Inspire the Next

Realtime Data Processing with Pentaho Data Integration (PDI)

Change log (if you want to use it):

Date	Version	Author	Changes
9/7/2017	0.1	Alexander Schurman	Initial document
9/21/2017	0.2	Alexander Schurman	Adding samples

Contents

Overview

This document covers some best practices on realtime data processing on big data with Pentaho Data Integration (PDI). In it, you will learn reasons to try this PDI implementation, and some of the things you should and should not do when implementing this solution.

Our intended audience is solution architects and designers, or anyone with a background in realtime ingestion, or messaging systems like Java Message Service (JMS), RabbitMQ, or WebSphere MQ.

The intention of this document is to speak about topics generally; however, these are the specific versions covered here:

Software	Version(s)
Pentaho	7.x, 8.0

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

Before beginning, use the following information to prepare for the procedures described in the main section of the document.

Prerequisites

This document assumes that you have some knowledge of messaging systems such as Kafka, RabbitMQ, Message Queue Telemetry Transport (MQTT), or JMS, as well the Hadoop Data File System (HDFS), microbatching, and Spark.

Use Cases

Use cases employed in this document include the following:

Use Case 1: Immediate Data Push to Customers

Marc wants to use a messaging system or channel that will push data to customers as soon as it is received. It will then be the customers' responsibility to collect and store or process the data.

See Use Case 1 later in the document for the solution.

Use Case 2: Event Messaging

Janice is using a messaging system like JMS or RabbitMQ to send an event message that will trigger a process of Extract/Transform/Load (ETL) alerts. In Janice's case, the data is not part of the message.

See Use Case 2 later in the document for the solution.

PDI Transformation

Successful PDI transformation of your data involves several considerations. This section contains preliminary information you will need to understand before you encounter the solutions to the use cases from the introduction of this document.

You can find information on these topics in the following sections:

- How PDI Works
- Neverending Transformations

How PDI Works

PDI is designed to work as a flow of data. In fact, it may be useful to think of PDI as a series of pipes through which water flows and is joined with other flows, mixed, and so on. Water pipes do not really know how much water is in their source, but no matter how big the source, if you keep the water flowing, eventually all the water will be processed. In a water pipe system, if any junctions or containers slow down or block the flow of water, then all the water in the pipe before that point will slow down in a chain reaction.

PDI joins, merges, and mixes data in the same way: all the data will eventually be processed as long as the data keeps flowing without blockages. The size of the "pipe" in PDI is directly linked to the number of data records, and to the amount of memory needed to hold all those records.

The key to successfully transform your data with high performance is to understand which PDI steps slow or block your flow.

Problem Steps and Situations

Some steps and situations you should pay close attention to include:

- **Sort step**: To sort rows, you first must collect all the rows. The flow of data can be blocked here until you have collected all the rows, so no data will move to the next step until all the data is received. [1]
- **Joins and stream lookup**: When a stream of data depends on another stream, pay attention to where the reference records come from, since all the reference records will be needed in memory before the data can move ahead.
- **Split and re-join on the same flow**: When you split one flow and then recombine it, make sure the flow is not slowed or stopped in one of the streams, because that will likely slow or block the other flow as well.

[1] Pentaho 8.0's microbatch strategy reduces the risk because the microbatch is executed on a different thread/executor. Therefore, it will not affect the incoming data.

Neverending Transformations

A neverending transformation is a process that starts, and then is always waiting for new data to come in. All the steps continue running, awaiting new data.

How to Implement a Neverending Transformation

In a PDI transformation, the input steps are the steps that collect and push the data records through the stream. There are currently several input steps that support constant data flow:

- **Kafka Consumer** step
- **JMS Consumer** step (which can connect to things like RabbitMQ)
- **IBM WebSphereMQ Consumer** step
- **MQTT** (available as a plugin from the Pentaho Marketplace)
- **User Defined Java Class** step (see an example at Realtime streaming data integration)

Where to Run a Neverending Transformation

Although there are several different places you can execute your neverending transformation, choose the best option for your use case based on your organization's needs, fault tolerance, resources, and available infrastructure.

Table 1: Locations

Location	Details
Adaptive Execution Layer (AEL) Spark	Pentaho 8 introduces Spark Streaming integration. The current implementation is linked to the **Kafka Consumer** and **Kafka Producer** steps only. New releases will include new input/output technologies.
Pentaho Server or Carte Server	When you execute the transformation this way, you can use the API to track the status. In case of failure, you will need to create a process to detect and handle your next steps.
Kitchen/Pan	This technique issues a specific Java Virtual Machine (JVM) for the process, with specific memory settings for the process required. It is not possible to get execution statistics unless you include the code in the transformation design, either with database logging, Simple Network Management Protocol (SNMP), or another custom implementation. In case of failure, the JVM will be terminated, and a wrapping operating system (OS) batch process will be necessary to handle your next steps.

How to Stop a Neverending Transformation

There is currently no *easy* way to stop a neverending transformation gracefully.[2] Instead, to stop a transformation, incoming records first must stop arriving, and then the transformation will wait until all the records move through the transformation pipe.

One technique to force a transformation to stop is to introduce a `SIGNAL` communication into the transformation. One of the ways you can do this is to include an independent event listener based on JMS, which will trigger an `ABORT` or `HALT` signal to the input step:

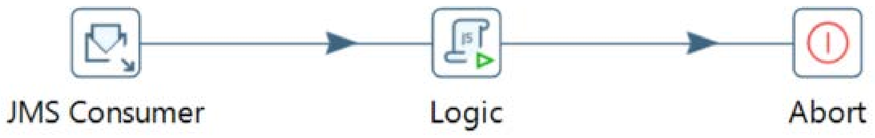


Figure 1: Abort Signal in a Transformation

When a transformation terminates, you can lose records that are in the transformation pipe. The number of potentially lost records is linked to the setting `number of steps` multiplied by the setting `Nr of rows in row-set`.[3]

Keep this lost record potential in mind as you design your transformations.

Pentaho 8.0 and Kafka Consumer have an offset setting that allows realtime processing tools to reprocess records that have been previously collected and flagged as committed/processed.

Managing Data Duplication

Data duplication and reprocessing is an important part of realtime data processing.

Handle data duplication detection and management outside of the realtime data processing, because it may require additional steps or processes that could slow down your data processing. This slowdown may cause your data processing to move even slower than the number of records per second received, causing a records traffic jam.

[2] This is generally true, although if you run your transformation on Pentaho Server or Carte Server, there *is* a button that can stop the currently running transformation.

[3] PDI's API contains internal mechanisms that you can call. See the stopRunning documentation for more information.

Realtime Data Processing with PDI

Both use cases in this document are presented here with their descriptions and solutions.

- Use Case 1: Streaming Solution
- Use Case 2: Event-Driven Solution

Use Case 1: Streaming Solution

The first use case detailed in the introduction can be addressed with a streaming solution.

Streaming-driven solutions are usually designed for a single purpose. Data can come from a broker, channel, or socket, and the architecture in this type of implementation most commonly includes Kafka or MQTT.

There are two approaches to designing this solution:

- Focus on collecting and storing the data as it comes in, and process it in microbatching.
- Process the data as it is received, transform it, and then store it.

You can find the similarity in how data is collecting from the messages/channel and pushed to the processing/storing in the Neverending Transformations section of this document.

The key to making your solutions successful is being able to cope with the required data throughput. This means you need to measure the solution execution at the data volumes and speed you require.

We recommend you test your process with at least 10-20% above your real data volume and speed requirements.

Split your data stream into two parts by following lambda architecture best practices:

- The **first stream** will be dumped directly into a data bucket such as HDFS, Amazon Simple Storage Service (S3), or other, which can be used for:
  - Batch processing
  - Model building
  - Processing asynchronous data chunks
- The **second stream** can be used for:
  - Realtime processing
  - Data decisions based on window events
  - Gathering statistics
  - Alerting

Because realtime processing may cause data duplication, gaps, missing records that arrive late, and other unique issues, you may want to reprocess in batch mode towards the end of your process so that you can rebuild your data while realtime data continues to arrive.

Pentaho 8.0 introduced stream processing capabilities. It can process data incoming from Kafka Source and create microbatching processes. The design solution works in the Pentaho engine or in Spark Streaming.

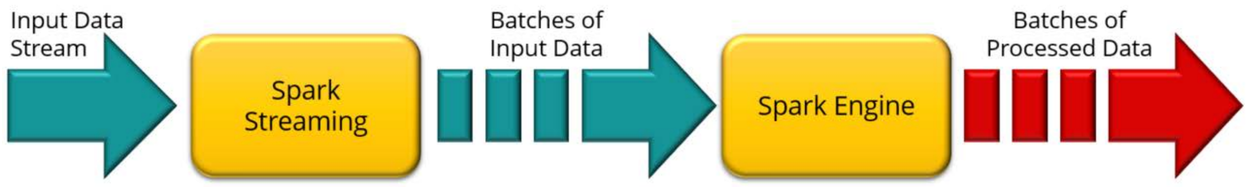


Figure 2: Pentaho Stream Processing

Setting Up Stream Processing

To set up and use stream processing:

1. Use a **Kafka Consumer** step to continuously listen to Kafka topics for messages.
2. Enable long-running stream data processing by setting parameters in the Kafka Consumer step for the size of your configurable microbatch of data.
3. Use a **Get Records from Stream** step to process a microbatch of a continuous stream of records.
4. Process and blend retrieved messages using other steps running in the Kettle engine or in AEL Spark.
5. When you use AEL Spark, use Spark Streaming to microbatch the streaming data.
6. Publish to Kafka from the Kettle engines using a **Kafka Producer** step, or in parallel from AEL Spark.

Here is an example of how the messages move and how that relates to the movement of your data:

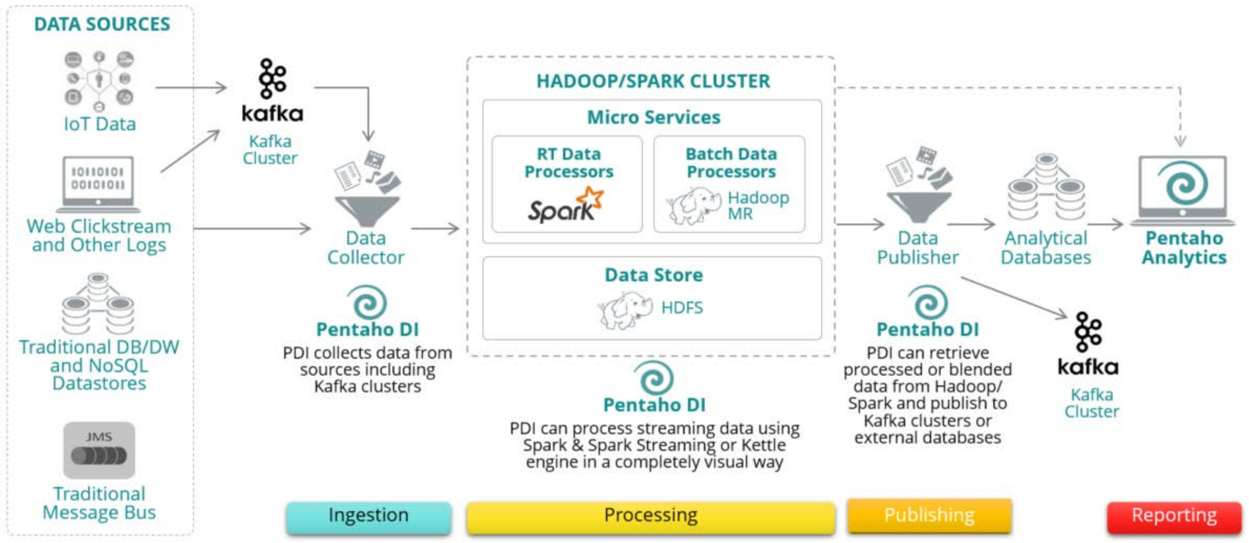
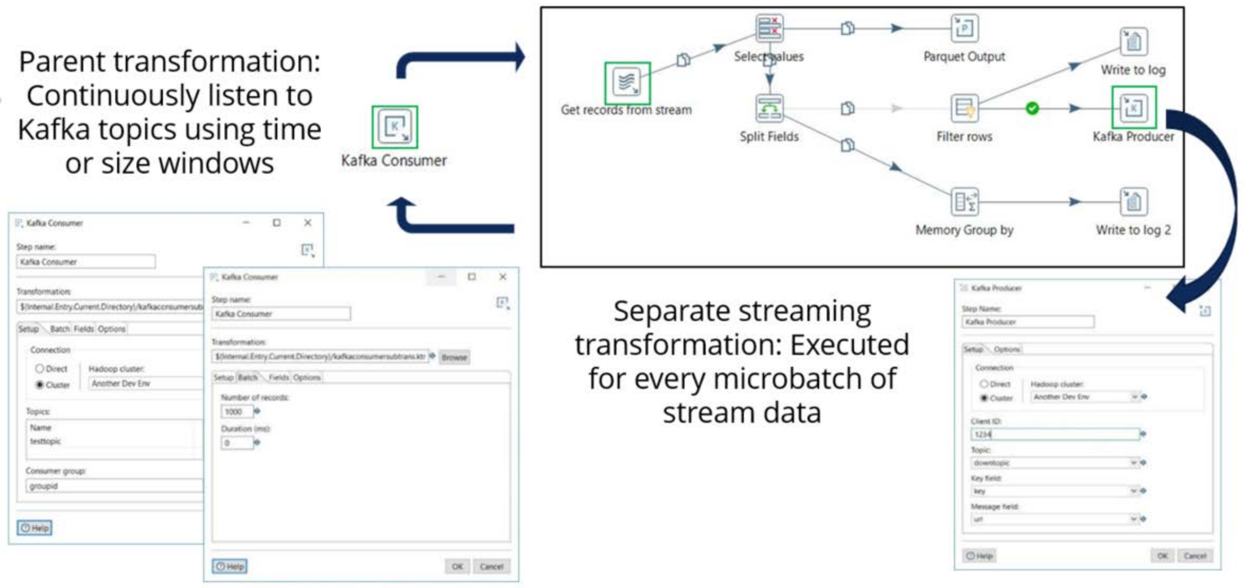

DATA SOURCES
IoT Data
Web Clickstream and Other Logs
Traditional DB/DW and NoSQL Datastores
JMS
Traditional Message Bus
kafka
Kafka Cluster
Data Collector
Pentaho DI
PDI collects data from sources including Kafka clusters
HADOOP/SPARK CLUSTER
Micro Services
RT Data Processors
Spark
Batch Data Processors
Hadoop MR
Data Store
HDFS
Pentaho DI
PDI can process streaming data using Spark & Spark Streaming or Kettle engine in a completely visual way
Data Publisher
Pentaho DI
PDI can retrieve processed or blended data from Hadoop/ Spark and publish to Kafka clusters or external databases
Analytical Databases
Pentaho Analytics
kafka
Kafka Cluster
Ingestion
Processing
Publishing
Reporting

Figure 3: Stream Processing Data Flow

An illustration of the potential PDI transformation you could use:


Parent transformation: Continuously listen to Kafka topics using time or size windows
Kafka Consumer
Get records from stream
Select values
Parquet Output
Write to log
Split Fields
Filter rows
Kafka Producer
Memory Group by
Write to log 2
Separate streaming transformation: Executed for every microbatch of stream data
Help
OK
Cancel
Browse

Kafka Consumer
Step name: Kafka Consumer
Transformation: ${Internal.Entry.Current.Directory}/kafkaconsumersubtrans.ktr
Setup | Batch | Fields | Options
Connection: Direct [✓ Cluster]
Hadoop cluster: Another Dev Env
Topics: Name: testtopic
Consumer group: groupid
Number of records: 1000
Duration (ms): 0

Kafka Producer
Step Name: Kafka Producer
Setup | Options
Connection: Direct [✓ Cluster]
Hadoop cluster: Another Dev Env
Client ID: 1234
Topic: downtopic
Key field: key
Message field: url

Figure 4: PDI Transformation for Streaming

In the above illustration:

- **Kafka Consumer** is where you collect the raw data with all its settings in real time. This step calls a subtransformation which will represent the microbatching processing.
- **Get Records from Stream** is the input step for the subtransformation and has information like key, message, partition, offset, and others.
- The remaining steps are the specific processes that need to be done: writing to HDFS in text, Avro or Parquet; and parsing and creating new messages to be pushed through **Kafka Producer** to Kafka on a different topic.

It is very important to have error handling on any of the steps that could cause a message parsing fatal error.

Learn more about Pentaho 8.0 streaming at [Kafka and Streaming Ingestion on PDI](#).

Example 2: Event-Driven Solution

The second use case detailed in the introduction can be addressed with an event-driven solution.

Event-driven solutions listen to a queue or queues for messages, either for a specific purpose or for a purpose defined within the message itself. Technologies like JMS or RabbitMQ act as message brokers, allowing you to collect the event and fire the necessary processes linked to the trigger event.

If you decide to use an event-driven process, you can use PDI or a third-party tool to push the filenames to the queue. Create the queue consumer with PDI based on a neverending transformation, or a third-party consumer. Any of the message consumers can kick a PDI transformation to begin processing the files received in the messages.

This diagram shows data moving from the cloud to your event handler of choice, and from there to your process task. Setting the number of rows in the rowset very low will reduce the number of messages that are consumed from the queue and still awaiting processing.

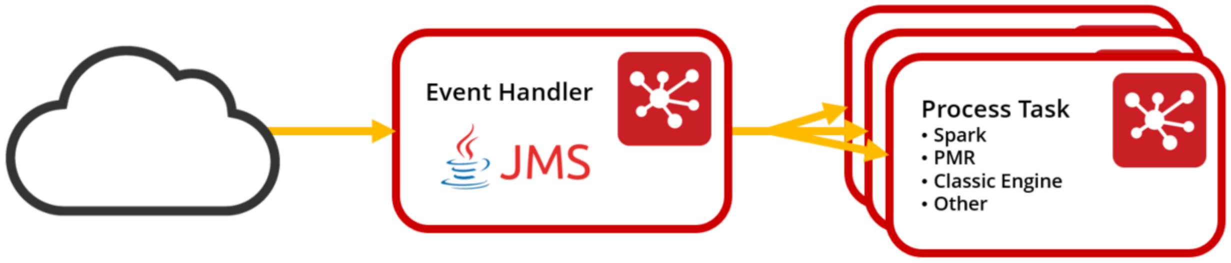


Figure 5: Event-Driven Solution Data Flow

In the next diagram, you will see what you can do with your message afterward:

1. First, the realtime event listener (Kafka, JMS, RabbitMQ, or whatever you choose) collects the message and pushes it to the next step.
2. Next, the **Process Event** step represents whatever steps you use to validate the message and action, including parsing, splitting, and other tasks. Again, keep the rows processing low to avoid slowdown.
3. Then, use **Job Executor** to start a fire and forget job. That job will process your message and then either run a transformation or a job afterward. It will be executed for every row.
4. If you choose to finish with a transformation to call the event processing, make sure the option **Wait for remote transformation to finish** is *not* checked. You can run the transformation on a Carte server.
5. If you choose to finish with a job, make sure the option **Wait for remote job to finish** is *not* checked. You can then:
    a. Run a Spark execution (AEL or Submit)
    b. Run on a remote Carte server
    c. Run PMR
    d. Run on an external shell (a Pentaho process or another external tool)

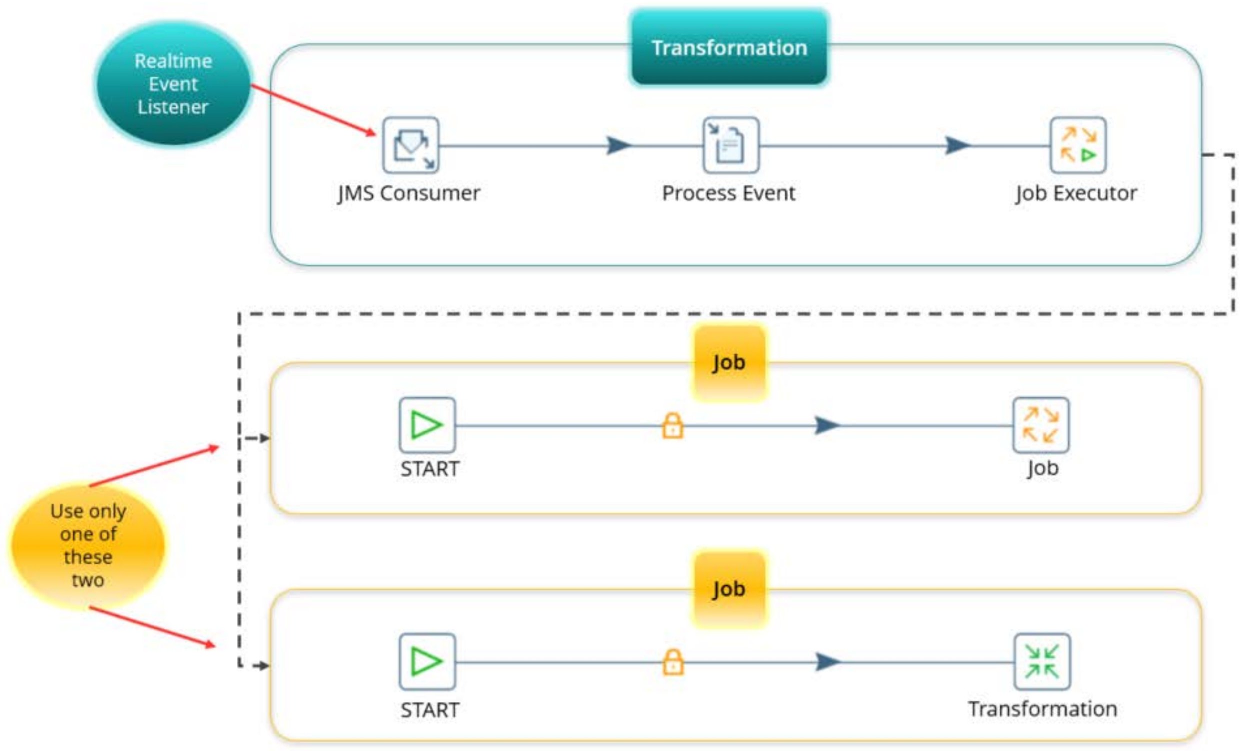


Figure 6: PDI Transformation

Whether you choose a job or transformation at the end, we recommend you run it asynchronously on a separate OS process.

A processing job can call PMR, Spark, a shell execution for tasks external to Pentaho, or other processes, but what it calls will need to run unlinked to the main process for proper scaling.

This example has multiple possible scale factors, depending on the task processor type you selected.

Related Information

Here are some links to information that you may find helpful while using this best practices document:

- Apache
    - Kafka
    - Spark Streaming
- JMS
- Lambda Architecture
- MQTT
- Pentaho
    - Checking Status Using Kettle
    - Integrate Pentaho with Third-Party Monitoring Through SNMP
    - Kafka and Streaming Ingestion on PDI
    - Pentaho 8 Streaming
    - Pentaho Components Reference
    - Pentaho Marketplace
    - Pentaho and MQTT
    - Pentaho Performance Tuning
    - stopRunning
- Realtime Streaming Data Aggregation (Matt Casters' blog prior to Pentaho 8.0)
- RabbitMQ

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed. (Compose specific questions about the topics in the document and put them in the table.)

Name of the Project:____

Date of the Review:____

Name of the Reviewer:____

Item	Response	Comments
Did you check for problem steps that might slow down your data flow?	YES_______ NO_______	
Did you set up a neverending transformation using the best practices explained in this document?	YES_______ NO_______	
Did you create an event-driven solution?	YES_______ NO_______	
Did you create a data-driven solution?	YES_______ NO_______	