Hortonworks DataFlow

Overview

(February 28, 2018)

docs.cloudera.com

Hortonworks DataFlow: Overview

Table of Contents

List of Figures

List of Tables

1. Overview

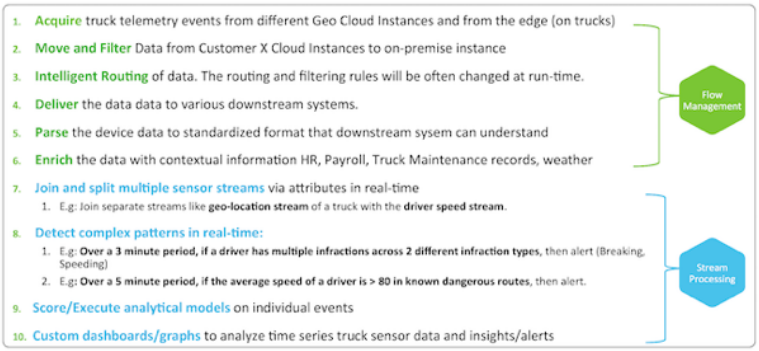
To build real-world data in motion apps such as those based on the Internet of Things (IoT), you need both flow management and stream processing capabilities. What is the difference between the two?

Data in motion apps typically have the following key requirements:

- **Acquisition of Data** from data sources within the data center, and across cloud environments and edge devices.
- **Moving and Filtering of Data** from edge devices (such as telematic panels on trucks), and across cloud environments and core data centers.
- **Intelligent and Dynamic Routing of Data** across regional data centers to core processing data centers.
- **Delivering Data** to different downstream systems.
- **Joining and Splitting Streams of Data** as they move.
- **Detecting complex patterns** in the streams of data.
- **Scoring/Executing Analytics Models** within the stream.
- **Creating Custom Dashboards** to visualize and analyze the streams and insights.

To explain how flow management and stream processing relate to these requirements, we employ a fictitious use case for trucking company X, which installed sensors on its fleet of trucks. These sensors emit streams of event data such as speed, braking frequency, and geo-code location. In this use case, the trucking company is building an IoT trucking app that monitors trucks in real time.

The following diagram illustrates how each of these requirements would be implemented in the context of stream processing and flow management:



As part of the stream processing suite available in HDF, Streaming Analytics Manager provides capabilities for implementing the requirements outlined in blue in the previous diagram.

To summarize, Streaming Analytics Manager provides the following core capabilities:

- Building stream apps, using the following primitives:
  - Connecting to streams
  - Joining streams
  - Forking streams
  - Aggregating over windows
  - Extensibility: adding custom processors and user-defined-functions (UDFs)
  - Stream analytics: descriptive, predictive, and prescriptive
  - Rules engine
  - Transformations
  - Filtering and routing
  - Notifications and alerts
- Deploying stream apps:
  - Deploying the stream app on a supported streaming engine:
  - Monitoring the stream app with application-specific metrics.
- Exploring and analyzing streaming data; discovering insights:
  - Creating dashboards of streaming data
  - Exploring streaming data
  - Creating streaming cubes

2. Apache NiFi Overview

2.1. Apache NiFi Overview

Table of Contents

What is Apache NiFi?

Put simply NiFi was built to automate the flow of data between systems. While the term 'dataflow' is used in a variety of contexts, we use it here to mean the automated and managed flow of information between systems. This problem space has been around ever since enterprises had more than one system, where some of the systems created data and some of the systems consumed data. The problems and solution patterns that emerged have been discussed and articulated extensively. A comprehensive and readily consumed form is found in the *Enterprise Integration Patterns*.

Some of the high-level challenges of dataflow include:

Systems fail	Networks fail, disks fail, software crashes, people make mistakes.
Data access exceeds capacity to consume	Sometimes a given data source can outpace some part of the processing or delivery chain - it only takes one weak-link to have an issue.
Boundary conditions are mere suggestions	You will invariably get data that is too big, too small, too fast, too slow, corrupt, wrong, or in the wrong format.
What is noise one day becomes signal the next	Priorities of an organization change - rapidly. Enabling new flows and changing existing ones must be fast.
Systems evolve at different rates	The protocols and formats used by a given system can change anytime and often irrespective of the systems around them. Dataflow exists to connect what is essentially a massively distributed system of components that are loosely or not-at-all designed to work together.

Compliance and security	Laws, regulations, and policies change. Business to business agreements change. System to system and system to user interactions must be secure, trusted, accountable.
Continuous improvement occurs in production	It is often not possible to come even close to replicating production environments in the lab.

Over the years dataflow has been one of those necessary evils in an architecture. Now though there are a number of active and rapidly evolving movements making dataflow a lot more interesting and a lot more vital to the success of a given enterprise. These include things like; Service Oriented Architecture, the rise of the API, Internet of Things, and Big Data. In addition, the level of rigor necessary for compliance, privacy, and security is constantly on the rise. Even still with all of these new concepts coming about, the patterns and needs of dataflow are still largely the same. The primary differences then are the scope of complexity, the rate of change necessary to adapt, and that at scale the edge case becomes common occurrence. NiFi is built to help tackle these modern dataflow challenges.

2.1.1. The core concepts of NiFi

NiFi's fundamental design concepts closely relate to the main ideas of Flow Based Programming. Here are some of the main NiFi concepts and how they map to FBP:

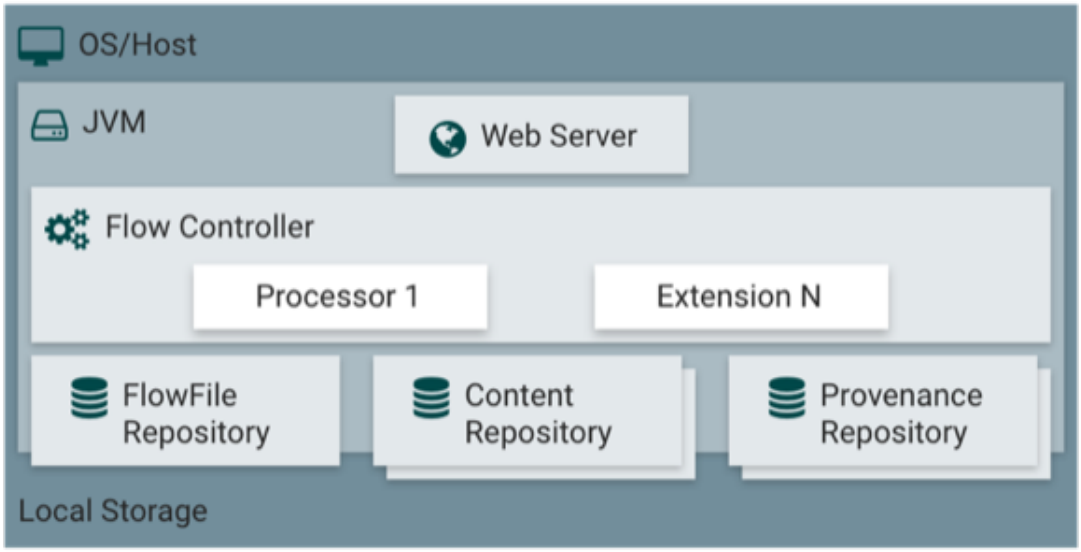
NiFi Term	FBP Term	Description
FlowFile	Information Packet	A FlowFile represents each object moving through the system and for each one, NiFi keeps track of a map of key/value pair attribute strings and its associated content of zero or more bytes.
FlowFile Processor	Black Box	Processors actually perform the work. In terms a processor is doing some combination of data routing, transformation, or mediation between systems. Processors have access to attributes of a given FlowFile and its content stream. Processors can operate on zero or more FlowFiles in a given unit of work and either commit that work or rollback.
Connection	Bounded Buffer	Connections provide the actual linkage between processors. These act as queues and allow various processes to interact at differing rates. These queues can be prioritized dynamically and can have upper bounds on load, which enable back pressure.
Flow Controller	Scheduler	The Flow Controller maintains the knowledge of how processes connect and manages the threads and allocations thereof which all processes use. The Flow Controller acts as the broker facilitating the exchange of FlowFiles between processors.
Process Group	subnet	A Process Group is a specific set of processes and their connections, which can receive data via input ports and send data out via output ports. In this

NiFi Term	FBP Term	Description
		manner, process groups allow creation of entirely new components simply by composition of other components.

This design model, also similar to, provides many beneficial consequences that help NiFi to be a very effective platform for building powerful and scalable dataflows. A few of these benefits include:

- Lends well to visual creation and management of directed graphs of processors
- Is inherently asynchronous which allows for very high throughput and natural buffering even as processing and flow rates fluctuate
- Provides a highly concurrent model without a developer having to worry about the typical complexities of concurrency
- Promotes the development of cohesive and loosely coupled components which can then be reused in other contexts and promotes testable units
- The resource constrained connections make critical functions such as back-pressure and pressure release very natural and intuitive
- Error handling becomes as natural as the happy-path rather than a coarse grained catch-all
- The points at which data enters and exits the system as well as how it flows through are well understood and easily tracked

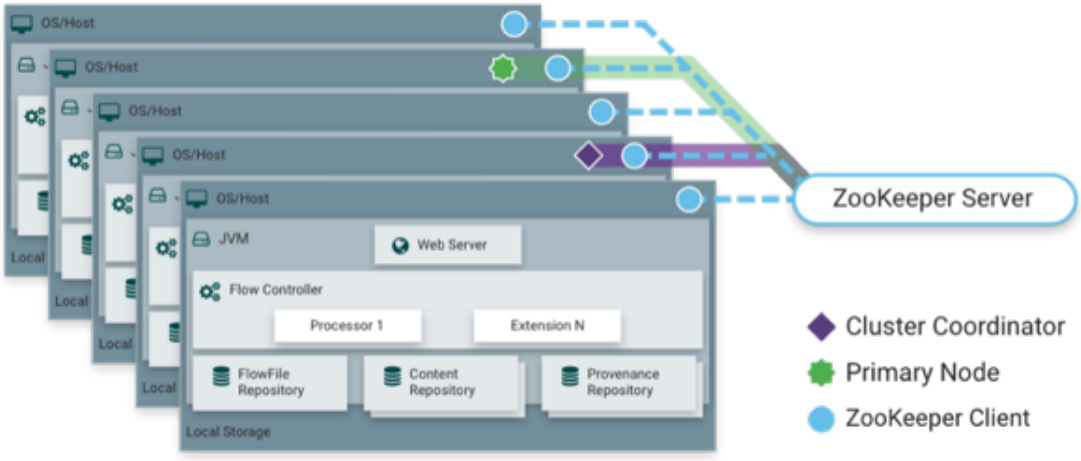
2.1.2. NiFi Architecture



NiFi executes within a JVM on a host operating system. The primary components of NiFi on the JVM are as follows:

Web Server	The purpose of the web server is to host NiFi's HTTP-based command and control API.
Flow Controller	The flow controller is the brains of the operation. It provides threads for extensions to run on, and manages the schedule of when extensions receive resources to execute.
Extensions	There are various types of NiFi extensions which are described in other documents. The key point here is that extensions operate and execute within the JVM.
FlowFile Repository	The FlowFile Repository is where NiFi keeps track of the state of what it knows about a given FlowFile that is presently active in the flow. The implementation of the repository is pluggable. The default approach is a persistent Write-Ahead Log located on a specified disk partition.
Content Repository	The Content Repository is where the actual content bytes of a given FlowFile live. The implementation of the repository is pluggable. The default approach is a fairly simple mechanism, which stores blocks of data in the file system. More than one file system storage location can be specified so as to get different physical partitions engaged to reduce contention on any single volume.
Provenance Repository	The Provenance Repository is where all provenance event data is stored. The repository construct is pluggable with the default implementation being to use one or more physical disk volumes. Within each location event data is indexed and searchable.

NiFi is also able to operate within a cluster.



Starting with the NiFi 1.0 release, a Zero-Master Clustering paradigm is employed. Each node in a NiFi cluster performs the same tasks on the data, but each operates on a different set of data. Apache ZooKeeper elects a single node as the Cluster Coordinator,

and failover is handled automatically by ZooKeeper. All cluster nodes report heartbeat and status information to the Cluster Coordinator. The Cluster Coordinator is responsible for disconnecting and connecting nodes. Additionally, every cluster has one Primary Node, also elected by ZooKeeper. As a DataFlow manager, you can interact with the NiFi cluster through the user interface (UI) of any node. Any change you make is replicated to all nodes in the cluster, allowing for multiple entry points.

2.1.3. Performance Expectations and Characteristics of NiFi

NiFi is designed to fully leverage the capabilities of the underlying host system on which it is operating. This maximization of resources is particularly strong with regard to CPU and disk. For additional details, see the best practices and configuration tips in the Administration Guide.

For IO — The throughput or latency one can expect to see varies greatly, depending on how the system is configured. Given that there are pluggable approaches to most of the major NiFi subsystems, performance depends on the implementation. But, for something concrete and broadly applicable, consider the out-of-the-box default implementations. These are all persistent with guaranteed delivery and do so using local disk. So being conservative, assume roughly 50 MB per second read/write rate on modest disks or RAID volumes within a typical server. NiFi for a large class of dataflows then should be able to efficiently reach 100 MB per second or more of throughput. That is because linear growth is expected for each physical partition and content repository added to NiFi. This will bottleneck at some point on the FlowFile repository and provenance repository. We plan to provide a benchmarking and performance test template to include in the build, which allows users to easily test their system and to identify where bottlenecks are, and at which point they might become a factor. This template should also make it easy for system administrators to make changes and to verify the impact.

For CPU — The Flow Controller acts as the engine dictating when a particular processor is given a thread to execute. Processors are written to return the thread as soon as they are done executing a task. The Flow Controller can be given a configuration value indicating available threads for the various thread pools it maintains. The ideal number of threads to use depends on the host system resources in terms of numbers of cores, whether that system is running other services as well, and the nature of the processing in the flow. For typical IO-heavy flows, it is reasonable to make many dozens of threads to be available.

For RAM — NiFi lives within the JVM and is thus limited to the memory space it is afforded by the JVM. JVM garbage collection becomes a very important factor to both restricting the total practical heap size, as well as optimizing how well the application runs over time. NiFi jobs can be I/O intensive when reading the same content regularly. Configure a large enough disk to optimize performance.

2.1.4. High Level Overview of Key NiFi Features

This sections provides a 20,000 foot view of NiFi's cornerstone fundamentals, so that you can understand the Apache NiFi big picture, and some of its the most interesting features.

The key features categories include flow management, ease of use, security, extensible architecture, and flexible scaling model.

Flow Management	Guaranteed Delivery	A core philosophy of NiFi has been that even at very high scale, guaranteed delivery is a must. This is achieved through effective use of a purpose-built persistent write-ahead log and content repository. Together they are designed in such a way as to allow for very high transaction rates, effective load-spreading, copy-on-write, and play to the strengths of traditional disk read/ writes.
	Data Buffering w/ Back Pressure and Pressure Release	NiFi supports buffering of all queued data as well as the ability to provide back pressure as those queues reach specified limits or to age off data as it reaches a specified age (its value has perished).
	Prioritized Queuing	NiFi allows the setting of one or more prioritization schemes for how data is retrieved from a queue. The default is oldest first, but there are times when data should be pulled newest first, largest first, or some other custom scheme.
	Flow Specific QoS (latency v throughput, loss tolerance, etc.)	There are points of a dataflow where

		the data is absolutely critical and it is loss intolerant. There are also times when it must be processed and delivered within seconds to be of any value. NiFi enables the fine-grained flow specific configuration of these concerns.
Ease of Use	Visual Command and Control	Dataflows can become quite complex. Being able to visualize those flows and express them visually can help greatly to reduce that complexity and to identify areas that need to be simplified. NiFi enables not only the visual establishment of dataflows but it does so in real-time. Rather than being 'design and deploy' it is much more like molding clay. If you make a change to the dataflow that change immediately takes effect. Changes are fine-grained and isolated to the affected components. You don't need to stop an entire flow or set of flows just to make some specific modification.
	Flow Templates	Dataflows tend to be highly pattern oriented and while there are often many different ways to solve a problem, it helps greatly to

		be able to share those best practices. Templates allow subject matter experts to build and publish their flow designs and for others to benefit and collaborate on them.
	Data Provenance	NiFi automatically records, indexes, and makes available provenance data as objects flow through the system even across fan-in, fan-out, transformations, and more. This information becomes extremely critical in supporting compliance, troubleshooting, optimization, and other scenarios.
	Recovery / Recording a rolling buffer of fine-grained history	NiFi's content repository is designed to act as a rolling buffer of history. Data is removed only as it ages off the content repository or as space is needed. This combined with the data provenance capability makes for an incredibly useful basis to enable click-to-content, download of content, and replay, all at a specific point in an object's lifecycle which can even span generations.
Security	System to System	A dataflow is only as good as it is secure. NiFi at every point in a dataflow offers secure

		exchange through the use of protocols with encryption such as 2-way SSL. In addition NiFi enables the flow to encrypt and decrypt content and use shared-keys or other mechanisms on either side of the sender/recipient equation.
	User to System	NiFi enables 2-Way SSL authentication and provides pluggable authorization so that it can properly control a user's access and at particular levels (read-only, dataflow manager, admin). If a user enters a sensitive property like a password into the flow, it is immediately encrypted server side and never again exposed on the client side even in its encrypted form.
	Multi-tenant Authorization	The authority level of a given dataflow applies to each component, allowing the admin user to have fine grained level of access control. This means each NiFi cluster is capable of handling the requirements of one or more organizations. Compared to isolated topologies, multi-tenant authorization enables a self-service model for dataflow

		management, allowing each team or organization to manage flows with a full awareness of the rest of the flow, to which they do not have access.
Extensible Architecture	Extension	NiFi is at its core built for extension and as such it is a platform on which dataflow processes can execute and interact in a predictable and repeatable manner. Points of extension include: processors, Controller Services, Reporting Tasks, Prioritizers, and Customer User Interfaces.
	Classloader Isolation	For any component-based system, dependency problems can quickly occur. NiFi addresses this by providing a custom class loader model, ensuring that each extension bundle is exposed to a very limited set of dependencies. As a result, extensions can be built with little concern for whether they might conflict with another extension. The concept of these extension bundles is called 'NiFi Archives' and is discussed in greater detail in the Developer's Guide.
	Site-to-Site Communication Protocol	The preferred communication

		protocol between NiFi instances is the NiFi Site-to-Site (S2S) Protocol. S2S makes it easy to transfer data from one NiFi instance to another easily, efficiently, and securely. NiFi client libraries can be easily built and bundled into other applications or devices to communicate back to NiFi via S2S. Both the socket based protocol and HTTP(S) protocol are supported in S2S as the underlying transport protocol, making it possible to embed a proxy server into the S2S communication.
Flexible Scaling Model	Scale-out (Clustering)	NiFi is designed to scale-out through the use of clustering many nodes together as described above. If a single node is provisioned and configured to handle hundreds of MB per second, then a modest cluster could be configured to handle GB per second. This then brings about interesting challenges of load balancing and fail-over between NiFi and the systems from which it gets data. Use of asynchronous queuing based protocols like messaging services, Kafka, etc., can help. Use of NiFi's 'site-to-site' feature is also very effective as it is a

		protocol that allows NiFi and a client (including another NiFi cluster) to talk to each other, share information about loading, and to exchange data on specific authorized ports.
	Scale-up & down	NiFi is also designed to scale-up and down in a very flexible manner. In terms of increasing throughput from the standpoint of the NiFi framework, it is possible to increase the number of concurrent tasks on the processor under the Scheduling tab when configuring. This allows more processes to execute simultaneously, providing greater throughput. On the other side of the spectrum, you can perfectly scale NiFi down to be suitable to run on edge devices where a small footprint is desired due to limited hardware resources. To specifically solve the first mile data collection challenge and edge use cases, you can find more details here: https://cwiki.apache.org/confluence/display/NIFI/MiNiFi regarding a child project effort of Apache NiFi, MiNiFi (pronounced "minify", [min-uh-fahy]).

2.1.5. References

- [eip] Gregor Hohpe. Enterprise Integration Patterns [online]. Retrieved: 27 Dec 2014, from: http://www.enterpriseintegrationpatterns.com/

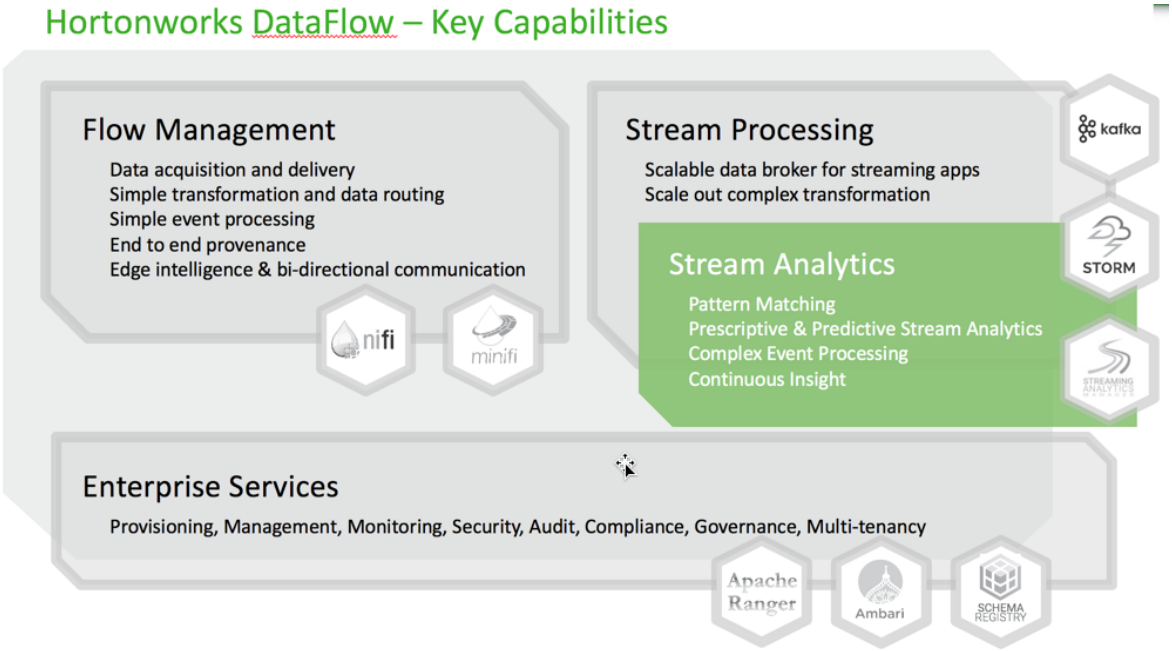
- [soa] Wikipedia. Service Oriented Architecture [online]. Retrieved: 27 Dec 2014, from: http://en.wikipedia.org/wiki/Service-oriented_architecture
- [api] Eric Savitz. Welcome to the API Economy [online]. Forbes.com. Retrieved: 27 Dec 2014, from: http://www.forbes.com/sites/ciocentral/2012/08/29/welcome-to-the-api-economy/
- [api2] Adam Duvander. The rise of the API economy and consumer-led ecosystems [online]. thenextweb.com. Retrieved: 27 Dec 2014, from: http://thenextweb.com/dd/2014/03/28/api-economy/
- [iot] Wikipedia. Internet of Things [online]. Retrieved: 27 Dec 2014, from: http://en.wikipedia.org/wiki/Internet_of_Things
- [bigdata] Wikipedia. Big Data [online]. Retrieved: 27 Dec 2014, from: http://en.wikipedia.org/wiki/Big_data
- [fbp] Wikipedia. Flow Based Programming [online]. Retrieved: 28 Dec 2014, from: http://en.wikipedia.org/wiki/Flow-based_programming#Concepts
- [seda] Matt Welsh. Harvard. SEDA: An Architecture for Highly Concurrent Server Applications [online]. Retrieved: 28 Dec 2014, from: https://sourceforge.net/projects/seda

Last updated 2018-01-09 02:06:38 -07:00

3. Streaming Analytics Manager Overview

The Hortonworks DataFlow Platform (HDF) provides flow management, stream processing, and enterprise services for collecting, curating, analyzing and acting on data in motion across on-premise data centers and cloud environments.

As the following diagram illustrates, Hortonworks Streaming Analytics Manager (SAM) is an application within the stream processing suite of the HDF platform:



Use Streaming Analytics Manager to design, develop, deploy and manage streaming analytics apps with a drag-and-drop visualization paradigm. Streaming Analytics Manager allows you to build streaming analytics apps for event correlation, context enrichment, complex pattern matching, and analytic aggregations. You can create alerts and notifications when insights are discovered.

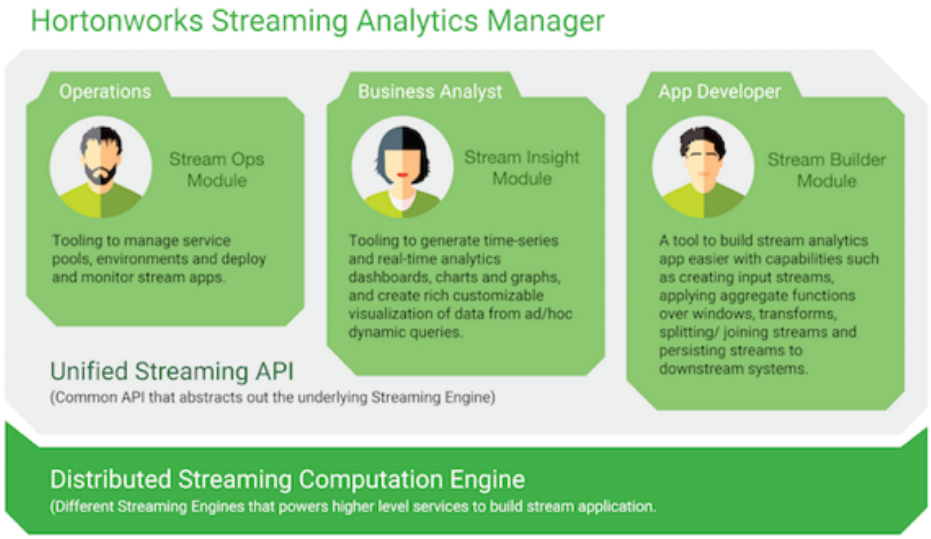
Streaming Analytics Manager is agnostic to the underlying streaming engine, and it can support multiple streaming substrates such as Storm, Spark Streaming, Flink, etc. The first streaming engine fully supported is Apache Storm.

This overview chapter describes fundamental concepts related to Streaming Analytics Manager:

- Streaming Analytics Manager Modules [17]
- Streaming Analytics Manager Taxonomy [17]
- Streaming Analytics Manager Personas [18]

3.1. Streaming Analytics Manager Modules

Streaming Analytics Manager is composed of several different modules that cater to different user personas. The following diagram illustrates Streaming Analytics Manager modules



You can think of Streaming Analytics Manager as an application that operates on top of a streaming engine (a "gray box") that allows you build stream apps faster on top of your selected streaming substrate. The unified streaming API provides the abstraction that allows you to plug in different streaming engines.

3.2. Streaming Analytics Manager Taxonomy

The following table describes the taxonomy for Streaming Analytics Manager. This taxonomy will be used throughout the rest of this guide.

Term	Description
Streaming Analytics Manager	The name of the graphical application for building, deploying, and managing stream apps.
Stream App	A streaming application built using Streaming Analytics Manager.
My Applications	The landing page for the Streaming Analytics Manager application. The Dashboard has a list of stream apps.
App Tile	Located on the dashboard, an app tile provides metrics, status and lifecyle actions for a stream. Each stream app is displayed as an app tile.
Stream Builder	The Streaming Analytics Manager tool that is used to build stream apps.
Builder Canvas	The canvas of the Stream Builder, on which stream apps are built. The canvas includes a palette of Builder components that can be used to build a stream app.
Builder Components	Building blocks available on the Builder Canvas palette, which can be used to build stream apps. There are four types of Builder components:

Term	Description
	• Source Builder Component: for creating streams from data sources such as Kafka topics or HDFS files. • Processor Builder Component: for manipulating and processing events in a stream, such as routing, applying transformations, performing windowing operations, and applying rules. • Sink Builder Component: for sending events to other systems such as HBase, HDFS, and Kafka. • Custom Builder Component: for creating custom requirements and adding them to the canvas palette.
Tile component	A component tile that has been moved onto the Builder Canvas, configurable for use in a specific stream app.
Connectors	Define connections between component tiles, directing a flow of tuples and how they flow (such as shuffle grouping).
Stream Operation	A view showing a running stream app, providing metrics for the app. After you use Stream Builder to build and deploy a stream app, the Stream Operation view allows you to monitor the running app.
Service Pool	A pool of services that can be used to create different environments. Services can come from two sources: • Ambari-managed cluster: if you specify an Ambari URL, a service pool is populated with all of the services managed by that Ambari Instance; for example, Storm, HDFS, HBase, and Kafka. • Custom service pool: for services not managed by Ambari, you can create a custom service and add that to a pool. Examples include Elastic Service and the Schema Registry Service.
Environment	A set of services you choose from one or more service pools. The environment is then associated with a stream app, which uses those services in that environment for various configurations.
Stream Insight Superset	The name of the Stream Insight module within SAM for Business Analysts to create dashboards and visualizations
Insight Data Source	A analytics cube powered by Druid where events can be streamed into for rollups/aggregations/analytics
Insight Slice	An insight visualization that can be created from a cube. An insight can be added to a dashboard
Insight Dashboard	Consists of a set of insight slices. Dashboards are created by the Business analyst in the Stream Insights Superset module.

3.3. Streaming Analytics Manager Personas

Four main modules within Streaming Analytics Manager offer services to different personas in an organization:

User Persona	Module	Module Features and Functionality
IT Engineer, Operations Engineer, Platform Engineer, Platform Operator	Stream Management	• Create service pools and environments. • Provision, manage and monitor stream apps.

User Persona	Module	Module Features and Functionality
		• Scale out or scale in stream apps based on resource consumption.
Application Developer	Stream Builder	• The Stream Builder tool assists in building analytic-focused stream apps. • The tool creates streams for event correlation, context enrichment, complex pattern matching, and aggregation. It can create alerts and notifications when patterns are detected and insights are discovered. • The interface uses a drag-and-drop visual programming paradigm.
Business Analyst, Data Analyst	Stream Insight Superset	• The Stream Insight tool assists in generating time-series and real-time analytics dashboards, charts, and graphs of metrics, alerts and notifications. • The tool provides interactive, ad-hoc analytics. You can issue ad-hoc queries, perform multidimensional analyses, and visualize the results in rich configurable dashboards. • The tool offers a self-service ability to create alerts and notification dashboards based on insights derived from the real-time streaming data flows.
SDK Developer	Unified Streaming API	• The unified streaming API abstracts out the underlying streaming engine, making it more straightforward to implement custom components. Initial support is for Storm.

The following subsections describe responsibilities for each persona. For additional information, see the following chapters in this guide:

Persona	Chapter Reference
IT Engineer, Operations Engineer, Platform Engineer, Platform Operator	Installing and Configuring Streaming Analytics Manager Managing Stream Apps
Application Developer	Running the Sample App Building an End-to-End Stream App
Business Analyst, Data Analyst	Creating Visualizations: Insight Slices
SDK Developer	Adding Custom Builder Components

3.3.1. Platform Operator Persona

A platform operator typically manages the Streaming Analytics Manager platform, and provisions various services and resources for the application development team. Common responsibilities of a platform operator include:

- Installing and managing the Streaming Analytics Manager application platform.

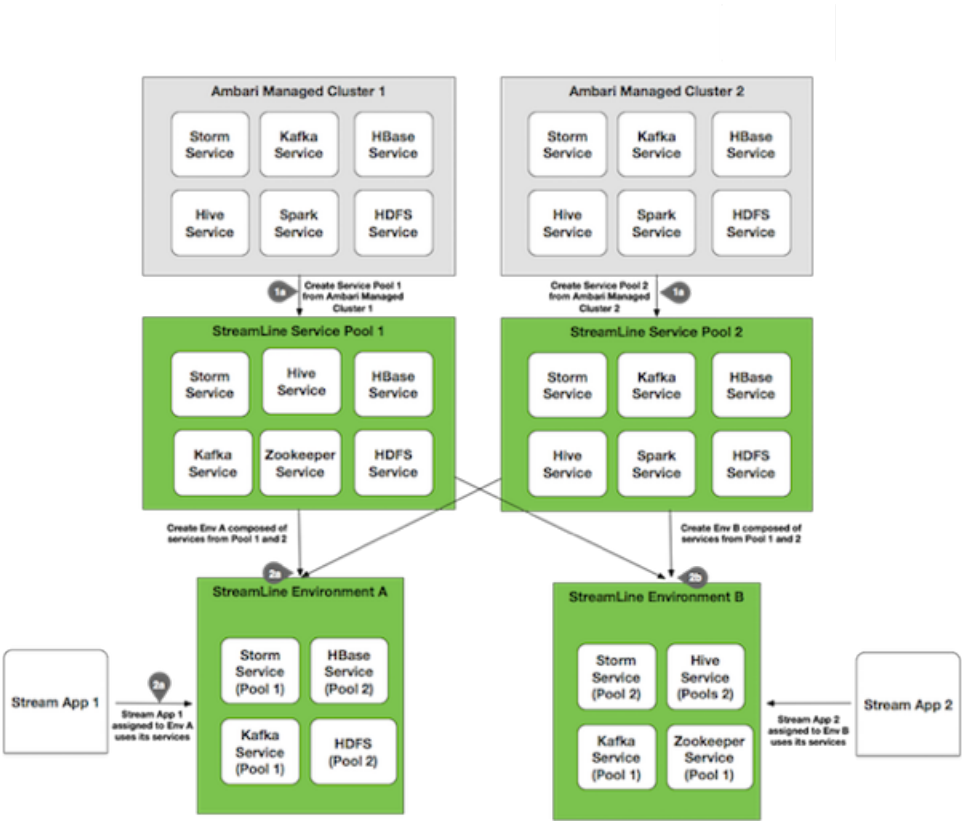
- Provisioning and providing access to services (e.g. big data services like Kafka, Storm, HDFS, HBase) for use by the development team when building stream apps.
- Provisioning and providing access to environments such as development, testing, and production, for use by the development team when provisioning stream apps.

3.3.1.1. Services, Service Pools and Environments

To perform these responsibilities, a platform operator works with three important abstractions in Streaming Analytics Manager:

- **Service** is an entity that an application developer works with to build stream apps. Examples of services could be a Storm cluster that the stream app will be deployed to, a Kafka cluster that is used by the stream app to create a streams, or a HBase cluster that the stream app writes to.
- **Service Pool** is a set of services associated with an Ambari managed cluster
- **Environment** is a named entity that represents a set of services chosen from different service pools. A stream app is assigned to an environment and the app can only use the services associated with an environment.

The following diagram illustrates these constructs:



The Service, Service Pool, and Environment abstractions provide the following benefits:

1. **Simplicity and ease of use**: An application developer can use the Service abstraction without needing to focus on configuration details. For example, to deploy a stream app

to a Storm cluster, the developer does not need to consider how to configure the Storm cluster (Nimbus host, ports, and so on). Instead, the developer simply selects the Storm service from the environment associated with the app. The service abstract out all the details/complexities.

2. **Ease of propagating a stream app between environments**: With Service as an abstraction, it is easy for the stream operator or application developer to move a stream app from one environment to another. They simply export the stream app and import it into a different environment.

More Information

See Managing Stream Apps for more information about creating and managing the Streaming Analytics Manager environment.

3.3.2. Application Developer Persona

The application developer uses the Stream Builder component to design, implement, deploy, and debug stream apps in Streaming Analytics Manager.

The following subsections describe component building blocks and schema requirements.

More Information

- Getting Started with Streaming Analytics

3.3.2.1. Component Building Blocks

Stream Builder offers several building blocks for stream apps: sources, processors, sinks, and custom components.

3.3.2.1.1. Sources

Source builder components are used to create data streams. SAM has the following sources:

- Kafka
- Azure Event Hub
- HDFS

3.3.2.1.2. Processors

Processor builder components are used to manipulate events in the stream.

The following table lists processors that are available with Streaming Analytics Manager.

Processor Name	Description
Join	• Joins two streams together based on a field from each stream. • Two join types are supported: inner and left. • Joins are based on a window that you can configure based on time or count.

Processor Name	Description

Rule	• Allows you to configure rule conditions that route events to different streams.
	• Standard conditional operators are supported for rules.
	• Configuring a rule has two modes:
	  • General: Guided rule creation using drop-down menus.
	  • Advanced: Write complex SQL to construct a rule.
	• Rules are translated to SQL to be applied on the stream.
	• An event goes through all the conditions and if it matches multiple rules the event is sent to all the matching output streams.

Aggregate	• Performs functions over windows of events. • Two types of windows are supported: tumbling and sliding. • You can create window criteria based on time interval and count. • Window functions supported out of the box include: stddev, stddevp, variance, variancecep, avg, min, max, sum, count. The system is extensible to add custom functions as well.
Projection	• Applies transformations to the events in the stream • Extensive set of OOO functions and the ability to add your own functions
Branch	• Performs a standard if-else construct for routing. • The even is routed to the first rule it matches. Once an event has matched a rule, no further condition search is performed.
PMML	• Executes a PMML model that is stored in the Model Registry. PMML has been minimally tested as part of the Tech Preview, and should not be used.

3.3.2.1.3. Sinks

Sink builder components are used to send events to other systems.

Streaming Analytics Manager supports the following sinks:

- Kafka
- Druid
- HDFS
- HBase
- Hive
- JDBC
- OpenTSDB

- Notification (OOO support Kafka and the ability to add custom notifications)
- Cassandra
- Solr

3.3.2.1.4. Custom Components

For more information about developing custom components, see SDK Developer Persona.

3.3.2.2. Schema Requirements

Unlike NiFi (the flow management service of the HDF platform), Streaming Analytics Manager requires a schema for stream apps. More specifically, every Builder component requires a schema to function.

The primary data stream source is Kafka, which uses the HDF Schema Registry.

The Builder component for Apache Kafka is integrated with the Schema Registry. When you configure a Kafka source and supply a Kafka topic, Streaming Analytics Manager calls the Schema Registry. Using the Kafka topic as the key, Streaming Analytics Manager retrieves the schema. This schema is then displayed on the tile component, and is passed to downstream components.

3.3.3. Analyst Persona

A business analyst uses the Streaming Analytics Manager Stream Insight module to create time-series and real-time analytics dashboards, charts and graphs; and create rich customizable visualizations of data.

Stream Insight Key Concepts

The following table describes key concepts of the Stream Insights module.

Stream Insight Concept	Description
Analytics Engine	• Stream Insight analytics engine is powered by Druid, an open source data store designed for OLAP queries on event data. • Data can be streamed into the Analytics engine via the Druid/Analytics Engine Sink that app developers can use when building streaming apps. The analytics engine sink can stream data into new/existing insight cubes.
Insight Data Source	• A insight data source is powered by Druid that represents the store for streaming data. The cube can be queried to do rollups, aggregations and other powerful analytics
Insight Slice	• A visualization that can be created from asking questions of the data source. An insight can be added to the dashboard
Dashboard	• Consists of a set of slices. Dashboards are created by the Business analysts to perform descriptive analytics

A business analyst can create a wide array of visualizations to gather insights on streaming data.

The platform supports over 30+ visualizations the business analyst can create.

More Information

- Creating Insight Slices
- See the Gallery of Superset Visualizations for visualization examples

3.3.4. SDK Developer Persona

Streaming Analytics Manager supports the development of custom functionality through the use of its SDK.

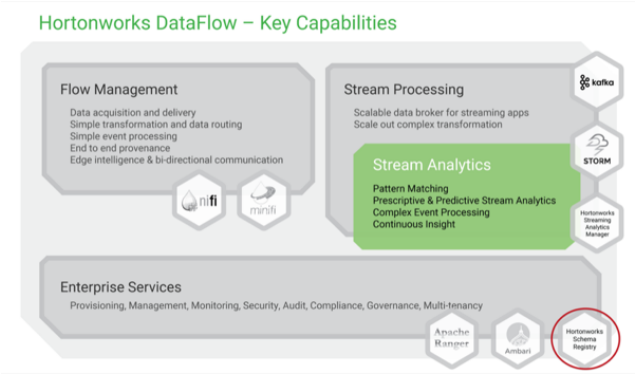
More Information

Adding Custom Builder Components

4. Schema Registry Overview

The Hortonworks DataFlow Platform (HDF) provides flow management, stream processing, and enterprise services for collecting, curating, analyzing and acting on data in motion across on-premise data centers and cloud environments.

As the diagram below instructions, Hortonworks Schema Registry is part of the enterprise services that powers the HDF platform.



Schema Registry provides a shared repository of schemas that allows applications and HDF components HDF (NiFi, Storm, Kafka, Streaming Analytics Manager, and similar) to flexibly interact with each other.

Applications built using HDF often need a way to share metadata across 3 dimensions:

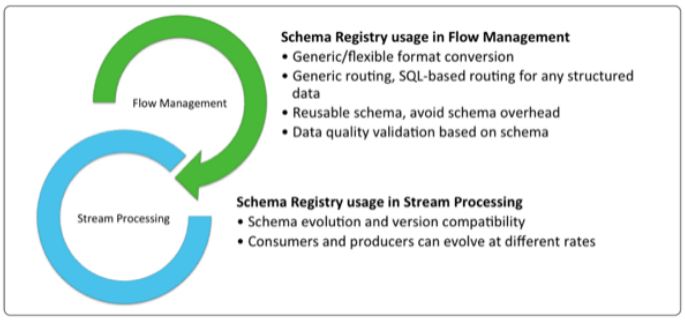
- Data format
- Schema
- Semantics or meaning of the data

The Schema Registry design principle is to provide a way to tackle the challenges of managing and sharing schemas between the components of HDF and in such a way that the schemas are designed to support evolution such that a consumer and producer can understand different versions of those schemas but still read all information shared between both versions and safely ignore the rest.

Hence, the value that Schema Registry provides for HDF and the applications that integrate with it are the following:

- Centralized registry – Provide reusable schema to avoid attaching schema to every piece of data
- Version management – Define relationship between schema versions so that consumers and producers can evolve at different rates
- Schema validation – Enable generic format conversion, generic routing and data quality

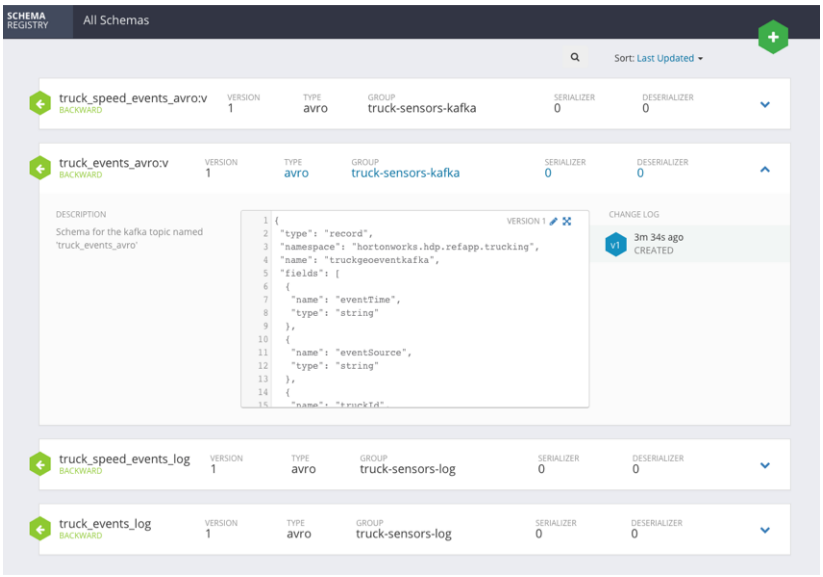
Figure 4.1. Schema Registry Usage in Flow Management



4.1. Examples of Interacting with Schema Registry

Schema Registry UI

You can use the Schema Registry UI to create schema groups, schema metadata, and add schema versions.



Schema Registry API

You can access the Schema Registry API Swagger documentation directly from the UI.

To do this, append your URL with: /swagger/

For example: http://localhost:9090/swagger/

Java Client

You can review the following GitHub repositories for examples of how to interact with the Schema Registry Java Client:

- https://github.com/georgevetticaden/hdp/blob/master/reference-apps/iot-trucking-app/trucking-data-simulator/src/main/java/hortonworks/hdp/refapp/trucking/simulator/schemaregistry/TruckSchemaRegistryLoader.java#L48

- https://github.com/hortonworks/registry/blob/HDF-2.1.0.0/schema-registry/README.md#api-examples
- https://github.com/hortonworks/registry/blob/HDF-2.1.0.0/examples/schema-registry/avro/src/main/java/com/hortonworks/registries/schemaregistry/examples/avro/SampleSchemaRegistryClientApp.java

Kafka Serdes

See the following example of using the Schema Registry Kafka Serdes:

https://github.com/hortonworks/registry/blob/HDF-2.1.0.0/examples/schema-registry/avro/src/main/java/com/hortonworks/registries/schemaregistry/examples/avro/TruckEventsKafkaAvroSerDesApp.java

4.2. Schema Registry Use Cases

With a basic understanding of Schema Registry, the below sections walks through common use cases for Schema Registry.

4.2.1. Use Case 1: Registering and Querying a Schema for a Kafka Topic

When Kafka is integrated into enterprise organization deployments, you typically have many different Kafka topics used by different apps and users. With the adoption of Kafka within the enterprise, some key questions that often come up are the following:

- What are the different events in a given Kafka topic?
- What do I put into a given Kafka topic?
- Do all Kafka events have a similar type of schema?
- How do I parse and use the data in a given Kafka topic?

While Kafka topics do not have a schema, having an external store that tracks this metadata for a given Kafka topic helps to answer these common questions. Schema Registry addresses this use case.

One important point to note is that Schema Registry is not just a metastore for Kafka. Schema Registry was designed to be generic schema store for any type of entity or store (log files, or similar.)

4.2.2. Use Case 2: Reading/Deserializing and Writing/Serializing Data from and to a Kafka Topic

In addition to storing schema metadata, another key use case is to store metadata for the format of how data should be read and how it should be written. Schema Registry supports this use case as well by providing capabilities to store JAR files for serializers and deserializers and then mapping the serdes to the schema.

4.2.3. Use Case 3: Dataflow Management with Schema-based Routing

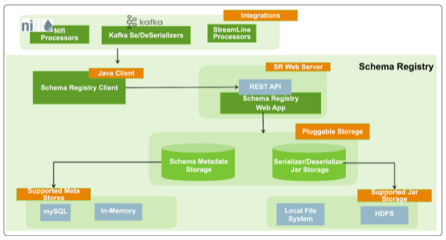
Image if you are using NiFi to move different types of syslog events to downstream systems. You have data movement requirements where you need to parse the syslog event to extract the event type, and route the event to a certain downstream system (different Kafka topics, for example) based on the event type.

Without Schema Registry, NiFi uses regular expressions or other utilities to parse the event type value from the payload and store into a flowfile attribute. Then NiFi uses routing processors (`RouteOnAttribute`, for example) to use the parsed value for routing decisions. If the structure of the data changes considerably, this type of extract and routing pattern is brittle and requires frequent changes.

With the introduction of Schema Registry, NiFi queries the registry for schema and then retrieves the value for a certain element in the schema. In this case, even if the structure changes, as long as compatibility policies are adhered to, NiFi's extract and routing rules do not change. This is another common use case for Schema Registry.

4.3. Schema Registry Component Architecture

The below diagram represents the component architecture of Schema Registry.



Schema Registry has three main components:

- Registry web server – Web Application exposing the REST endpoints you can use to manage schema entities. You can use a web proxy and load balancer with multiple Web Servers to provide HA and scalability.
- Pluggable storage – Schema Registry uses the following two types of storages:
  - Schema Metadata Storage – Relational store that holds the metadata for the schema entities. Inn-memory storage (for development purposes) and mySQL databases are supported.
  - Serdes Storage – File storage for the serializer and deserializer jars. Local file system and HDFS storage are supported. Local file system storage is the default.
- Schema Registry Client – A java client that HDF components can use to interact with the RESTful services.

There are three integration points with HDF:

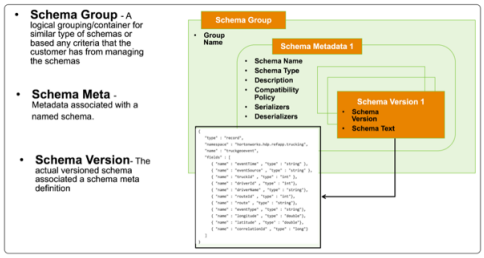
- Custom NiFi Processors – New processors and controller services in NiFi that interact with the Schema Registry.
- Kafka Serializer and Deserializer – A Kafka serializer and deserializer that uses Schema Registry. The Kafka serdes can be found on GitHub.
- Hortonworks Streaming Analytics Manager Processors –

4.4. Schema Registry Concepts

4.4.1. Schema Entities

You can use Schema Registry to work with three types of schema entities:

Figure 4.2. Schema entities



This table provides a more detailed description of the schema entities:

Table 4.1. Schema entity types

Entity Type	Description	Example
Schema Group	A logical grouping of similar schemas. A Schema Group can be based on any criteria you have for managing schemas. Schema Groups can have multiple Schema Metadata definitions.	• Group Name – truck-sensors-log • Group Name – truck-sensors-kafka
Schema Metadata	Metadata associated with a named schema. A metadata definition is applied to all the schema versions that are assigned to it. Key metadata elements include: • Schema Name – A unique name for each schema. Used as a key to look up schemas. • Schema Type – The format of the schema. **Note:** Avro is currently the only supported type. • Compatibility Policy – The compatibility rules that exist when the new schemas are registered. See Compatibility Policies for more information. • Serializers/Deserializers – A set of serializers and deserializers that you can upload to the registry and associate with schema metadata definitions.	• Schema Name – truck_events_avro:v • Schema Type – avro • Compatibility Policy – SchemaCompatibility.BACKWARD
Schema Version	The versioned schema associated a schema metadata definition.	`{ "type" : "record", "namespace" : "hortonworks.hdp.refapp.trucking",`

Entity Type	Description	Example
		<pre>{ "name" : "truckgeoevent", "fields" : [{ "name" : "eventTime" , "type" : "string" }, { "name" : "eventSource" , "type" : "string" }, { "name" : "truckId" , "type" : "int" }, { "name" : "driverId" , "type" : "int"}, { "name" : "driverName" , "type" : "string"}, { "name" : "routeId" , "type" : "int"}, { "name" : "route" , "type" : "string"}, { "name" : "eventType" , "type" : "string"}, { "name" : "longitude" , "type" : "double"}, { "name" : "latitude" , "type" : "double"}, { "name" : "correlationId" , "type" : "long"}] }</pre>

4.4.2. Compatibility Policies

A key Schema Registry feature is the ability to version schemas as they evolve. Compatibility policies are created at the schema metadata level, and define evolution rules for each schema.

After a policy has been defined for a schema, any subsequent version updates must honor the schema's original compatibility, otherwise you experience an error.

Compatibility of schemas can be configured with any of the below values:

Backward Compatibility	Indicates that new version of a schema would be compatible with earlier version of that schema. That means the data written from earlier version of the schema, can be deserialized with a new version of the schema. When you have a Backward Compatibility policy on your schema, you can evolve schemas by deleting portions, but you cannot add information.
Forward Compatibility	Indicates that an existing schema is compatible with subsequent versions of the schema. That means the data written from new version of the schema can still be read with old version of the schema.
Full Compatibility	Indicates that a new version of the schema provides both backward and forward compatibilities.
None	Indicates that no compatibility policy is in place.

The default value is None.

You set the compatibility policy when you are adding a schema. Once set, you cannot change it.

5. Navigating the HDF Library

To navigate the Hortonworks DataFlow (HDF) documentation library, begin by deciding your current goal.

If you want to...	See this document...
Install or upgrade an HDF cluster using Apache Ambari	• Release Notes • Support Matrix • Planning Your Deployment • Ambari Upgrade • MiNiFi Java Agent Quick Start
Get started with HDF	• Getting Started with Apache NiFi • Getting Started with Stream Analytics
Use and administer HDF Flow Management capabilities	• Apache NiFi User Guide • Apache NiFi Administration Guide • Apache NiFi Developer Guide • Apache NiFi Expression Language Guide • MiNiFi Java Agent Administration Guide
Use and administer HDF Stream Analytics capabilities	• Streaming Analytics Manager User Guide • Schema Registry User Guide • Apache Storm Component Guide • Apache Kafka Component Guide