



StreamSets Starter's Guide

Architecture, minimum requirements,
and a full walkthrough to build your first pipeline



What You Will Learn

This document will introduce you to the StreamSets DataOps Platform architectural concepts and the steps you'll need to take to build and run your smart data pipelines.

Need technical help along the way?

Here are 3 ways you can contact a live person and ask for help:

- Email cloudsuccess@streamsets.com
- Use the chat icon directly in app
- [Book an expert session](#)

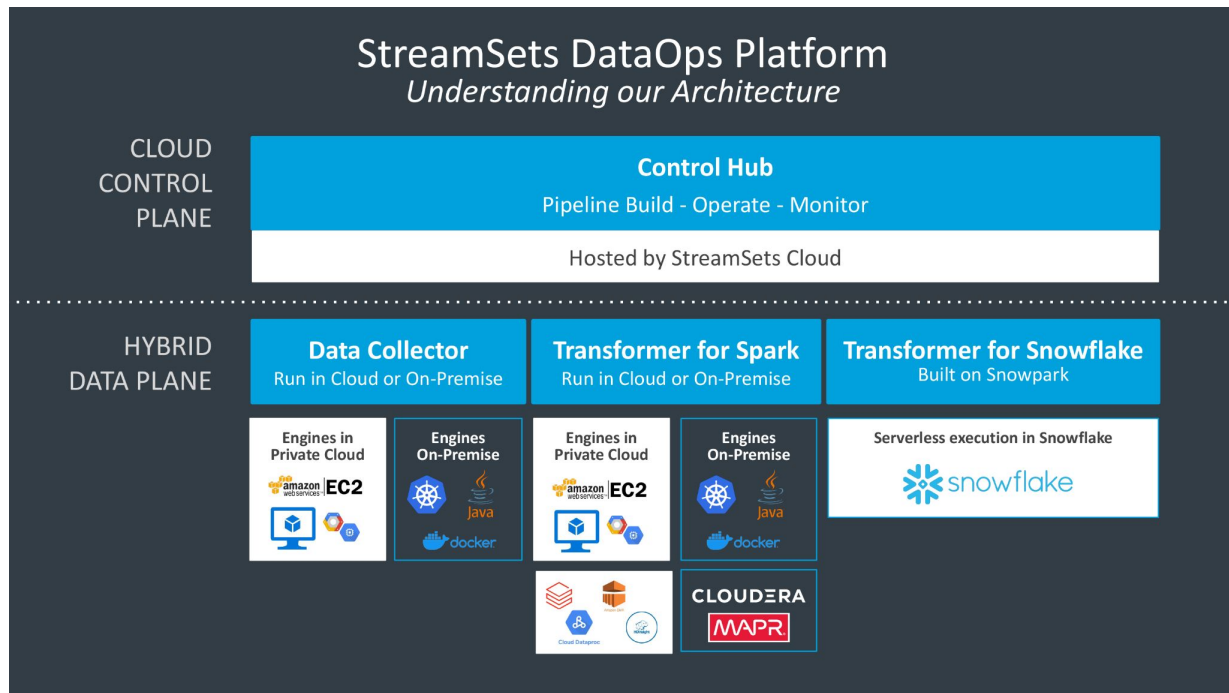
Introduction to the DataOps Platform Architecture & Core Components

The StreamSets DataOps Platform is made up of a Cloud Control Plane (Control Hub) and a Hybrid Data Plane. The Data Plane is where ingestion engines (Data Collector) and transformation engines (Transformer) reside. The Data Plane operates securely in your own environments (on-premise or in the cloud) while being fully monitored and managed by Control Hub hosted by StreamSets.

Note:

Seen in the diagram, StreamSets' hybrid data plane allows for flexible multi-cloud hybrid deployment. Move easily between on-premises and multiple cloud environments without rework.

StreamSets DataOps Platform *Understanding our Architecture*



Environments, Deployments & Engines

An Environment is the foundational component and defines where you will set up your Deployments and run your Engine. **You can create the following types of Environments:**

- Self managed
- Amazon Web Services (AWS)
- Microsoft Azure (Azure)
- Google Cloud Platform (GCP)

A Deployment is then created in an Environment. The Deployment defines the resources that will be available to run 1n identical Engines.

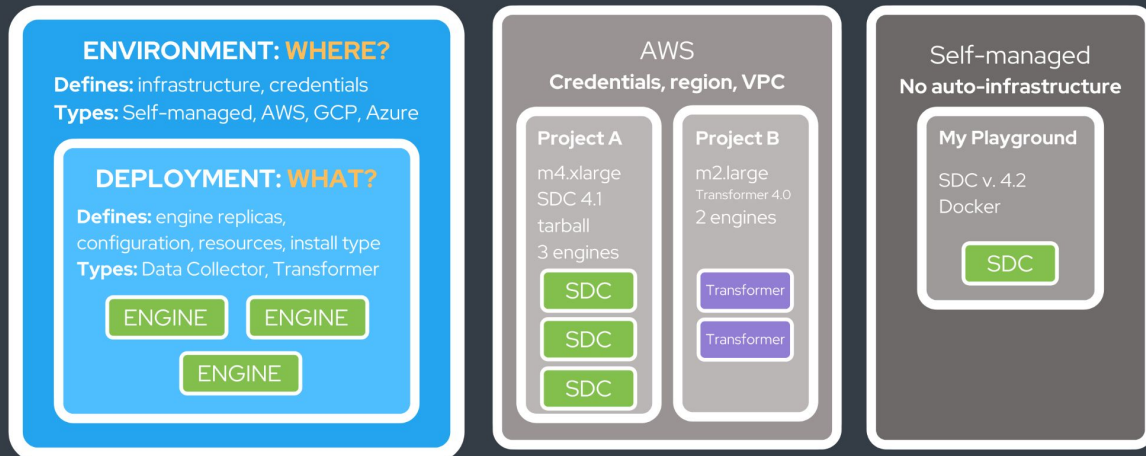
Finally, the type of Engine you want to run is selected and configuration parameters are set.

Engines can be 1 of 3 types.

1. Data Collector engine (records-based) to run ingestion pipelines
2. Transformer for Spark (set-based) to run processing pipelines on Spark
3. Transformer for Snowflake (set-based). Serverless architecture (**see next page**).

StreamSets DataOps Platform Data Collector & Transformer for Spark

3 Deployment Concepts



What you need to know:

Make sure you have your credentials and access rights for the environments you want to work in (AWS, Snowflake, Azure, GCP) or rights to download a self-managed SDC engine).

Transformer for Snowflake

Transformer for Snowflake uses the Snowpark client libraries to generate SnowSQL that runs natively in the Snowflake Data Cloud.

It is a serverless engine embedded within the platform so there is nothing to download, install or deploy. Just log in, navigate to Transformer for Snowflake, enter your Snowflake credentials and begin building & running data pipelines on Snowflake.

This addition to the Snowflake Data Cloud provides greatly extended functionality.

- Running complex data processing logic such as Slowly Changing Dimensions, JSON parsing, and standardization functions
- Running custom libraries in your data pipelines
- ML model training
- And more.

When using this engine, skip Part 1.

Instead [start with updating your Snowflake credentials](#) inside of StreamSets.

Design, Manage and Monitor - DataOps Platform

Transformer for Snowflake (Simple, Complex, Custom Transformations)

The screenshot displays a data pipeline in the Transformer for Snowflake interface. The pipeline starts with three source nodes: 'CUSTOMER', 'ORDERS', and 'LINEITEM'. 'CUSTOMER' flows into 'MKTSEGMENT IS FURNITURE', which then flows into 'Remove Fields'. 'ORDERS' flows into 'Expression Evaluator 1', which flows into 'Remove Fields'. 'LINEITEM' flows into 'Remove Fields'. The 'Remove Fields' nodes connect to 'JOIN_CUSTOM...' and 'JOIN_LINEITEMS'. 'JOIN_CUSTOM...' flows into 'SORT ORDERDATE AND LINENUMBER', which flows into 'Field Order 1', which finally flows into 'CUSTORD'. The interface includes a top bar with 'CustOrd', 'v5-DRAFT', 'Check In', 'Edit Mode', and 'All Changes Saved'. Below the pipeline, a dashed box contains the text 'Built on Snowpark (Native SnowSQL generation)'. At the bottom, the Snowflake logo and name are centered.

Built on Snowpark (Native SnowSQL generation)

Snowflake

About this Setup Guide

For the remainder of this guide, we will use a Self Managed Environment and Deployment and use a Data Collector engine as the example.

Before you proceed, make sure your machine meets the [minimum requirements](#) for a Data Collector engine and that you have installed [OpenJDK 8](#) or [Java 8 JDK](#) which is required to run the Data Collector engine.

Environments can also be configured using Docker or native in AWS, Azure or GCP. To learn more about using these deployment options, see our [documentation](#).

Note:

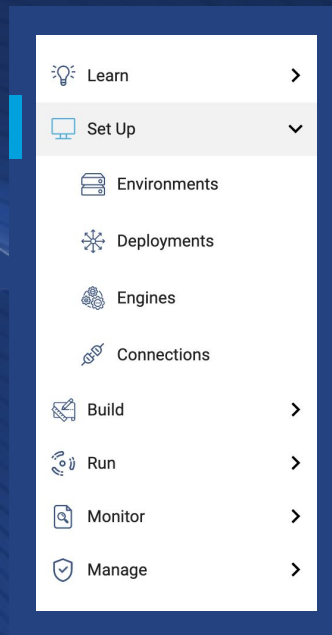
If you have a MacBook with an M1 chip, you may see an error when attempting to run the Docker install script. If you receive an error about the detected host platform not matching the requested image's platform, add: `--platform linux/amd64`.

The major steps we will walk you through include:

1. Configuring your Environment and Deployment
.....
2. Building a Pipeline
.....
3. Creating and running a job
.....
4. Monitoring your job

PART 1: Configure Your Environment and Deployment

You must create and activate environments before creating deployments, which allow you to manage all deployed engine instances with a single configuration change.



Creating Your Environment

For this tutorial, you are welcome to use the pre-built environment you see called Default Self-Managed Environment (SELF) that's already available. Or, you can create a new environment with the following three steps.

1. Under **Setup**, in the left navigation, click **Environments**.
2. Define your environment by giving it a **Name**. Keep **Self Managed** selected for **Type**, and add any Tags you'd like. Click **Save and Next** twice, leaving default values.
3. Then, click **Activate & Add Deployment** in the Review & Activate Section.

Having issues with your engine or laptop setup? Check out [7 Tips for Using StreamSets on Your Laptop](#).

New Environment

1 Define Environment
Define environment details. Once saved, you cannot change the environment type. [Learn more](#)

Environment Name: ⓘ

Environment Type: ⓘ

Environment Tags: ⓘ

[Show Advanced Options](#) ▾

2 Share Environment

3 Review & Activate

Define the Environment

Define the environment essentials - the environment name and type, and optional tags to identify similar environments.

1. Configure the following properties:

Define Environment Property	Description
Environment Name	Name of the environment. Use a brief name that informs your team of the environment use case.
Environment Type	Select one of the following types: Self-managed Amazon Web Services (AWS) Google Cloud Platform (GCP) Microsoft Azure Once saved, you cannot change the environment type.
Environment Tags	Optional tags that identify similar environments within Control Hub. Use environment tags to easily search and filter environments. Enter nested tags using the following format: <code><tag1>/<tag2>/<tag3></code>

Creating Your Deployment

1. Under **Deployment Name**, enter *Tutorial* as the name. Then, use the preset default values for the remaining properties. Click **Save & Next**.
2. In the **Configure Engine (2a)** section, Click **Save & Next**
 - a. **Note:** we are using the default of 3 stage libraries selected. No need to change that as additional libraries can be installed in future Deployments when setting up Engines.
3. Configure **Install Type**, leave as *Tarball* and Click **Save & Next**
4. **Share Deployment**, leave defaults and Click **Save & Next**
5. Review & Launch
 - a. Validate you have set up Java
 - b. Open a command prompt and [set your file descriptors limit](#) to at least 32768 (example for MacOS `ulimit -n 32768`)
 - c. Click **Start & Generate Install Script**.

New Deployment

1 Define Deployment

Define deployment details. Once saved, you cannot change the deployment type, the engine version, or the environment that the deployment belongs to. [Learn more](#)

Deployment Name: Tutorial

Deployment Type: Self-Managed

Environment: My First Environment (SELF)

Engine Type:


- Data Collector** - Runs data ingestion pipelines that perform record-based data transformations in streaming, CDC, or batch modes
- Transformer** - Runs data processing pipelines on Spark that perform set-based transformations such as joins, aggregates, and sorts on the entire data set

Engine Version: 4.4.0

Deployment Tags: dev x Enter New...

Cancel Save & Next Save & Exit

2 Configure Deployment

6. Click the **Copy to Clipboard icon** () to copy the generated command, and then click **Check Engine Status after Running the Script**. Control Hub displays an Engine Status window. Before you can view the engine status, you need to set up your machine and run the installation script command.

7. Paste the copied installation script command into your Terminal and run it. This will start your engine. Respond to the command prompts to enter download and installation directories for the Data Collector engine. When the engine installation is complete, Control Hub informs you that the engine is successfully running. **This can take a few minutes.**

You have successfully created an Environment and a Deployment with an available Data Collector Engine.

PART 2: Building Your Pipeline

When you build a pipeline, you are defining how data flows from your origin to your destination systems and how the data is processed along the way.

- 💡 Learn >
- 🖥️ Set Up >
- 🔧 Build ▾
 - ⚙️ Fragments
 - 🔗 Pipelines
 - 🏗️ Sample Pipelines
- 🔄 Run >
- 🔍 Monitor >
- 🛡️ Manage >

Building Your Pipeline

For part 2 of this tutorial, we will use a sample data set (New York taxi data) that we make available to users. This data set contains a CSV file which is read from an HTTP resource URL. You will add Processors to convert the data type of several fields, and then write the data to a JSON file on your local machine. The sample CSV file includes some invalid data, so you'll also see how StreamSets handles errors when you preview the pipeline.

Sample pipelines in StreamSets help you learn about pipeline design. View a sample pipeline to explore how the pipeline and stages are configured. Duplicate a sample pipeline to use it as the basis for building your own pipeline.

The screenshot displays the StreamSets DataOps Platform Control Hub interface. At the top, there's a navigation bar with the StreamSets logo, 'DataOps Platform', 'Control Hub', a search bar, and a 'Quick Start' button. A left sidebar contains navigation options: Learn, Set Up, Build (expanded to show Fragments, Pipelines, and Sample Pipelines), Run, Monitor, and Manage. The main area shows a sample pipeline titled 'Parse Web Logs To ...' with stages: 'Load Retail Web App Logs', 'Field Type Converter', 'Cart vs Page Views', 'Generate Avro Schema', and 'Cart Views (Avro)'. Below the pipeline, the configuration panel for 'Parse Web Lo...' is open, showing the 'General' tab with details like Name, Description, Labels, and Execution Mode.

Field	Value
Name	Parse Web Logs To JSON & Avro
Description	Tutorial - https://github.com/streamsets/pipeline-library/tree/master/datacol1
Labels	origin:Dev Raw Data Source destination:Local FS
Execution Mode	Standalone

Building Your Pipeline

1. In the Engine Status window, click **Create a pipeline**. Or, if you already closed the Engine Status window, click **Quick Start > Create a pipeline**.
2. Enter the following name: Tutorial.

New Pipeline

1 Define Pipeline

Define the pipeline name, the type of engine for the pipeline, and whether to start with a blank canvas or with a sample pipeline. [Learn more](#)

Name: ⓘ

Description: ⓘ

Engine Type:

- Data Collector** - Runs data ingestion pipelines that perform record-based data transformations in streaming, CDC, or batch modes
- Transformer** - Runs data processing pipelines on Spark that perform set-based transformations such as joins, aggregates, and sorts on the entire data set

Start with:

- Blank Pipeline**
- Sample Pipeline**

3. Use the defaults to create a blank Data Collector pipeline, and then click Next. In the **Configure Pipeline** section, the engine that you deployed is selected as the default authoring engine.

1 Define Pipeline

2 Configure Pipeline

If starting with a sample pipeline, select the sample to use. Select the authoring engine to use for pipeline design. The engine with the most recent reported time is selected by default. [Learn more](#)

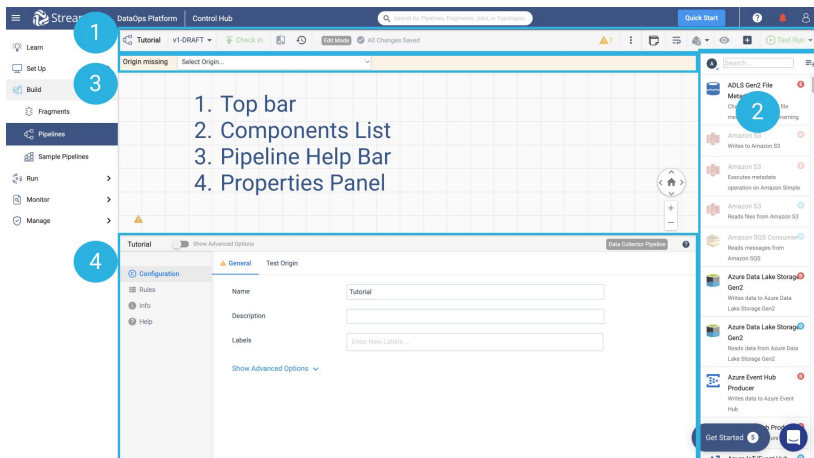
Authoring Engine: Annie's Deployment (Self-Managed) - f612f47c5072:18630 ⓘ

[Click here to select](#)

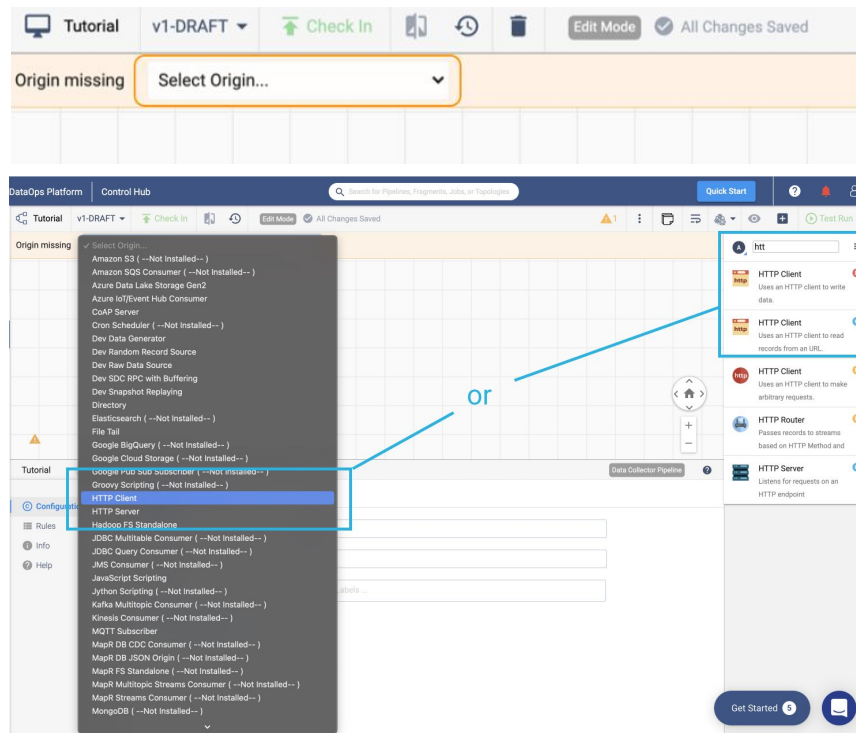
Building Your Pipeline

- 4. By clicking Save and Next, you'll have the ability to share your pipeline. But, let's click "Save and Open in Canvas" to get started quickly.

Here's the canvas:



- 5. From the Pipeline help bar, click **Select Origin** > **HTTP Client**:



The origin is added to the canvas.

Building Your Pipeline

- Configure the HTTP properties in the Properties Panel as shown in the image below:
 - Resource URL:**
https://docs.streamsets.com/datacollector/sample_data/tutorial/nyc_taxi_data.csv
 - Mode: Polling
 - Polling Interval: 600000
- Use the default values for the remaining properties.
The **HTTP** tab should be configured as seen in the figure at right.
- Click the **Data Format** tab.

The screenshot displays the configuration interface for 'HTTP Client 1' in the StreamSets Data Collector. The 'HTTP' tab is selected, and the following properties are visible:

- Resource URL:** `https://docs.streamsets.com/datacollector/sample_data/tutorial/nyc_taxi_data.csv`
- Mode:** Polling
- Polling Interval (ms):** 600000
- HTTP Method:** GET
- Authentication Type:** None
- Use OAuth 2:**

The 'Data Collector Pipeline' tab is also visible in the top right corner.

Building Your Pipeline

9. Configure the **data format** as follows:
 - a. Change **Data Format** from JSON to Delimited
 - b. Header Line: With Header Line
10. Use the default values for the remaining properties.

The screenshot shows the configuration interface for 'HTTP Client 1' in the StreamSets Data Collector Pipeline. The interface is divided into a top status bar, a central canvas, and a bottom configuration panel.

Top Status Bar: Displays 'HTTP Client 1 has open stream' and provides dropdown menus for 'Select New Processor to connect...' and 'Or Select New Destination to connect...'.

Central Canvas: A grid-based workspace containing a blue box labeled 'HTTP Client 1' with a small orange triangle at its bottom right corner.

Bottom Configuration Panel: Features a sidebar with 'Configuration' selected, and a main area with tabs for 'General', 'HTTP', 'Pagination', 'Credentials', 'OAuth 2', 'TLS', and 'Data Format'. The 'Data Format' tab is active, showing the following settings:

- Data Format:** Delimited
- Header Line:** With Header Line
- Delimiter Format Type:** Default CSV (ignores empty lines)
- Lines to Skip:** 0

A 'Show Advanced Options' toggle is visible, currently turned off. A 'Data Collector Pipeline' label with a help icon is also present in the top right of the configuration panel.

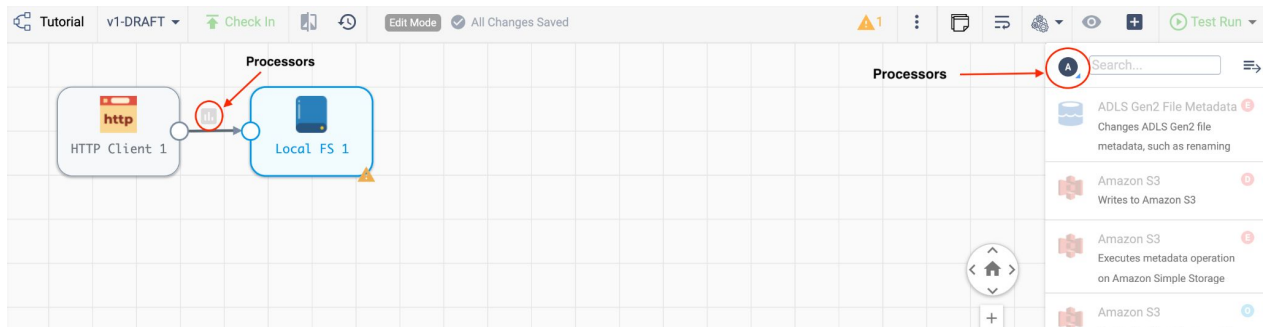
Building Your Pipeline

- From the **Pipeline Creation help bar**, click **Select New Processor to connect > Field Type Converter**

Note:

If you don't see your Pipeline Creation help bar, processors can also be found by clicking the little grey square between the elements in your pipeline. Alternatively, you can click on the box with 3 lines and a plus sign in the upper right hand corner of your canvas and then click on the "A" in a circle- it will open a drop down menu where you will see "Processors." You can also select processors from here.

- Click the **Conversions tab**. *Shown on next page.*



Building Your Pipeline

13. Convert fields with datetime data to Datetime as follows:

a. Conversion Method: By Field Name (default)

b. Fields to Convert

c. `/droppoff_datettime`

d. `/pickup_datettime`

*Note: To reference a field, you enter the path of the field. For simple records of data such as the sample CSV file, you reference a field as follows: `</field name>`.

e. Convert to Type: Datetime


f. Date Format: yyyy-MM-dd HH:mm:ss

The screenshot shows the configuration page for a 'Field Type Converter' in a 'Data Collector Pipeline'. The 'Conversions' tab is active. The configuration includes:

- Conversion Method:** A dropdown menu set to 'By Field Name'.
- Fields to Convert:** A list containing two entries: `/droppoff_datettime` and `/pickup_datettime`.
- Convert to Type:** A dropdown menu set to 'DATETIME'.
- Source Field is Empty:** A dropdown menu set to 'Send to error'.
- Date Format:** A dropdown menu set to 'yyyy-MM-dd HH:mm:ss'.

At the bottom of the configuration area, there are buttons for '+ ADD ANOTHER' and 'BULK EDIT MODE'. On the right side, there is a 'Get Started' button with a '5' notification badge and a chat icon.

Building Your Pipeline

- In the toolbar above the pipeline canvas, click the **Preview** icon: . When you preview the pipeline, you can view several records of source data.

Preview Configuration

Preview Source:

Preview Batch Size:

Preview Timeout (in milliseconds):

Run Preview Through Stage:

Time zone:

Write to Destinations and Executors:

Execute Pipeline Lifecycle Events:

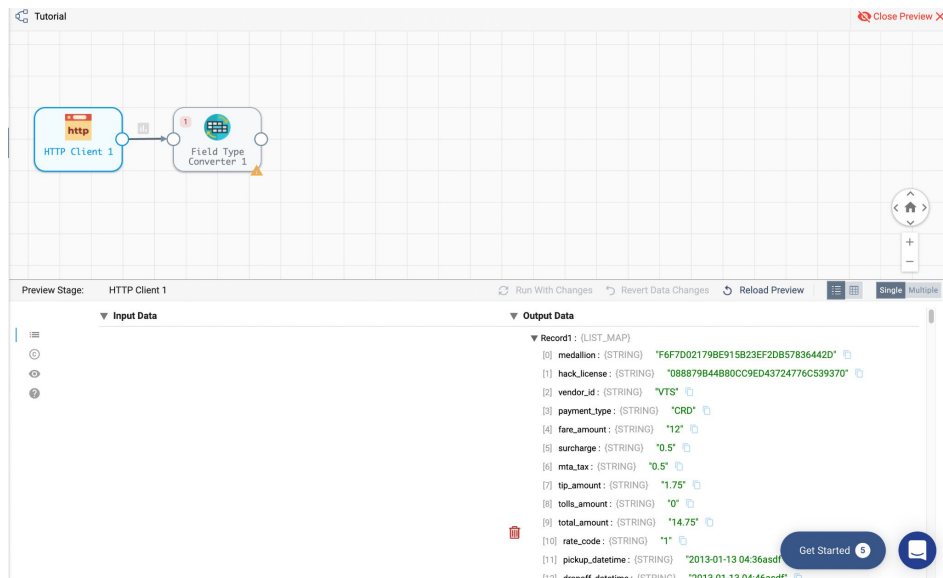
Show Record/Field Header:

Show Field Type:

Save Preview Record Schema:

Remember the Configuration:

- In the **Preview Configuration** dialog box, use the default values and then click Run Preview. The HTTP Client origin is selected in the pipeline canvas, and preview displays several records of output data read by the origin. Since this is the origin of the pipeline, no input data displays. Notice how the Field Type Converter processor displays a red square with a counter of 1, indicating that the stage has encountered an error.



The screenshot shows a pipeline canvas with two stages: "HTTP Client 1" and "Field Type Converter 1". The "Field Type Converter 1" stage has a red square icon with the number "1", indicating an error. Below the canvas, the "Preview Stage: HTTP Client 1" dialog is open, showing "Input Data" (empty) and "Output Data" (a list of records). The output data is a list of records, each with a key-value pair. The records are:

- Record1: (LIST_MAP)
- [0] medallion: (STRING) "F6F7D021798E915B23EF20B57836442D"
- [1] hack_license: (STRING) "088879B44B80CC9ED43724776C539370"
- [2] vendor_id: (STRING) "VTS"
- [3] payment_type: (STRING) "CRD"
- [4] fare_amount: (STRING) "12"
- [5] surcharge: (STRING) "0.5"
- [6] mta_tax: (STRING) "0.5"
- [7] tip_amount: (STRING) "1.75"
- [8] tolls_amount: (STRING) "0"
- [9] total_amount: (STRING) "14.75"
- [10] rate_code: (STRING) "1"
- [11] pickup_datetime: (STRING) "2013-01-13 04:36:58"
- [12] dropoff_datetime: (STRING) "2013-01-13 04:46:58"

Building Your Pipeline

16. Select the **Field Type Converter** processor in the canvas.
Preview highlights the first record in red and displays an error message indicating that the first record has an unparseable date. The date data includes invalid characters at the end, as follows:

By default, the stage passes error records to the pipeline for error handling, and then the pipeline discards the error records. Since this is sample data, you can leave the default error record handling. When you run the pipeline, this first record will not be passed to the next stage for processing. [Learn more about Error Handling.](#)

The screenshot shows the StreamSets Data Collector interface. At the top, there's a 'Tutorial' header and a 'Close Preview' button. The main canvas displays a pipeline with two processors: 'HTTP Client 1' and 'Field Type Converter 1'. Below the canvas, the 'Preview Stage' is set to 'Field Type Converter 1'. The interface includes buttons for 'Run With Changes', 'Revert Data Changes', and 'Reload Preview'. On the right, there are navigation controls for 'Single' and 'Multiple' views.

The 'Input Data' section shows a list of records. Record 12 is highlighted in red and contains an error message: 'Record1-Error Record1 CONVERTER_00 - Failed to convert field '/dropoff_datetime' of type 'STRING' with value '2013-01-13 04:46asdf' to type 'DATETIME': (View Stack Trace...)'.

The 'Output Data' section shows the same list of records, but Record 12 is missing, indicating it was filtered out due to the error.

```
Input Data
Record1: (LIST_MAP)
[0] medallion: (STRING) "F6F7D02179BE915B23EF2DB57836442D"
[1] hack_license: (STRING) "088879B44B80CC9ED43724776C539370"
[2] vendor_id: (STRING) "VTS"
[3] payment_type: (STRING) "CRD"
[4] fare_amount: (STRING) "12"
[5] surcharge: (STRING) "0.5"
[6] mta_tax: (STRING) "0.5"
[7] tip_amount: (STRING) "1.75"
[8] tolls_amount: (STRING) "0"
[9] total_amount: (STRING) "14.75"
[10] rate_code: (STRING) "1"
[11] pickup_datetime: (STRING) "2013-01-13 04:36asdf"
[12] dropoff_datetime: (STRING) "2013-01-13 04:46asdf"
[13] passenger_count: (STRING) "5"
[14] trip_time_in_secs: (STRING) "600"

Output Data
Record1-Error Record1 CONVERTER_00 - Failed to convert field '/dropoff_datetime' of type 'STRING' with value '2013-01-13 04:46asdf' to type 'DATETIME': (View Stack Trace...)
(LIST_MAP)
[0] medallion: (STRING) "F6F7D02179BE915B23EF2DB57836442D"
[1] hack_license: (STRING) "088879B44B80CC9ED43724776C539370"
[2] vendor_id: (STRING) "VTS"
[3] payment_type: (STRING) "CRD"
[4] fare_amount: (STRING) "12"
[5] surcharge: (STRING) "0.5"
[6] mta_tax: (STRING) "0.5"
[7] tip_amount: (STRING) "1.75"
[8] tolls_amount: (STRING) "0"
[9] total_amount: (STRING) "14.75"
[10] rate_code: (STRING) "1"
[11] pickup_datetime: (STRING) "2013-01-13 04:36asdf"
[12] dropoff_datetime: (STRING) "2013-01-13 04:46asdf"
```


Building Your Pipeline

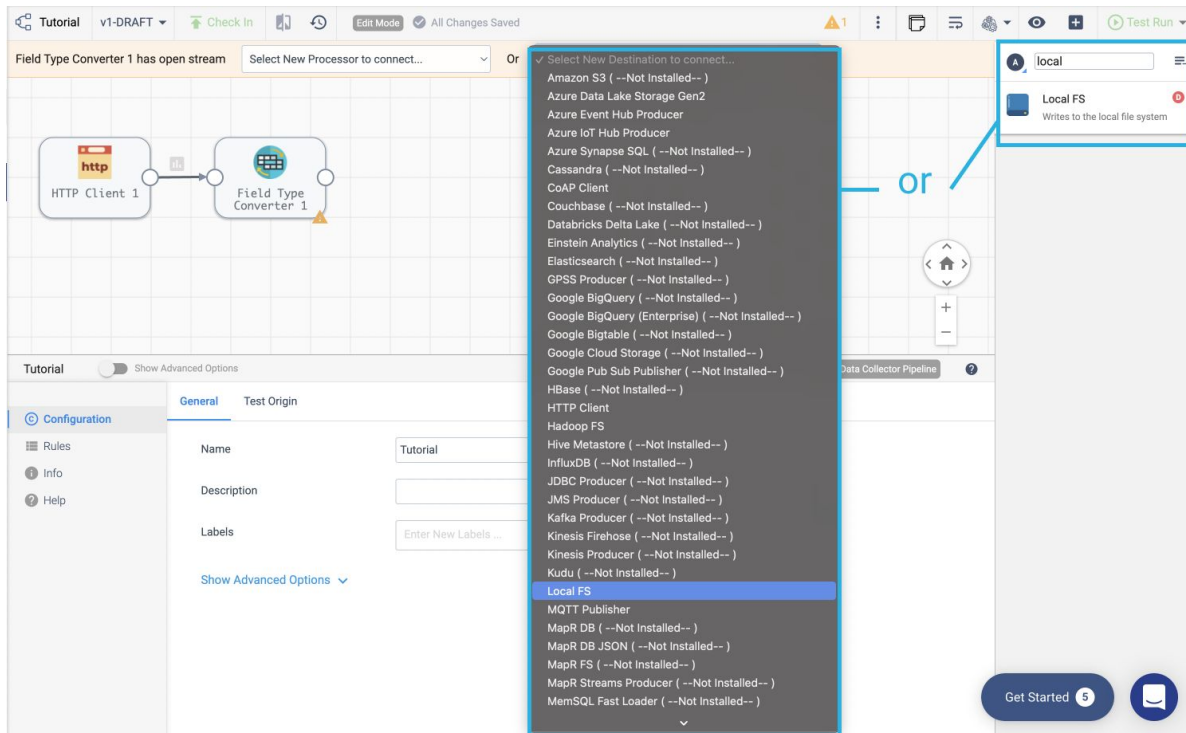
- With the processor still selected in the canvas, scroll down in the preview panel to display the input and output data of the second record. You can see that the date fields in the second record were successfully converted to the Datetime data type:

Preview Stage: Field Type Converter 1 Run With Changes Revert Data Changes Reload Preview Single Multiple

Record2 : (LIST_MAP)	Record2-Output Record1 : (LIST_MAP)
[0] medallion : (STRING) "BE386D8524FCD16B3727DCF0A32D9B25"	[0] medallion : (STRING) "BE386D8524FCD16B3727DCF0A32D9B25"
[1] hack_license : (STRING) "4EB96EC9F3A42794DEE233EC8A2616CE"	[1] hack_license : (STRING) "4EB96EC9F3A42794DEE233EC8A2616CE"
[2] vendor_id : (STRING) "VTS"	[2] vendor_id : (STRING) "VTS"
[3] payment_type : (STRING) "CRD"	[3] payment_type : (STRING) "CRD"
[4] fare_amount : (STRING) "12"	[4] fare_amount : (STRING) "12"
[5] surcharge : (STRING) "0.5"	[5] surcharge : (STRING) "0.5"
[6] mta_tax : (STRING) "0.5"	[6] mta_tax : (STRING) "0.5"
[7] tip_amount : (STRING) "3.12"	[7] tip_amount : (STRING) "3.12"
[8] tolls_amount : (STRING) "0"	[8] tolls_amount : (STRING) "0"
[9] total_amount : (STRING) "16.12"	[9] total_amount : (STRING) "16.12"
[10] rate_code : (STRING) "1"	[10] rate_code : (STRING) "1"
[11] pickup_datetime : (STRING) "2013-01-13 04:37:00"	[11] pickup_datetime : (DATETIME) Jan 13, 2013 4:37:00 AM [Browser Timezone]
[12] dropoff_datetime : (STRING) "2013-01-13 04:48:00"	[12] dropoff_datetime : (DATETIME) Jan 13, 2013 4:48:00 AM [Browser Timezone]
[13] passenger_count : (STRING) "2"	
[14] trip_time_in_secs : (STRING) "660"	

Building Your Pipeline

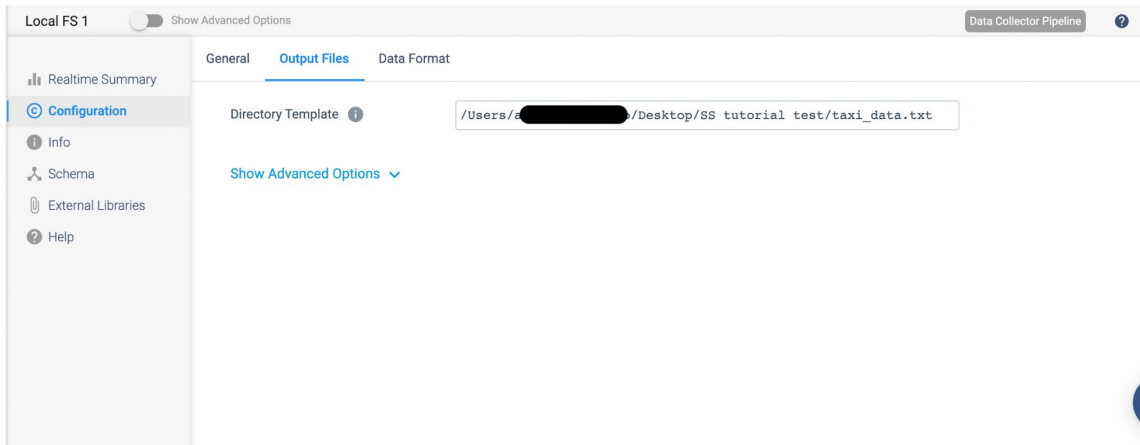
18. Click  **Close Preview** to close the preview.
19. From the pipeline creation help bar, click **Select New Destination to connect > Local FS**. The Local FS destination writes to files in a local file system.



Building Your Pipeline

- Click the **Output Files** tab, and configure the following property.
Use the defaults for the advanced options:

Output Files Property	Description
Directory Template	<p>By default, the directory template includes datetime variables to create a directory structure for output files. This is intended for writing large volumes of data.</p> <p>Since you are only processing the sample file, you don't need the datetime variables. Go ahead and delete the default and enter a local directory where you want the files to be written. Be sure to include what you would like to name the file at the end.</p> <p>For example: <code></basedirectory>/tutorial/destination/taxi_data.txt</code></p>



The screenshot shows the configuration interface for a pipeline named 'Data Collector Pipeline'. The 'Output Files' tab is active, and the 'Directory Template' property is configured with the path: `/Users/.../Desktop/SS tutorial test/taxi_data.txt`. A 'Show Advanced Options' dropdown menu is visible below the field.

Note:

Ensure the end of the file path is what you would like to name the file that is being created. In this case it is `'Taxi_data.txt'`

- Click the **Data Format** tab, and select **JSON** to write the data using the JSON format. Use the defaults for the remaining properties.

PART 3:


Run Your Job

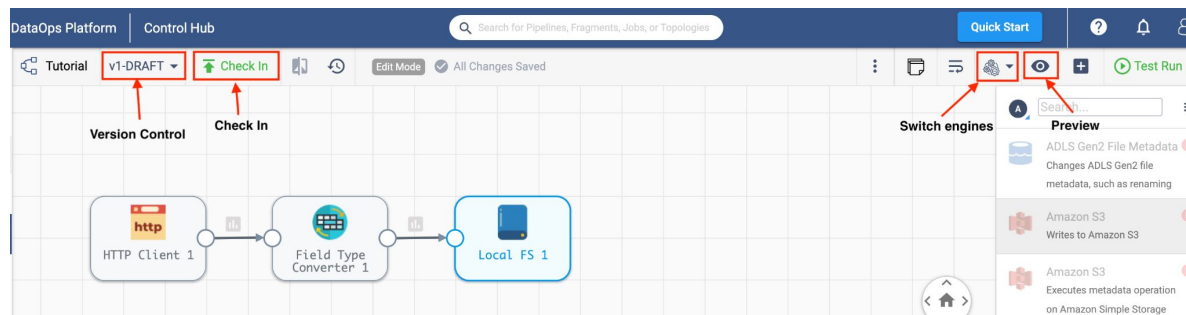
Jobs are the execution of the dataflow. Jobs enable you to manage and orchestrate large scale dataflows that run across multiple engines.

- 💡 Learn >
- 🖥️ Set Up >
- 🔧 Build >
- 🔄 Run ▾
- 📅 Job Templates
- 📁 Job Instances
- 📅 Scheduled Tasks
- 🔍 Monitor >
- 🛡️ Manage >

Run Your Job

Since this pipeline processes one file, there's no need to enable the job to start on multiple engines or to increase the number of pipeline instances that run for the job. As a result, you can simply use the default values when creating the job. As you continue to use StreamSets, you can explore how to run pipelines at scale.

1. With the pipeline open in the canvas, click the **Check In** icon: .
2. Enter a commit message. For now, you can simply use the default: **New Pipeline**.
3. As a best practice, state what changed in this pipeline version so that you can track the commit history of the pipeline.
4. Click **Publish** and then **Create and Start New Job**. The New Job window appears.



5. Use the defaults in the **Define Job** section, and click **Next**.
6. In the **Select Pipeline** section, click **Next**.
7. In the **Configure Job** section, select **Tutorial (Self-Managed)** for the **Deployment** property. Notice how the Engine Labels property is automatically populated with the default **Tutorial** label assigned to the deployment.
8. Use the defaults for the remaining properties and click **Save & Next**.
9. Click **Start & Monitor Job**. The job displays in the canvas, and Control Hub indicates that the job is active.

PART 4: Monitor Your Job

Next, you'll monitor the progress of the job. When you start a job, Control Hub sends the pipeline to the Data Collector engine. The engine runs the pipeline, sending status updates and metrics back to Control Hub.

- 💡 Learn >
- 🖥️ Set Up >
- 🏗️ Build >
- 🏃 Run >
- 🔍 Monitor ▾
- 📊 Operations Dashboard
- 📊 Topologies Dashboard
- 📡 Subscriptions
- 🔗 Topologies
- 📄 Reports
- 🔔 Alerts
- 🛡️ Manage >

Monitor Your Job

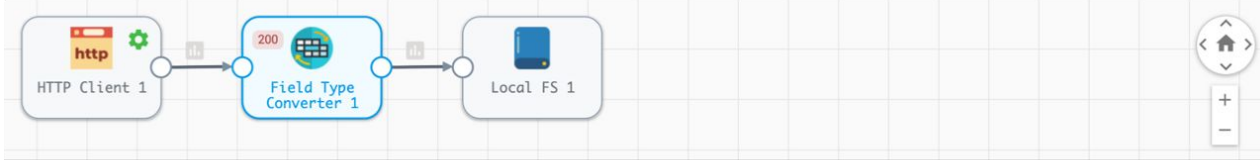
1. As the job runs, click the **Realtime Summary** tab in the monitor panel to view the real-time statistics for the job.

Notice how the Record Count chart displays more input records than output records. That's because the pipeline is configured to discard the error records encountered by the Field Type Converter processor.



Monitor Your Job

2. Select the Field Type Converter processor in the canvas, and then click the **Errors** tab. The tab displays an error message for each error record, as follows:




The screenshot shows a pipeline in the StreamSets interface. The pipeline consists of three processors: 'HTTP Client 1', 'Field Type Converter 1', and 'Local FS 1'. The 'Field Type Converter 1' processor is selected, and the 'Errors' tab is active. The error records are as follows:


Error Records	Timestamp	Error Message
▶ Record 1: (View Stack Trace...)	May 4, 2021, 8:46:14 PM	CONVERTER_00 - Failed to convert field '/dropoff_datetime' of type 'STRING' with value '2013-01-13 10:36asdf' to type 'DATETIME'
▶ Record 2: (View Stack Trace...)	May 4, 2021, 8:46:14 PM	CONVERTER_00 - Failed to convert field '/dropoff_datetime' of type 'STRING' with value '2013-01-13 06:57asdf' to type 'DATETIME'
▶ Record 3: (View Stack Trace...)	May 4, 2021, 8:46:14 PM	CONVERTER_00 - Failed to convert field '/dropoff_datetime' of type 'STRING' with value '2013-01-13 10:35asdf' to type 'DATETIME'

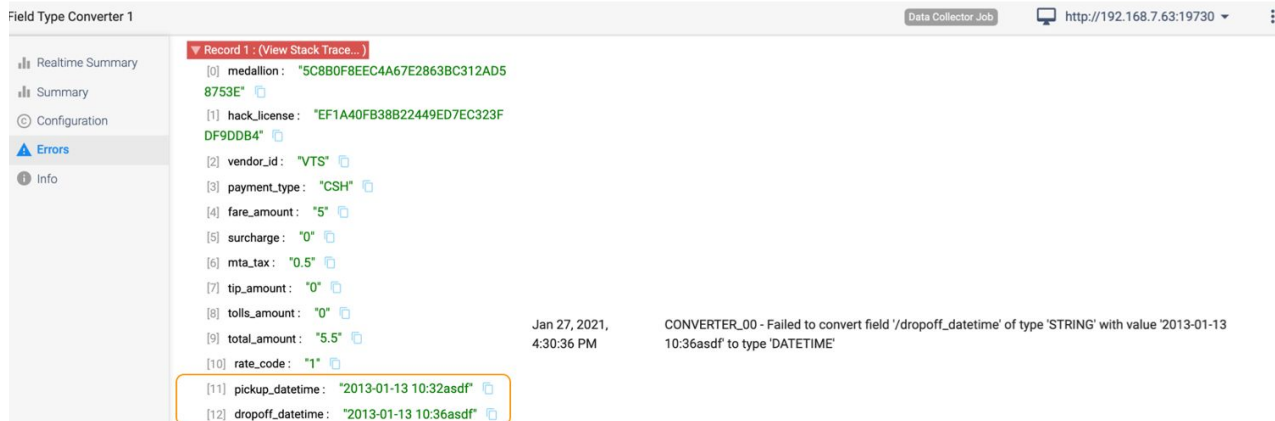
Monitor Your Job

- Expand one of the error records, and you'll see the same invalid data causing the error that you saw during the preview of the pipeline, as follows:

Notice how the HTTP Client origin has displayed the running icon () for the last several minutes although the input record count has not increased. That's because you configured the origin to poll the specified URL every 10 minutes. So the origin waits for that interval, then reads the file again.

When you've finished monitoring the data, stop the job so that the pipeline doesn't run indefinitely.

- Click the **Stop Job** icon:  and the **OK** button in the **Confirmation Dialog**.
- After the job successfully stops, click Close.



Field Type Converter 1 Data Collector Job http://192.168.7.63:19730

Realtime Summary
Summary
Configuration
Errors
Info

Record 1: (View Stack Trace...)

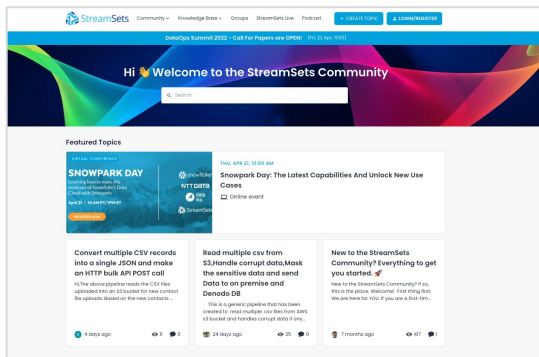
```
[0] medallion: "5C8B0F8EECA67E2863BC312AD58753E"
[1] hack_license: "EF1A40FB38B22449ED7EC323FDF9DDB4"
[2] vendor_id: "VTS"
[3] payment_type: "CSH"
[4] fare_amount: "5"
[5] surcharge: "0"
[6] mta_tax: "0.5"
[7] tip_amount: "0"
[8] tolls_amount: "0"
[9] total_amount: "5.5"
[10] rate_code: "1"
[11] pickup_datetime: "2013-01-13 10:32asdf"
[12] dropoff_datetime: "2013-01-13 10:36asdf"
```

Jan 27, 2021, 4:30:36 PM CONVERTER_00 - Failed to convert field '/dropoff_datetime' of type 'STRING' with value '2013-01-13 10:36asdf' to type 'DATETIME'

- Locate the local directory configured for the destination, `<base directory>/tutorial/destination`, and open the file to verify that the data was written in JSON format.

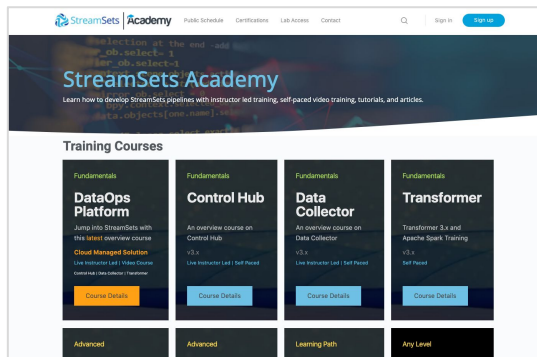
That's it! You've finished building, running, and monitoring your first pipeline.

Additional Resources to Get Started Fast



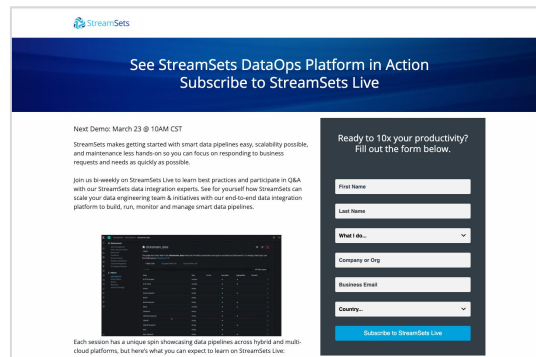
Community

Connect with data engineers to learn, share best practices, and expand your skills.



Academy

Get StreamSets certified to expand your skills and accelerate your success.



StreamSets Live

Join us for a full StreamSets walkthrough and participate in Q&A with StreamSets experts.



About StreamSets

At StreamSets, our mission is to make data engineering teams wildly successful. Only StreamSets offers a platform dedicated to building the smart data pipelines needed to power DataOps across hybrid and multi-cloud architectures. That's why the largest companies in the world trust StreamSets to power millions of data pipelines for modern business intelligence, data science, and AI/ML. With StreamSets, data engineers spend less time fixing and more time doing.

To learn more, visit www.streamsets.com