Apache NiFi Overview

Date of Publish: 2019-03-15

https://docs.hortonworks.com/

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. It is often not possible to come even close to replicating production environments in the lab.
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	

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 (SOA), 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.

The core concepts of NiFi

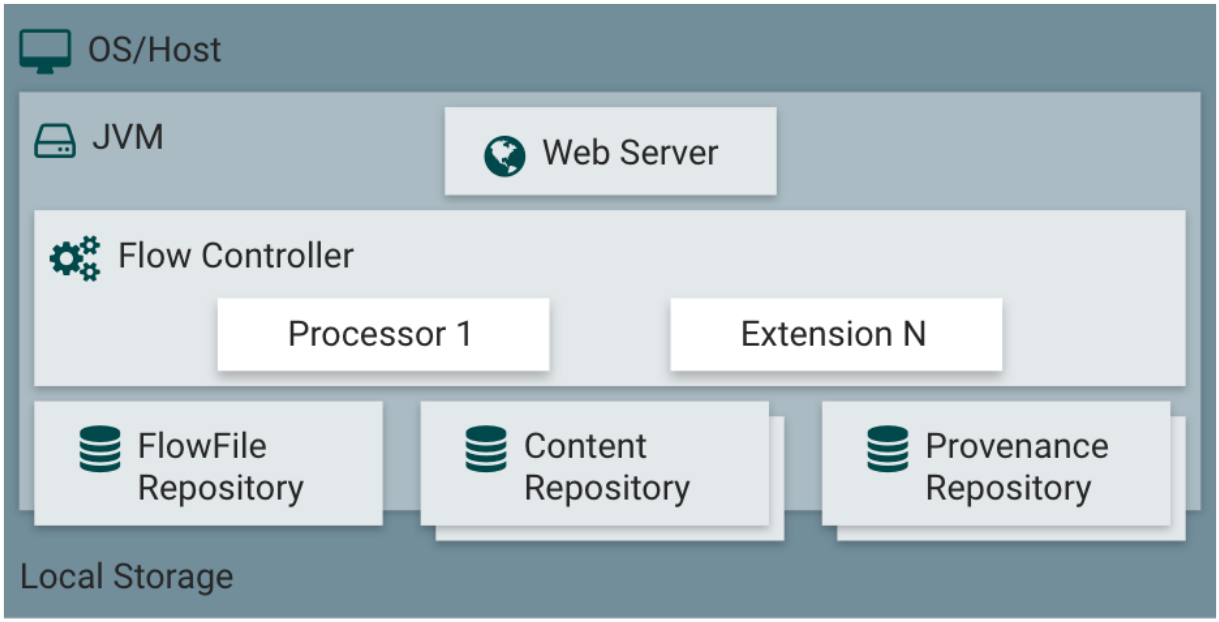
NiFi's fundamental design concepts closely relate to the main ideas of Flow Based Programming. Here are some of the main NiFi concepts and how they map to FBP:

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 EIP 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 manner, process groups allow creation of entirely new components simply by composition of other components.

This design model, also similar to SEDA, 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

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

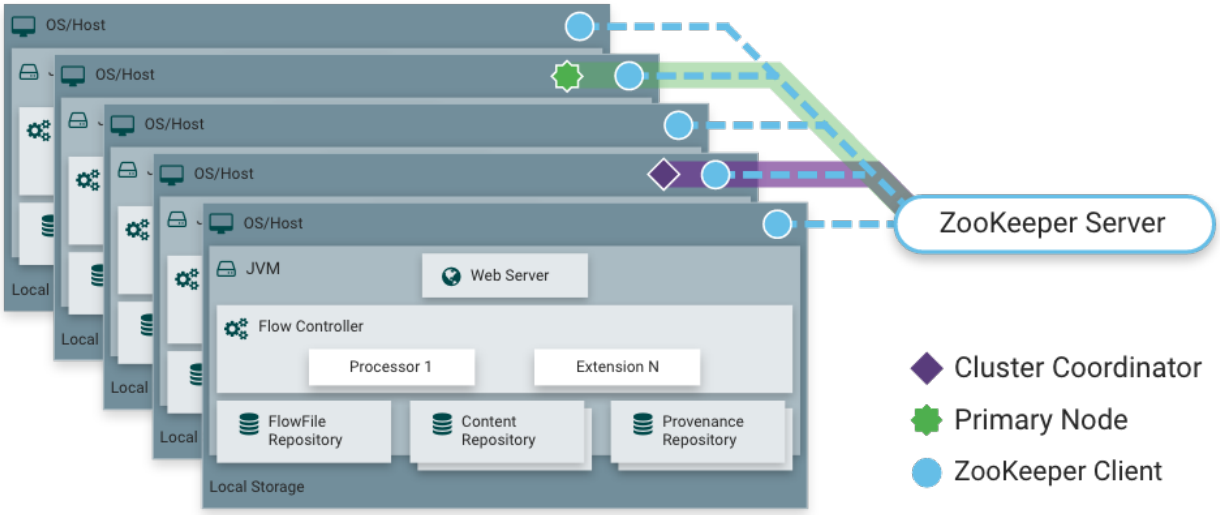
The Content Repository is where the actual content bytes of a given FlowFile live. The implementation of the repository is pluggable. The default approach is a fairly simple mechanism, which stores blocks of data in the file system. More than one file system storage location can be specified so as to 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.

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.

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

Ease of Use

Security

Extensible Architecture

Flexible Scaling Model

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	
Prioritized Queuing	
Flow Specific QoS (latency v throughput, loss tolerance, etc.)	

	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). 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. 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.
Visual Command and Control **Flow Templates** **Data Provenance** **Recovery / Recording a rolling buffer of fine-grained history**	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.

	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.
	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.
	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.
System to System **User to System** **Multi-tenant Authorization**	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. 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. 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.
Extension **Classloader Isolation** **Site-to-Site Communication Protocol**	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. 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.
	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.
Scale-out (Clustering) **Scale-up & down**	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.

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]).