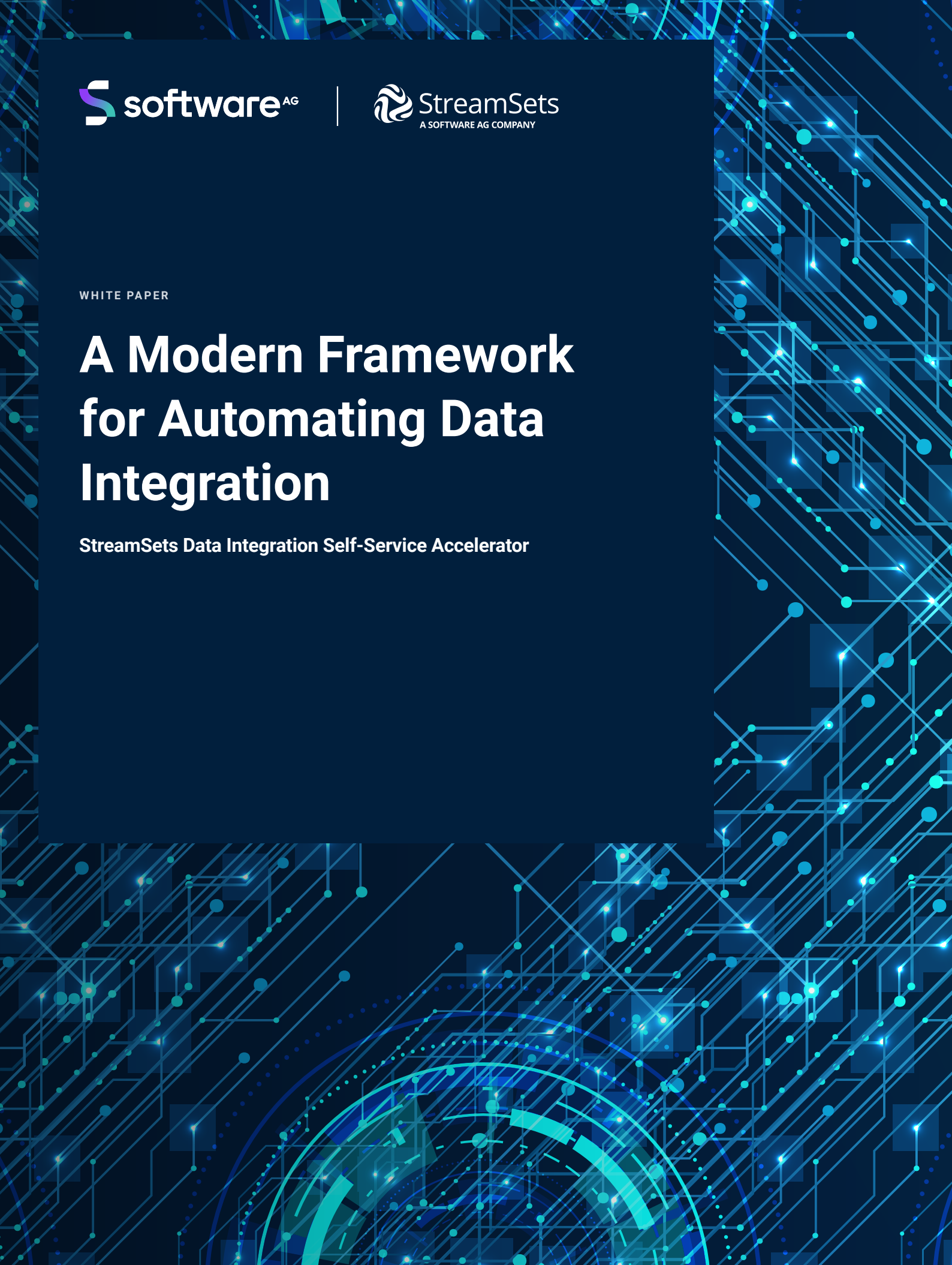


WHITE PAPER

A Modern Framework for Automating Data Integration

StreamSets Data Integration Self-Service Accelerator

1 Introduction

Every organization gathers data, presenting an opportunity for meaningful actions: informed decision-making, problem-solving, and fostering business growth. Achieving these objectives necessitates the presence of reliable and high-performing data practices that enjoy trust throughout the organization.

The challenge of timely access to the right data for business analytics can be a major hurdle for companies modelled on a centralized IT distribution approach. Even the smallest of delays can have a significant impact on competitive advantage. One way around this is to decentralise the data ingestion process by using standardised patterns and automation. Imagine a scenario where the business can self-serve the procurement of their data to develop their insights within their timeframes and capabilities. This white paper describes a framework for delivering such a capability.

Customers often present highly intricate requirements, and while the available technology can meet their needs, the complexity of the required tools demands a significant level of expertise for successful implementation. StreamSets software stands out by offering a distinctive combination of graphical and programmatic approaches across the entire platform lifecycle, encompassing deployment, configuration, design, and pipeline release. This unique feature enables us to establish best practice implementations, referred to as "patterns," for common scenarios. These patterns provide companies with comprehensive guidance, including architecture, design, and code samples, thereby reducing risk and accelerating delivery. Crafted by StreamSets engineers, these patterns consider recurring scenarios encountered during customer interactions, leverage deep product knowledge, and encompass all core elements necessary for a correct initial implementation. Furthermore, they are designed to be extensible, allowing customers to tailor them to their specific needs and architectures.

1.1 StreamSets

StreamSets, a Software AG company, eliminates data integration friction in complex hybrid and multi-cloud environments to keep pace with “need-it-now” business data demands. Our platform lets data teams unlock data, in a safe and governed way, to enable a data-driven enterprise. Resilient and repeatable pipelines deliver analytics-ready data that improve real-time decision-making and reduce the costs and risks associated with data flow across an organisation. That’s why the largest companies in the world trust StreamSets to power millions of data pipelines for modern analytics, data science, smart applications, and hybrid integration.

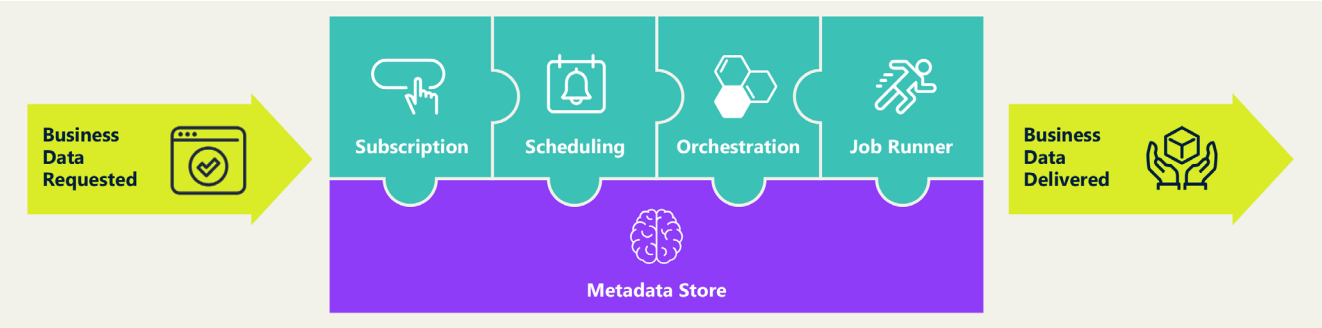
1.2 Intended Use

This document is intended to be used by a central IT function (aka data platform provider) with responsibility for maintaining **ingestion patterns** (i.e., data ingestion types) in conjunction with business unit functions (aka data platform tenant subscribers) responsible for maintaining **job templates** (i.e., parameterised data ingestion jobs customized to meet business needs).

It is common for organisations to have relatively simple data movement tasks, which are used for ingestion of data from source systems into an Analytical platform, be it a data lake or data warehouse (or combination). Other, similar tasks could be the movement of data between source systems and MDM repositories, or publication of enriched data from data warehouses to applications. All these data movement tasks are simple enough to implement as one-offs, when the organisation is equipped with a data ingestion platform providing drag and drop interfaces, but organisations typically have hundreds of these feeds to implement, manage and execute. And the drag and drop user interfaces themselves, while easy to use by data engineers, are not really designed for self-service consumption by end-users; this is where StreamSets' automation comes into play, delivering the self-service experience for the Enterprise needs.

2 Framework Overview

In the creation of a sophisticated automated data ingestion capability, our pattern development encompasses key functionalities tailored to streamline the process. This includes a **subscription function** to capture and manage requirements effectively, a **scheduling function** for seamless process coordination, an **orchestration function** to efficiently build and manage the environment, a **job runner** function for precise delivery execution, and a **metadata store** that enhances logic intelligence and provides comprehensive process instrumentation. These elements work in tandem to ensure a robust and automated data ingestion solution, aligning with the strategic needs of modern businesses.



A modern approach to automating Data Integration

2.1 Subscription function

The subscription function is the front end of the pattern process where the business requirements for data provision are captured and validated. This can be accomplished through a straightforward Excel or spreadsheet method, enhanced as a more feature-rich website capability, or incorporated into an existing version control system.

It is the responsibility of the Data Subscription function to ensure that the metadata store is populated with the correct values for downstream scheduling (pattern selection) and downstream orchestration (parametrized job template creation).

2.2 Scheduling function

The scheduling function identifies new data ingestion requests, triggers the automation process, including ingestion pattern selection for downstream orchestration processing and if required, establishes a future runtime job schedule. All the details required to support the scheduling function are stored in the metadata store.

For a basic level of functionality, the StreamSets scheduler can be used to run a regular polling job to scan the metadata store for new requests. However, it is more likely that organizations will want to integrate the automation process into their scheduling capability to accommodate more advanced levels of functionality such as data governance compliance, processing order, exception handling, notifications, and data quality checks.

2.3 Orchestration function

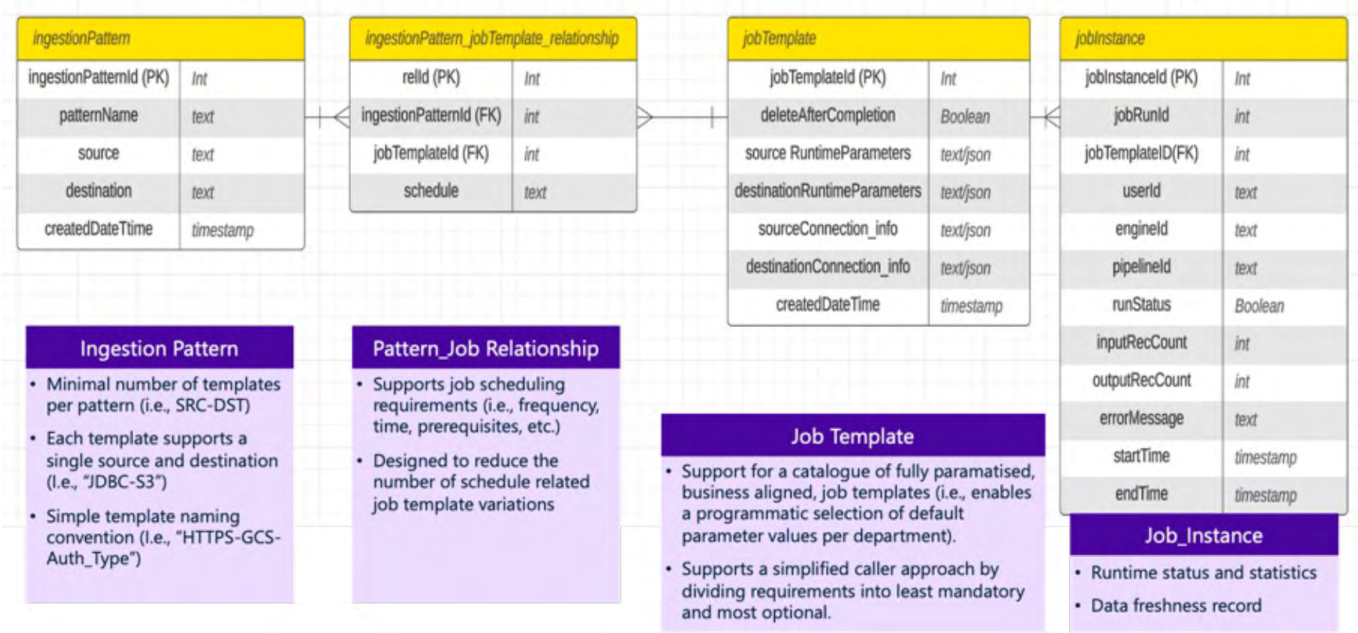
Invoked by the scheduling function and using the pattern details provided, the orchestration function reads the metadata store to capture the caller request parameters (i.e., the mandatory and optional values provided during the data subscription phase) and select the parameterised job template(s) and default runtime settings. The orchestration function uses these details to populate the job templates, build the pipelines and jobs, and if required, provision the necessary runtime infrastructure for downstream job runner execution. capture the caller request parameters (i.e., the mandatory and optional values provided during the data subscription phase) and select the parameterised job template(s) and default runtime settings. The orchestration function uses these details to populate the job templates, build the pipelines and jobs, and if required, provision the necessary runtime infrastructure for downstream job runner execution.

2.4 Job Runner function

Using the StreamSets environment, the Job Runner function runs the automatically orchestrated pipelines and jobs to read the required data from the source, apply the required transformations, and write to the required destination. To meet operational support requirements, runtime events and statistics are written to the metadata store.

2.5 Metadata Store function

The metadata store provides an organised information repository that enables scalable pattern complexity and ease of integration into already established data platforms. The following data model has been provided as an organisational example that minimises the duplication of pattern types and paramatised job templates.



Reference **Appendix A1** for table details and examples.

Typically, a central platform team will be responsible for maintaining the ingestion patterns whereas job templates will be filled in by individual tenants which in turn will be agreed upon in consultation with the platform team. A system automation will manage the job instance (holding the job run and audit details) and the relationship between ingestion patterns and job templates.

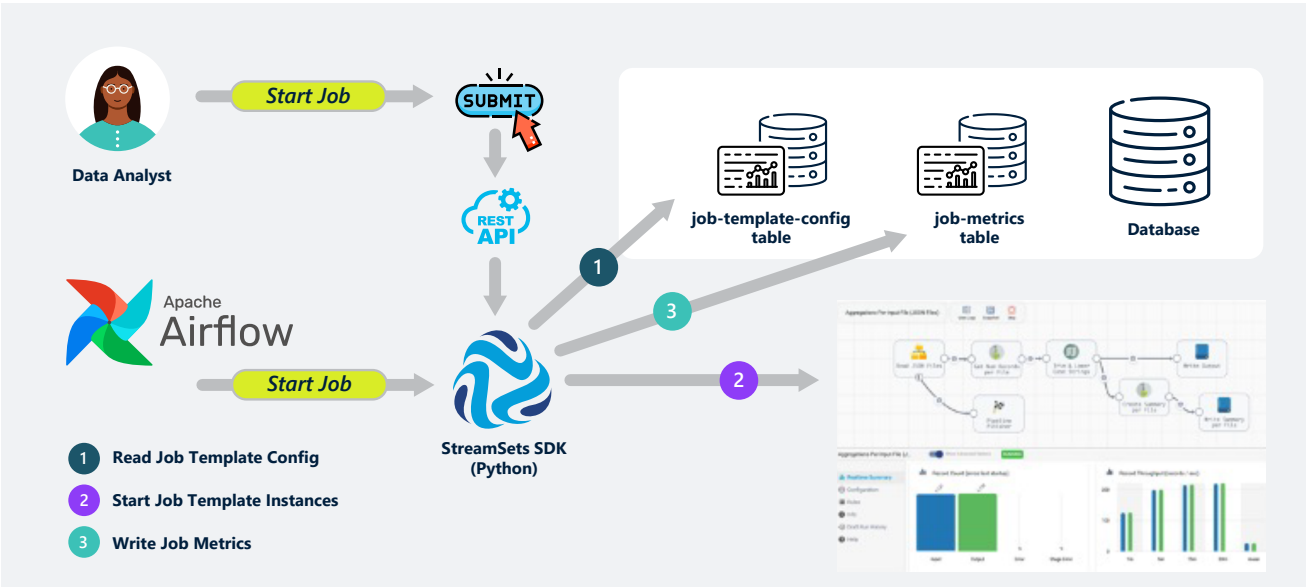
3 Proof of Concept Example

This PoC provides an example of how to use the **StreamSets Platform SDK** to parameterize and start Job Template instances based on parameters retrieved from a database table. The source code and files to the PoC can be found **here** in the GitHub repository.

3.1 Behold, a guided stroll through the process!

In this example:

- A Data Analyst submits a request to run a Job to an app that makes REST API calls to the Job-Template-Service, or a scheduler like Apache Airflow uses Python bindings to directly call the Python Job Template Runner script.
- The Job Template that is run is dynamically selected based on rules applied to the request's source-type and target-type values.
- A subset of the Job's runtime parameters are passed in by the caller as part of the request, which we can consider as "dynamic" runtime parameters, and additional pipeline and connection parameters are retrieved from the configuration store, which we can consider as "static" runtime parameters.
- A Python application built using the StreamSets SDK selects the appropriate Job Template and retrieves the Job Template configuration and static parameters from a set of database tables.
- The Python Application creates and starts Job Template Instance(s) that StreamSets Control Hub schedules on engines.
- The Python Application spawns a new thread per Job Template Instance, and each thread waits until its instance completes, then gathers the instance metrics, and inserts the metrics into a database table.

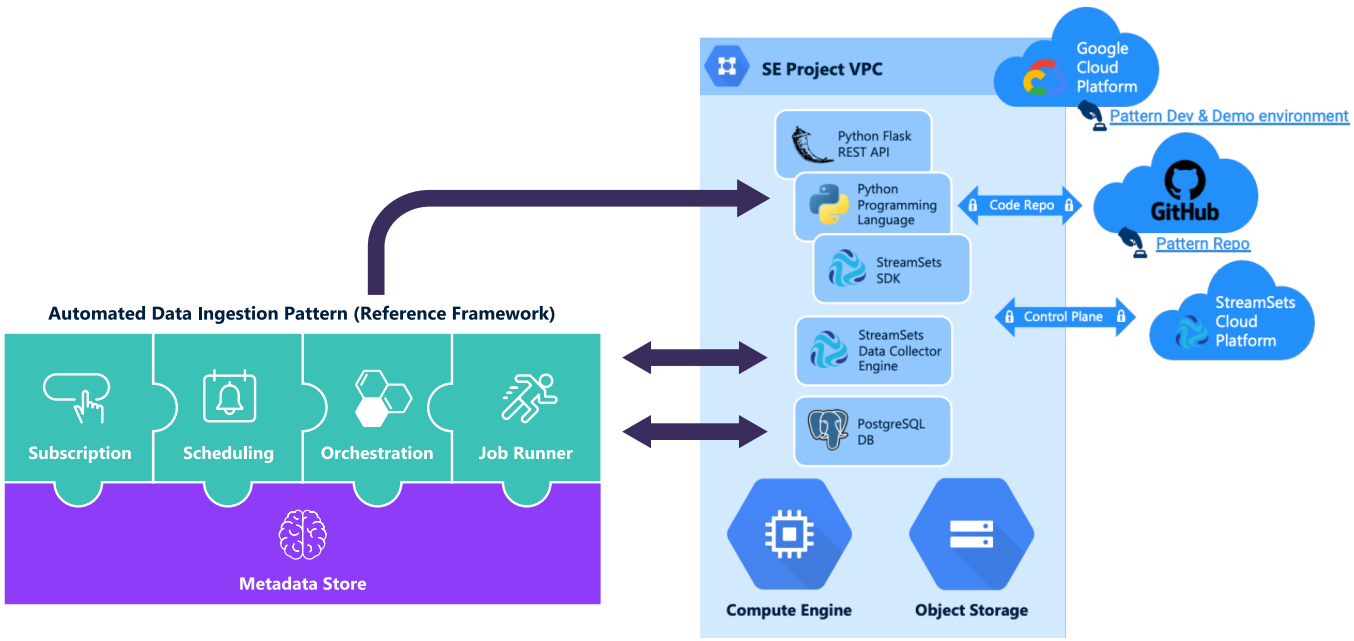


3.2 Prerequisites

In this example, the following components are required:

- A PostgreSQL database
- Python 3.8+
- Psycopg - PostgreSQL database adapter for Python
- Flask - Python web application framework
- StreamSets Platform SDK for Python v6.0.1+
- StreamSets Platform API Credentials for a user with permissions to start Jobs

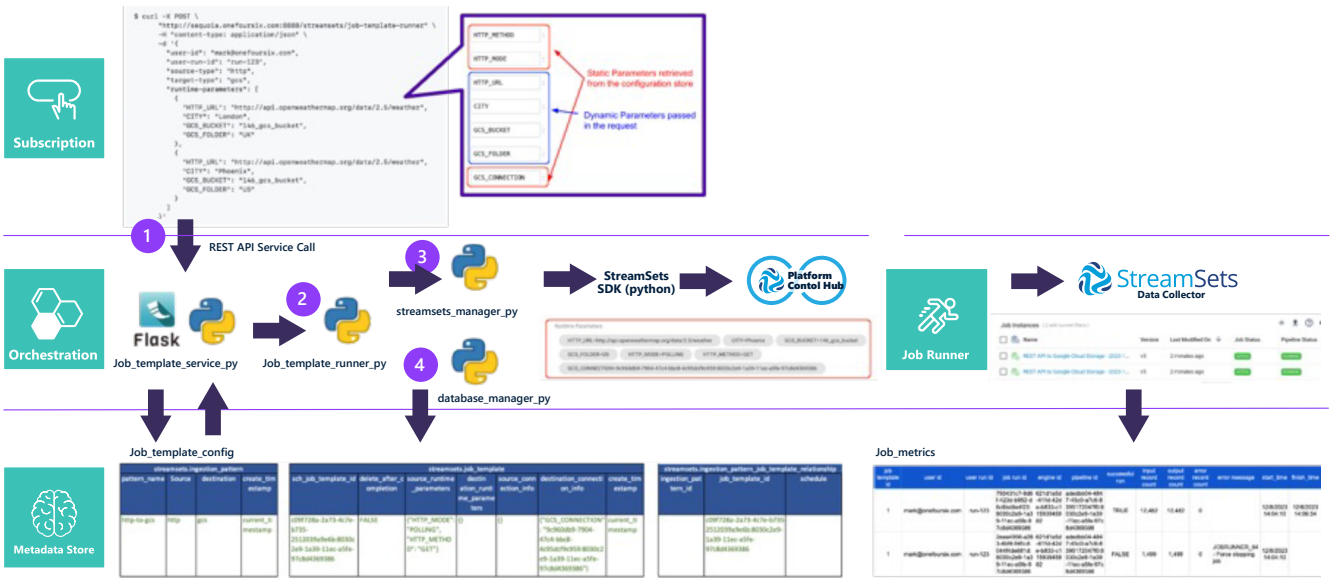
The following is an example Cloud environment used to develop a pattern framework.



3.3 The Nuts and Bolts Unveiling - Implementation details

In this example:

1. The caller's input requirements are posted as a JSON payload to a REST API service provided by Flask, a lightweight web framework for building applications in Python. Runtime parameters are captured as dynamic values passed by the user (i.e., HTTP_URL, "GCS_Bucket") and as static values pre-defined and referenced from the metadata store (i.e., "HTTP_Method", "HTTP_Mode", "GCS_Connection").
2. The REST API endpoint calls a "run_job_template" method in the Python file "job_template_runner.py". The type of job template selected is pre-defined in the metadata store using the caller provided source and target values.
3. Interaction with the StreamSets Platform is managed by the class "StreamSetsManager" in the Python file "streamsets_manager.py".
4. Interaction with the database is managed by the class "DatabaseManager" in the Python file "database_manager.py".

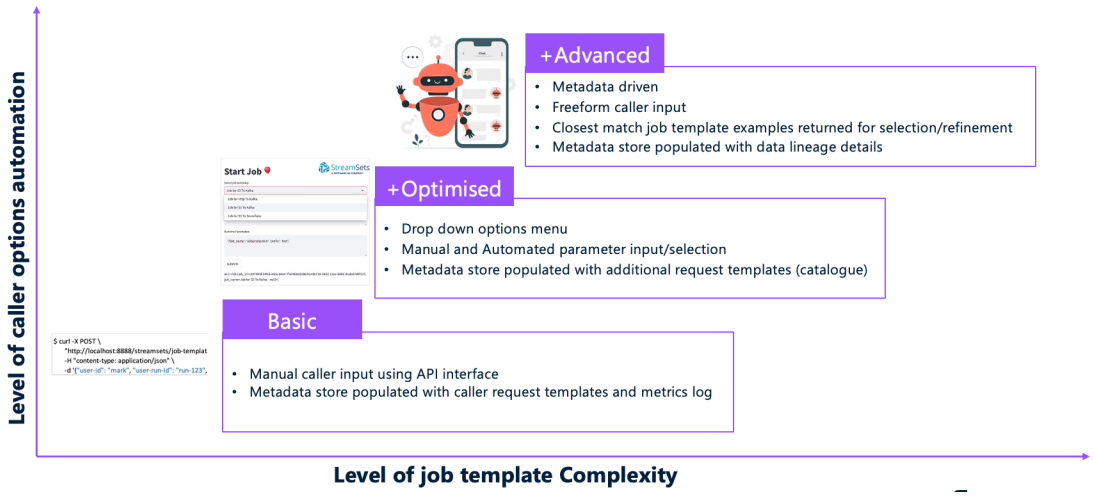


3.4 Configuration and runtime details

Please reference the GitHub repository for details: https://github.com/streamsets/data-integration-patterns

4 Different Approaches to Implementation

Exploring alternative avenues for deploying this framework (such as utilizing an SDK or other automation methods) and examining the flexibility of parameter application are dimensions to consider.



4.1 Basic Pattern

A pattern implementation in its simplest form (i.e., with minimal IT platform integration requirements) is achieved using a REST API data subscription front end where the callers' requirements are submitted at the command line level. This does place an emphasis on the caller submitting a correctly formatted input but removes the more time and resource consuming requirement to develop a new or integrate with an existing front-end system (i.e., web portal, GitLab service, etc.).

Post data ingestion request, the caller inquires upon the metadata store to determine job status and outcome.

4.2 Optimized Pattern

With an automated pattern capability established, further step improvements can be introduced to simplify the customer experience and expand the scope of ingestion scenarios, such as.

- The addition of a drop-down selection menu for patterns and parameters (i.e., subscribing to a webforms provider or capability that takes the callers input using drop down menu selection and submits their request in a REST API format).
- Reducing the number of caller parameter selection requirements by aligning predetermined settings to department level defaults.
- Expanding the metadata store with a wider range of caller request templates.

4.3 Advanced Pattern

With the addition of data lineage and chatbot technology, it is not inconceivable that data analysts could reach their data ingestion outcomes using free form data inquires and metadata store guided intelligence.

Other considerations could include:

- Improved logging, auditing, monitoring, and error handling
- Improved security and access control
- Improved scalability and maintenance support

The application of this pattern is suitable for various scenarios, including migration to the cloud, data ingestion into a data lake, and data ingestion into a staging area in a data warehouse. Attempting these tasks without the proper conditions poses challenges, such as dealing with numerous feeds, extensive manual efforts, and the risk of losing control and visibility over the process.

5 Customer Case: A Large Energy Company

5.1 Overview and Challenges

To expedite insights for the analyst community and minimize IT costs, A large energy tech pioneer sought a platform capable of automating data collection to a public cloud for processing. The challenge was to make this data accessible to diverse applications serving groups with distinct analytic needs without duplicating data storage. Due to the extensive volume and diversity of data, a one-size-fits-all approach to data management and processing proved impractical.

The system handled a substantial daily data load, reaching billions of records from systems spanning Billing, Customer Identity, Payments, Industry data, Metering, Pricing/Markets, Forecasting, and Regulatory categories. The data originated from various technologies like Oracle, Postgres, SQL Server, S3, flat files, and APIs. Simultaneously, there was an ongoing effort to migrate data and workloads from the existing SQL Server infrastructure to enhance efficiency and accommodate evolving requirements.

The objective was to establish a secure, multitenant architecture facilitating self-service data access and the ability to transform data into formats required by data scientists.

5.2 Solution

StreamSets facilitated the ingestion of data into cloud data warehouses, specifically Snowflake and Databricks, as well as the egress of data from the cloud to SQL Server and other existing legacy systems. The implementation of StreamSets involves the utilization of 'pipelines as code,' enabling streamlined operations with minimal maintenance requirements. The platform supports a diverse array of use cases and can accommodate a substantial workload, handling up to 1000 jobs per day, with the potential for growth, particularly during peak periods like month-end processes.

5.3 Result

The implementation of StreamSets has yielded significant outcomes, ensuring a dependable daily provisioning of data to over 400 users, addressing their analytic needs effectively. This has resulted in a notable reduction in the time required for the analyst community to deliver new assets, simultaneously lowering the entry bar for technical proficiency among analysts. The platform's efficiency has further led to a decrease in pressure and expenses for central IT teams, particularly in relation to data pipeline builds and fixes. With 500 jobs executed daily, StreamSets demonstrates robust performance, ingesting data from 100 different databases, APIs, and Filesystems, including the ingestion of billions of records from large tables.

The evaluation criteria, encompassing the speed of provisioning necessary components in the cloud, ease of development, customer support, and the ability to influence product development, showcase StreamSets as a comprehensive solution that not only meets operational demands but also provides a user-friendly experience with strategic impact.

A. Metadata Store Data Model

A.01 Ingestion Pattern

Ingestion Pattern table that holds the key details for a given ingestion pattern and provides the base definition of the pattern.

Field Name	Field Type	Description	Example
ingestionPatternId (PK)	int	The is the primary key that will be auto incremented.	12345
patternName	text	This field contains the name of the pattern.	HTTP_To_GCS
source	text	This field will indicate the source name.	HTTP
destination	text	This field will indicate the destination name.	GCS
createdDateTime	timestamp	A timestamp to indicate when a given pattern was created.	2023-11-13 19:10:25-07

A.02 Job Template

The Job Template table holds the information on the job definition and the associated parameters and is used to create a job instance that will perform the data ingestion.

Field Name	Field Type	Description	Example
jobTemplateId (PK)	int	The is the primary key that will be auto incremented.	12345
deleteAfterCompletion	Boolean	A Boolean field to indicate if the job instance needs to be deleted or kept after the execution.	
sourceRruntimeParameters	text/json	This field contains the details of Source run time parameters (Static) that will control / define the execution of the job instance associated with the template.	{ "RunTimeParameters": { "ResourceURL": "https://sample.com/sample.json", "HTTPMethod": "GET", "DataFormat": "JSON" } })
DestinationRuntimeParameters	text/json	This field contains the details of the destination run time parameters (Static) that will control / define the execution of the job instance associated with the template.	{ "RunTimeParameters": { "ResourceURL": "https://sample.com/sample.json", "HTTPMethod": "GET", "DataFormat": "JSON" } })
sourceConnection_info	text/json	This field will contain the details of the Source connection that need to be used for the execution	2023-11-13 19:10:25-07

Field Name	Field Type	Description	Example
destinationConnection_info	text/json	This field will contain the details of the Destination connection that need to be used for the execution	
createdDateTime	timestamp	A timestamp to indicate when a given pattern was created.	2023-11-13 19:10:25-07

A.02 Job Template

The Job Template table holds the information on the job definition and the associated parameters and is used to create a job instance that will perform the data ingestion.

Field Name	Field Type	Description	Example
jobTemplateId (PK)	int	The is the primary key that will be auto incremented.	12345
deleteAfterCompletion	Boolean	A Boolean field to indicate if the job instance needs to be deleted or kept after the execution.	
sourceRruntimeParameters	text/json	This field contains the details of Source run time parameters (Static) that will control / define the execution of the job instance associated with the template.	{ "RunTimeParameters": { "ResourceURL": "https://sample.com/sample.json", "HTTPMethod": "GET", "DataFormat": "JSON" } })
DestinationRuntimeParameters	text/json	This field contains the details of the destination run time parameters (Static) that will control / define the execution of the job instance associated with the template.	{ "RunTimeParameters": { "ResourceURL": "https://sample.com/sample.json", "HTTPMethod": "GET", "DataFormat": "JSON" } })
sourceConnection_info	text/json	This field will contain the details of the Source connection that need to be used for the execution	2023-11-13 19:10:25-07

A.03 Ingestion Pattern & Job Template Relationship

The Ingestion Pattern and Job Template Relationship table holds the relationship between ingestion patterns and the associated job templates.

Field Name	Field Type	Description	Example
relId (PK)	int	The is the primary key that will be auto incremented.	12345
ingestionPatternId (FK)	int	A field to hold the primary key of the associated ingestion pattern id	
jobTemplateID(FK)	int	A field to hold the primary key of the associated job template Id.	12345
schedule	text	A field to hold the details of the schedule when the jobs will run. It will store the information in Cron expression providing detail of “seconds”, “minutes”, “Hours”, “Day of Month”, “Month”, “Day of Week” and “Year” of the execution (e.g., 0 15 10 ? * * the job will run every day at 10:15	

A.04 Job Instance

The Job Instance table holds the information on a particular job instance and other logging information on the job runs.

Field Name	Field Type	Description	Example
jobInstanceId (PK)	int	The is the primary key that will be auto incremented.	12345
jobRunId	int	A field to hold the primary key of the associated ingestion pattern id	
jobTemplateID(FK)	int	A field to hold the primary key of the associated job template Id.	12345
userId	text	A field to hold the email of the user that triggered the job.	`prateek@streamsets.com)`
engineId	text	A field to hold the engine id associated with the job run.	`“51712c55-f8f8-4dee-bb86-5b2112023f0d")`
pipelineId	text	The field to hold the associated pipeline id.	`“f64fa1c8-69fe-4ce3-8a3c-a7abf7699514:b0550502-ea35-11eb-a03d-53c722bc5504"`
runStatus	Boolean	A field to indicate if the run was successful or not.	1 = successful, 2 = unsuccessful
inputRecCount	Int	This file will hold the total number of input records from a given run.	12202020030
outputRecCount	Int	This file will hold the total number of output records from a given run.	5656565656
errorMessage	text	A field to capture any error message associated with a job run.	`“GCS_09 - Error happened when writing to Output stream”`
startTime	timestamp	A timestamp to indicate when a given job run started.	`2023-11-13 19:10:25-07`
endTime	timestamp	A timestamp to indicate when a given job run completed.	`2023-11-13 19:10:25-07`

Take the next step

Contact your Software AG representative or visit us at www.softwareag.com

Learn more

ABOUT SOFTWARE AG

Software AG helps you create effortlessly connected experiences for your customers, employees and partners with an enterprise-grade iPaaS that integrates anything, anywhere, any way you want. By bringing application, data, API and B2B integration together in the same generative AI-enabled platform, you can run a high-performing enterprise and constantly improve it based on data. Get end-to-end visibility and governance across geographies, IT environments, and complex business ecosystems, with hybrid multi-cloud connectivity, and enterprise-grade security relied on by the most powerful banks, governments, and corporations in the world. Trusted by the world's best brands for more than 50 years, our technology and team of integration enthusiasts will make sure that integration is a driver of innovation for your enterprise. Ready to make integration really work for you? Now you can. Finally.

Learn more at **www.SoftwareAG.com**. Follow us on **LinkedIn** and **X**.

software AG