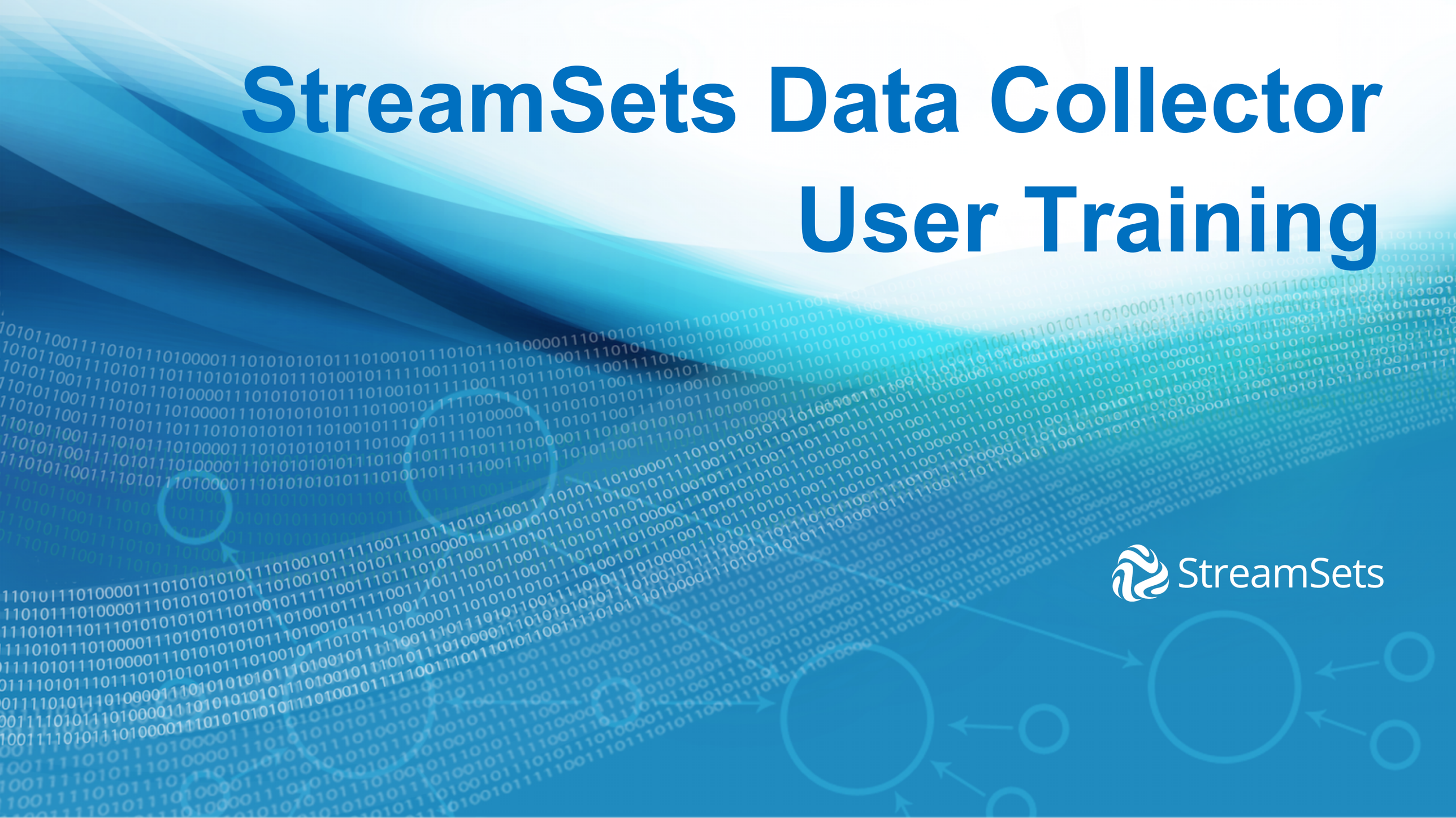
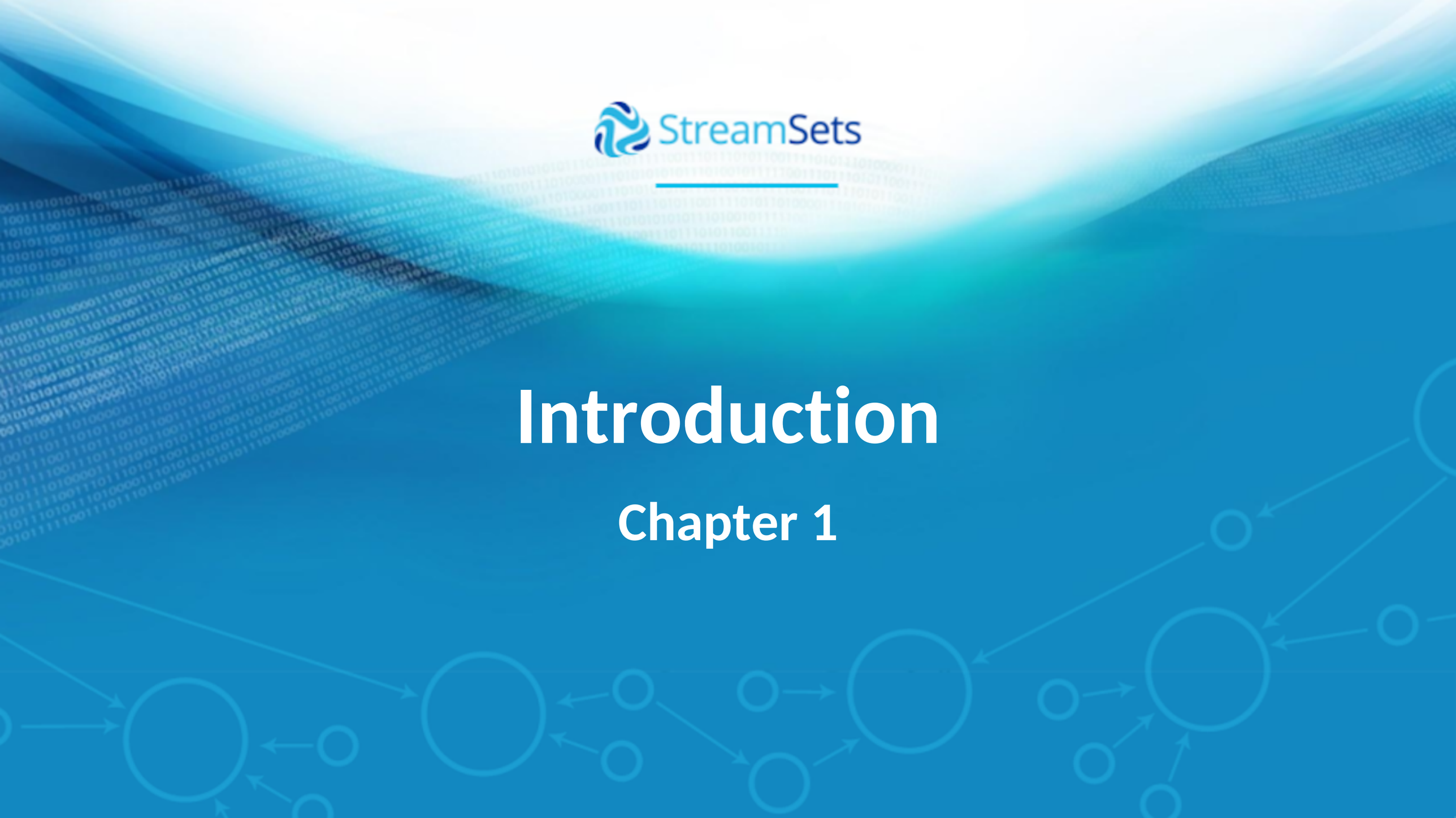
# StreamSets Data Collector User Training






# Introduction

## Chapter 1

# Course Contents

# Introduction

In this course, you will learn:

- **How to navigate the StreamSets Data Collector user interface**
- **How to build data pipelines**
- **How to modify data as it passes through the pipeline**
- **More advanced features of the StreamSets Data Collector**

# Introduction

- **Course Logistics**
- *About StreamSets*
- *StreamSets Data Collector*
- *StreamSets Dataflow Performance Manager*
- *About You, and Your Instructor*

# Course Logistics


- Start/End times
- Breaks/lunch
- Restrooms, WiFi, other administrivia
- Accessing the course materials

# Introduction

- *Course Logistics*
- **About StreamSets**
- *StreamSets Data Collector*
- *StreamSets Dataflow Performance Manager*
- *About You, and Your Instructor*

# About StreamSets


- **StreamSets was founded in 2015**
  - Co-founders came from Informatica and Cloudera
- **Formed to provide a solution to moving data when…**
  - …the data sources can change
  - …the data format can change
  - …the meaning of the data can change
- **We call the combination of these problems *data drift***

# StreamSets Products

- **StreamSets develops two products:**
  - StreamSets Data Collector (SDC)
  - StreamSets Dataflow Performance Manager (DPM)

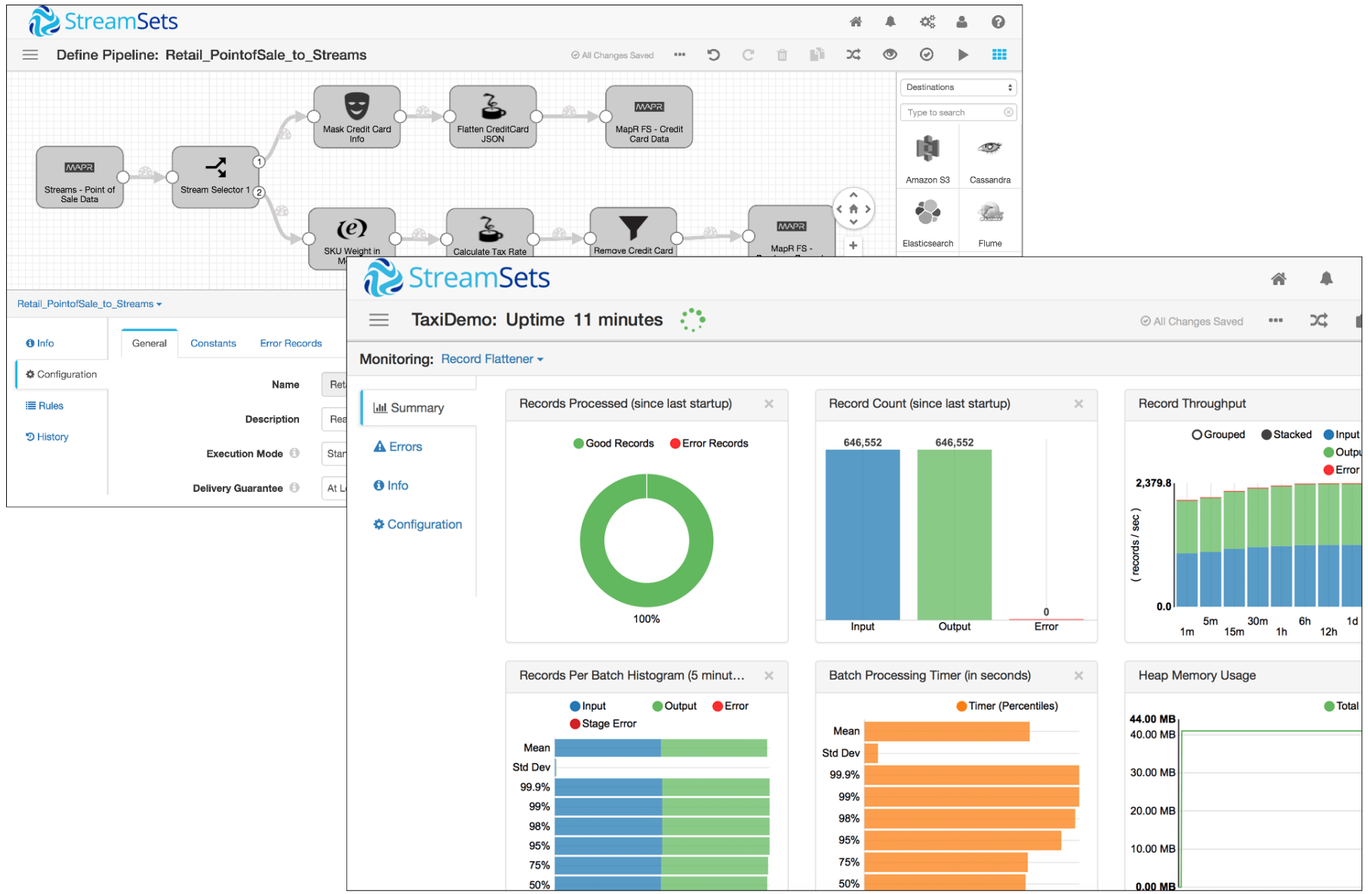
# Introduction

- *Course Logistics*
- *About StreamSets*
- **StreamSets Data Collector**
- *StreamSets Dataflow Performance Manager*
- *About You, and Your Instructor*

# StreamSets Data Collector (1)

- **StreamSets Data Collector: Design complex any-to-any data pipelines in an easy-to-use, graphical environment**
- **Create smart pipelines that handle *data drift***
  - Schema and semantic changes in the data
- **Monitor dataflows with real-time statistics and metrics for each stage of a pipeline**
- **100% open source**

# StreamSets Data Collector (2)



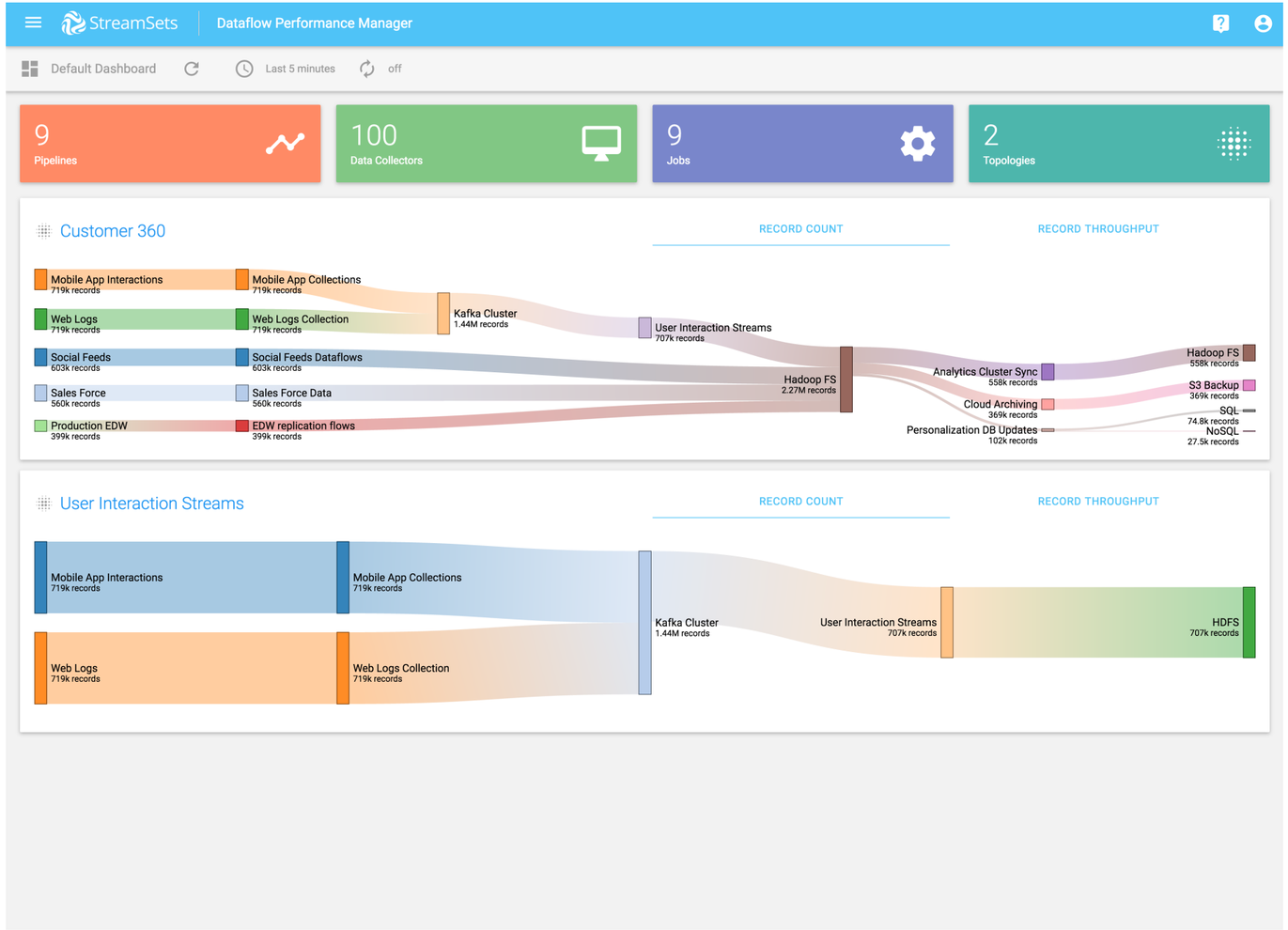
# Introduction

- *Course Logistics*
- *About StreamSets*
- *StreamSets Data Collector*
- **StreamSets Dataflow Performance Manager**
- *About You, and Your Instructor*

# StreamSets Dataflow Performance Manager (1)

- **Dataflow Performance Manager: a single application to manage and monitor all your pipelines**
  - Version, manage, release, and revert pipelines
  - Track data movement SLAs
  - Receive proactive alerts, and view pipeline status through interactive dashboards

# StreamSets Dataflow Performance Manager (2)



# Introduction

- *Course Logistics*
- *About StreamSets*
- *StreamSets Data Collector*
- *StreamSets Dataflow Performance Manager*
- **About You, and Your Instructor**

# About Your, and Your Instructor

- **About your instructor**
- **About you**
  - Your name
  - What do you do?
  - Have you use SDC before?
  - What other data movement tools are you familiar with?
  - What do you expect to get out of the course?


# Introduction to StreamSets Data Collector (SDC)

## Chapter 2

# Course Contents

# Introduction to SDC

In this chapter, you will learn:

- **The key features of StreamSets Data Collector (SDC)**
- **How SDC can be used for real-world data movement**
- **What the SDC Integrated Development Environment (IDE) looks like**
- **How to build a data pipeline in SDC**
- **How to preview the results of your pipeline**

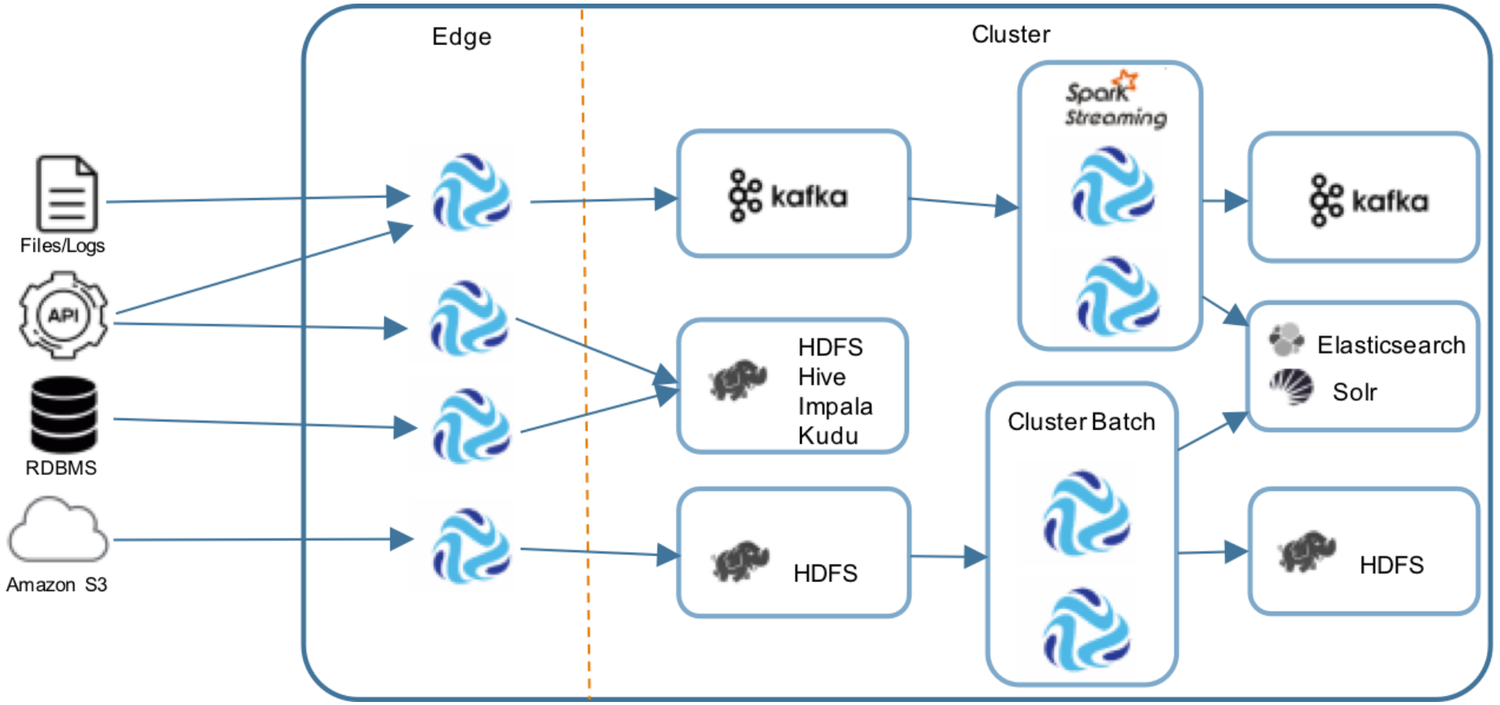
# Introduction to StreamSets Data Collector

- **An Overview of SDC**
- *The SDC User Interface*
- *Building Pipelines*
- *Previewing Data and Running the Pipeline*
- *Hands-On Exercise: Build and Run a Pipeline*
- *Conclusion*

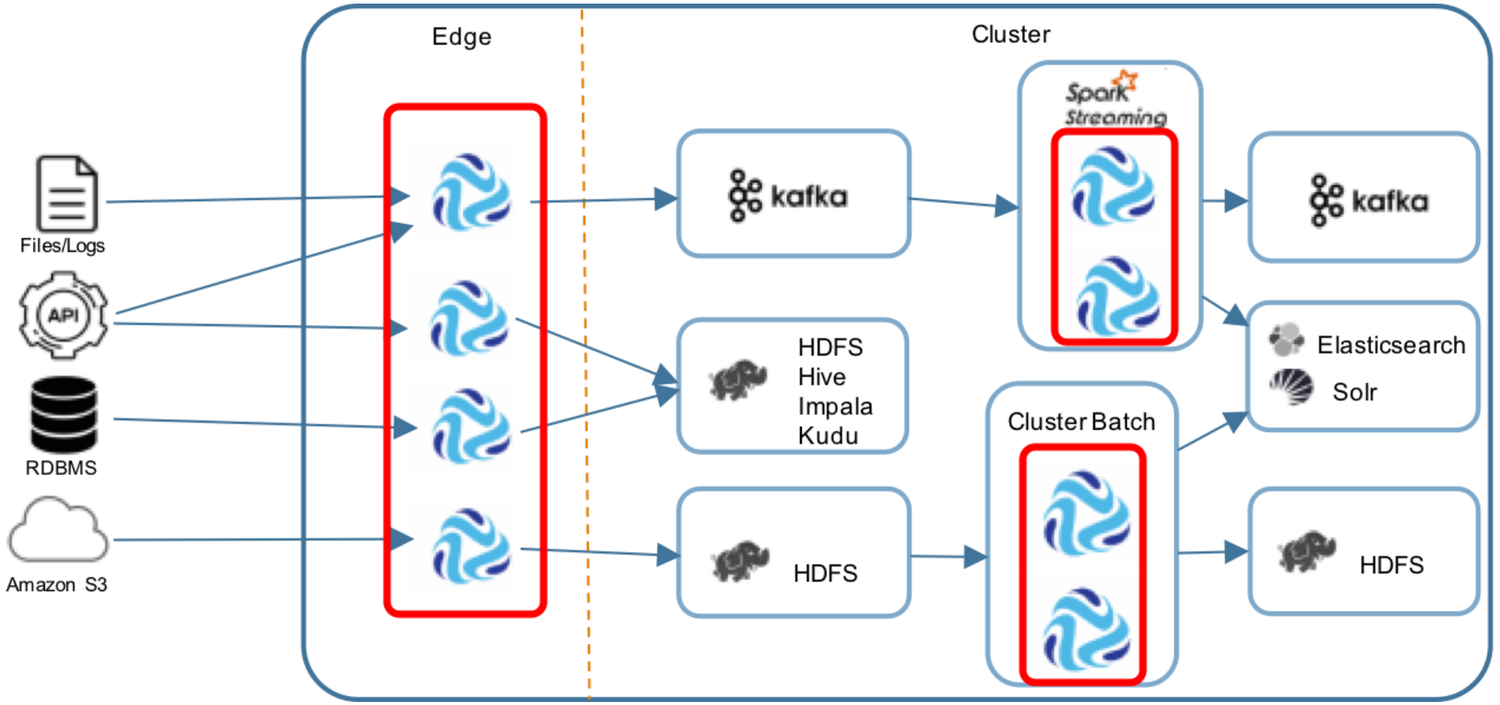
# What is SDC? (1)

- **Any-to-any data movement tool**
- **Integrated Development Environment (IDE) to create data pipelines**
  - Pipelines define the flow of data from origin to destination
- **Works with structured, semi-structured, and unstructured data**
- **Runs in standalone or clustered mode**
- **Open source – Apache 2 license**

# What is SDC? (2)



# What is SDC? (3)



# Introduction to StreamSets Data Collector

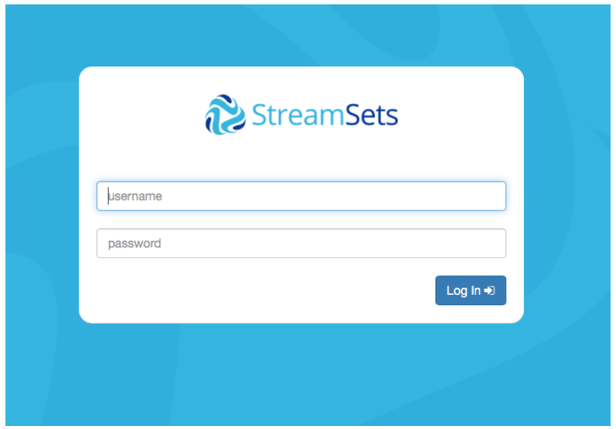
- *An Overview of SDC*
- **The SDC User Interface**
- *Building Pipelines*
- *Previewing Data and Running the Pipeline*
- *Hands-On Exercise: Build and Run a Pipeline*
- *Conclusion*

# Launch and Log In to SDC

- **Launch SDC from the command line by running**

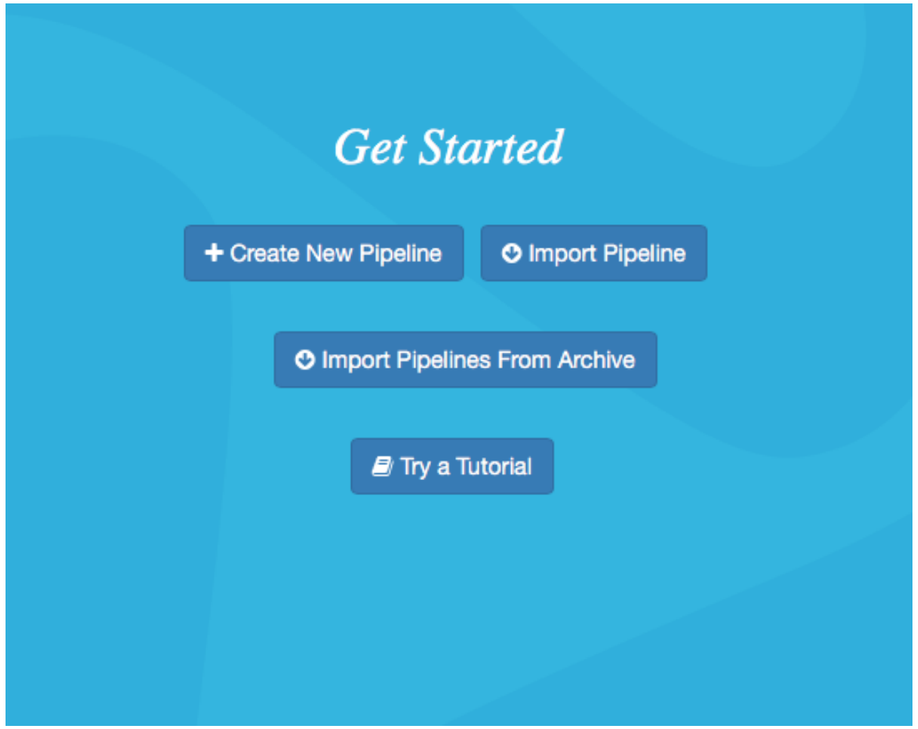
```
$ <sdc_home>/bin/streamsets dc
```

- **In your Web browser, visit `http://<datacollector_host>:18630`**
- **You will be asked to log in**
  - By default, username is `admin`, password is `admin`



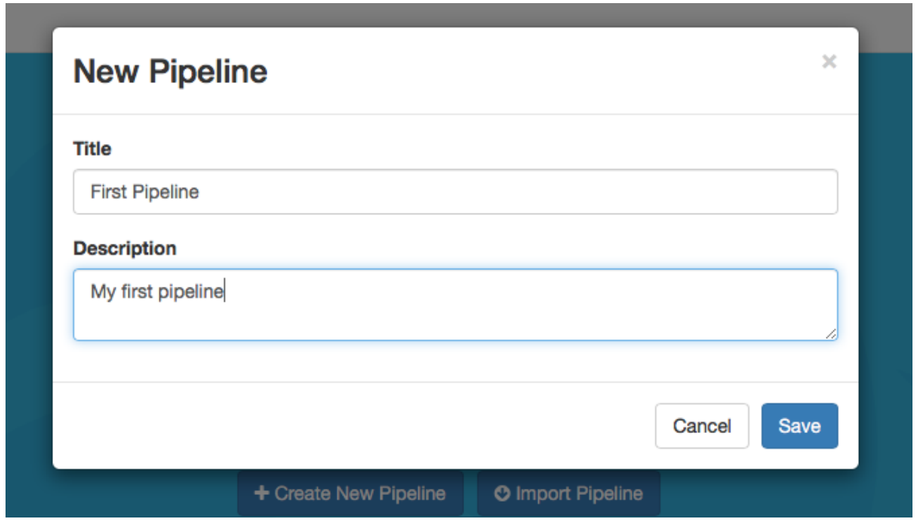
# Initial SDC Screen

- **The first time you launch SDC, you will see this screen**

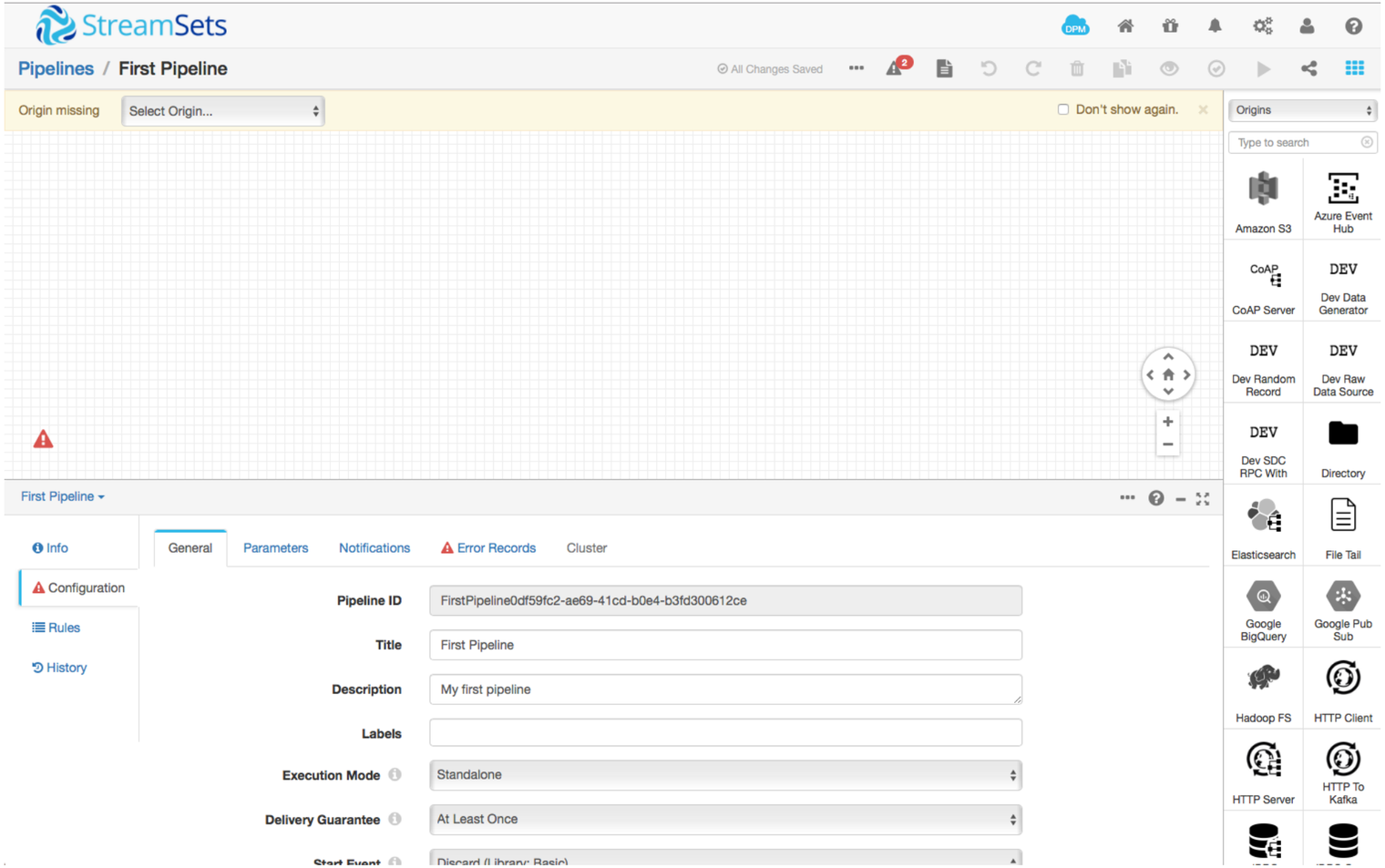


# Create a Pipeline

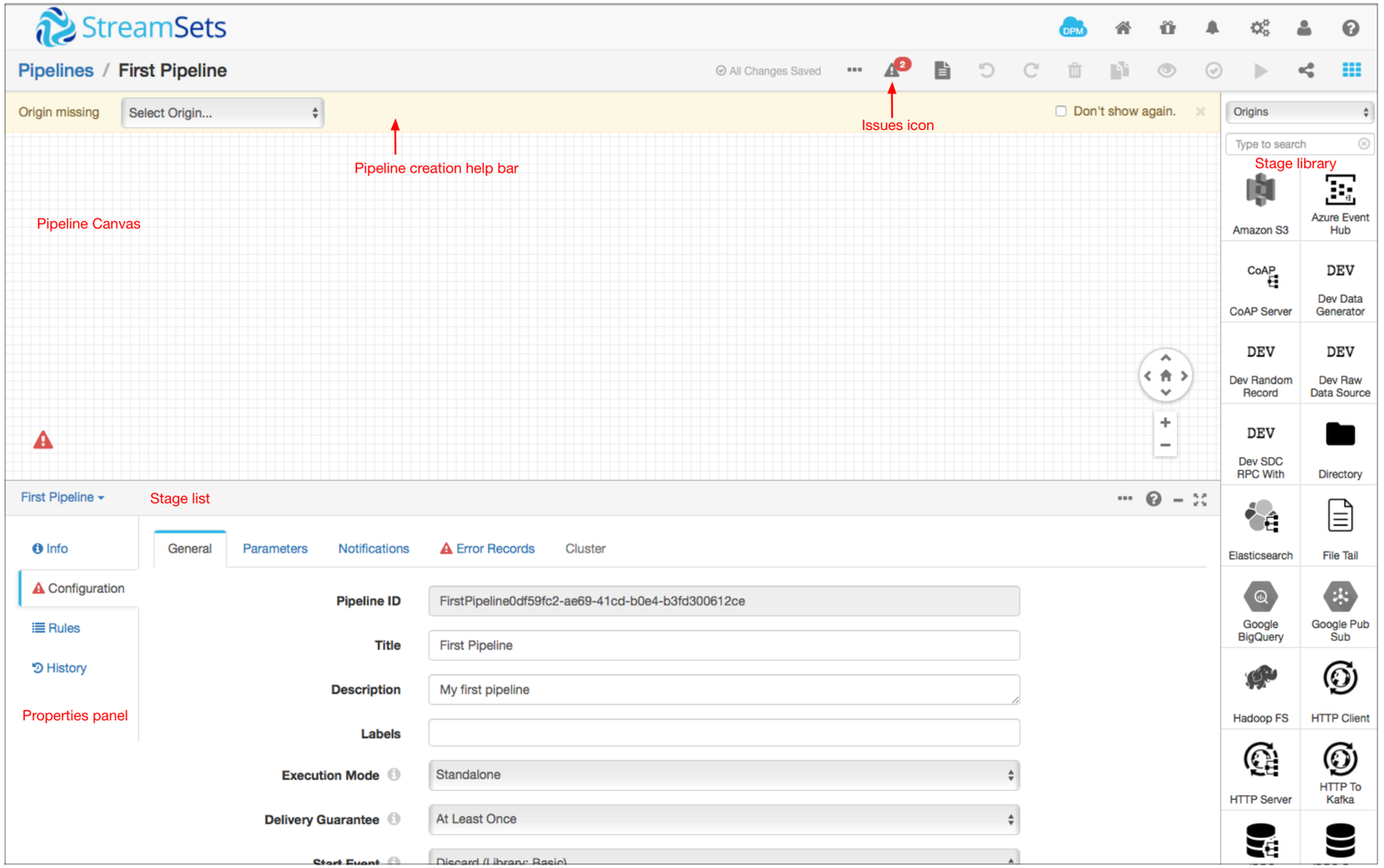
- **Select ‘Create New Pipeline’, and enter a name and description of your pipeline**
  - The pipeline name can be changed later



# The Blank Pipeline Canvas (1)



# The Blank Pipeline Canvas (2)



# Introduction to StreamSets Data Collector

- *An Overview of SDC*
- *The SDC User Interface*
- **Building Pipelines**
- *Previewing Data and Running the Pipeline*
- *Hands-On Exercise: Build and Run a Pipeline*
- *Conclusion*

# Pipeline Terminology

- **Pipeline terminology:**
  - **Origin** – Source of the data
  - **Processor** – Typically modifies the data in some way
  - **Executor** – Triggers a task such as running a shell command, sending an email, running a Spark application etc.
  - **Destination** – Exports the data
- **Pipelines are made up of *stages***
  - One ‘step’ in the pipeline
  - Each stage is an instance of an Origin, Processor, Executor, or Destination

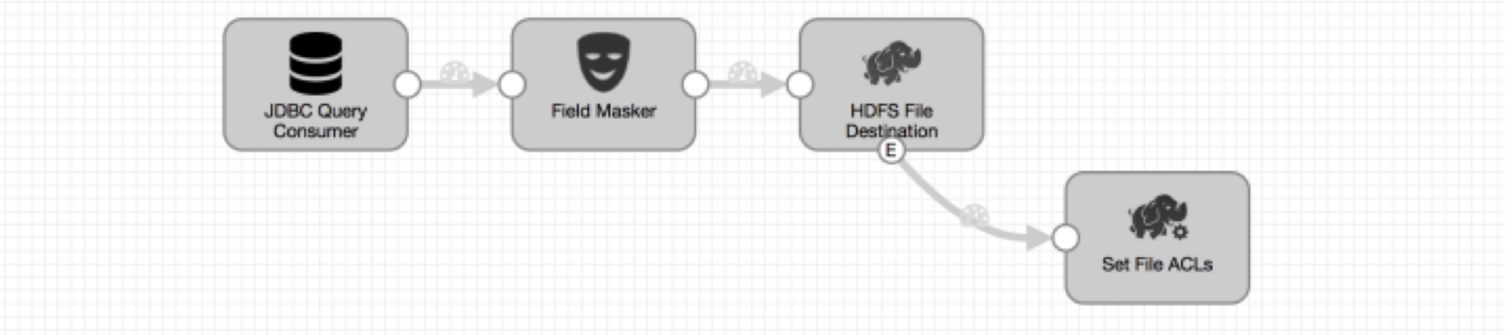
# Data is Comprised of Records

- **Data moves through SDC in the form of *records***
- **Records have *fields***
  - The individual parts of the record
  - Exactly how the record is broken down into fields depends on the configuration of the origin

# Record Headers

- **Each record has a *header* which contains metadata about the record**
  - Example: what file or database table the record came from
- **This metadata can be used to determine how to process a record (more later)**

# Simple Example Pipeline



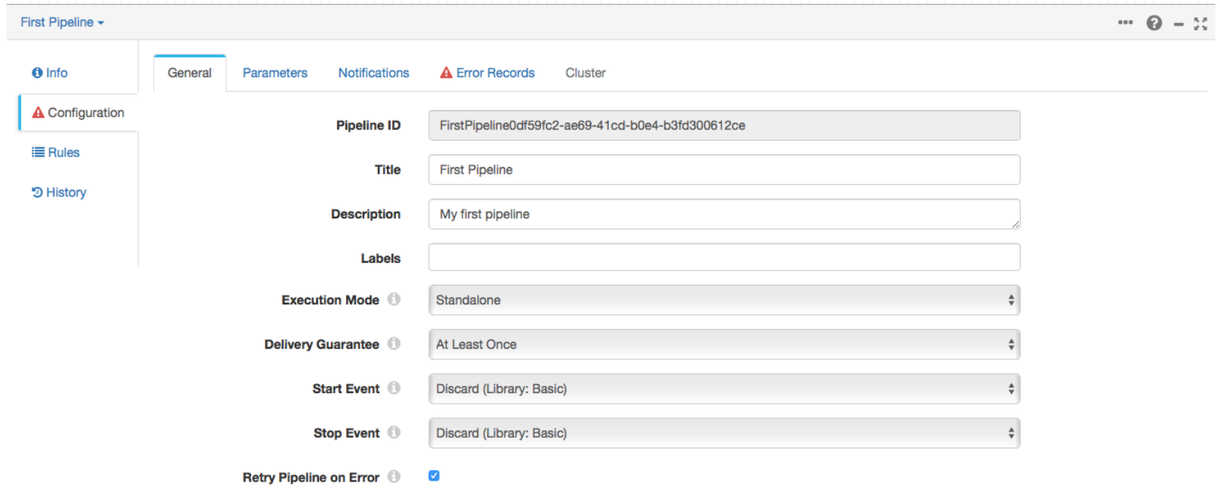

- **This example pipeline does the following:**
  - Reads data from a relational database table
  - Masks (hides) one or more fields
    - Example: personally identifiable information (PII)
  - Writes the records to a file in the Hadoop Distributed File System
  - On completion, sets file attributes via an Executor

# Adding Stages to the Pipeline

- **Add stages to the pipeline by selecting them from the Pipeline Creation Help Bar, or by choosing them from the Stage Library**
  - Note: Only one Origin per pipeline is allowed
- **To adjust the data flow, click and drag between stage 'handles'**
  - To break a connection between two stages, click on the line and hit the `Delete` key

# Configuring the Pipeline

- **To configure overall pipeline options, click on a blank part of the canvas, and choose the `Configuration` tab in the Properties panel**




- **Note that the `Error Records` tab shows a red triangle**
  - This denotes that you need to change something in this section
  - In this case, you need to determine what should happen to error records

# Instructor Demonstration: A Simple Pipeline

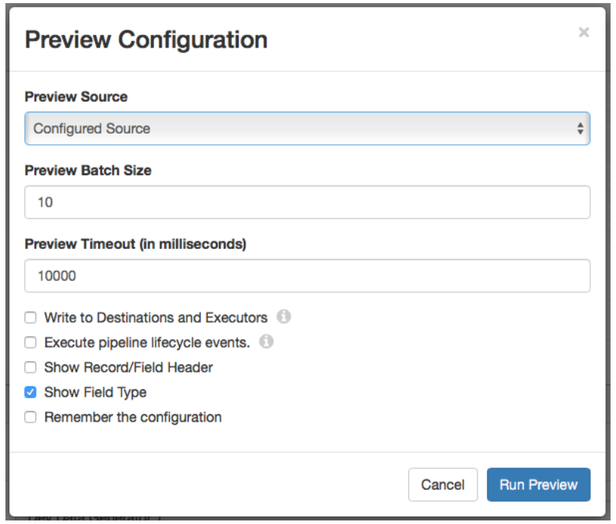

- **Your instructor will walk through creating a simple pipeline**
- **Points to note:**
  - The `Dev` Origins are useful for generating test data
  - During building and testing a pipeline, the `Trash` destination can be used to throw away data

# Introduction to StreamSets Data Collector

- *An Overview of SDC*
- *The SDC User Interface*
- *Building Pipelines*
- **Previewing Data and Running the Pipeline**
- *Hands-On Exercise: Build and Run a Pipeline*
- *Conclusion*

# Previewing Data (1)

- **Before you run the pipeline, it's a good idea to preview the data**
- **Previewing allows you to see records at each stage of the pipeline**
  - Extremely useful when creating complex pipelines
- **To preview, click on the preview icon**

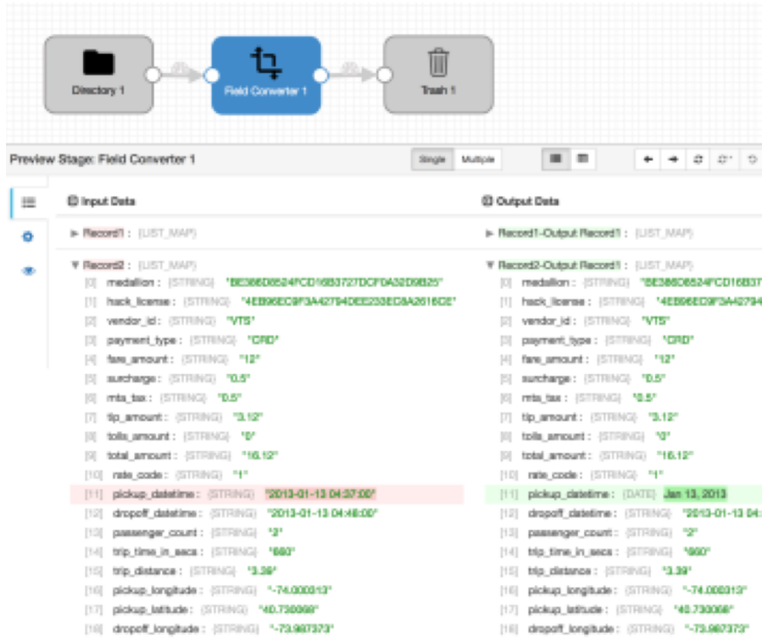


# Previewing Data (2)

- **Preview Source options:**
  - Configured Source
    - Uses the source configured in the pipeline
    - You may need to reset the origin (from the ‘more’ icon) ···
  - Snapshot
    - Uses a portion of data from a previous run of the pipeline
    - Only available if you have snapshotted data during the pipeline run
- **Other configuration options include the number of records to preview (Batch Size)**

# Previewing Data (3)

- After the preview has generated data, click on a stage to view the input and output records to/from the stage

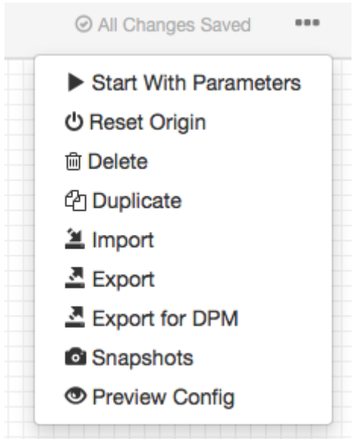


# Previewing Data (4)

- **When viewing the input and output records for a stage, you can edit the input record data to see the effect out the output from the stage**
  - Edit the record data as required, then click the ‘Run With Changes’ button 

# Running the Pipeline

- **To run the pipeline, click the run icon ▶**
- **Note: many origins store their last offset, and will resume a pipeline from that offset**
  - To start from the beginning of the data, choose ‘Reset Origin’ from the ‘more’ icon



# Pipeline Metrics

- As the pipeline runs, metrics will be displayed in the browser window
- Mouse over the graphs for more detail on individual metrics
- Metrics are also exposed via JMX

# Introduction to StreamSets Data Collector

- *An Overview of SDC*
- *The SDC User Interface*
- *Building Pipelines*
- *Previewing Data and Running the Pipeline*
- **Hands-On Exercise: Build and Run a Pipeline**
- *Conclusion*

# Hands-On Exercise: Build and Run a Pipeline

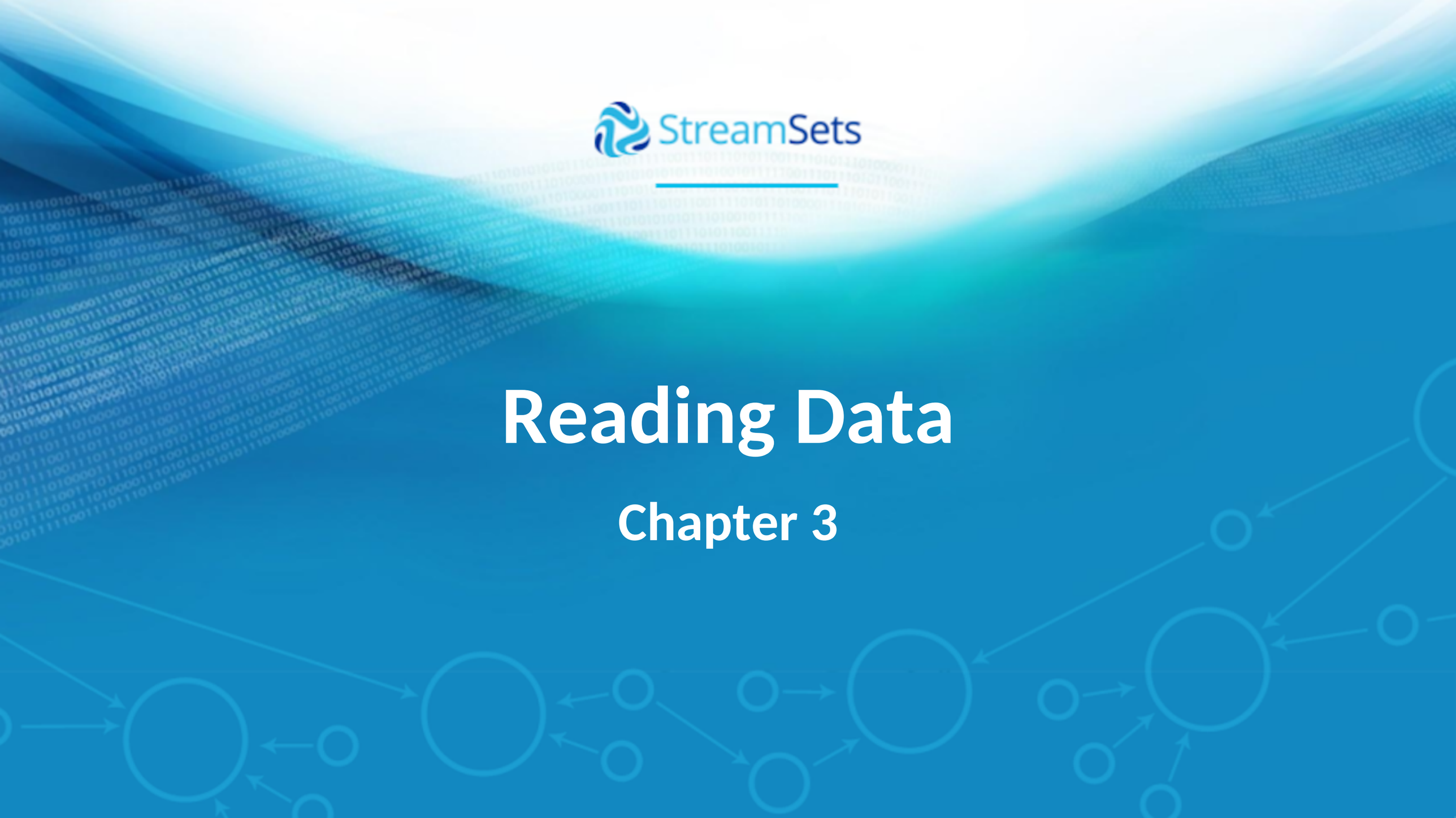

- In this Hands-On Exercise, you will familiarize yourself with the StreamSets Data Collector user interface, and build a simple data pipeline
- Please refer to the Hands-On Exercise Manual

# Introduction to StreamSets Data Collector

- *An Overview of SDC*
- *The SDC User Interface*
- *Building Pipelines*
- *Previewing Data and Running the Pipeline*
- *Hands-On Exercise: Build and Run a Pipeline*
- **Conclusion**

# Conclusion

- StreamSets Data Collector provides an intuitive user interface to create data pipelines
- Pipelines can be previewed, allowing you to inspect the input and output to and from stages




# Reading Data

## Chapter 3

# Course Contents

# Reading Data

In this chapter, you will learn:

- **How to read data from flat files**
- **How to read data from relational database tables**
- **How to read data from Apache Kafka**
- **What other common data origins are available in SDC**

# Reading Data

- **General Origin Considerations**
- *Reading from Flat Files*
- *Reading from RDBMSs*
- *Reading from Apache Kafka*
- *Other Common Origins*
- *Hands-On Exercise*
- *Conclusion*

# Batches of Data

- **SDC typically sends data through the pipeline in 'batches'**
- **Most Origins have a setting to determine the batch size**
  - The number of records sent
- **The memory allocated to SDC must be large enough to hold a single batch of records in memory**
  - Important if each record is large

# Batch Size and Stopping Pipelines

- **You can stop a pipeline from the SDC UI**
- **As of SDC 2.7.1.0, this can take a long time**
  - SDC will wait for a batch to finish processing before stopping the pipeline
    - This is to ensure that data is not lost
  - If a batch takes a long time to generate, this will mean a pipeline may take several minutes to stop
- **It is possible to force-stop a pipeline, if desired**
  - Note that this may cause data loss, depending on the origin being used

# Reading Data

- *General Origin Considerations*
- **Reading from Flat Files**
- *Reading from RDBMSs*
- *Reading from Apache Kafka*
- *Other Common Origins*
- *Hands-On Exercise*
- *Conclusion*

# Flat File Origins

- **SDC can read data from flat files on the local disk, or from remote systems**
  - Amazon S3
  - The Hadoop Distributed File System (HDFS)
  - The MapR File System
  - From an SFTP server

# Reading from Local Disk

- **Two options to read from local disk:**
  - File Tail 

    - Reads new lines as they are written to the file
    - Can handle related archived files
    - Handles files as they are rolled with a utility such as `logrotate`
  - Directory 

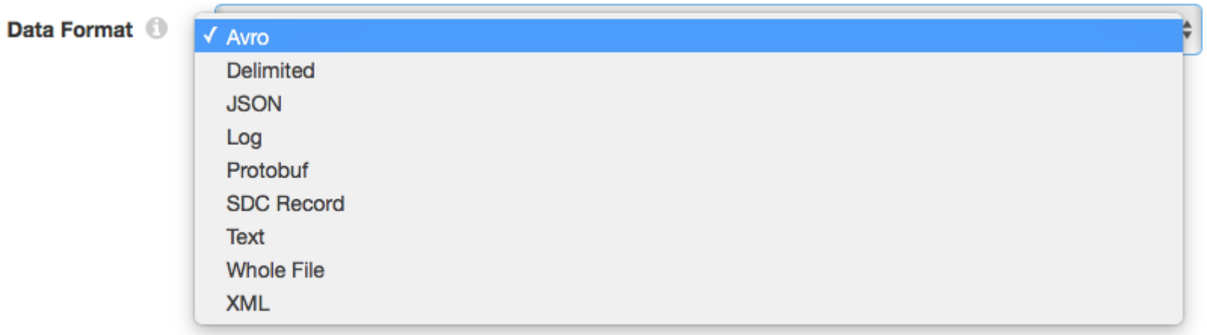
    - Reads files in a directory (and subdirectories) matching a pattern
    - Includes newly-arrived files

# File Tail Origin

- **File Tail origin can treat the data in files as plain text, JSON, or log format**
  - Includes built-in patterns for several standard log formats
  - Can specify a delimiter for fields if treating the data as text
- **Supports compressed files, archives, and compressed archives**
- **After a file has been completely processed it can be archived to a different directory or deleted, if desired**

# Directory Origin

- **Directory Origin supports files in a wide range of formats**




- **Supports compressed files, archives, and compressed archives**
- **Caution: Do not use the Directory Origin to read from files which are being updated**
  - Instead, write files to a different directory, and move them to the processing directory once they are complete

# Other Flat File Origins

- **SFTP origin will read data from files via an SFTP server**
  - As with Directory Origin, problems will occur if you try to read partially-written files
- **Amazon S3 origin will read files from an Amazon S3 bucket**
- **Hadoop FS origin reads from HDFS**
  - Only works in pipelines configured for cluster batch execution mode
- **MapR FS origin reads from the MapR file system**
  - Again, requires cluster execution mode

# Reading Data

- *General Origin Considerations*
- *Reading from Flat Files*
- **Reading from RDBMSs**
- *Reading from Apache Kafka*
- *Other Common Origins*
- *Hands-On Exercise*
- *Conclusion*

# Relational Database Origins


- SDC has origins for JDBC data sources (single and multitable)
- Also provides origins for SQL Server Change Tracking, Oracle Change Data Capture (CDC), and MySQL Binary Log

# JDBC Query Consumer (1)

- **JDBC Query Consumer origin reads data using a user-defined SQL query via a JDBC (Java database connectivity) driver**
- **Query can be for a single table, or join multiple tables together**
- **Uses an offset column (typically a primary key) to keep track of where to read from if performing incremental queries**
  - This is optional for full (bulk) queries
- **Can be used to process change capture data from Microsoft SQL Server**

# JDBC Query Consumer (2)

- **Ideally, the offset column should be a column in the table with unique values**
  - Typically a primary key or other indexed column
- **Questions to consider:**
  - Does the table ever receive updates or deletes?
  - Is there a last modified or insertion timestamp?
  - Is there a primary key?
  - How is the table indexed?

# JDBC Multitable Consumer

- **Reads data from multiple tables**
- **Uses multiple threads for better performance**
- **Often used for database replication**
- **By default, reads the entire contents of each table every time the pipeline starts**
  - Possible to override this and provide offset columns and values for each table

# Other RDBMS Connectors

- **Oracle CDC Client processes change data capture (CDC) information provided by Oracle LogMiner redo logs**
  - LogMiner must be enabled for this origin to work
- **MySQL Binary Log origin processes CDC information from the MySQL binary log**
  - Essentially acts as a MysQL replication slave
- **SQL Server Change Tracking origin processes data from SQL Server change tracking tables**
- **Typically, for each of these you would run a separate pipeline with the JDBC Query Consumer to read the initial data from a table before starting to track the changes**

# Reading Data

- *General Origin Considerations*
- *Reading from Flat Files*
- *Reading from RDBMSs*
- **Reading from Apache Kafka**
- *Other Common Origins*
- *Hands-On Exercise*
- *Conclusion*

# What Is Kafka?

- At its simplest, Apache Kafka is a distributed, replicated, many-to-many pub/sub log
- Used by many companies to transfer data between systems
- *Producers* write data to Kafka, *Consumers* read data from Kafka
- SDC supports both writing to and reading from Kafka

# Reading from Kafka

- **The Kafka Consumer origin becomes a new Kafka Consumer Group for the topic when you start a pipeline for the first time**
  - You can specify the Consumer Group name
  - If multiple pipelines consume from the same topic using the same Consumer Group name, each will receive a portion of the data from the topic (based on the number of partitions in the topic)
- **When you stop and restart the pipeline, processing continues from the last saved offset**
  - Offset is saved in a special Kafka topic as of Kafka v0.9

# Reading Data


- *General Origin Considerations*
- *Reading from Flat Files*
- *Reading from RDBMSs*
- *Reading from Apache Kafka*
- **Other Common Origins**
- *Hands-On Exercise*
- *Conclusion*

# Other Common Origins: General

- SDC has 40+ Origins
- It is possible to write a custom origin if none of the existing ones meet your needs
- Following are just some of the origins, listed by origin type

# Message Broker-Type Origins

- Google Pub/Sub
- JMS
- Kafka
- Kinesis
- MapR Streams
- RabbitMQ
- Redis

# Data Store Origins

- Elasticsearch
- Google BigQuery
- JDBC Query/JDBC Multitable
- MapR DB
- MongoDB
- MySQL Binary Log
- Oracle CDC Client
- Salesforce
- SQL Server CDC Client
- SQL Server Change Tracking

# Lower-Level Origins

- Azure Event Hub Consumer
- HTTP Client/HTTP Server
- MQTT
- TCP Server
- UDP Source
- WebSocket Server

# Reading Data


- *General Origin Considerations*
- *Reading from Flat Files*
- *Reading from RDBMSs*
- *Reading from Apache Kafka*
- *Other Common Origins*
- **Hands-On Exercise**
- *Conclusion*

# Hands-On Exercise: Reading Data From Database Tables

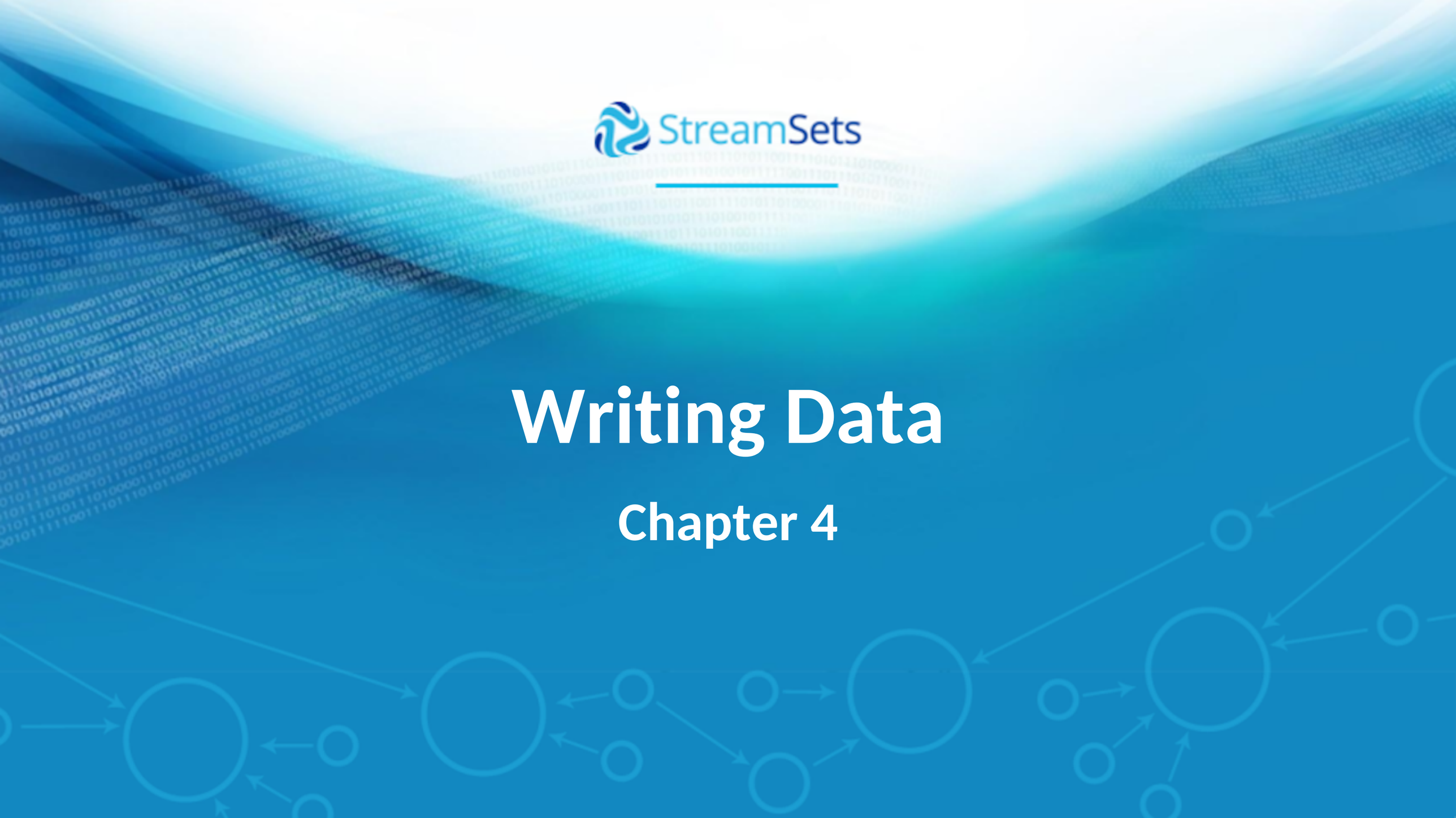

- In this Hands-On Exercise, you will read data from a MySQL server using the JDBC Query Consumer
- Please refer to the Hands-On Exercise Manual

# Reading Data


- *General Origin Considerations*
- *Reading from Flat Files*
- *Reading from RDBMSs*
- *Reading from Apache Kafka*
- *Other Common Origins*
- *Hands-On Exercise*
- **Conclusion**

# Conclusion

- SDC has a wide range of data origins, ranging from high-level to very low-level
- If a standard origin does not fit your needs, you can write your own




# Writing Data

## Chapter 4

# Course Contents

# Writing Data

In this chapter, you will learn:

- **How to write data to flat files**
- **How to write data to relational database tables**
- **How to write data to Apache Kafka**
- **What other common data destinations are available in SDC**

# Writing Data

- **Writing to Flat Files**
- *Writing to Relational Database Tables*
- *Writing to Apache Kafka*
- *Writing to Hadoop*
- *Other Destinations*
- *Hands-On Exercise*
- *Conclusion*

# Flat File Destinations

- **SDC has numerous destinations for writing flat files**
  - Local FS
  - Amazon S3
  - Azure Data Lake Store
  - Flume
  - Hadoop FS (see later)
  - MapR FS

# The Local FS Destination

- **Specify a Directory Template for destination files**
  - Directory/directories will be created as necessary
  - Example to create a new directory every hour:
    `/tmp/out/${YYYY()}-${MM()}-${DD()}-${hh()}`
- **Files will be rolled:**
  - After a specified idle time
  - After a specified number of records have been written
  - After a file reaches a specific size
  - If a record contains a special 'roll attribute'
- **File formats include Avro, JSON, delimited, Protobuf, binary…**
- **Supports file compression**

# Writing Data

- *Writing to Flat Files*
- **Writing to Relational Database Tables**
- *Writing to Apache Kafka*
- *Writing to Hadoop*
- *Other Destinations*
- *Hands-On Exercise*
- *Conclusion*

# The JDBC Producer Destination

- **The JDBC Producer destination writes data to an RDBMS table**
- **Each record can be used for an `INSERT`, `UPDATE`, or `DELETE` operation**
  - Operation is specified in the `sdc.operation.type` value in the record header
    - If this does not exist, either a default operation can be used, the record can be discarded, or it can be sent to an error stream
- **JDBC Producer supports multi-row inserts for improved performance**
  - This is not supported by all RDBMSs; check your documentation
- **By default, fields are written to columns with the same name**
  - You can specify a different mapping if necessary

# Writing Data

- *Writing to Flat Files*
- *Writing to Relational Database Tables*
- **Writing to Apache Kafka**
- *Writing to Hadoop*
- *Other Destinations*
- *Hands-On Exercise*
- *Conclusion*

# The Kafka Producer Destination

- **The Kafka Producer destination writes records to a Kafka topic**
  - Specify which partition each record is written to in the configuration
    - Round-Robin, Random, Expression, or Default (specify the field to use as the key)
- **Records can be written in a variety of formats**
  - Avro, JSON, Delimited, Protobuf,…

# Writing Data


- *Writing to Flat Files*
- *Writing to Relational Database Tables*
- *Writing to Apache Kafka*
- **Writing to Hadoop**
- *Other Destinations*
- *Hands-On Exercise*
- *Conclusion*

# The Hadoop FS Destination


- **The Hadoop FS Destination is configured very similarly to the Local FS destination**
  - Writes to HDFS, rather than local disk
- **Similarly, the MapR FS destination writes to the MapR File System**
- **Supports Kerberos authentication**
- **Supports file compression**

# Writing Data


- *Writing to Flat Files*
- *Writing to Relational Database Tables*
- *Writing to Apache Kafka*
- *Writing to Hadoop*
- **Other Destinations**
- *Hands-On Exercise*
- *Conclusion*

# Other Destinations

- **A wide range of other destinations are available**
- **Examples:**
  - Cassandra, Elasticsearch, Google Bigtable, HBase, InfluxDB, MapR DB, MongoDB, Redis, Salesforce, Solr
  - Amazon S3, Azure Data Lake Store, Flume, Kudu
  - Azure IoT Hub Producer, HTTP Client, JMS Producer, Kinesis Firehose, MQTT Publisher, RabbitMQ Producer, WebSocket Client

# Writing Data


- *Writing to Flat Files*
- *Writing to Relational Database Tables*
- *Writing to Apache Kafka*
- *Writing to Hadoop*
- *Other Destinations*
- **Hands-On Exercise**
- *Conclusion*

# Hands-On Exercise: Writing Data to Hadoop


- In this Hands-On Exercise, you will write data to a Hadoop cluster
- Please refer to the Hands-On Exercise Manual

# Writing Data


- *Writing to Flat Files*
- *Writing to Relational Database Tables*
- *Writing to Apache Kafka*
- *Writing to Hadoop*
- *Other Destinations*
- *Hands-On Exercise*
- **Conclusion**

# Conclusion

- SDC provides a wide range of destinations to which data can be written
- You can write your own destination if a suitable one does not exist


# Modifying and Enriching Data

## Chapter 5

# Course Contents

# Modifying and Enriching Data


In this chapter, you will learn:

- **How to use the StreamSets Expression Language**
- **What Data Processors and Lookup Processors are available**
- **How to write more complex data manipulation scripts with the Language Evaluation Processors**

# Modifying and Enriching Data

- **The StreamSets Expression Langage**
- *Data Processors*
- *Lookup Processors*
- *Language Evaluation Processors*
- *Hands-On Exercise*
- *Conclusion*

# The StreamSets Expression Language: Introduction


- StreamSets Expression Language can be used to create expressions that evaluate or modify data
- Based on the JSP 2.0 expression language
- Can be used to configure expressions and conditions in processors
- Can be used to determine some destination settings such as the directory in the LocalFS and Hadoop FS destinations

# StreamSets Expression Language: Sample Uses

- **Some uses for the Expression Language:**
  - Replace missing values in fields with a default value
  - Perform numeric calculations on a field's values
  - Capitalize strings in a field
  - Route records to different stages based on the value of a field
  - Generate new fields based on the values in existing fields, or from external data sources

# Expression Language: Basic Syntax (1)

- **Basic syntax: `${<expression>}`**

# Expression Language: Basic Syntax (2)

- **Basic syntax: `${<expression>}`**

```
${2 + 2}
```

Numeric, yields the value 4

# Expression Language: Basic Syntax (3)

- **Basic syntax: `${<expression>}`**

```
${2 + 2}
```

Numeric, yields the value 4

```
${record:value('/total_amount') * 2}
```

Numeric, returns the value of the ‘total amount’ field multiplied by 2

# Expression Language: Basic Syntax (4)

- **Basic syntax: `${<expression>}`**

```
${2 + 2}
```

Numeric, yields the value 4

```
${record:value('/total_amount') * 2}
```

Numeric, returns the value of the ‘total amount’ field multiplied by 2

```
${str:toUpper(record:value('/name')}
```

String, returns the string in the ‘name’ field with all characters turned to upper case

# Conditional Expressions

- **Conditional Expressions**

```
${record:value('/payment_type') == 'CC'}
```

Returns `true` if the value in the `payment_type` field is `CC` `false` otherwise

- **Conditional operators include:** ==, !=, >, <, >=, <=

# If-Then-Else

- **The ? operator gives an if-then-else construct:**

```
${record:value('/age') >= 18 ? 'Adult' : 'Child'}
```

If the value in the age field is 18 or higher, returns `Adult`, otherwise returns `Child`

# Types of Functions

- **The Expression Language has a variety of different types of functions, including:**
  - Record functions

    Example: `record:exists(<field path>)`
  - Base64 functions

    Example: `base64:decodeBytes(<string>)`
  - Math functions

    Example: `math:ceil(<number>)`
  - String functions

    Example: `str:replaceAll(<string>, <regEx>, <newString>)`
  - Time functions

    Example: `time:extractDateFromString(<string>, <format string>)`
  - Record functions (more later)

# Modifying and Enriching Data

- *The StreamSets Expression Langage*
- **Data Processors**
- *Lookup Processors*
- *Language Evaluation Processors*
- *Hands-On Exercise*
- *Conclusion*

# Field Processors

- **SDC includes a wide range of field processors to manipulate and modify records as they pass through the pipeline**
- **Examples:**
  - Field Flattener
    - Flattens a record to remove nested fields
  - Field Masker
    - Used to hide values in specific fields
  - Field Order
    - Re-orders the fields in a record
  - Field splitter
    - Breaks a field up into new fields based on a delimiter

# Expression Evaluator

- **The Expression Evaluator performs calculations and writes results to new (or existing) fields**
  - Can also change or add new record header attributes
- **Provides a powerful way to modify data based on your business logic**

# Modifying and Enriching Data


- *The StreamSets Expression Langage*
- *Data Processors*
- **Lookup Processors**
- *Language Evaluation Processors*
- *Hands-On Exercise*
- *Conclusion*

# JDBC Lookup Processor

- **The JDBC Lookup Processor queries an external database**
  - Used to enrich data in the pipeline
- **Specify the SQL query to be used. Example:**

```
SELECT CurrencyName FROM Conversions WHERE CountryCode = '${record:value('/country_code')}'
```

- **Can be configured to cache lookup results**

# Other Lookup Processors

- **Other Lookup Processors exist:**
  - HBase Lookup
  - Kudu Lookup
  - Redis Lookup
  - Salesforce Lookup
  - Static Lookup

# Modifying and Enriching Data

- *The StreamSets Expression Langage*
- *Data Processors*
- *Lookup Processors*
- **Language Evaluation Processors**
- *Hands-On Exercise*
- *Conclusion*

# Performing Custom Data Manipulation


- **SDC provides Groovy, Jython, and JavaScript Evaluators**
- **Use when none of the built-in processors is sufficient for your needs**
- **Each can process data record-by-record, or batch-by-batch**
- **Each has three sections:**
  - Init: Runs when the pipeline starts
  - Main: Processes the records or batches of records
  - Destroy: Run when the pipeline stops
- **Each can call external Java code**

# Example Jython Script (1)

```
try:
  for record in records:
    cc = record.value['credit_card']
    if cc == '':
      error.write(record, "Payment type was CRD, but credit card was null")
      continue

    cc_type = ''
    if cc.startswith('4'):
      cc_type = 'Visa'
    elif cc.startswith(('51','52','53','54','55')):
      cc_type = 'MasterCard'
    elif cc.startswith(('34','37')):

...
```

# Example Jython Script (2)

```
    else:
      cc_type = 'Other'

    record.value['credit_card_type'] = cc_type

    output.write(record)
except Exception as e:
  error.write(record, e.message)
```

# Modifying and Enriching Data

- *The StreamSets Expression Langage*
- *Data Processors*
- *Lookup Processors*
- *Language Evaluation Processors*
- **Hands-On Exercise**
- *Conclusion*

# Hands-On Exercise: Modifying Data


- In this Hands-On Exercise, you will modify data using SDC Processors
- Please refer to the Hands-On Exercise Manual

# Modifying and Enriching Data

- *The StreamSets Expression Langage*
- *Data Processors*
- *Lookup Processors*
- *Language Evaluation Processors*
- *Hands-On Exercise*
- **Conclusion**

# Conclusion

- The StreamSets Evaluation Language, along with the built-in Evaluators provides a powerful way to modify data as it passes through the pipeline
- The Groovy, JavaScript, and Jython evaluators are available for more complex data manipulation


# Pipeline Events, Rules, and Alerts

## Chapter 6

# Course Contents

# Pipeline Events, Rules, and Alerts

In this chapter, you will learn:

- **What *events* are in the StreamSets Data Collector**
- **How to perform actions based on specific metrics**
- **How to handle Data Drift**

# Pipeline Events, Rules, and Alerts

- **Dataflow Triggers**
- *Metric and Data Rules*
- *Data Drift Synchronization*
- *Conclusion*

# Dataflow Triggers

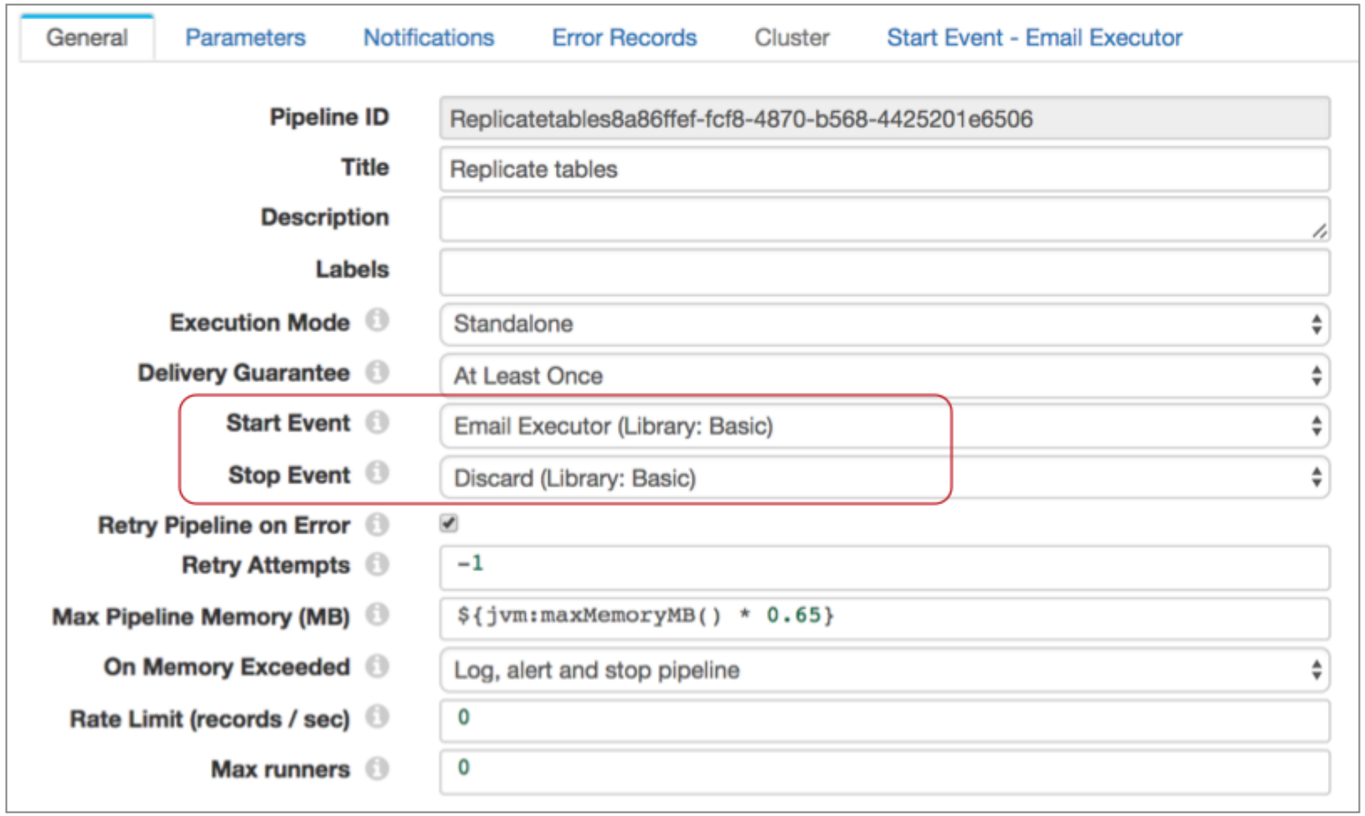
- **Dataflow Trigger**
  - Instruction to SDC to start a task in response to an event that occurs in the pipeline
- **Examples:**
  - Start a MapReduce job after a file has been written to HDFS
  - Stop a pipeline after all data has been read from a database table
  - Send an email after a directory of files has been processed

# Pipeline Events (1)

- **As of SDC 2.7.1, SDC generates events when a pipeline...**
  - Starts
    - Runs an executor, and waits for it to return before it initializes the pipeline stages
  - Stops
    - The pipeline stops after the executor completes
- **Pipeline events can be consumed by an executor or by another pipeline**
  - Pass to another pipeline if, e.g., you need multiple executors to run

# Pipeline Events (2)

- Configure the effect of a pipeline event from the main pipeline configuration options section



# Stage Events (1)

- **Many sources and destinations, and some processors, generate events**
- **Examples:**
  - Directory origin
    - Starts processing a file, completes processing a file, completes processing all available files
  - JDBC Query origin
    - Successfully completes a query, fails to complete a query, completely processes all data returned by a query
  - Hadoop FS destination
    - Closes a file, completely streams an entire file

# Stage Events (2)

- **Process stage events by enabling the 'Produce Events' checkbox in the stage configuration**
- **Route the event to a ‘task execution stream’ (an executor), or to an ‘event storage stream’ (a pipeline that stores the event record)**
- **Executors perform tasks when they receive event records**
  - Examples:
    - Email executor
    - Hive Query executor
    - Shell executor
    - Pipeline Finisher executor

# Pipeline Events, Rules, and Alerts

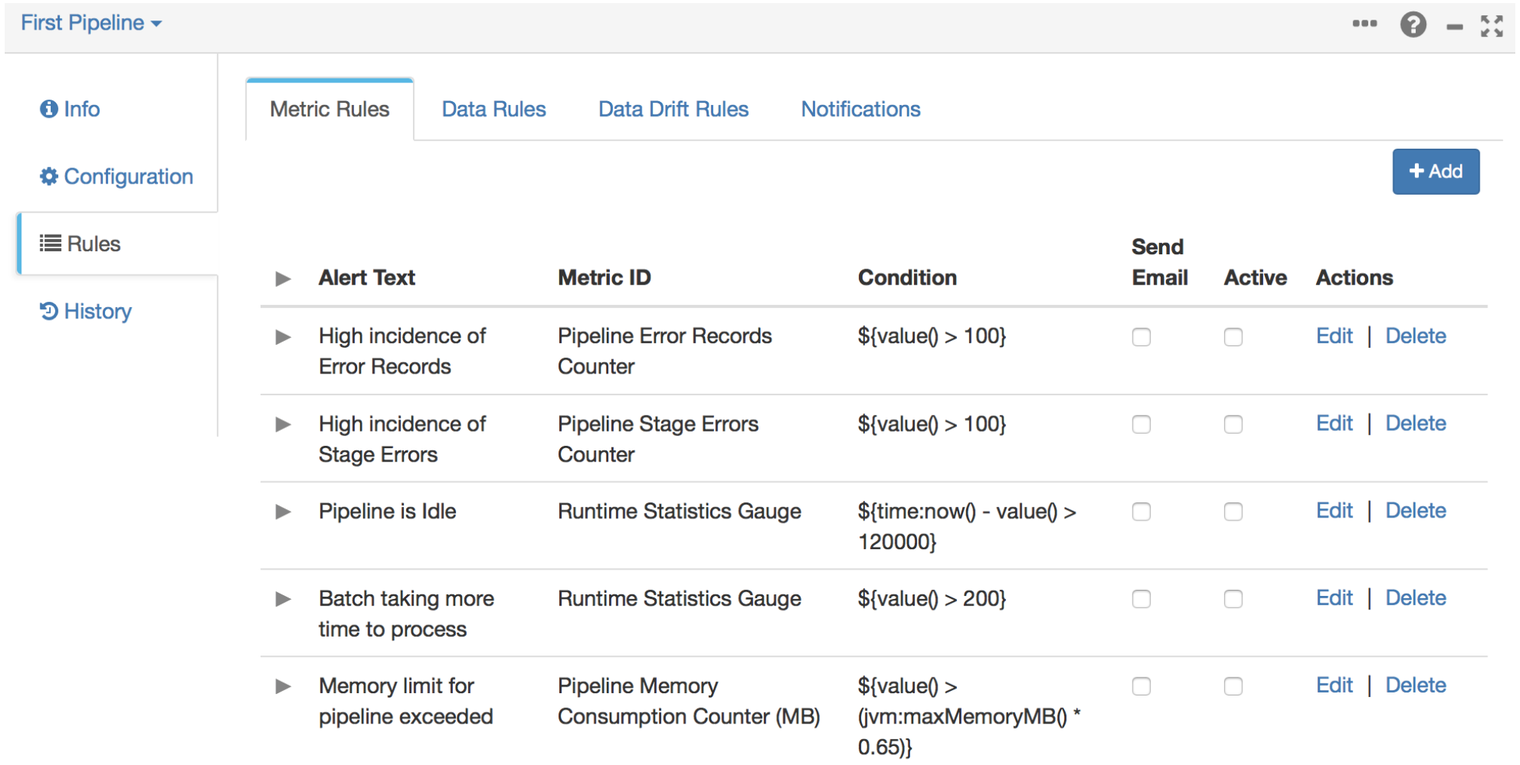
- *Dataflow Triggers*
- **Metric and Data Rules**
- *Data Drift Synchronization*
- *Conclusion*

# Metric Rules

- **Metric rules**
  - Rules to react to real-time statistics of running pipelines
- **Examples:**
  - Too many bad records
  - Too many error records
  - Memory limit exceeded

# Configuring Rules (1)

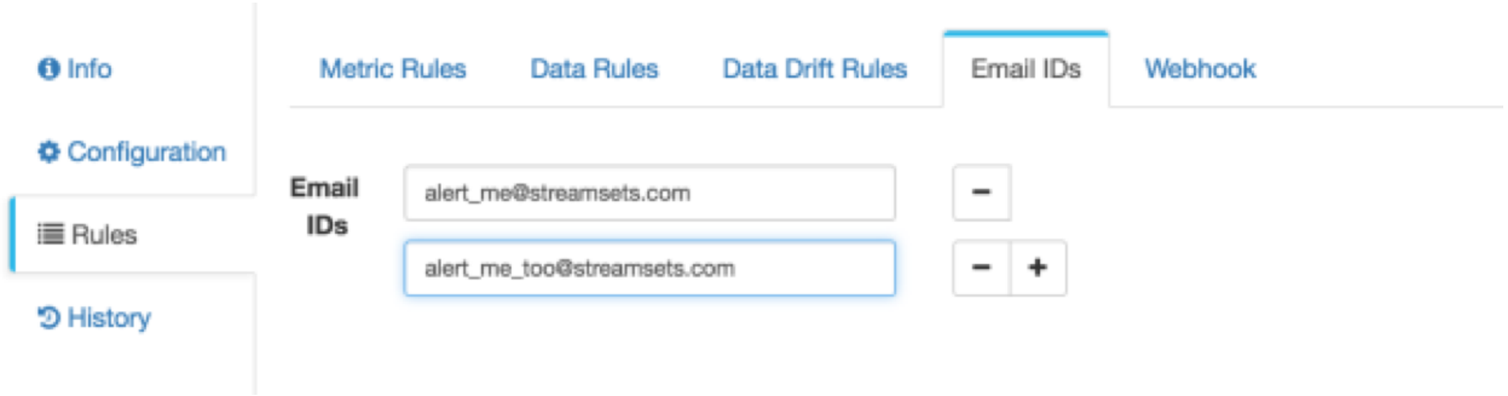
- Configure in the ‘Rules’ section of the pipeline configuration
- A set of default (but not active) rules are pre-populated




| ▶ | Alert Text | Metric ID | Condition | Send Email | Active | Actions |
|---|---|---|---|---|---|---|
| ▶ | High incidence of Error Records | Pipeline Error Records Counter | ${value() > 100} | ☐ | ☐ | Edit \| Delete |
| ▶ | High incidence of Stage Errors | Pipeline Stage Errors Counter | ${value() > 100} | ☐ | ☐ | Edit \| Delete |
| ▶ | Pipeline is Idle | Runtime Statistics Gauge | ${time:now() - value() > 120000} | ☐ | ☐ | Edit \| Delete |
| ▶ | Batch taking more time to process | Runtime Statistics Gauge | ${value() > 200} | ☐ | ☐ | Edit \| Delete |
| ▶ | Memory limit for pipeline exceeded | Pipeline Memory Consumption Counter (MB) | ${value() > (jvm:maxMemoryMB() * 0.65)} | ☐ | ☐ | Edit \| Delete |

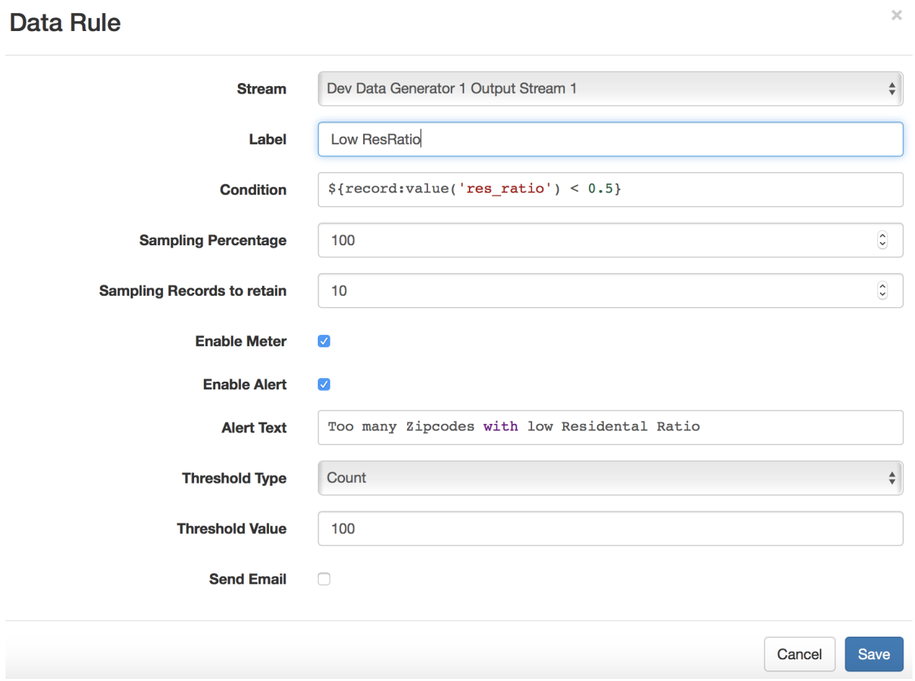
# Configuring Rules (2)

- **Alerts will be displayed in the SDC user interface**
- **Can also email on alert: configure email addresses in the 'Email IDs' tab**
  - Enable the 'send email' checkbox on the alert itself



# Data Rules

- You can define rules which relate to data passing through the pipeline
- Configured from Rules/Data Rules in the general pipeline tab



# Pipeline Events, Rules, and Alerts

- *Dataflow Triggers*
- *Metric and Data Rules*
- **Data Drift Synchronization**
- *Conclusion*

# What Is Data Drift? (1)

- **Data Drift:**
  - The unpredictable, unannounced and unending mutation of data characteristics caused by the operation, maintenance and modernization of the systems that produce the data

# What Is Data Drift? (2)

- **Data Drift:**
  - The unpredictable, unannounced and unending mutation of data characteristics caused by the operation, maintenance and modernization of the systems that produce the data
- **Erodes data fidelity**
- **Decreases operational reliability**
- **Affects productivity**
- **Increases costs**
- **Delays your time to analysis**
- **Leads to poor decisions**

# Data Drift Expression Language Functions

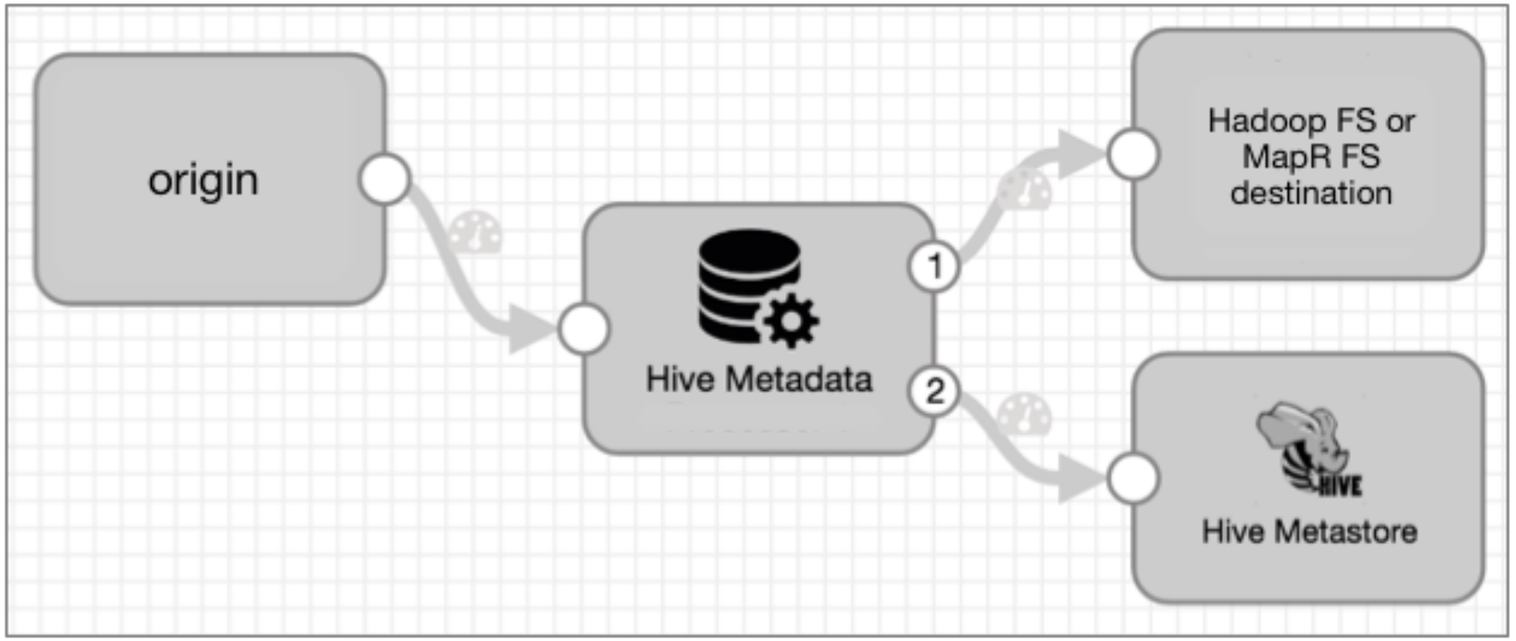
- **The StreamSets Expression Language has functions for creating data drift rules:**
  - `drift:name()` — Field name changes
  - `drift.order()` — Field order changes
  - `drift.size()` — Number of fields change
  - `drift.type()` — Field data type changes

# The Drift Synchronization Solution (1)

- **The Drift Synchronization Solution for Hive creates and updates Hive tables based on a changing data definition**
- **Uses the Hive Metadata processor**
- **Supports writing data as Avro or Parquet**
- **Hive Metadata processor detects columnar drift and the need for new tables and partitions**
  - Generates metadata records describing the required changes
  - These records are passed to the Hive Metastore destination

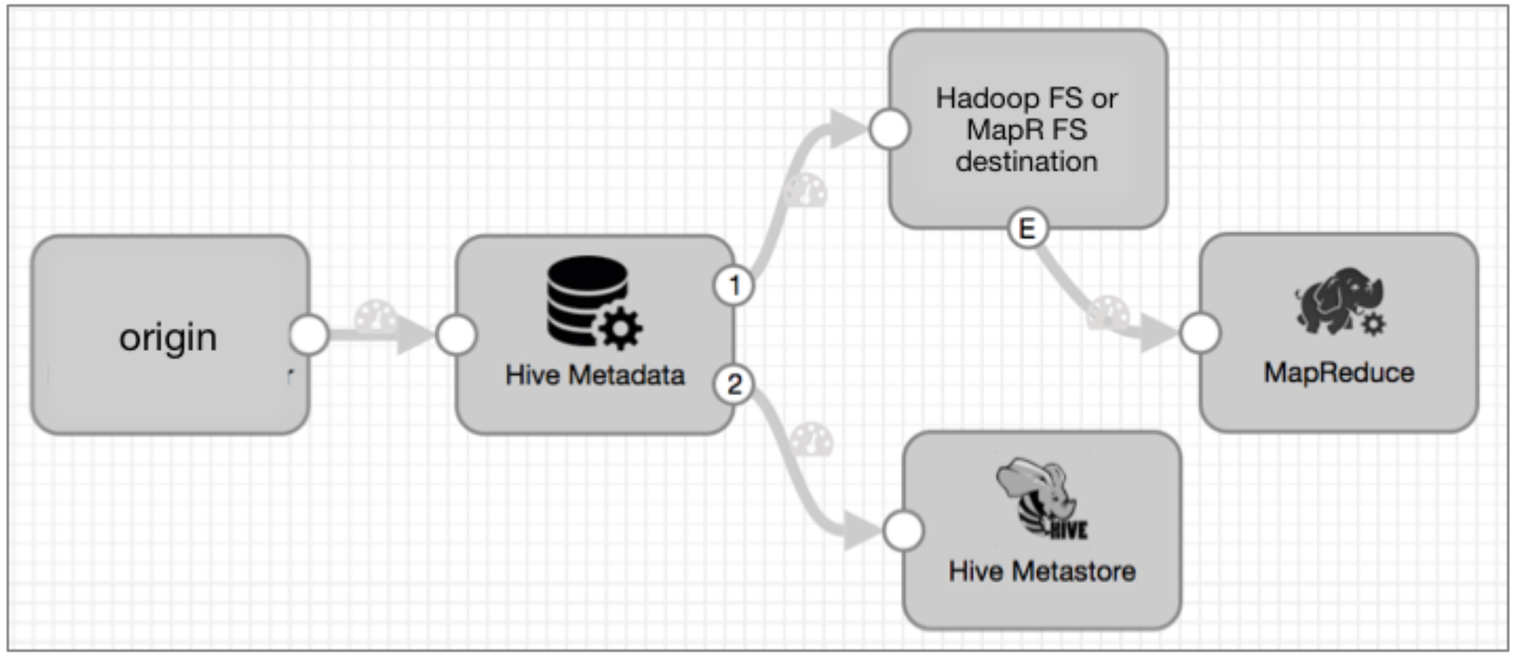
# The Drift Synchronization Solution (2)

- **Example pipeline for Avro:**



# The Drift Synchronization Solution (3)

- **Example pipeline for Parquet:**

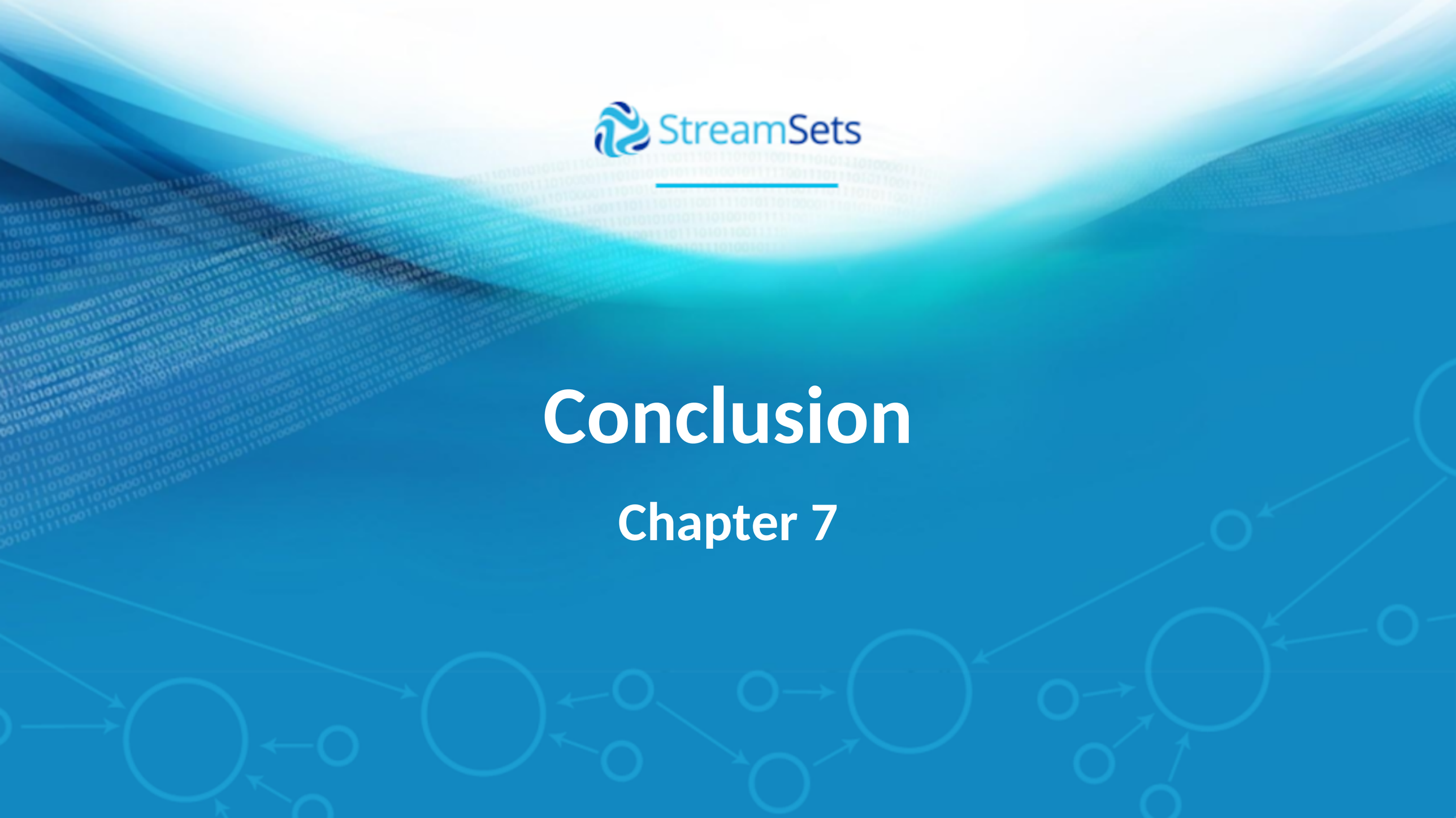


# Pipeline Events, Rules, and Alerts

- *Dataflow Triggers*
- *Metric and Data Rules*
- *Data Drift Synchronization*
- **Conclusion**

# Conclusion

- Dataflow triggers and pipeline events allow you to construct pipelines that react to specific conditions
- StreamSets Data Collector includes a built-in drift synchronization solution for data being written to Hive and Impala




# Conclusion

## Chapter 7

# Course Contents

# Conclusion

In this course, you have learned:

- **How to navigate the StreamSets Data Collector user interface**
- **How to build data pipelines**
- **How to modify data as it passes through the pipeline**
- **More advanced features of the StreamSets Data Collector**