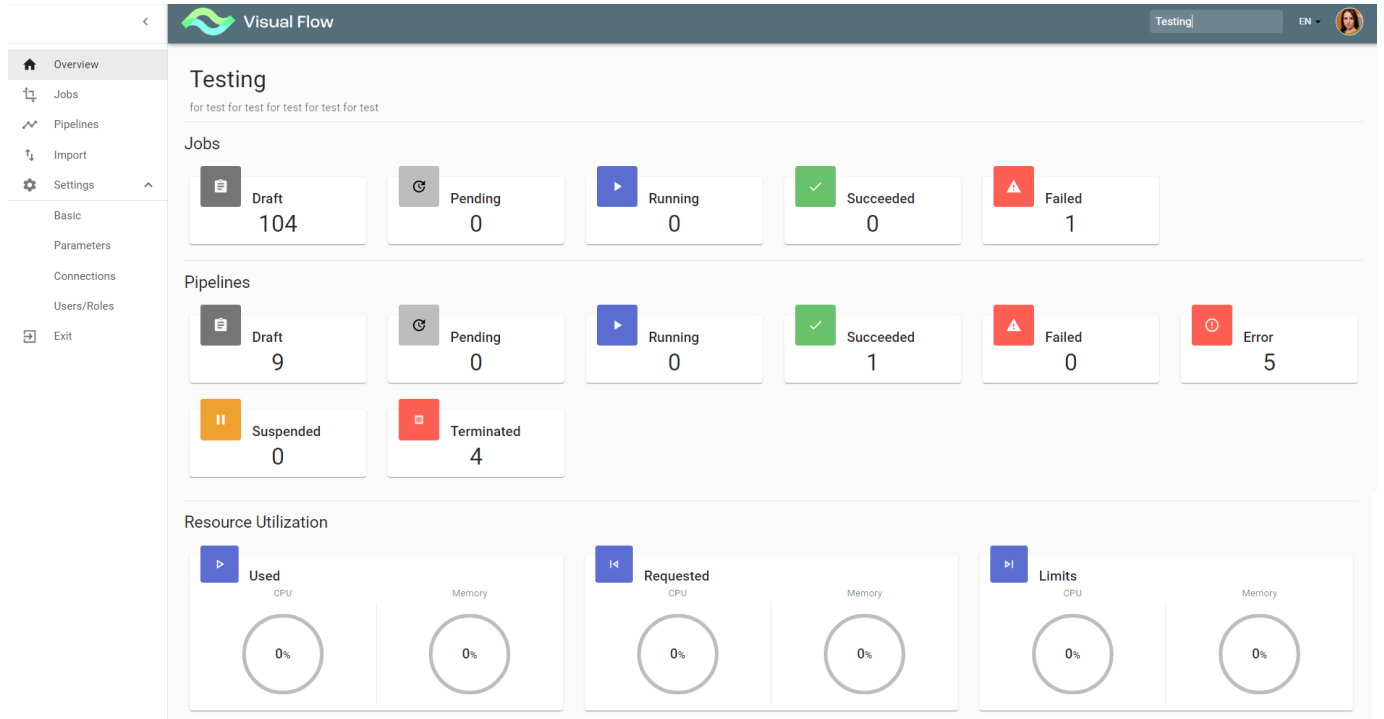
- Project Name
- Project Description
- Requests (CPU/Memory)
- Limits (CPU/Memory)

After saving *Create Project Form* the project is created under the given name and then can be found on the initial screen:



### 3.3. Project Overview

The screen contains the project left menu and displays information about the project jobs, pipelines and resource utilization (applicable for running jobs).

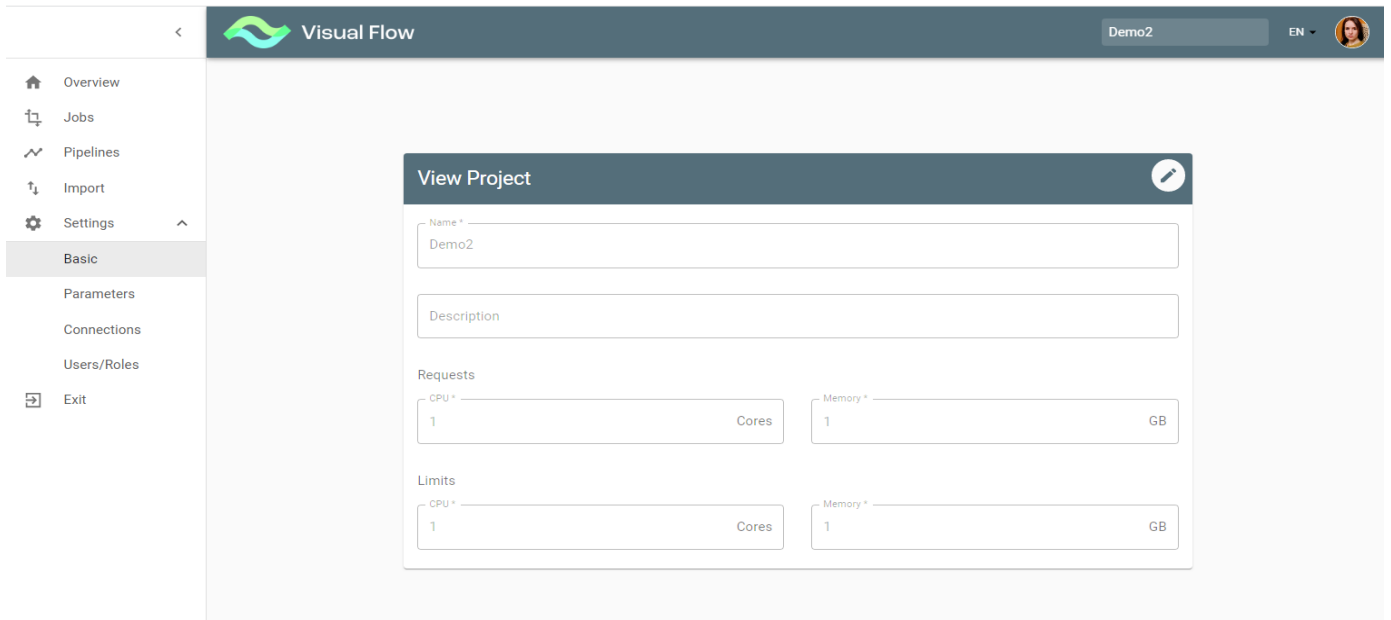


### 3.4. Manage Project Settings

The *Settings* submenu contains:

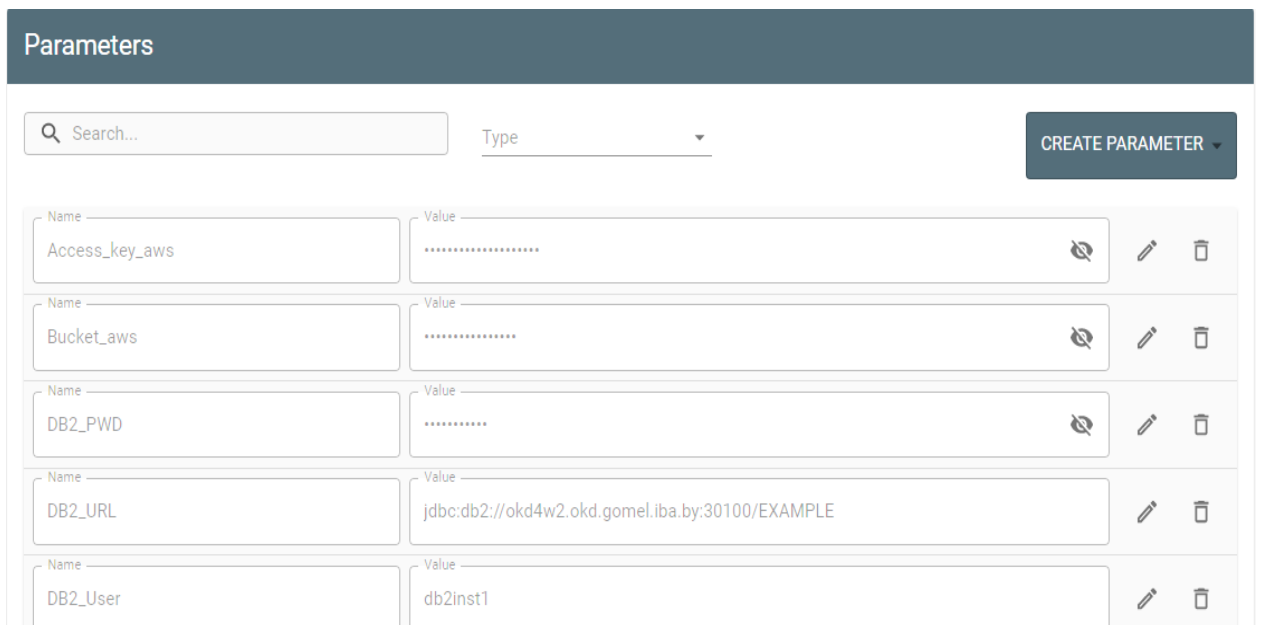
- Basic
- Parameters
- Connections
- Users/Roles

1) The *Basic* is already there after project creation. The *Edit* button turns on the edit mode for updates.



The screenshot shows the 'Visual Flow' application interface. On the left is a sidebar with navigation options: Overview, Jobs, Pipelines, Import, Settings (expanded), Basic (selected), Parameters, Connections, Users/Roles, and Exit. The main area displays the 'View Project' dialog for a project named 'Demo2'. The 'Basic' tab is active, showing fields for Name (Demo2), Description, Requests (CPU: 1, Memory: 1 GB), and Limits (CPU: 1, Memory: 1 GB). An 'Edit' button is visible in the top right corner of the dialog.

2) The *Parameters* stores values required for the entire project, e.g., JDBC connection, DB2 credentials, or table schema can be the same for multiple jobs within a project and therefore stored at the project level. The *Create Parameter* button opens dialog on the right so you can introduce a new parameter.



The screenshot shows the 'Parameters' dialog in the Visual Flow application. It features a search bar, a 'Type' dropdown, and a 'CREATE PARAMETER' button. Below these are five parameter entries, each with a Name field, a Value field, and icons for edit and delete. The parameters are:





















Name	Value
Access_key_aws	.....
Bucket_aws	.....
DB2_PWD	.....
DB2_URL	jdbc:db2://okd4w2.okd.gomel.iba.by:30100/EXAMPLE
DB2_User	db2inst1

- 3) The *Connections* option enables the user to manage connections to a storage. Here you see a list of all existing connections with their name/storage type and available actions (view, edit, delete, ping). Also, you can create a new connection with the *Create Connection* button.


### Manage Connections

Storage ▼


CREATE CONNECTION ▼


Storage Mongo	Name Mongo	   
Storage AWS S3	Name aws	   
Storage Cassandra	Name cassandra	   
Storage DB2	Name db2	   
Storage Oracle	Name oracle	   


- 4) The *Users/Roles* is meant for user access management or to view user access depending on your authorization. Users cannot set roles to themselves. This operation can be performed by Admin or Super-admin only. So if you try to change your role, you will get the error message: “You cannot change your role”. The *Edit* button and therefore *Edit* mode is only available for an *Admin* within the project or for a *Super-admin*.


 Visual Flow


Visual Flow Demo


EN 

 Overview

 Jobs

 Pipelines

 Import


 Settings ^

Basic

Parameters

Connections

**Users/Roles**

 Exit

### Users and Roles


<input type="checkbox"/> ID	Name	Role
<input type="checkbox"/> aapanasko-ibagroup-eu	Apanasko Alina	vf-operator
<input type="checkbox"/> agsadvovsky	Alexander Sadovsky	vf-editor
<input type="checkbox"/> akutsenka-ibagroup-eu	Alina Kutsenka	vf-operator
<input type="checkbox"/> ashubenok	ashubenok-gomel-iba-by Shubenok	vf-operator

Rows per page: 5 ▼ 1-4 of 4 < >

14

## 4. Job operations

### 4.1. Jobs Overview

Clicking the Jobs menu item leads to the Jobs overview screen, which allows you to see a list of jobs existing within a project. If a job is used in a pipeline, it is indicated by  (pipeline) icon. The Jobs overview screen displays the following information:

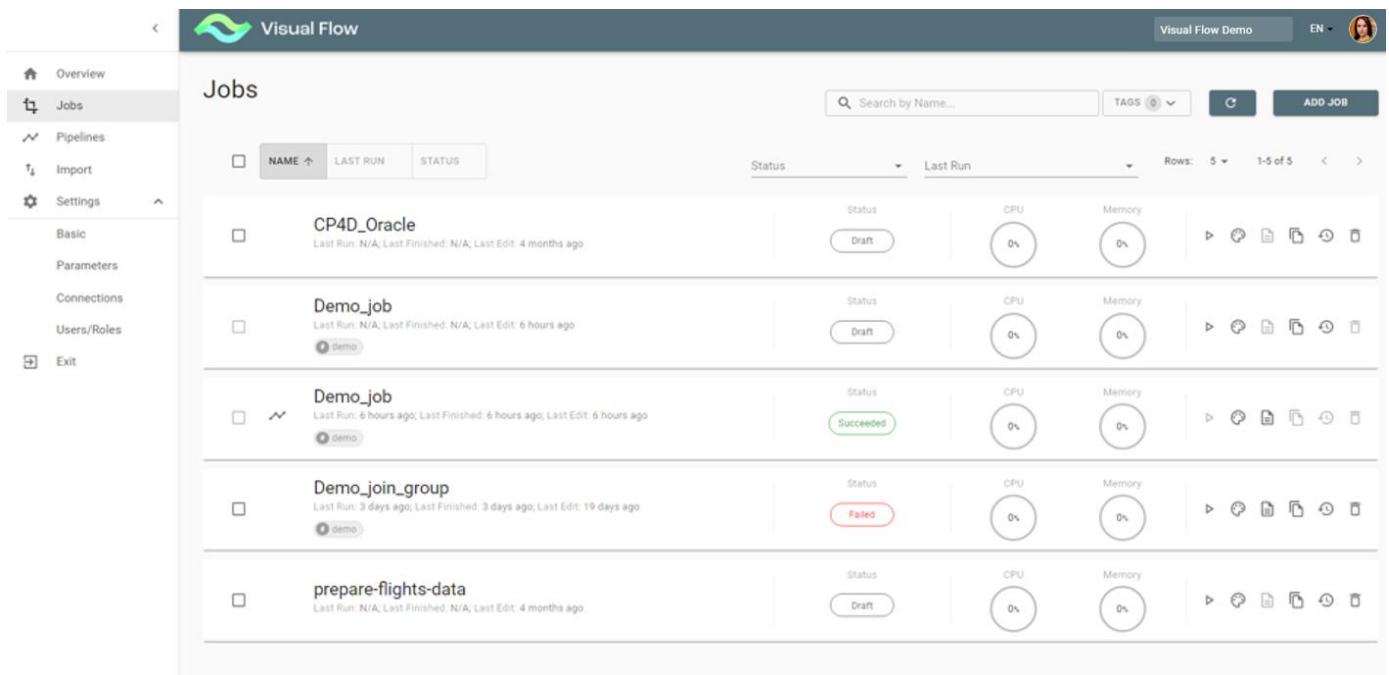
- Job Name
- Job Last run/Last finished/Last edit
- Job Status
- Resource Utilization (CPU/Memory)
- Available Actions (Run/Job Designer/Logs/Copy/Job History/ Delete)

A job has a certain status at various phases of execution:

- Draft
- Pending
- Running
- Succeeded
- Failed
- Unknown (This status appears very rarely in case of undefined error)

Notes:

- The actions availability and, therefore, visibility depends on user authorizations
- One cannot delete a job that compounds a pipeline



NAME	LAST RUN	STATUS	CPU	Memory	Actions
CP4D_Oracle	Last Run: N/A; Last Finished: N/A; Last Edit: 4 months ago	Draft	0%	0%	Run, Copy, Job History, Delete
Demo_job	Last Run: N/A; Last Finished: N/A; Last Edit: 6 hours ago	Draft	0%	0%	Run, Copy, Job History, Delete
Demo_job	Last Run: 6 hours ago; Last Finished: 6 hours ago; Last Edit: 6 hours ago	Succeeded	0%	0%	Run, Copy, Job History, Delete
Demo_join_group	Last Run: 3 days ago; Last Finished: 3 days ago; Last Edit: 19 days ago	Failed	0%	0%	Run, Copy, Job History, Delete
prepare-flights-data	Last Run: N/A; Last Finished: N/A; Last Edit: 4 months ago	Draft	0%	0%	Run, Copy, Job History, Delete

## 4.2. Create a Job. Available Stages.

With the *Add Job* button pushed you get to *Job Designer* to create a new job.

← Please enter name and save params

PALETTE PARAMS Zoom: 20 50 100 150 Status: Draft

Name \*  
Job\_1

Tags  
Tag your job to categorize it

Resources

Driver Request Cores  
0.1

Driver Cores  
1

Driver Memory  
1 GB

Executor Request Cores  
0.1

Executor Cores  
1

Executor Memory  
1 GB

Executor Instances  
2

Shuffle Partitions  
10

CONFIRM DISCARD

You must provide a name for the job on the left configuration panel.

Tags can be used to classify your job.

Update parameters or keep their default values and then push the *Confirm* button:

← Your job has not been saved. If you leave Job Designer unsaved changes will be lost. Press "SAVE" to save your job

PALETTE PARAMS Zoom: 20 50 100 150 Status: Draft

Name \*  
Job\_1

Tags  
Tag your job to categorize it

Resources

Driver Request Cores  
0.2

Driver Cores  
1

Driver Memory  
1 GB

Executor Request Cores  
0.1

Executor Cores  
1

Executor Memory  
1 GB

Executor Instances  
2

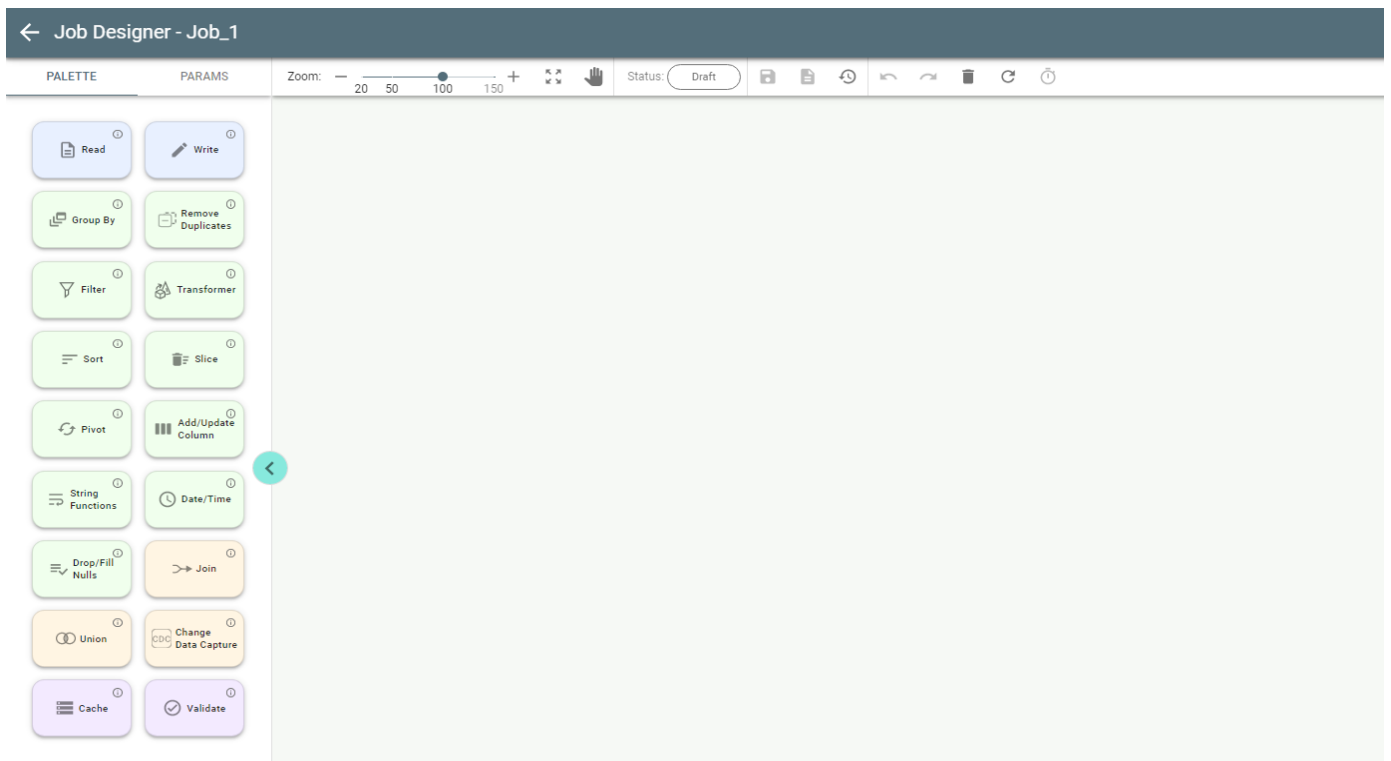
Shuffle Partitions  
10

CONFIRM DISCARD

Save the job by pushing the *Save* button on *Job Designer* header.

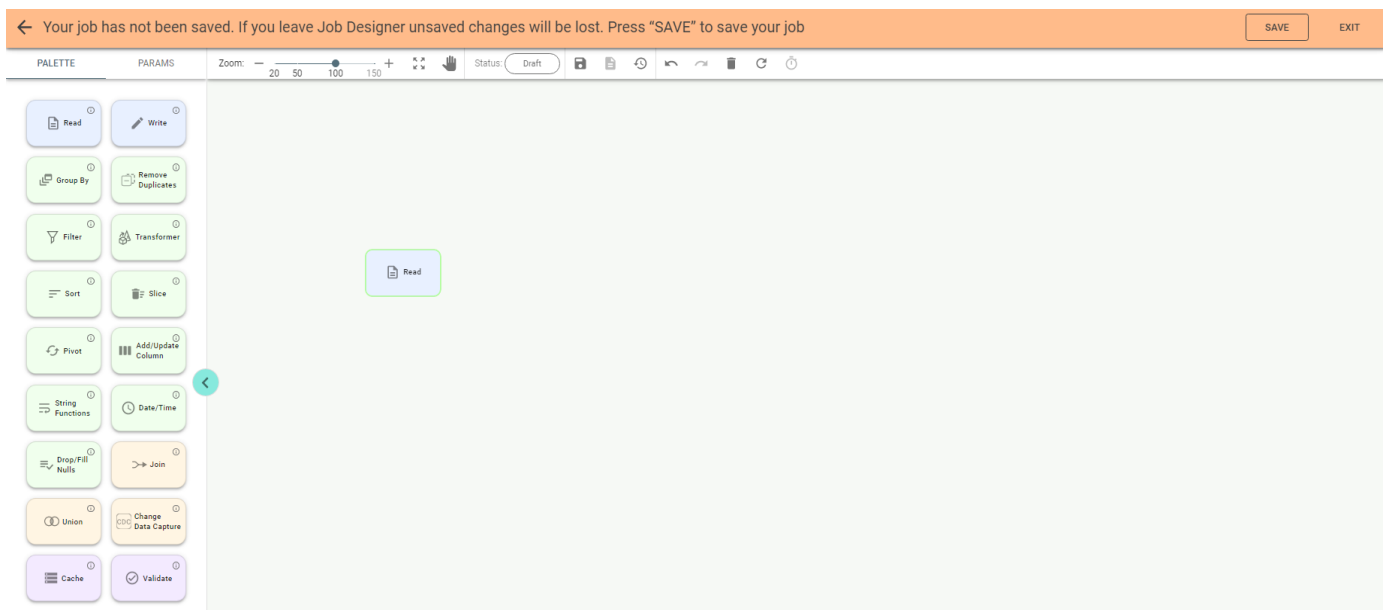
Now you see the *Palette* tab with all available stages:





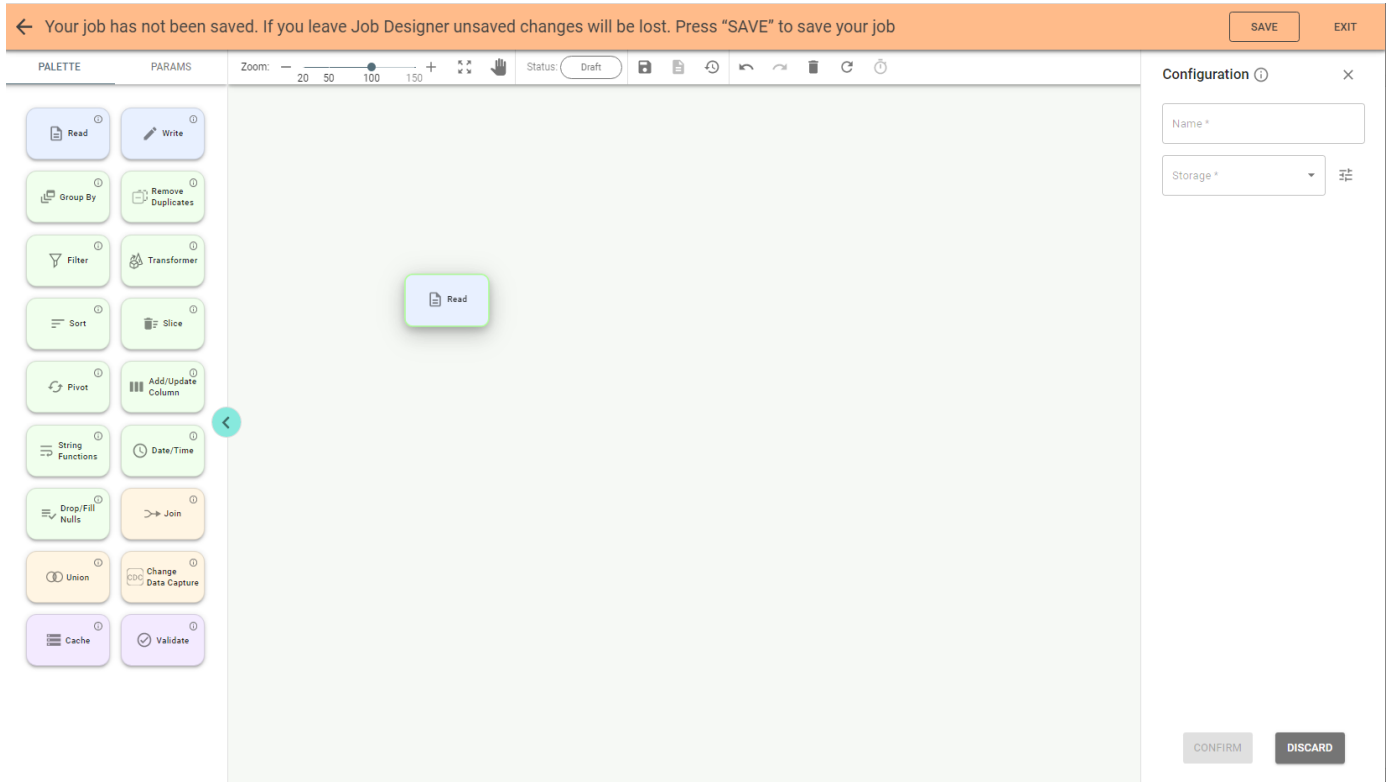
#### 4.2.1. Read Stage.

You can start creating a job by dragging a stage to the canvas, e.g., you can drag the *Read* stage:



Note: you can also add a stage by double clicking its tile on the palette.

Double-click on a stage on canvas opens the configuration panel on the right:



Enter a name for the stage and select storage e.g. *DB2* if you want to read data from the DB2 table.  
Available *Storage* values for *Read* stage are:

- ✓ API
- ✓ AWS S3
- ✓ Cassandra
- ✓ ClickHouse
- ✓ Dataframe
- ✓ DB2
- ✓ Elasticsearch
- ✓ IBM COS
- ✓ Local file
- ✓ Mongo
- ✓ MySQL
- ✓ MSSQL
- ✓ Oracle
- ✓ PostgreSQL
- ✓ Redis
- ✓ Redshift
- ✓ Redshift-jdbc

Important: you can select an existing connection with the *Parameters* button near the Storage field:

Configuration ⓘ

×

Name \*

Read\_tab1

Storage \*

DB2

⚙️

If you do so, its parameters, e.g., JDBC URL, user, etc., are filled automatically.

But now, suppose you don't have a connection created previously, so fill required parameters for *DB2* storage manually:

Configuration ⓘ

×

Name \*

Read\_tab1

Storage \*

DB2

⚙️

JDBC URL \*

jdbc:db2://10.224.0.52

⚙️

×

User \*

db2inst1

⚙️

×

Password \*

.....

👁️ ⚙️

×

Custom SQL

False

⚙️

×

Schema \*

db2inst1

⚙️

×

Table \*

user\_tab1

⚙️

×

CertData

⚙️

×

CONFIRM

DISCARD

Important: you can pick up a parameter value with the *Parameters* ⚙️ button on the right panel if it is previously created as a project parameter.

Configuration ⓘ

Name \*

Read\_tab1

Storage \*

DB2

JDBC URL \*

#jdbc\_url#

The *Read* stage for DB2 storage has an option *Custom SQL* to read data with SQL statement (e.g., *select \* from table where field = value*). If you select *Custom SQL - True* you need to enter the SQL statement and specify the schema:

Custom SQL

True

SQL statement

Select \* from  
db2inst1.user\_tab1 where  
DEPT = 'I'

For the *Redis* source you need to define *Key column*, *Model*, *SSL*, *Read mode*, *Keys pattern* or *Table* fields in the Configuration of the *Read* stage.

- ✓ *Key column*. The Column name to store the hash key.
  - ✓ *Model* (*binary*, *hash*) defines the *Redis* model used to persist dataframe. By default, it is *hash*.
  - ✓ *Read mode* (*key*, *pattern*) defines how the read operation is handled. If *Key* is selected, then the read is done based on the *table* field. In the case of *Pattern* the provided pattern (option *Keys Pattern*) dictates what *Redis* key to read.
- Keys pattern*. If the pattern ends with \* (e.g., “table: \*”), all keys from it are read.  
If one pattern is defined (e.g., “table: first value”), then only one key is read.

Configuration ⓘ

×

Name \*

ReadStage

Storage \*

Redis

⌵

⌵

Host \*

okd4w1.okd.gomel.iba.

⌵

×

Port \*

31380

⌵

×

Password \*

.....

👁

⌵

×

SSL

False

⌵

×

Key column

⌵

×

Model \*

hash

⌵

×

Read mode \*

Pattern

⌵

×

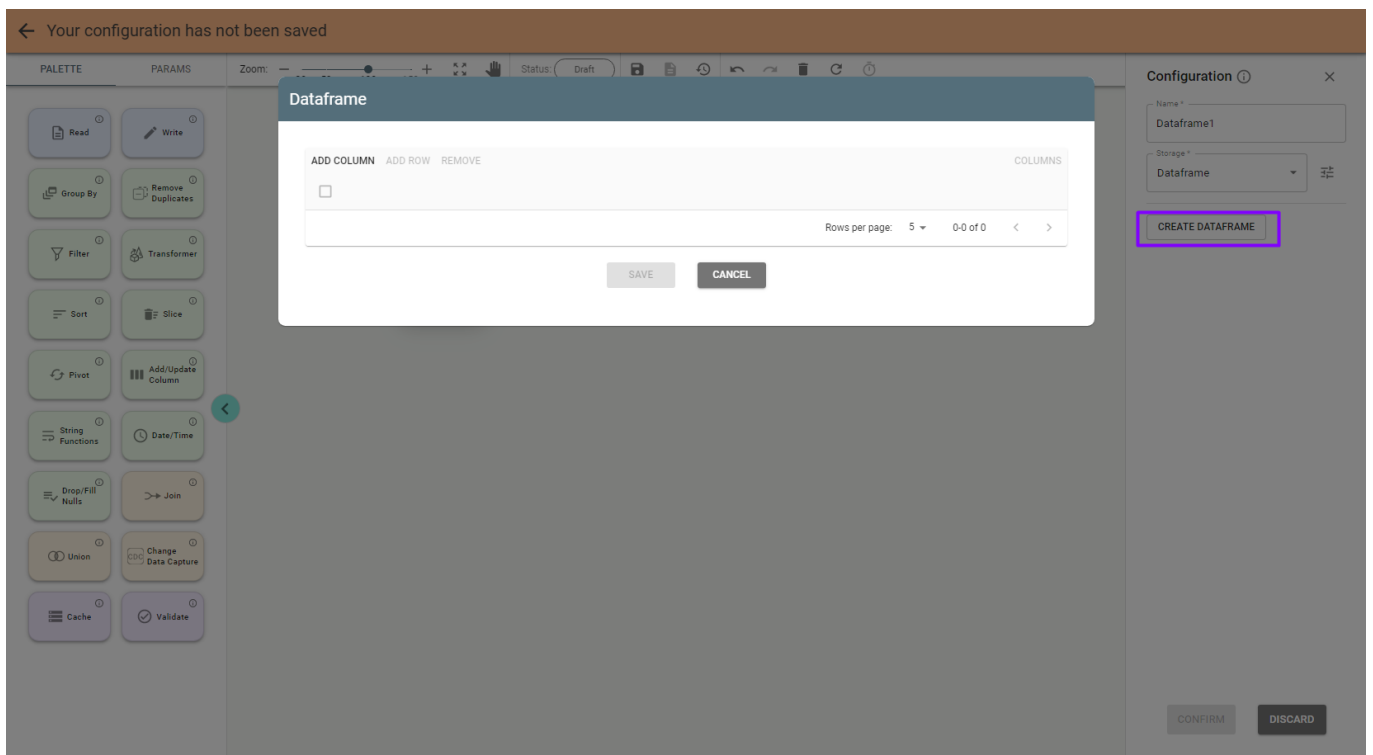
Keys pattern \*

table\_test:\*

⌵

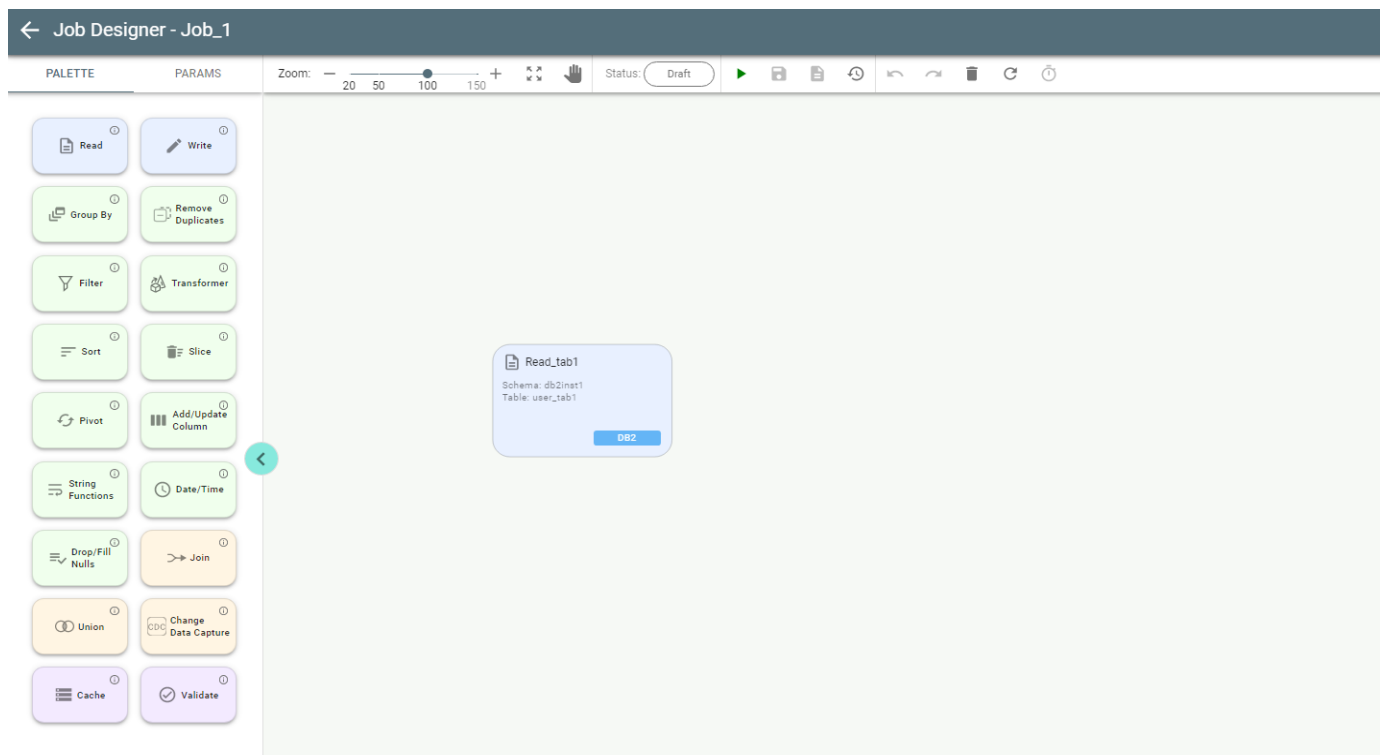
×

Note: with *Dataframe* storage option you can create your own dataset:



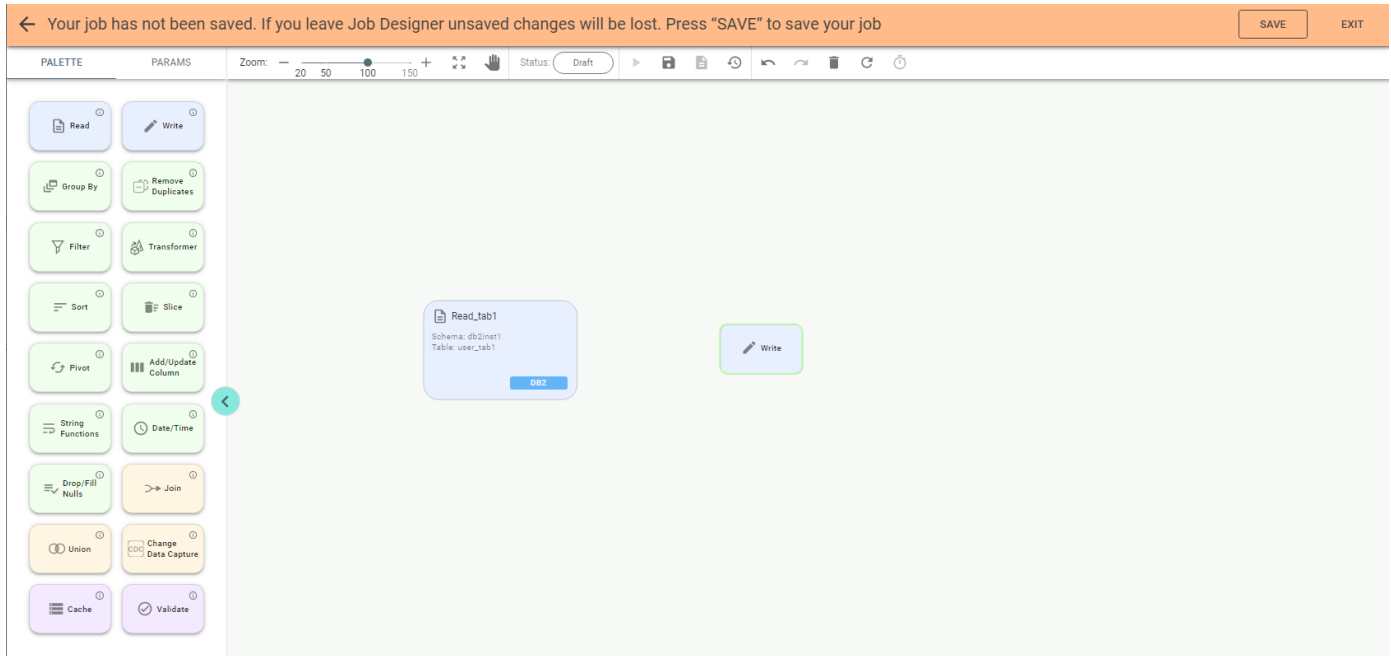
Save the stage by pushing the *Confirm* button on the configuration panel. Push the *Save* button above if you want to save the job at this step.

When the first stage of the job is configured, the canvas looks like this:



### 4.2.2. Write Stage.

Now drag another stage, e.g., Write stage:



Enter a name for the stage and select storage *IBM COS* if you want to post data from the DB2 table to the Cloud Object Storage file. Fill required parameters for the *IBM COS* storage.

Available storage values for the *Write* stage are:

- ✓ AWS S3
- ✓ Cassandra
- ✓ ClickHouse
- ✓ DB2
- ✓ Elasticsearch
- ✓ IBM COS
- ✓ Local file
- ✓ Mongo
- ✓ MSSQL
- ✓ MySQL
- ✓ Oracle
- ✓ PostgreSQL
- ✓ Redis
- ✓ Redshift
- ✓ Redshift-jdbc
- ✓ STDOUT

*IBM COS* storage has two options of *Authentication type*: *HMAC* and *IAM*.

If *HMAC* is selected, you should fill *accessKey* and *secretKey*.

If *IAM* is selected, *iamApiKey* and *iamServiceId* should be entered.

For the storages *IBM COS* and *AWS S3* the function *Partition By* can be used in the *Write* stage. It partitions the output on the file system by given columns. If specified, the output is laid out on the file system similar to Hive's partitioning scheme.

As an example, when we partition a dataset by year and then month, the directory layout looks like this:

- year=2016/month=01/
- year=2016/month=02/

In the case of importing table data with *Write* stage to *Cassandra* source from another storage, the table layout for output must be previously created in *Cassandra*. Columns, key fields, and data types of the fields must be specified in the table.

Important: All the above points must match the imported table.

If the column names have uppercase characters in the imported table when data is output to *Cassandra*, the job will fail as column names are stored there in lowercase only. You can resolve this issue with the *Transformer* stage. The output recorded to *STDOUT* storage can be seen in Logs. The number of records shown in the logs can be specified in the *Quantity* field. The available range is from 1 to 2147483631 records.

For *Redis* source in *Write* stage, the following fields must be specified: *Key column*, *Model*, *SSL*, *TTL*, *Table* and *Write mode*.

- ✓ *Key column*. For writing. It specifies the unique column used as the Redis key. By default, the key is auto-generated.

The screenshot shows a 'Configuration' dialog box with the following fields and values:

- Name: Write2
- Storage: Redis
- Host: okd4w1.okd.gomel.iba
- Port: 31380
- Password: (masked with dots)
- SSL: (dropdown menu)
- Key column: NAME (highlighted with a red box)
- Model: hash
- Table: table\_test
- TTL: 0
- Write mode: Overwrite

At the bottom of the dialog are two buttons: 'CONFIRM' and 'DISCARD'.



- ✓ *TTL*. Data expiration time in seconds. Data doesn't expire if TTL is negative or 0. By default, it is 0. A positive value of *TTL* means the number of seconds in which the data will be removed.

Important:

The *Write mode* field defines how data is posted to its destination. Available values are:

- ✓ Overwrite
- ✓ Append
- ✓ Error if Exists

With Overwrite Write mode Truncate mode can be used for DB2, Oracle, MySQL, PostgreSQL, MSSQL and Redshift:

- ✓ *None*. No truncation occurs, but the target table is deleted and recreated. Note that all the indexes, constraints and other modifiers assigned for this table will be lost.
- ✓ *Simple*. The standard truncation that deletes data from the target table but keeps the indexes, constraints and other modifiers intact. However, if the target table has a primary key referenced as a foreignkey in other tables, the truncation will fail. To resolve this, either use Cascade mode or manually drop constraints (outside of VF) prior to accessing the table with VF.
- ✓ *Cascade* (only for *Oracle* and *PostgreSQL*). The cascade truncation not only deletes the data from the target table but also from other tables that use the target table's primary key as a foreign key constraint.

*File format* is to choose a format of the destination file. Available formats are:

- ✓ CSV
- ✓ JSON
- ✓ Parquet
- ✓ ORC
- ✓ Text
- ✓ Avro

Confirm the stage by pushing *Confirm* on the panel. Now there are two stages to connect.



Important: to connect stages, hover a mouse on a stage edge until you see a green rectangle. Click and drag it to the border of another stage and its green rectangle. When you reach it, a green arrow should appear.



Other stages available are:

- ✓ Group By
- ✓ Remove Duplicates
- ✓ Filter
- ✓ Transformer
- ✓ Sort
- ✓ Slice
- ✓ Pivot
- ✓ Add/Update Column
- ✓ String Functions
- ✓ Date/Time
- ✓ Drop/Fill Nulls
- ✓ Join
- ✓ Union
- ✓ Change Data Capture
- ✓ Cache
- ✓ Validate

#### 4.2.3. Group By Stage.

The stage allows grouping by columns, which must be specified in the Configuration panel. There is an option *Drop grouping columns* for removing them from the output. Also you can use the aggregate function, e.g., Count, Avg, etc.

Configuration ⓘ
✕

Name \*

Group\_by\_dept

☐ Drop grouping columns

Group By \*

DEPT ✕

Aggregate functions \*
+

Count ▾

Column

NAME

—

#### 4.2.4. Remove Duplicates Stage.

Specify a key column for the operation. To specify more than one key, use a comma or Enter. For *Order By* operation, you need to select columns to sort by and sort order. The default is ascending.

The screenshot shows a configuration window titled 'Configuration' with a close button (X) in the top right corner. It contains three main sections: 1. 'Name \*' with a text input field containing 'Remove duplicates'. 2. 'Key \*' with a text input field containing 'name' and a small 'X' icon to its right. 3. 'Order By \*' with a dropdown menu set to 'asc' and a text input field containing 'name'. To the right of the 'Order By' section is a '+' icon, and below the 'name' input is a '-' icon.

#### 4.2.5. Filter Stage.

Enter any boolean expression. You can combine two or more ones by using logical operators (AND,OR). Examples: (column1 < 10) and (column2 between 10 and 25).

The screenshot shows a configuration window titled 'Configuration' with a close button (X) in the top right corner. It contains two main sections: 1. 'Name \*' with a text input field containing 'Filter\_DEPT\_I'. 2. 'Filter \*' with a large text area containing the expression 'DEPT='I''.

#### 4.2.6. Transformer Stage.

The *Transformer* stage allows you to modify columns to fill them with data further. *Transformer* mode defines the type of SQL query (Spark SQL dialect) which is taken to execute.

*Simple* mode only allows you to specify the part between SELECT and FROM. You can do things like these:

```
col1, concat(col1, col2)
count(*) as count
a, b, row_number() OVER (PARTITION BY a ORDER BY b)
col, exists(col, x -> x % 2 == 0)
col, collect_list(col)
```

The syntax is: <column\_name\_1> as <alias\_1>, function(<column\_name\_2>) as <alias\_2>.

*Full SQL* mode allows you to write a full-blown Spark SQL query. In this case, you have to specify a table name manually or reference a table name from a parameter.

*Table name*. The name of the table which should be used within the Spark SQL query. Applicable for *Full SQL* transformer mode only.

#### 4.2.7. Sort Stage.

There are two types of sorting available: *Full sort* and *Sort within partitions*.

*Full sort* classifies dataframe by the specified column(s).

*Sort within partitions* sorts each dataframe partition by the specified column(s). In this case, the order of the output data is not guaranteed because the data is ordered at the partition level.

Select column(s) to sort by and the sort order (default value is Asc).

Available sort options:

- asc
- asc nulls first
- asc nulls last
- desc
- desc nulls first
- desc nulls last

#### 4.2.8. Slice Stage.

The *Slice* stage allows you to remove unnecessary columns from your data stream. There are 2 modes of *Slice* stage:

- *Keep*
- *Drop*

The default mode is Drop. In this case, you specify columns you would like to slice from your dataflow.

With Keep mode selected, you specify the columns you want to keep.

#### 4.2.9. Pivot Stage.

The *Pivot* stage has two operation types:

- *Pivot*
- *Unpivot*

*Pivot* function rotates data from one column into multiple columns (transpose a row to a column).

With aggregation one of the grouping column values is transposed into individual columns with distinct data.

Product	Amount	Country
Banana	1000	USA
Carrots	1500	USA
Beans	1600	USA
Orange	2000	USA
Orange	2000	USA
Banana	400	China
Carrots	1200	China
Beans	1500	China

Orange	4000	China
Banana	2000	Canada
Carrots	2000	Canada
Beans	2000	Mexico

From the above dataframe, to get the total amount exported to each country of each product we group by Product, pivot by Country and sum the Amount. This transposes the countries from the dataframe rows into columns and produces the below output. Missing data is represented as Null.

Product	Canada	China	Mexico	USA
Orange	Null	4000	Null	4000
Beans	Null	1500	2000	1600
Banana	2000	400	Null	1000
Carrots	2000	1200	Null	1500

Unpivot is a reverse operation used to transform it back by rotating column values into rows values. For *Pivot* operation you need to specify:

- *Group By.* Columns for grouping
- *Pivot columns*
- *Aggregate function and column*
- *Pivot values.* Values from pivot column to be used (optional)

For *Unpivot* operation you need to specify:

- *Unchanged columns*
- *Unpivot columns.* Columns for unpivoting
- *Unpivot names.* Column names for unpivoting

#### 4.2.10. Add/Update Column Stage.

The *Add/Update Column* stage is used to add a new column to a dataframe, change value of an existing column, convert the data type of a column or derive a new column from an existing column. Also it gives the ability to use conditions or window functions to provide column values.

It has the following operations types:

- *deriveColumn*
- *addConstant*
- *changeType*
- *renameColumn*
- *useConditions*
- *useWindowFunctions*

If you select *useWindowFunctions* you need to pick a function from the dropdown list. Each one has its own parameters to enter.

The available window functions are:

- *rank*
- *dense\_rank*
- *percent\_rank*
- *cume\_dist*
- *row\_number*

- *ntile*
- *lag*
- *lead*
- *count*
- *sum*
- *avg*
- *max*
- *min*

#### 4.2.11. String Functions Stage.

The *String Functions* stage is handy when we need to make string operations on a dataframe column. The *Operation type* dropdown list contains the following Spark built-in standard string functions:

- *ascii*
- *base64*
- *concat\_ws*
- *decode*
- *encode*
- *format\_number*
- *format\_string*
- *title\_case*
- *instr*
- *length*
- *lower*
- *locate*
- *lpad*
- *ltrim*
- *regexp\_extract*
- *ubase64*
- *rpadd*
- *repeat*
- *rtrim*
- *split*
- *substring*
- *substring\_index*
- *trim*
- *upper*

#### 4.2.12. Date/Time Stage.

Date/Time stage leverages Spark standard built-in date and timestamp functions which come in handy when we need to make operations on date and time. All these operations accept input of type: date, timestamp or string. If string, it should be in a format that can be cast to date, such as *yyyy-MM-dd* and timestamp in *yyyy-MM-dd HH:mm:ss.SSSS*. It returns date and timestamp respectively. Null is returned if the input data was a string that could not be cast to date or timestamp.

#### 4.2.13. Drop/Fill Nulls Stage.

Drop/Fill Null stage allows gracious null handling. There are 2 modes:

- *Drop*
- *Fill*

With *Drop* mode you can:

- 1) Remove all columns where all the rows have null values. For that you need to set *Drop Type* to *Column* and *Drop Choice* to *All*.
- 2) Remove specific columns where all the rows have null values. For that you need to set *Drop Type* to *Column*, *Drop Choice* to *Names* and specify column names you want to drop.
- 3) Remove all rows where all the values are null. For that you need to set *Drop Type* – *Row* and *Drop Choice* – *All*.
- 4) Remove all rows, where all the values are null in specific columns. (Works as a filter). For that you need to set *Drop Type* to *Row*, *Drop Choice* to *Names* and specify drop column names.

*Fill* mode is used to replace NULL values on a dataframe column with any constant value or using aggregate function. ( *Fill Value Type* - *Custom* or *Agg* respectively ).

*Fill Value Type* - *Custom* allows you to replace all columns with one value at once or specify a specific value per column you want to fill with. For that you need to select *Fill Choice* - *All* or *Names* respectively.

*Fill Value Type* - *Agg* allows you to fill a specific column with a *Strategy*, which stands for the aggregation types: "mean", "median" or "mode".

#### 4.2.14. Join Stage.

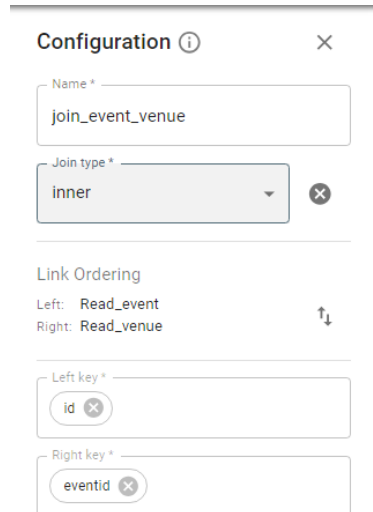
Available types of join are:

- *Inner* join. Transfers records from input data sets whose key columns contain equal values to the output data set. Records whose key columns do not contain equal values are dropped.
- *Left outer* join. Transfers all values from the left data set but transfers them from the right one only when key columns match. The stage drops the key column from the right data set.
- *Right outer* join. Transfers all values from the right and left data sets. Intermediate them only when key columns match. The stage drops the key column from the left and intermediate data sets.
- *Full outer* join. Transfers records in which the contents of the key columns are equal from the left and right input data sets to the output. It also transfers records whose key columns contain unequal values from input and output data sets.
- *Cross* join. Returns a result data set where each row from the first table is combined with each from the second table.

- *Left semi* join. Returns values from the left relation that has a match with the right.
- *Left anti* join. Returns values from the left relation that has NO match with the right.

*Link Ordering* option allows you to specify which input link is regarded as the left and which as the right. By default, the first link added is left, and the last one is right.

With *Left Key* and *Right Key* you must enter the key of joining.



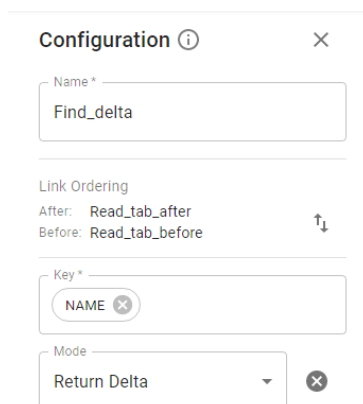
The screenshot shows a 'Configuration' dialog box with the following fields:

- Name \***: A text input field containing 'join\_event\_venue'.
- Join type \***: A dropdown menu set to 'inner'.
- Link Ordering**: A section with 'Left: Read\_event' and 'Right: Read\_venue', accompanied by a swap icon (two arrows forming a square).
- Left key \***: A text input field containing 'id'.
- Right key \***: A text input field containing 'eventid'.

#### 4.2.15. Change Data Capture Stage.

This stage is intended to find all differences between before (old) and after (new) datasets. Based on them, CDC produces an additional column — 'Operation', which indicates the state of the row from the old dataset considering its presence/absence in the new one. CDC compares each row of the new and the old datasets based on keys and columns to compare values and sets Operation value.

Note: old and new datasets must not contain duplicates (rows with the same key) based on key column(s). Old and new datasets columns to compare, as well as key columns, must be present in both datasets with the same name. If there are duplicated rows in at least one dataset, the result of the CDC gets unpredictable.



The screenshot shows a 'Configuration' dialog box for the CDC stage with the following fields:

- Name \***: A text input field containing 'Find\_delta'.
- Link Ordering**: A section with 'After: Read\_tab\_after' and 'Before: Read\_tab\_before', accompanied by a swap icon (two arrows forming a square).
- Key \***: A text input field containing 'NAME'.
- Mode**: A dropdown menu set to 'Return Delta'.



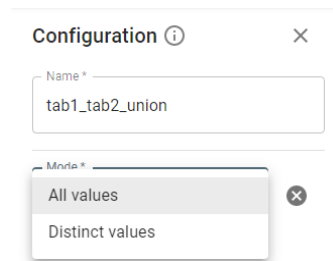
#### 4.2.16. Union Stage.

You can union two datasets.

Note: Columns sequence, names, and types are important for union operation.

*Mode* contains 2 options: *All values* and *Distinct values*.

- ✓ *Distinct values*, which is the default, eliminates duplicate records from the second dataset.
- ✓ *All values* needs to be specified explicitly, and it tolerates duplicates from the second dataset.



Configuration ⓘ X

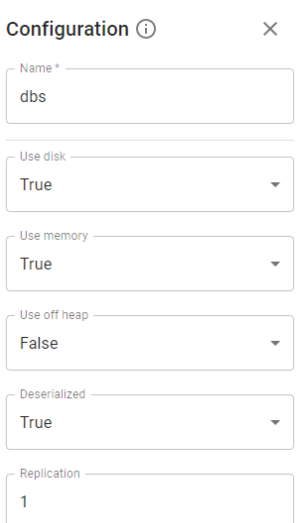
Name \*  
tab1\_tab2\_union

Mode \*  
All values X  
Distinct values

#### 4.2.17. Cache Stage.

The *Cache* stage persists dataset in some storage. You can tweak the storage type by specifying parameters. The configuration gives you the ability to define:

- ✓ Whether to use memory.
- ✓ Whether to drop the RDD to disk if it falls out of memory.
- ✓ Whether to keep the data in memory in a serialized format.
- ✓ Whether to replicate the RDD partitions on multiple nodes.



Configuration ⓘ X

Name \*  
dbs

Use disk  
True ▼

Use memory  
True ▼

Use off heap  
False ▼

Deserialized  
True ▼

Replication  
1

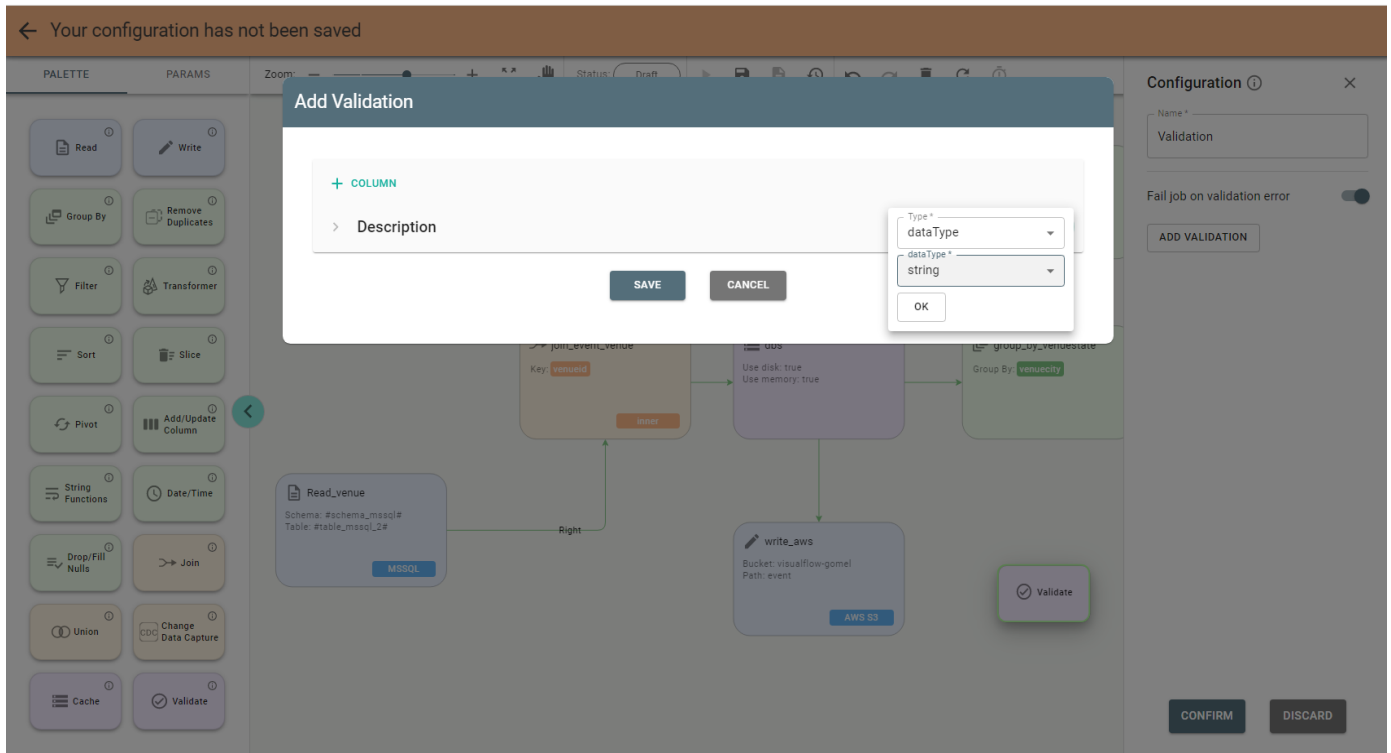
#### 4.2.18. Validate Stage

With the *Validate* stage you can run the data validation using Spark job validation types:

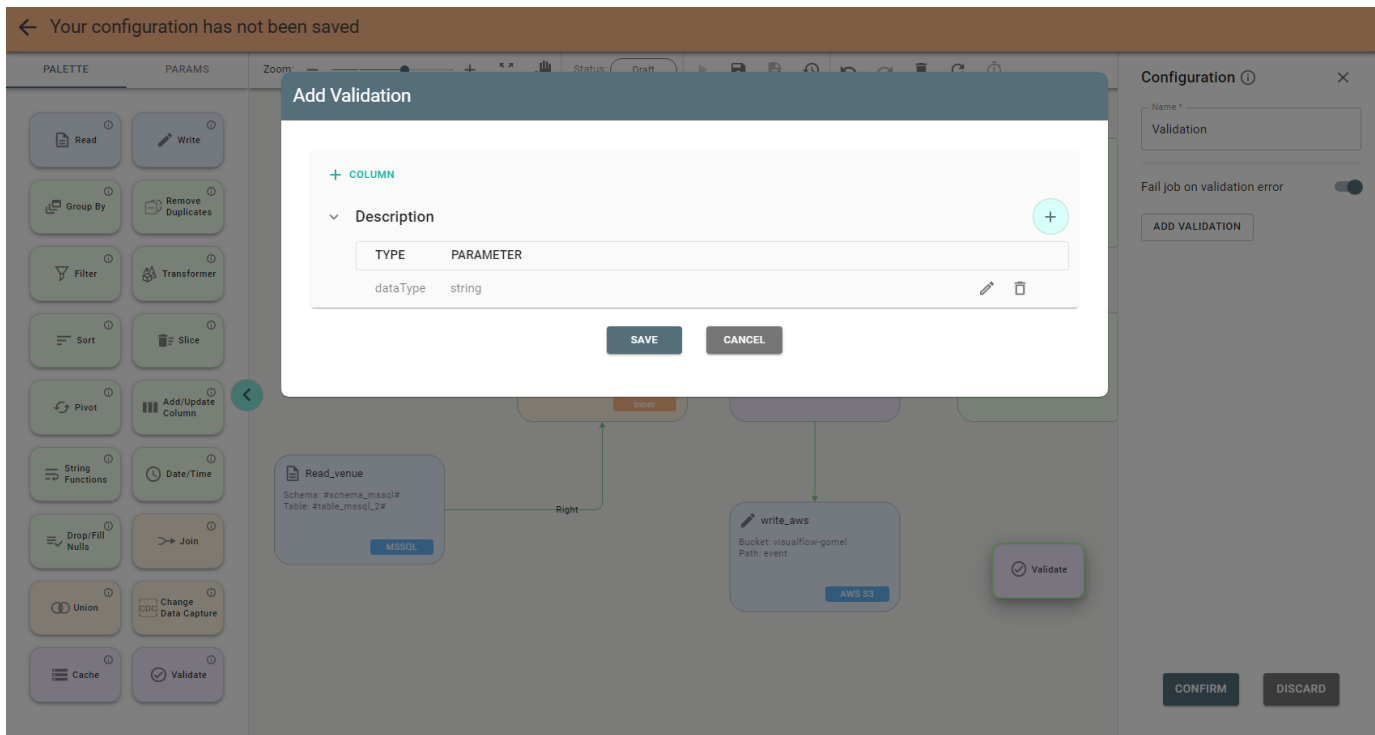
- dataType
- minValue

- maxValue
- minLength
- maxLength
- notNull
- inValues
- regex
- uniqueness

To add a validation you need to push *Add Validation* button on the stage configuration panel and start adding a column validation:



When you hit Ok the validation row appears:

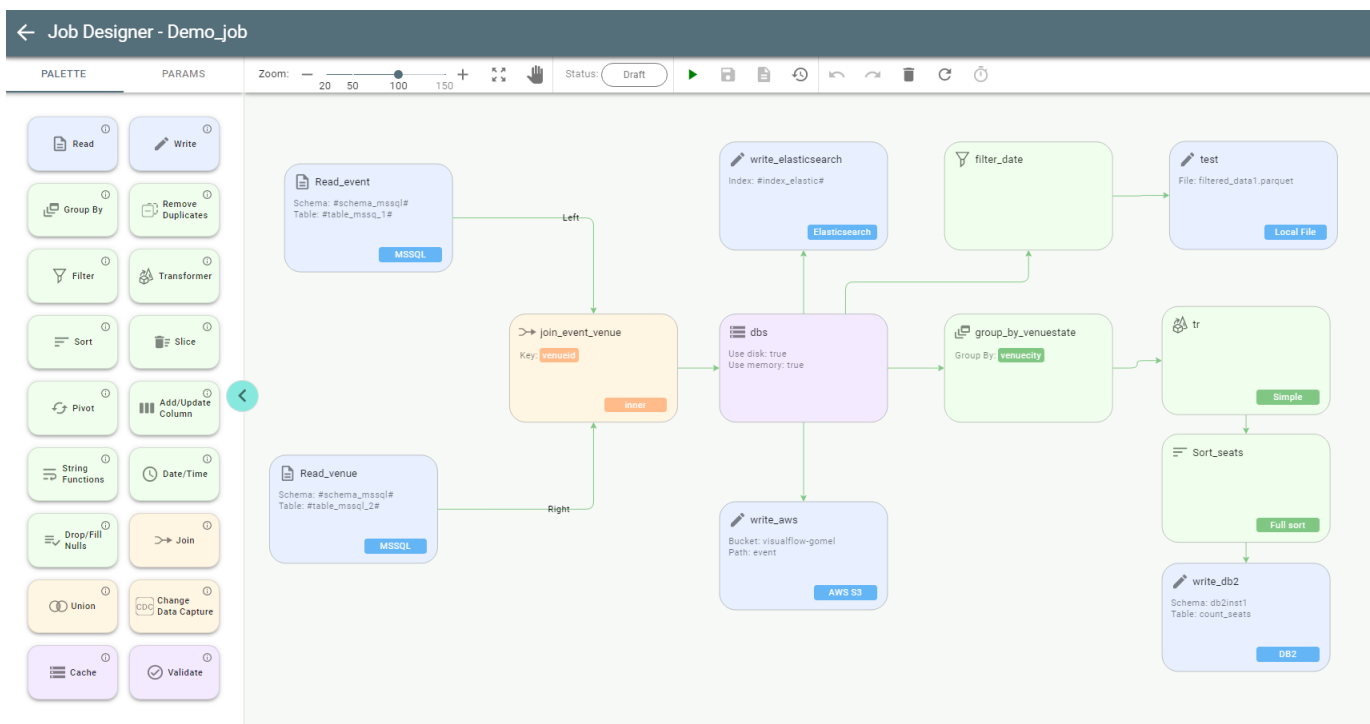


Once you are done with adding all validations push *Save* on the dialog screen and then confirm the stage configuration.

Save the job by pushing *Save* on *Job Designer* header.

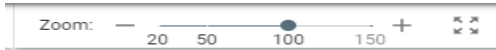













For a newly created job, as long as it is not yet run, its status is *Draft*:  
 Drag other stages according to the data flow from source to destination.  
 See the job with more stages as an example:

Status: Draft



### 4.3. Job Designer functions overview


The following functions are available in *Job Designer*:

- ✓ Zoom operations: 
- ✓ Switch mode between  
Move  and Select 
- ✓ Show job status 
- ✓ Run job  / Stop job  (for running)
- ✓ Save job 
- ✓ See job logs 
- ✓ See job history 
- ✓ Undo / Redo operation on canvas  
- ✓ Remove element from canvas 
- ✓ Refresh 
- ✓ Auto refresh 

Note: you can copy, edit or delete a stage on canvas using the mouse right-click menu options:

- ✓ Edit stage
- ✓ Copy stage
- ✓ Delete stage

### 4.4. Job Execution


Push the *Play* button  to run the job:

Its status changes from *Draft* to *Pending*



Push Refresh to update the status. It should turn to *Running*



While running, it can be interrupted with the *Stop* button.  When a job is completed, its status is *Succeeded* or *Failed*.

Use the *Logs* button



to analyze job logs. It gets you to *Logs* screen:

2022-09-23 12:59:14,947 - WARN - org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

2022-09-23 12:59:16,856 - INFO - org.apache.spark.internal.Logging - Running Spark version 3.2.2

2022-09-23 12:59:16,958 - INFO - org.apache.spark.internal.Logging - Submitted application: 99078258-e065-464e-bd7e-8a6a21d9c327

2022-09-23 12:59:16,959 - INFO - org.apache.spark.internal.Logging - No custom resources configured for spark.driver.

2022-09-23 12:59:16,959 - INFO - org.apache.spark.internal.Logging - Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)

2022-09-23 12:59:17,141 - INFO - org.apache.spark.internal.Logging - Limiting resource is cpus at 1 tasks per executor

2022-09-23 12:59:17,145 - INFO - org.apache.spark.internal.Logging - Added ResourceProfile id: 0

2022-09-23 12:59:17,345 - INFO - org.apache.spark.internal.Logging - Changing view acls to: job-user

2022-09-23 12:59:17,346 - INFO - org.apache.spark.internal.Logging - Changing modify acls to: job-user

2022-09-23 12:59:17,347 - INFO - org.apache.spark.internal.Logging - Changing view acls groups to:

2022-09-23 12:59:17,347 - INFO - org.apache.spark.internal.Logging - Changing modify acls groups to:

2022-09-23 12:59:17,348 - INFO - org.apache.spark.internal.Logging - SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(job-user); groups with view permissions: Set(); users with modify permissions: Set(job-user); groups with modify permissions: Set()

2022-09-23 12:59:17,931 - INFO - org.apache.spark.internal.Logging - Successfully started service 'sparkDriver' on port 14536.

2022-09-23 12:59:17,967 - INFO - org.apache.spark.internal.Logging - Registering MapOutputTracker

2022-09-23 12:59:18,137 - INFO - org.apache.spark.internal.Logging - Registering BlockManagerMaster

2022-09-23 12:59:18,173 - INFO - org.apache.spark.internal.Logging - Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information

2022-09-23 12:59:18,174 - INFO - org.apache.spark.internal.Logging - BlockManagerMasterEndpoint up

2022-09-23 12:59:18,233 - INFO - org.apache.spark.internal.Logging - Registering BlockManagerMasterHeartbeat

Logs can be filtered by level. Such as:

- ✓ INFO
- ✓ WARNING
- ✓ ERROR
- ✓ DEBUG
- ✓ RESULT

You can also view earlier job runs data with the *Job History* button:

Visual Flow

Visual Flow Demo

Jobs


Search by Name...

TAGS 0

ADD JOB


NAME	LAST RUN	STATUS
CP4D_Oracle	Last Run: N/A; Last Finished: N/A; Last Edit: 4 months ago	Draft
Demo_job	Last Run: 3 minutes ago; Last Finished: 2 minutes ago; Last Edit: 6 hours ago	Succeeded
Demo_join_group	Last Run: N/A; Last Finished: N/A; Last Edit: 6 hours ago	Draft
Job_1	Last Run: N/A; Last Finished: N/A; Last Edit: a day ago	Draft
prepare-flights-data	Last Run: N/A; Last Finished: N/A; Last Edit: 4 months ago	Draft





It takes you to *Job History* screen, containing each job run data, including logs:



Demo\_job

Last Run by: alemeshevskaya-ibagroup-eu



	STARTED	DURATION	RUN BY	LOG
▼ Today				
	At 14:21	1m 47s	alemeshevskaya-ibagroup-eu	
▼ Yesterday				
	At 06:36	2m 11s	alemeshevskaya-ibagroup-eu	

## 5. Pipeline Operations

### 5.1. Pipelines Overview

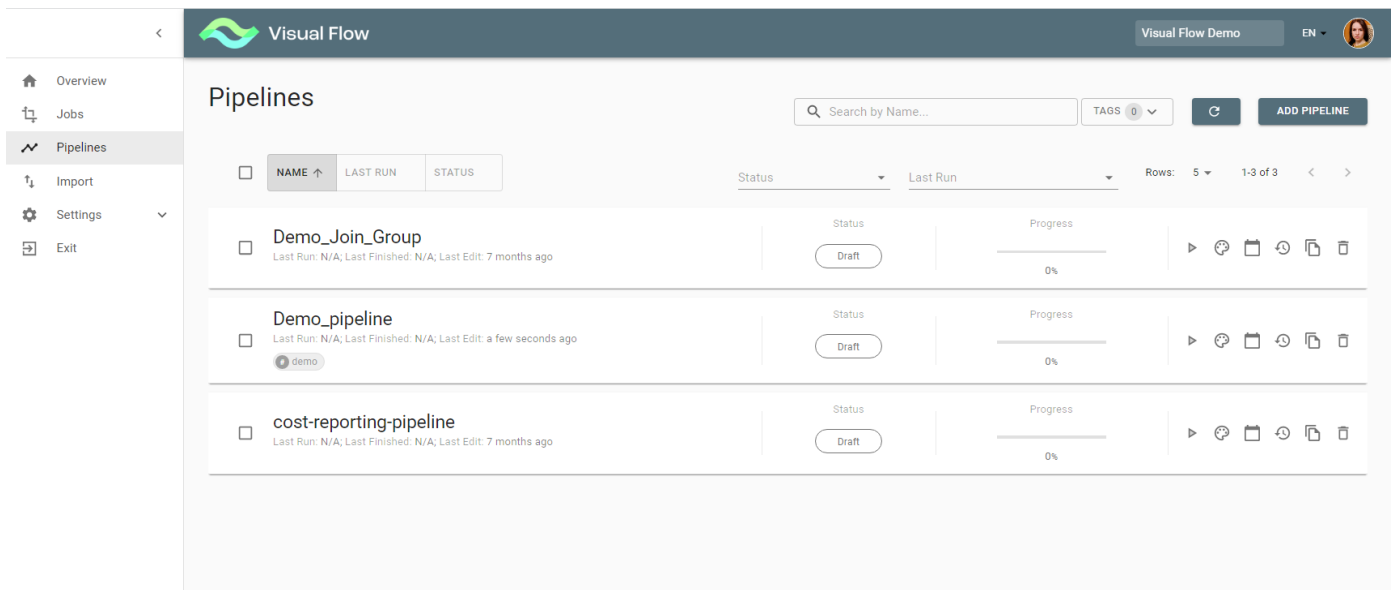
Clicking the *Pipelines* menu item takes you to the Pipelines overview screen, which shows a list of pipelines existing within a project.

It displays the following information:

- Pipeline Name
- Checkbox for deleting/exporting multiple pipelines
- Pipeline Last run/Last finished/Last edit
- Pipeline Status
- Pipeline Progress
- Available Actions (Run/Pipeline Designer/ Scheduling /History/Copy/Delete)

Pipeline has a certain status at various phases of execution:

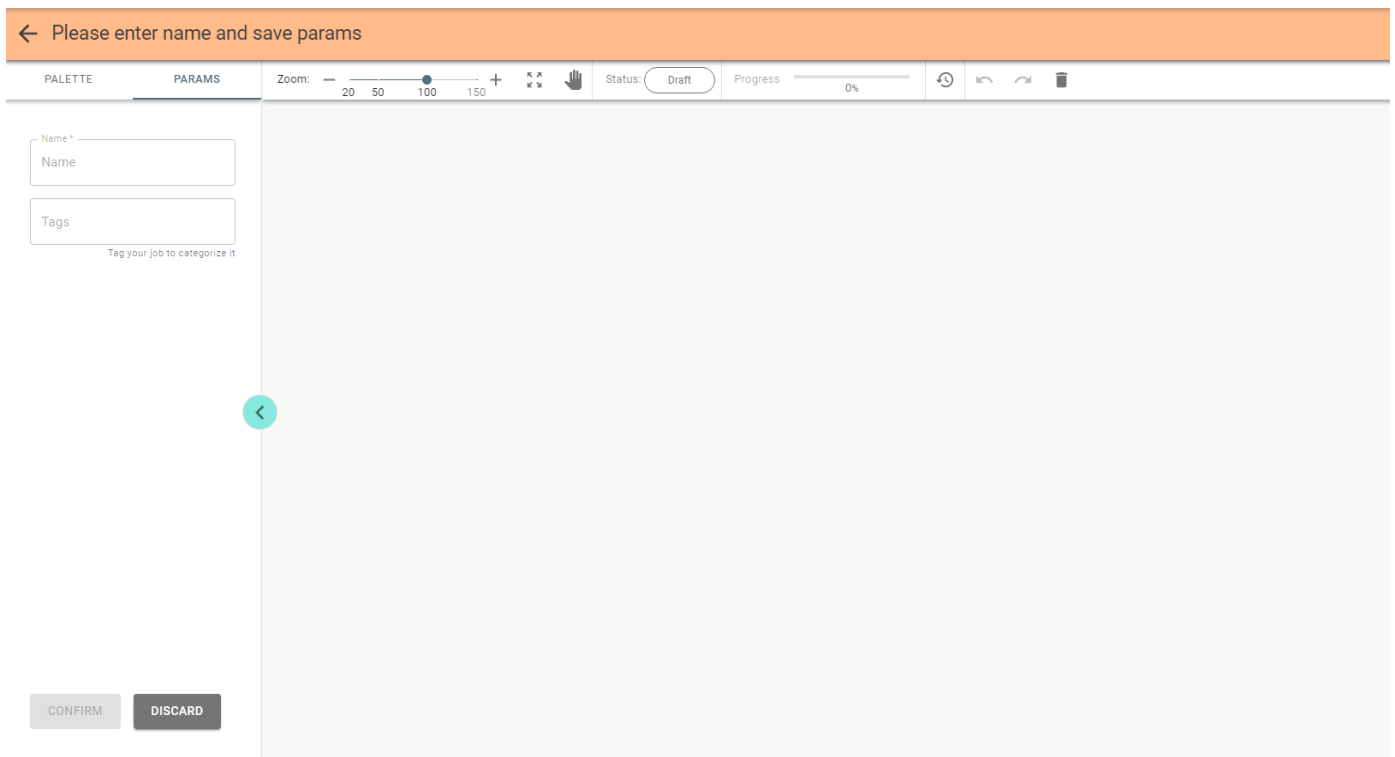
- Draft
- Error (This status may appear due to incorrectly entered data)
- Failed
- Pending
- Running
- Succeeded
- Suspended
- Terminated



Note: the actions availability and therefore visibility depends on user authorizations.

## 5.2. Create a Pipeline

With the *Add Pipeline* button pushed, you get to *Pipeline Designer* for creating a pipeline. On the left configuration panel the *Params* tab is open by default, where you can enter a pipeline name and tags for the pipeline classification:



Once you add a name, notifications become available to you:

← Please enter name and save params

PALETTE PARAMS Zoom: 20 50 100 150 + -

Name \*  
Pipe1

Tags  
Tag your job to categorize it

Notifications

SLACK EMAIL

Notify about failure ☐

Notify about success ☐

CONFIRM DISCARD

Set notifications to be notified on the pipeline events as you wish and push the *Confirm* button.

Note: you can set it specifically for Slack or email notifications.

Save the pipeline by pressing the *Save* button on *Pipeline Designer* header.

Once a pipeline is saved the *Palette* tab with all available stages is opened by default:

← Pipeline Designer - Pipe1

PALETTE PARAMS Zoom: 20 50 100 150 + -

Job Pipeline

Container Notification

Wait

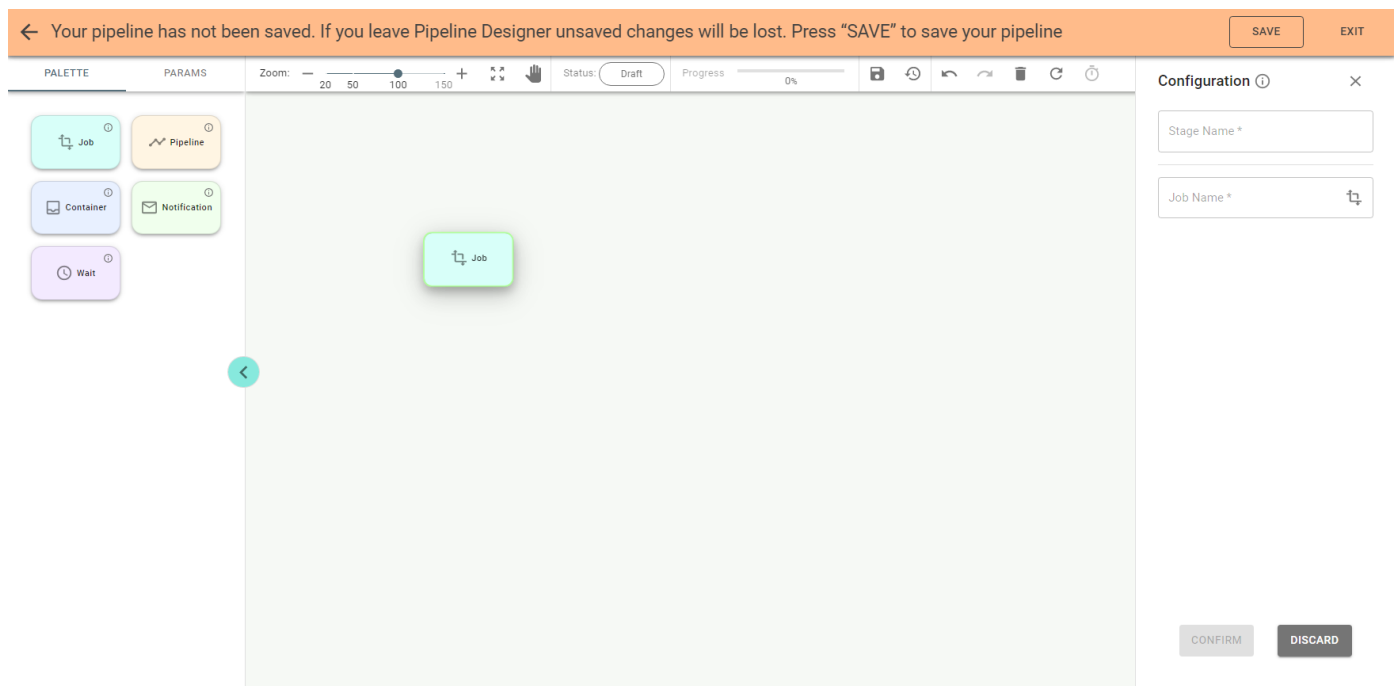



Pipeline is a combination of *Job*, *Pipeline*, *Notification*, *Container* and *Wait* stages.  
The *Notification* stage is most often added to the configuration to notify about job/pipeline stage failure/success.

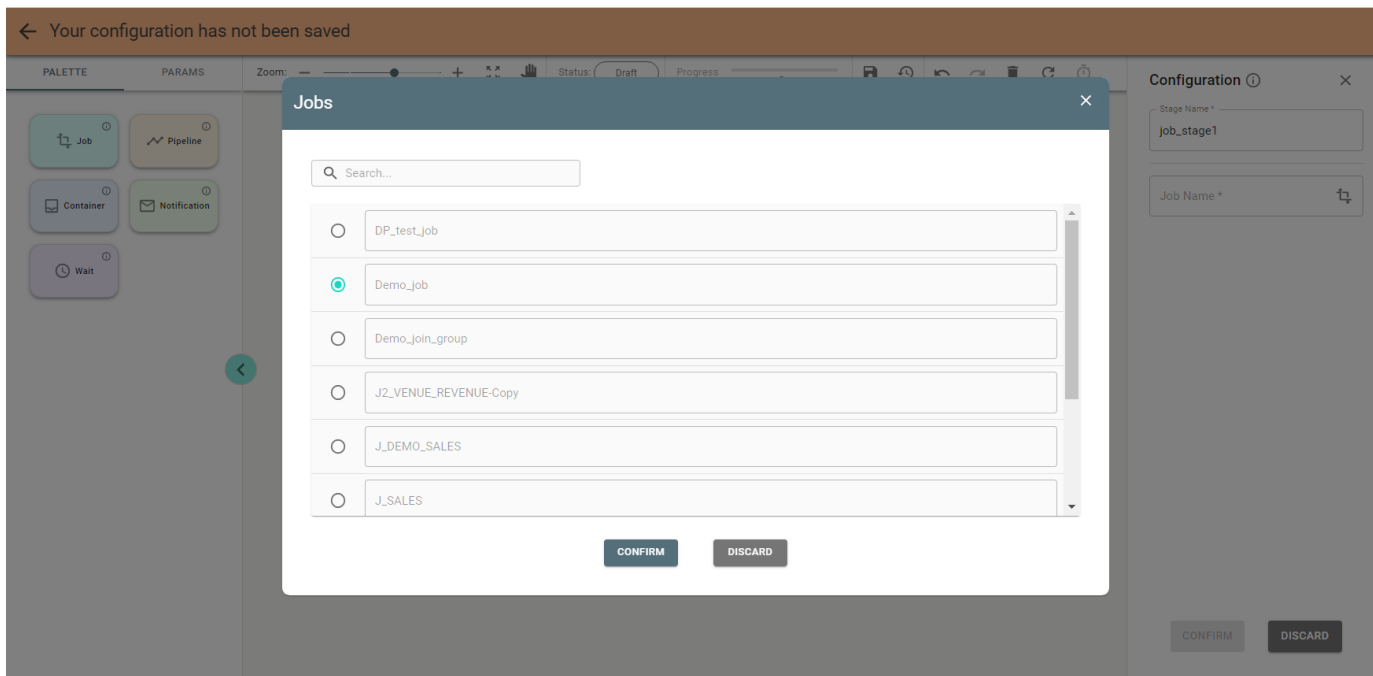
Drag the *Job* stage to the canvas:



Double-click on the stage opens the configuration panel on the right:



Enter the name for the stage and push  to select a job.



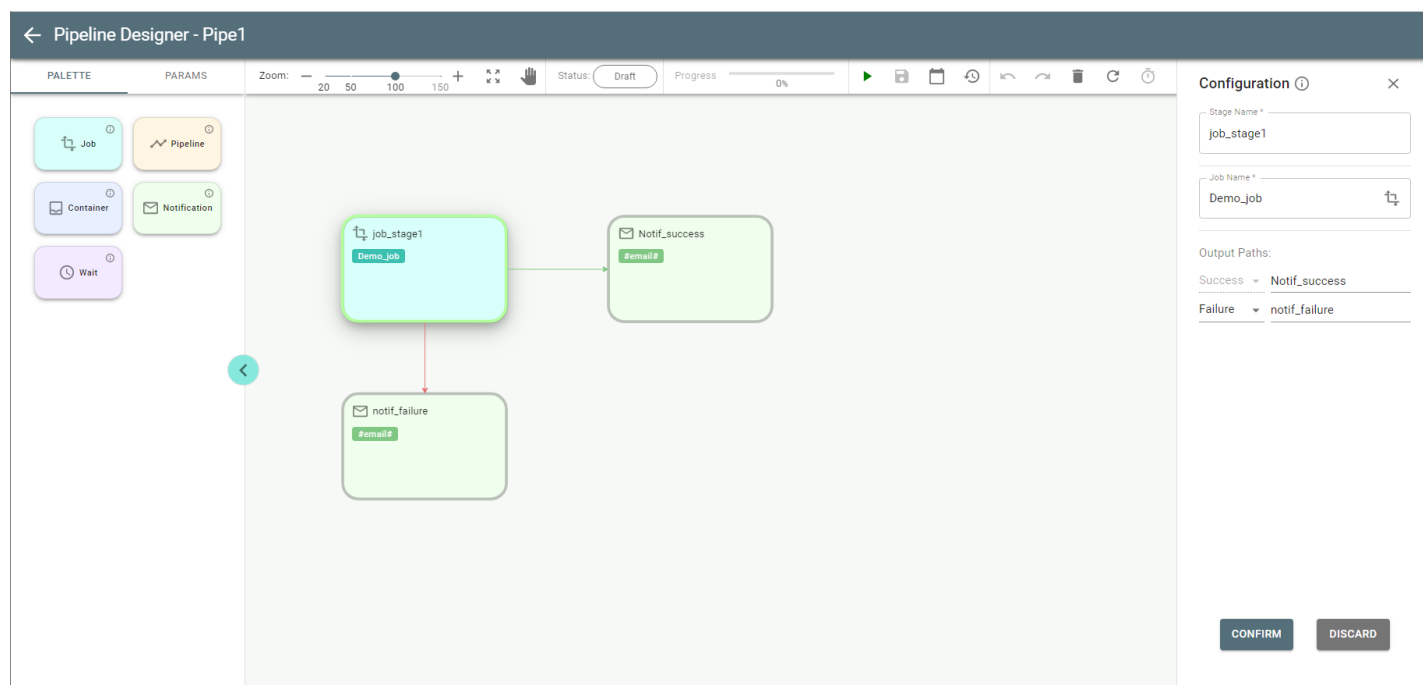
Pick the job and push the *Confirm* button.

Save the stage by pushing the *Confirm* button on the panel. Push the *Save* button on the header if you want to save the pipeline at this step.

Similarly to the *Job* stage the *Pipeline* stage can be used if you want to evoke existing pipeline within your pipeline. Drag and configure other stages. Connect them in the same manner as in *Job Designer*.

Note: you can also add a stage by double clicking its tile on the palette.

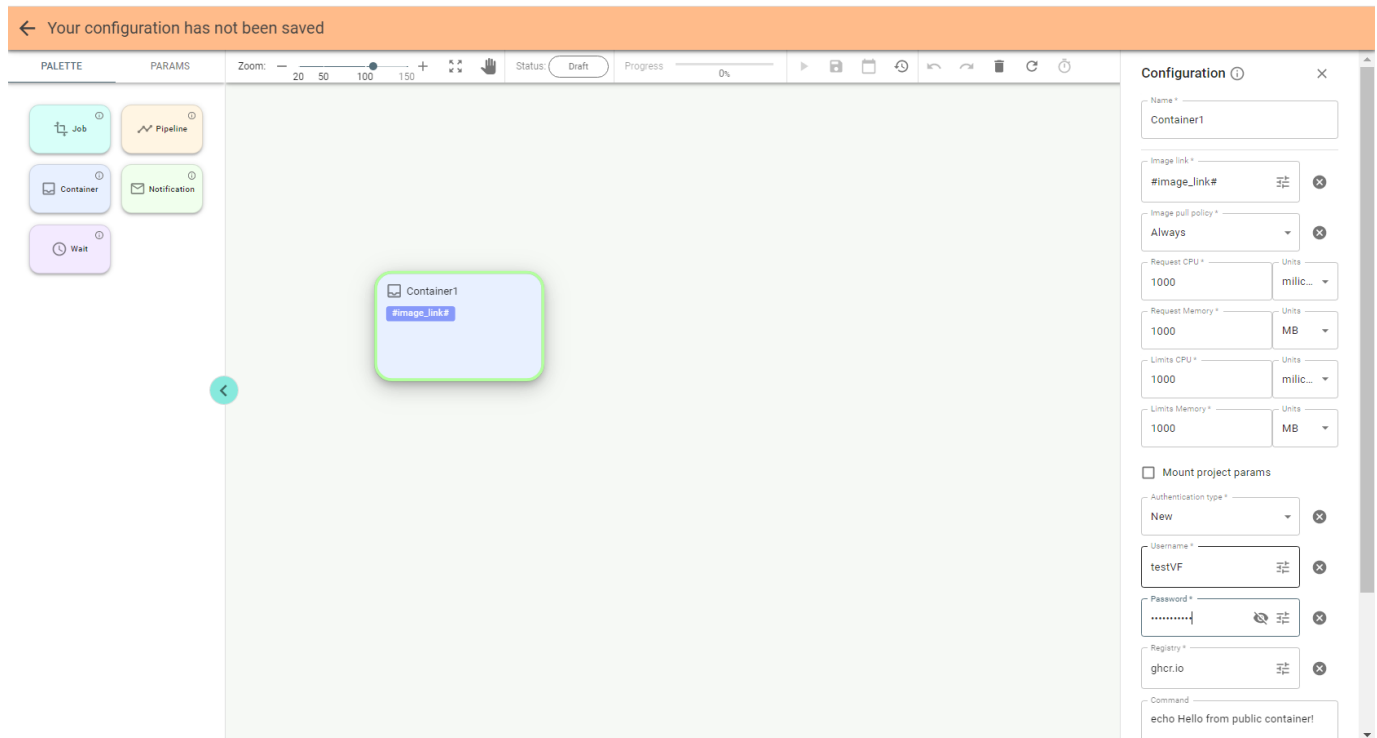
You can link stages based on their success or failure. After connecting them to each other you can choose the *Success* or *Failure* link on the *Job* stage configuration panel. There can be only one connection for failure. See the example of configured pipeline:



Before the first run or after updating, the status of the pipeline is *Draft*. See each stage border painted in a *Gray* color, which stands for *Draft*.

The *Container* stage is used to run custom commands for executing any logic in a pipeline. You can use docker images instead of custom commands.


Start creating a pipeline by dragging the *Container* stage to the canvas and entering parameters in the Configuration panel:



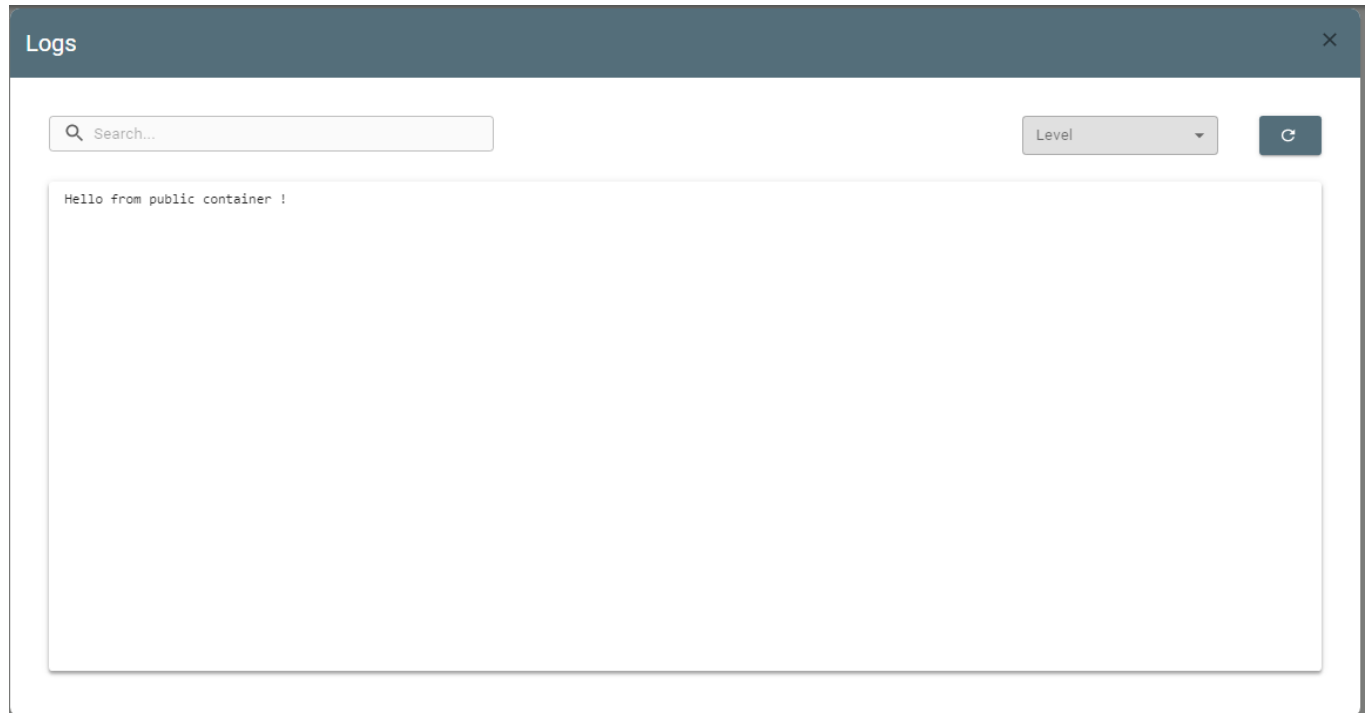
The *Container* stage has the following fields in the configuration:

- ✓ *Image link*. Docker image path.Examples:
  - ✓ mysql, mysql:latest,
  - ✓ bitnami/argo-cd:2.1.2, localhost:5000/bitnami/argo-cd:2.1.2,registry.redhat.io/rhel7:latest.
- ✓ *Image pull policy*. Defines when the image is pulled (downloaded).Possible values:
  - *If not present* – is downloaded only if it does not exist locally;
  - *Always* – is downloaded before each start;
  - *Never* – is not downloaded, a local copy is used.
- ✓ *Requests and Limits CPU*
- ✓ *Requests and Limits memory*
- ✓ *Mount project parameters*. Defines whether to mount all project parameters as environmentvariables inside the Pod.
- ✓ *Authentication type*
- ✓ *Authentication mode* can be one of these:
  - *Not applicable*: image pull secrets are not required as the image is pulled from the public registry;
  - *New*: create a new image pull secret on the fly by providing all necessary information;
  - *Provided*: use the existing image pull secret by providing its name (Image pull secret name).
- ✓ *Image pull secret name*. Name of the secret to pull the image. Note that it must exist within thesame k8s namespace as the current pipeline.
- ✓ *Username*
- ✓ *Password*

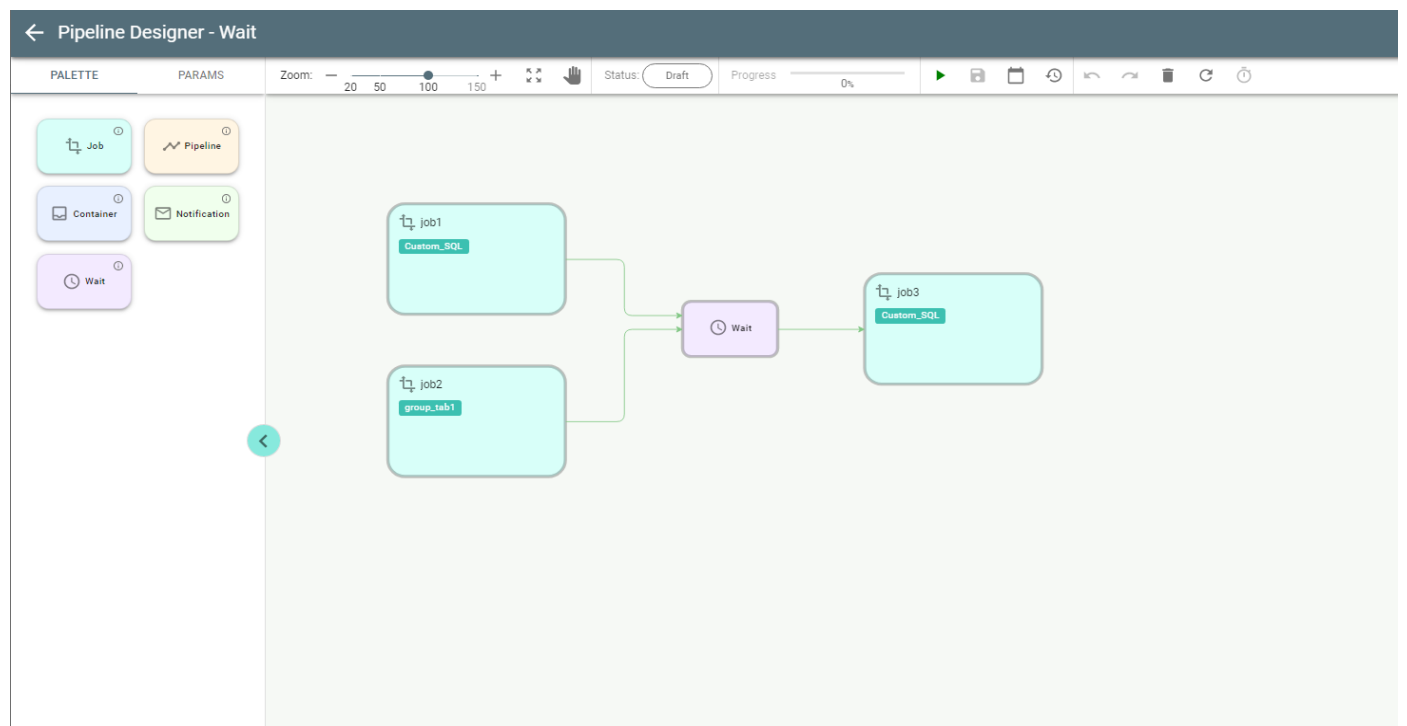
- ✓ *Registry*. Name of the registry for authentication.
- ✓ *Command*. The command to run once the Pod is created.

Important: the *Container* stage has Logs button  so you can see logs.

If the pipeline completed successfully, the logs display the message contained in the *Command* field in the configuration of the *Container* stage.

















The *Wait* stage is a dummy stage with no configuration, used for running multiple jobs in parallel as in this example:



### 5.3. Pipeline Designer Functions Overview



The following functions are available in *Pipeline Designer*:

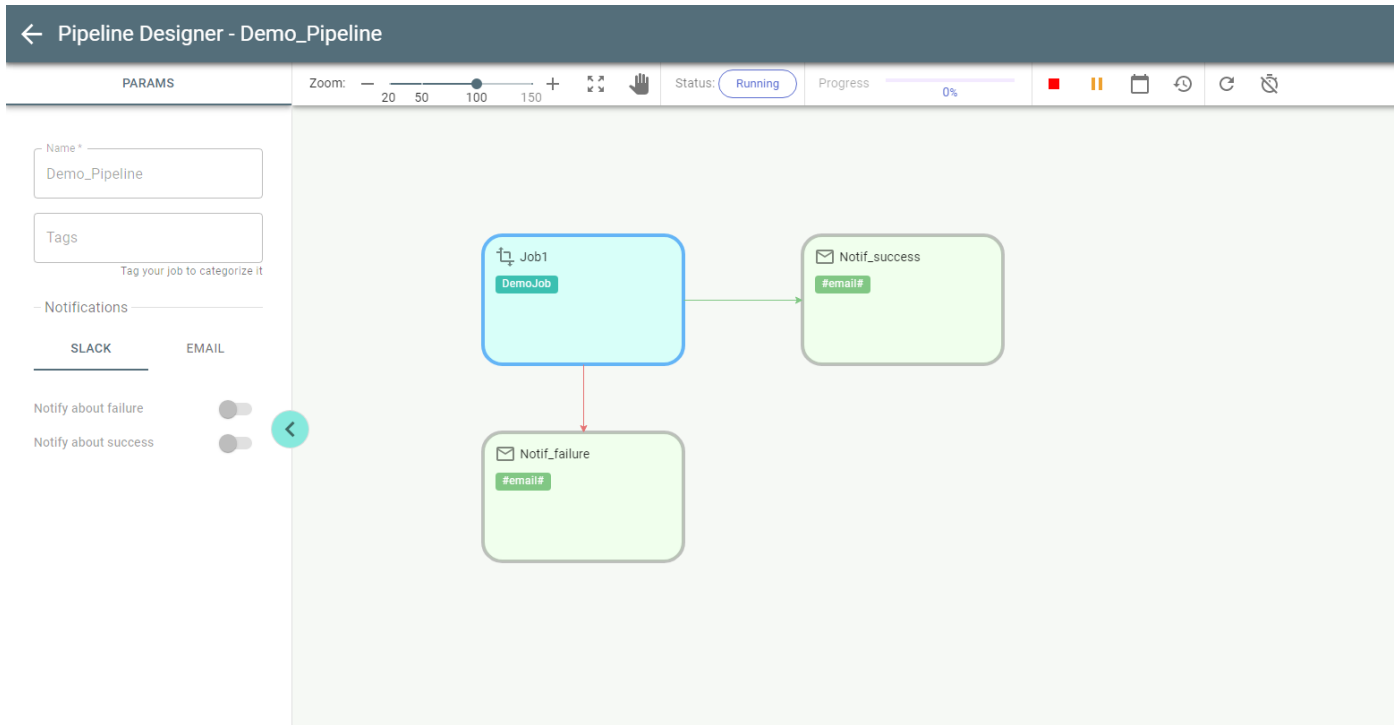
- Zoom functions: 
- Move elements: 
- Move elements/screen: 
- Show pipeline status: 
- Show pipeline progress: 
- Run pipeline  / Stop pipeline  (for running)
- Save pipeline 
- Create schedule for pipeline 
- View pipeline history 
- Undo / Redo operation on canvas 
- Remove element from canvas 
- Refresh 
- Auto refresh 

### 5.4. Pipeline Execution

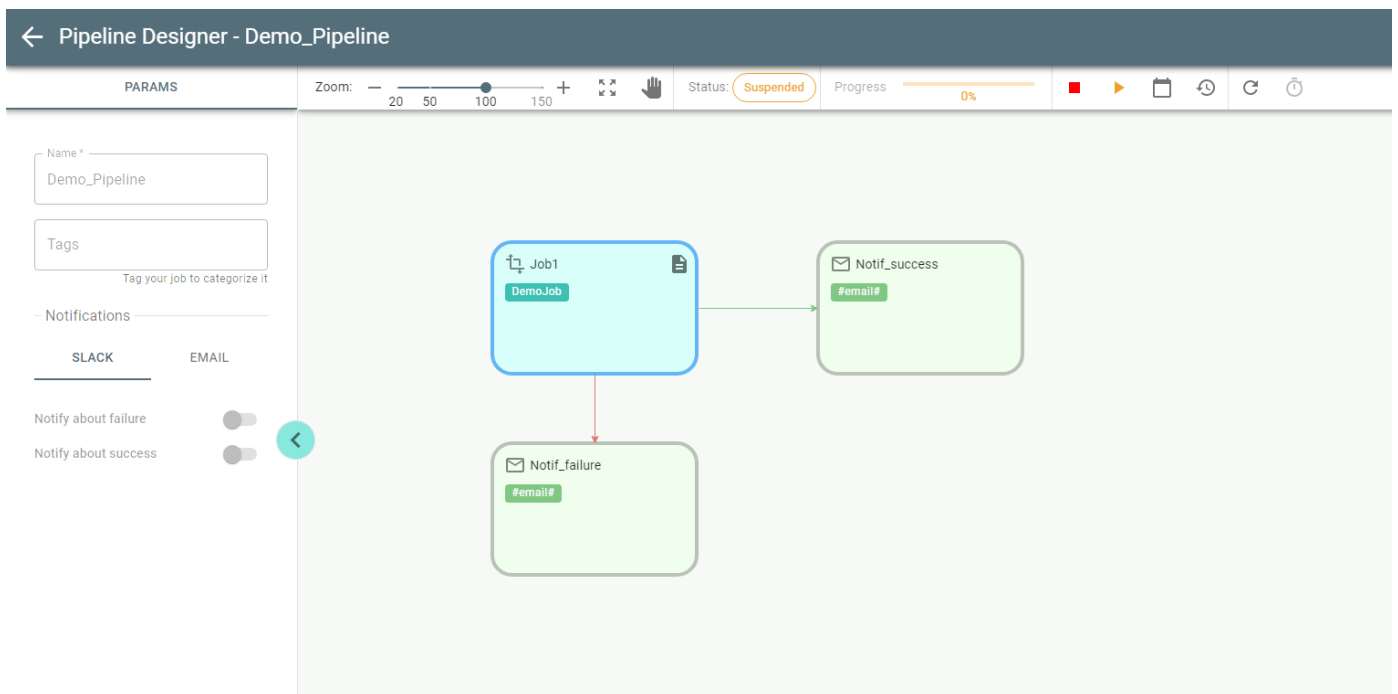
If you run a pipeline, as in the above example, its status changes from *Draft* to *Pending* and then to *Running*. Push Refresh to update the status.

The border of the stage currently running is painted in *Blue*:

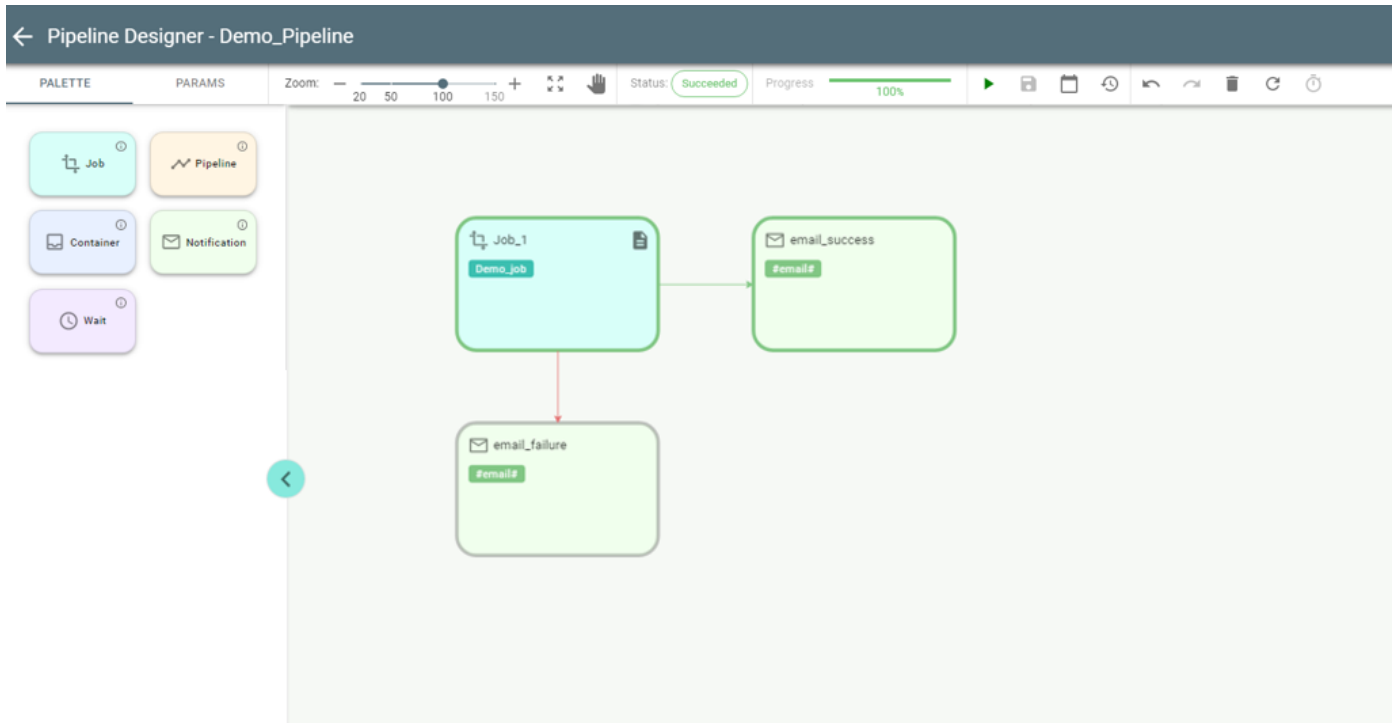
While running a pipeline can be stopped or suspended with Stop/Suspend buttons respectively:  



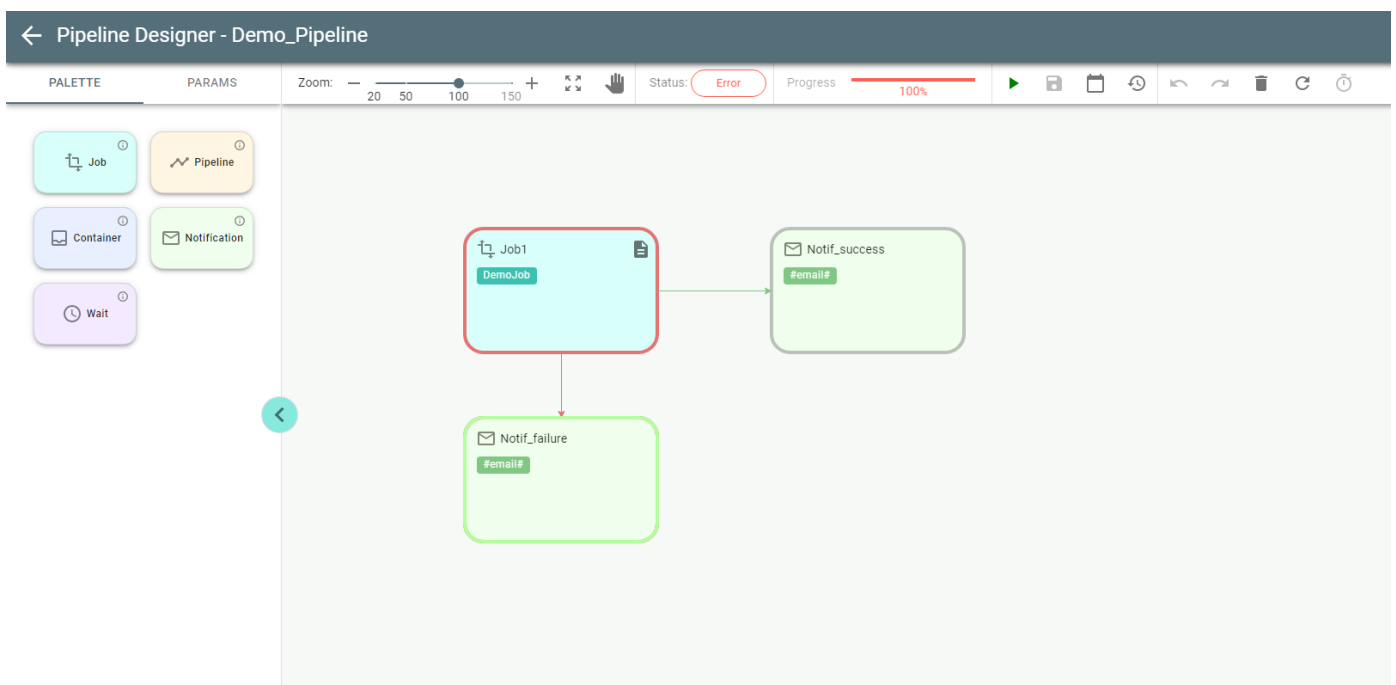
Once suspended it can be resumed with Resume button:



If a pipeline succeeds, all completed stages are painted in *Green*, indicating success.  
The stages configured for failure scenario (red arrow) remain *Gray* as a *Draft* as they have not been executed.




If a pipeline fails, then the **Red** border indicates the failed stage:



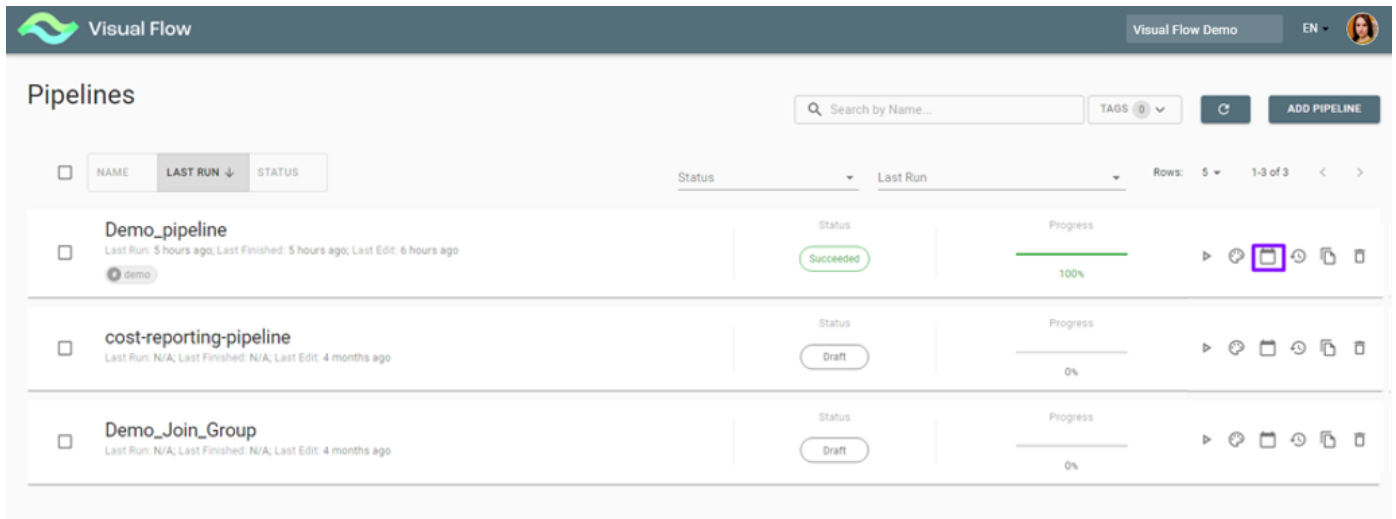
A failed pipeline can be re-run with **Error**  button from Pipelines overview page.

Important: *Job* stage has the *Logs* button  for analyzing logs of a certain job.

Use the *History* button  to view information about previous runs of the pipeline.

## 5.5. Scheduling a pipeline.

You can schedule a pipeline to run with the *Scheduling* button:



Visual Flow

Visual Flow Demo EN

### Pipelines

Search by Name... TAGS 0 ADD PIPELINE

NAME	LAST RUN	STATUS	
<input type="checkbox"/> Demo_pipeline Last Run: 5 hours ago; Last Finished: 5 hours ago; Last Edit: 6 hours ago <small>demo</small>		Succeeded	100% ▶ ⌚ 📅 ⌛ 📄 🗑️
<input type="checkbox"/> cost-reporting-pipeline Last Run: N/A; Last Finished: N/A; Last Edit: 4 months ago		Draft	0% ▶ ⌚ 📅 ⌛ 📄 🗑️
<input type="checkbox"/> Demo_Join_Group Last Run: N/A; Last Finished: N/A; Last Edit: 4 months ago		Draft	0% ▶ ⌚ 📅 ⌛ 📄 🗑️

It opens the *Scheduling* window:

### Scheduling

On/Off

MINUTE (0-59) HOUR (0-23) DAY OF MONTH (1-31) MONTH (1-12) DAY OF WEEK (0-6)

\* any value

,

- range of values

/ step values

SAVE CANCEL

Switch the toggle on and enter values of minute/day/day of month/month/day of the week through spaces according to the tips:



## Scheduling

☒ On/Off

"At 01:01"

Next at: 2022-09-23 21:01:00 UTC

1 1 \* \* \*

MINUTE	HOUR	DAY OF	MONTH	DAY OF
(0-59)	(0-23)	MONTH	(1-12)	WEEK
		(1-31)		(0-6)

*	any value
,	separator or value list
-	range of values
/	step values

SAVE

CANCEL