



APACHE NIFI

DATAFLOW MANAGEMENT PLATFORM

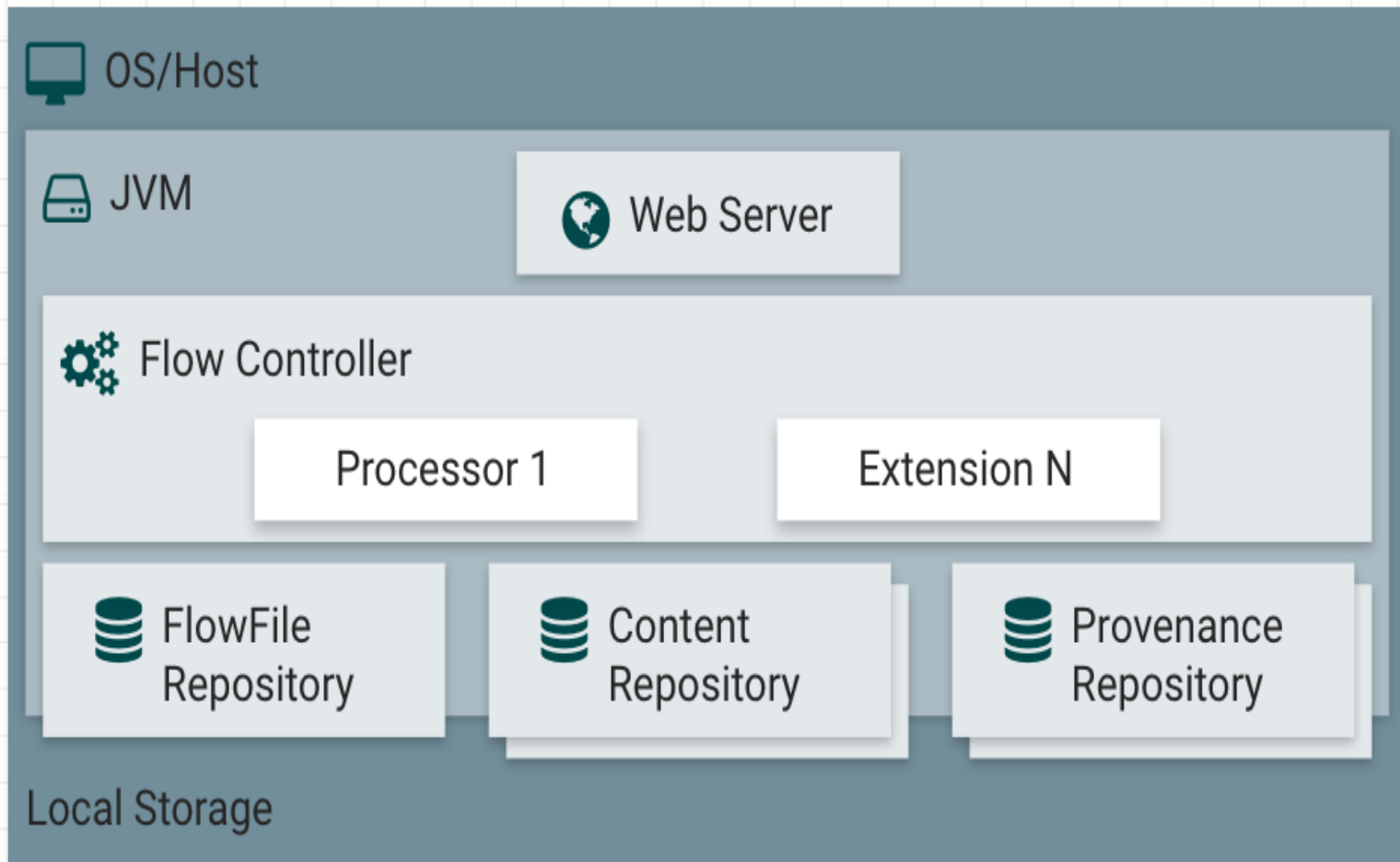
Bin Jiang

04/08/2017

What's Apache Nifi

Apache NiFi is an integrated data logistics platform for automating the movement of data between disparate systems. It provides real-time control that makes it easy to manage the movement of data between any source and any destination.

What's Apache Nifi



High-level Challenges of Dataflow

- Systems fail
- Data access exceeds capacity to consume
- Boundary conditions are mere suggestions
- What is noise one day becomes signal the next
- Systems evolve at different rates
- Compliance and security
- Continuous improvement occurs in production

NiFi Features

- Web-based user interface
 - Seamless experience between design, control, feedback, and monitoring
- Highly configurable
 - Loss tolerant vs guaranteed delivery
 - Low latency vs high throughput
 - Dynamic prioritization
 - Flow can be modified at runtime
 - Back pressure
- Data Provenance
 - Track dataflow from beginning to end
- Designed for extension
 - Build your own processors and more
 - Enables rapid development and effective testing
- Secure
 - SSL, SSH, HTTPS, encrypted content, etc...
 - Multi-tenant authorization and internal authorization/policy management

NiFi Features – Financial Service

- Support High volumes
- Recorded lineage for each message
- Isolation of processors to avoid conflicts

The core concepts of NiFi

- FlowFile
- FlowFile Processor
- Connection
- Flow Controller
- Process Group
- DataFlow Manager
- Relationship
- Controller Service
- Reporting Task
- Funnel
- Port
- Remote Process Group
- Template
- flow.xml.gz

The core concepts of NiFi

- DataFlow Manager
 - A DataFlow Manager (DFM) is a NiFi user who has permissions to add, remove, and modify components of a NiFi dataflow

The core concepts of NiFi

- FlowFile
 - 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

The core concepts of NiFi

- Core attributes for FlowFiles
 - **filename**: The filename of the FlowFile. The filename should not contain any directory structure
 - **uuid**: A unique universally unique identifier (UUID) assigned to this FlowFile
 - **path**: The FlowFile's path indicates the relative directory to which a FlowFile belongs and does not contain the filename
 - **absolute.path**: The FlowFile's absolute path indicates the absolute directory to which a FlowFile belongs and does not contain the filename.

The core concepts of NiFi

- Core attributes for FlowFiles
 - **priority**: A numeric value indicating the FlowFile priority
 - **mime.type**: The MIME Type of this FlowFile
 - **discard.reason**: Specifies the reason that a FlowFile is being discarded
 - **alternate.identifier**: Indicates an identifier other than the FlowFile's UUID that is known to refer to this FlowFile

The core concepts of NiFi

- FlowFiles REST API

Flow REST API:

<code>/flow/input-ports/{id}/status</code>	Gets status for an input port
<code>/flow/output-ports/{id}/status</code>	Gets status for an output port
<code>/flow/process-groups/{id}</code>	Gets a process group
<code>/flow/process-groups/{id}</code>	Schedule or unschedule components in the specified Process Group.
<code>/flow/process-groups/{id}/controller-services</code>	Gets all controller services
<code>/flow/process-groups/{id}/status</code>	Gets the status for a process group
<code>/flow/process-groups/{id}/status/history</code>	Gets status history for a remote process group
<code>/flow/processors/{id}/status</code>	Gets status for a processor
<code>/flow/processors/{id}/status/history</code>	Gets status history for a processor
<code>/flow/remote-process-groups/{id}/status/history</code>	Gets the status history

FlowFile Queue REST API:

<code>/flowfile-queues/{id}/drop-requests</code>	Creates a request to drop the contents of the queue in this connection.
<code>/flowfile-queues/{id}/drop-requests/{drop-request-id}</code>	Gets the current status of a drop request for the specified connection.
<code>/flowfile-queues/{id}/drop-requests/{drop-request-id}</code>	Cancels and/or removes a request to drop the contents of this connection.
<code>/flowfile-queues/{id}/flowfiles/{flowfile-uuid}</code>	Gets a FlowFile from a Connection.
<code>/flowfile-queues/{id}/flowfiles/{flowfile-uuid}/content</code>	Gets the content for a FlowFile in a Connection.
<code>/flowfile-queues/{id}/listing-requests</code>	Lists the contents of the queue in this connection.
<code>/flowfile-queues/{id}/listing-requests/{listing-request-id}</code>	Gets the current status of a listing request for the specified connection.
<code>/flowfile-queues/{id}/listing-requests/{listing-request-id}</code>	Cancels and/or removes a request to list the contents of this connection.

The core concepts of NiFi

- FlowFile Processor
 - 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

The core concepts of NiFi

- Processor - accomplish all of the following tasks
 - Create FlowFiles
 - Read FlowFile content
 - Write FlowFile content
 - Read FlowFile attributes
 - Update FlowFile attributes
 - Ingest data
 - Egress data
 - Route data
 - Extract data
 - Modify data

The core concepts of NiFi

- Extending Processor

- Loaded and instantiated using Java's ServiceLoader mechanism
- AbstractProcessor is the base class
- All extensions must be thread-safe
- Supporting API:
 - ProcessSession
 - ProcessContext
 - PropertyDescriptor
 - PropertyValue
 - Validator
 - Relationship
 - StateManager

The core concepts of NiFi

- AbstractProcessor API

The AbstractProcessor provides several methods that will be of interest to Processor developers:

- Processor Initialization
- Exposing Processor's Relationships
- Exposing Processor Properties
- Validating Processor Properties
- Responding to Changes in ConfigurationProcessSession

The core concepts of NiFi

- Processor Lifecycle

- @OnAdded
- @OnEnabled
- @OnRemoved
- @OnScheduled
- @OnUnscheduled
- @OnUnscheduled
- @OnShutdown

The core concepts of NiFi

- Processor REST API

Processors

Create a processor, Set properties, Schedule

GET	/processors/{id}	Gets a processor
------------	------------------	------------------

PUT	/processors/{id}	Updates a processor
------------	------------------	---------------------

DELETE	/processors/{id}	Deletes a processor
---------------	------------------	---------------------

GET	/processors/{id}/descriptors	Gets the descriptor for a processor property
------------	------------------------------	--

GET	/processors/{id}/state	Gets the state for a processor
------------	------------------------	--------------------------------

POST	/processors/{id}/state/clear-requests	Clears the state for a processor
-------------	---------------------------------------	----------------------------------

The core concepts of NiFi

- Relationship
 - Each Processor has zero or more Relationships defined for it. These Relationships are named to indicate the result of processing a FlowFile. After a Processor has finished processing a FlowFile, it will route (or “transfer”) the FlowFile to one of the Relationships

The core concepts of NiFi

- Connection
 - 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

The core concepts of NiFi

- Connection

DETAILS

SETTINGS

Name

Id

No value set

FlowFile Expiration ?

0 sec

Back Pressure Object Threshold ?

10000

Back Pressure Data Size Threshold ?

1 GB

Available Prioritizers ?

FirstInFirstOutPrioritizer

NewestFlowFileFirstPrioritizer

OldestFlowFileFirstPrioritizer

PriorityAttributePrioritizer

Selected Prioritizers ?

The core concepts of NiFi

- Connection
 - File expiration
 - Back Pressure
 - Prioritizers

The core concepts of NiFi

- Connection REST API

Connections

Create a connection, Set queue priority, Update connection destination

GET

`/connections/{id}`

Gets a connection

PUT

`/connections/{id}`

Updates a connection

DELETE

`/connections/{id}`

Deletes a connection

The core concepts of NiFi

- Flow Controller
 - 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

The core concepts of NiFi

- Process Group
 - 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

The core concepts of NiFi

- Process Group REST API

<code>/process-groups/{id}</code>	Gets a process group
<code>/process-groups/{id}</code>	Updates a process group
<code>/process-groups/{id}</code>	Deletes a process group
<code>/process-groups/{id}/connections</code>	Creates a connection
<code>/process-groups/{id}/connections</code>	Gets all connections
<code>/process-groups/{id}/controller-services</code>	Creates a new controller service
<code>/process-groups/{id}/funnels</code>	Creates a funnel
<code>/process-groups/{id}/funnels</code>	Gets all funnels
<code>/process-groups/{id}/input-ports</code>	Creates an input port
<code>/process-groups/{id}/input-ports</code>	Gets all input ports
<code>/process-groups/{id}/labels</code>	Creates a label
<code>/process-groups/{id}/labels</code>	Gets all labels
<code>/process-groups/{id}/output-ports</code>	Creates an output port
<code>/process-groups/{id}/output-ports</code>	Gets all output ports
<code>/process-groups/{id}/process-groups</code>	Creates a process group
<code>/process-groups/{id}/process-groups</code>	Gets all process groups
<code>/process-groups/{id}/processors</code>	Creates a new processor
<code>/process-groups/{id}/processors</code>	Gets all processors
<code>/process-groups/{id}/remote-process-groups</code>	Creates a new process group
<code>/process-groups/{id}/remote-process-groups</code>	Gets all remote process groups
<code>/process-groups/{id}/snippet-instance</code>	Copies a snippet
<code>/process-groups/{id}/template-instance</code>	Instantiates a template
<code>/process-groups/{id}/templates</code>	Creates a template
<code>/process-groups/{id}/templates/import</code>	Imports a template
<code>/process-groups/{id}/templates/upload</code>	Uploads a template

The core concepts of NiFi

- Remote Process Group
 - While NiFi provides many different mechanisms for transferring data from one system to another, Remote Process Groups are often the easiest way to accomplish this if transferring data to another instance of NiFi

The core concepts of NiFi

- Remote Process Group
 - Site-to-Site
 - NiFi Site-to-Site Protocol
 - Securely and efficiently transfer data to/from nodes in one NiFi instance
 - Easy to configure
 - Site-to-Site optionally makes use of Certificates in order to encrypt data and provide authentication and authorization

The core concepts of NiFi

- Remote Process Group
 - Checksums are automatically produced by both the sender and receiver and compared after the data has been transmitted
 - Automatically load balanced
 - FlowFiles maintain attributes
 - Push and Pull
 - Storm and Nifi Instance
 - Spark Streaming and Nifi Instance

The core concepts of NiFi

- Configure Site-to-Site
 - Remote Process Group
 - Transport Protocol - http or https plus proxy server
 - Input Port
 - Output Port
 - Access Control

The core concepts of NiFi

- Controller Service

Controller Services are extension points that, after being added and configured by a DFM in the User Interface, will start up when NiFi starts up and provide information for use by other components (such as processors or other controller services). A common Controller Service used by several components is the `StandardSSLContextService`

The core concepts of NiFi

- Reporting Task

Reporting Tasks run in the background to provide statistical reports about what is happening in the NiFi instance. The DFM adds and configures Reporting Tasks in the User Interface as desired. Common reporting tasks include the `ControllerStatusReportingTask`, `MonitorDiskUsage` reporting task, `MonitorMemory` reporting task, and the `StandardGangliaReporter`

The core concepts of NiFi

- Funnel

A funnel is a NiFi component that is used to combine the data from several Connections into a single Connection

The core concepts of NiFi

- Port

Dataflows that are constructed using one or more Process Groups need a way to connect a Process Group to other dataflow components. This is achieved by using Ports. A DFM can add any number of Input Ports and Output Ports to a Process Group and name these ports appropriately

The core concepts of NiFi

- Template

A Template is a way of combining these basic building blocks into larger building blocks. Once a DataFlow has been created, parts of it can be formed into a Template. This Template can then be dragged onto the canvas, or can be exported as an XML file and shared with others. Templates received from others can then be imported into an instance of NiFi and dragged onto the canvas.

The core concepts of NiFi

- Template
 - Creating a Template
 - Importing a Template
 - Instantiating a Template
 - Managing Templates
 - Exporting a Template
 - Removing a Template

The core concepts of NiFi

- flow.xml.gz

Everything the DFM puts onto the NiFi User Interface canvas is written, in real time, to one file called the flow.xml.gz. This file is located in the nifi/conf directory by default. Any change made on the canvas is automatically saved to this file

The core concepts of NiFi

- State Manager
 - Responsible for providing a simple API for storing and retrieving state
 - The state can be stored local to the node or across all nodes in a cluster
 - Only implementation that is currently supported for storing cluster-wide state is backed by ZooKeeper
 - Entire State Map must be less than 1 MB in size, after being serialized

NiFi - Data Provenance

- NiFi's Data Provenance page provides what happened to a FlowFile
- NiFi records and indexes data provenance details as objects flow through the system
- Users can perform searches, conduct troubleshooting and evaluate things like dataflow compliance and optimization in real time

NiFi - Data Provenance

- NiFi updates this information every five minutes
- Replaying a FlowFile
- Viewing FlowFile Lineage
- Find Parents
- Expanding an Event

NiFi – Scheduling Strategies

- Time Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Scheduling Strategy ?

Timer driven ✓

Timer driven ?

CRON driven ?

Run Schedule ?

0 sec

Run Duration ?

0ms 25ms 50ms 100ms 250ms 500ms 1s 2s

Lower latency Higher throughput

CANCEL

APPLY

NiFi – Scheduling Strategies

- CRON Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Scheduling Strategy ?

CRON driven ▼

Concurrent Tasks ?

1

Run Duration ?

0ms 25ms 50ms 100ms 250ms 500ms 1s 2s

Lower latency Higher throughput

Run Schedule ?

*****?

CANCEL

APPLY

NiFi – Scheduling Strategies

- CRON Driven

The CRON driven scheduling value is a string of six required fields and one optional field, each separated by a space. These fields are

NiFi – Scheduling Strategies

- CRON Driven

Field	Valid values
Seconds	0-59
Minutes	0-59
Hours	0-23
Day of Month	1-31
Month	1-12 or JAN-DEC
Day of Week	1-7 or SUN-SAT
Year (optional)	empty, 1970-2099

NiFi – Scheduling Strategies

- CRON Driven

Use one of the following ways to specify values:

- Number: Specify one or more valid value. You can enter more than one value using a comma-separated list.
- Range: Specify a range using the <number>-<number> syntax.
- Increment: Specify an increment using <start value>/<increment> syntax. For example, in the Minutes field, 0/15 indicates the minutes 0, 15, 30, and 45.

NiFi – Scheduling Strategies

- CRON Driven

Quartz documentation

<http://www.quartz-scheduler.org/documentation/quartz-2.x/tutorials/crontrigger.html>

NiFi – Scheduling Strategies

- Event Driven

The Processor will be triggered to run by an event, and that event occurs when FlowFiles enter Connections feeding this Processor

NiFi – Scheduling Strategies

- Email Driven

Displaying 8 of 224

Email

Type ▲	Version	Tags
ConsumeEWS	1.2.0.3.0.2.0-76	EWS, Exchange, Email, Consum...
ConsumeIMAP	1.2.0.3.0.2.0-76	Imap, Email, Consume, Ingest, ...
ConsumePOP3	1.2.0.3.0.2.0-76	Email, Consume, Ingest, Messa...
ExtractEmailAttachments	1.2.0.3.0.2.0-76	split, email
ExtractEmailHeaders	1.2.0.3.0.2.0-76	split, email
ExtractTNEFAttachments	1.2.0.3.0.2.0-76	split, email
ListenSMTP	1.2.0.3.0.2.0-76	smtp, listen, email
PutEmail	1.2.0.3.0.2.0-76	smtp, email, put, notify

NiFi – Scheduling Strategies

- Email Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value
User Name	?	No value set
Password	?	No value set
Folder	?	INBOX
Fetch Size	?	10
Delete Messages	?	false
Connection timeout	?	30 sec
Exchange Version	?	Exchange2010_SP2
EWS URL	?	No value set
Auto Discover URL	?	true
Mark Messages as Read	?	true
Original Headers to Include	?	Empty string set
Original Headers to Exclude	?	Empty string set

NiFi – Scheduling Strategies

- Web Socket Driven

Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Name ConnectWebSocket		<input checked="" type="checkbox"/> Enabled	Automatically Terminate Relationships ?
Id 037241dc-11b2-1175-f01f-11e86a23b9c6			<input type="checkbox"/> binary message The WebSocket binary message output
Type ConnectWebSocket 1.2.0.3.0.2.0-76			<input type="checkbox"/> connected The WebSocket session is established
Bundle org.apache.nifi - nifi-websocket-processors-nar			<input type="checkbox"/> text message The WebSocket text message output
Penalty Duration ? 30 sec	Yield Duration ? 1 sec		
Bulletin Level ? WARN ▼			

NiFi – Scheduling Strategies

- HTTP Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Name

ListenHTTP

☒ Enabled

Automatically Terminate Relationships [?](#)

☐ success

Relationship for successfully received FlowFiles

Id

037241de-11b2-1175-abe2-ce2a9fea5ca3

Type

ListenHTTP 1.2.0.3.0.2.0-76

Bundle

org.apache.nifi - nifi-standard-nar

Penalty Duration [?](#)

30 sec

Yield Duration [?](#)

1 sec

Bulletin Level [?](#)

WARN



NiFi – Scheduling Strategies


- Message Driven



Displaying 227 of 227

Type ▲	Version	Tags
ConsumeEWS	1.2.0.3.0.2.0-76	EWS, Exchange, Email, Consume...
ConsumeIMAP	1.2.0.3.0.2.0-76	Imap, Email, Consume, Ingest, ...
ConsumeJMS	1.2.0.3.0.2.0-76	jms, receive, get, consume, me...
ConsumeKafka	1.2.0.3.0.2.0-76	PubSub, Consume, Ingest, Get,...
ConsumeKafkaRecord_0_10	1.2.0.3.0.2.0-76	0.10.x, PubSub, Consume, Inge...
ConsumeKafka_0_10	1.2.0.3.0.2.0-76	0.10.x, PubSub, Consume, Inge...
ConsumeMQTT	1.2.0.3.0.2.0-76	MQTT, subscribe, consume, lis...
ConsumePOP3	1.2.0.3.0.2.0-76	Email, Consume, Ingest, Mess...
ConsumeWindowsEventLog	1.2.0.3.0.2.0-76	event, windows, ingest
ControlRate	1.2.0.3.0.2.0-76	throttle, rate, rate control, throu...
ConvertAvroSchema	1.2.0.3.0.2.0-76	convert, kite, avro
ConvertAvroToJSON	1.2.0.3.0.2.0-76	json, convert, avro

NiFi – Scheduling Strategies

- Signal Driven

	 Wait Wait 1.2.0.3.0.2.0-76 org.apache.nifi - nifi-standard-nar	
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

	 Notify Notify 1.2.0.3.0.2.0-76 org.apache.nifi - nifi-standard-nar	
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

NiFi – Custom Properties

- In addition to using FlowFile attributes, system properties, and environment properties within Express Language, you can also define custom properties for Expression Language use
- NiFi properties have resolution precedence of which you should be aware when creating custom properties

NiFi – Custom Properties

- Processor-specific attributes
- FlowFile properties
- FlowFile attributes
- User defined properties (custom properties)
- System properties
- Operating System environment variables

NiFi – FlowFile Expiration

Create Connection

DETAILS

SETTINGS

Name

Id

No value set

FlowFile Expiration ?

0 sec

Back Pressure Object Threshold ?

10000

Back Pressure Data Size Threshold ?

1 GB

Available Prioritizers ?

FirstInFirstOutPrioritizer

NewestFlowFileFirstPrioritizer

OldestFlowFileFirstPrioritizer

PriorityAttributePrioritizer

Selected Prioritizers ?

NiFi – Prioritization

Create Connection

DETAILS

SETTINGS

Name

Id

No value set

FlowFile Expiration ?

0 sec

Back Pressure Object Threshold ?

10000

Back Pressure Data Size Threshold ?

1 GB

Available Prioritizers ?

FirstInFirstOutPrioritizer

NewestFlowFileFirstPrioritizer

OldestFlowFileFirstPrioritizer

PriorityAttributePrioritizer

Selected Prioritizers ?

NiFi – Back Pressure

Create Connection

DETAILS

SETTINGS

Name

Id

No value set

FlowFile Expiration ?

0 sec

Back Pressure Object Threshold ?

10000

Back Pressure Data Size Threshold ?

1 GB

Available Prioritizers ?

FirstInFirstOutPrioritizer

NewestFlowFileFirstPrioritizer

OldestFlowFileFirstPrioritizer

PriorityAttributePrioritizer

Selected Prioritizers ?

NiFi – Control Rate

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property

Rate Control Criteria

Maximum Rate

Rate Controlled Attribute

Time Duration

Grouping Attribute

data rate



data rate



flowfile count



attribute value



NiFi – Control Rate

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property

Rate Control Criteria

Maximum Rate

Rate Controlled Attribute

Time Duration

Grouping Attribute

data rate



data rate



flowfile count



attribute value



NiFi – Ordering

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
Group Identifier	?	\${filename}	
Order Attribute	?	No value set	
Initial Order	?	0	
Maximum Order	?	No value set	
Batch Count	?	1000	
Wait Timeout	?	10 min	
Inactive Timeout	?	30 min	

NiFi – Wait and Notify

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
Release Signal Identifier	?	No value set	
Target Signal Count	?	1	
Signal Counter Name	?	No value set	
Wait Buffer Count	?	1	
Releasable FlowFile Count	?	1	
Expiration Duration	?	10 min	
Distributed Cache Service	?	No value set	
Attribute Copy Mode	?	Keep original	
Wait Mode	?	Transfer to wait relationship	

NiFi – Wait and Notify

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

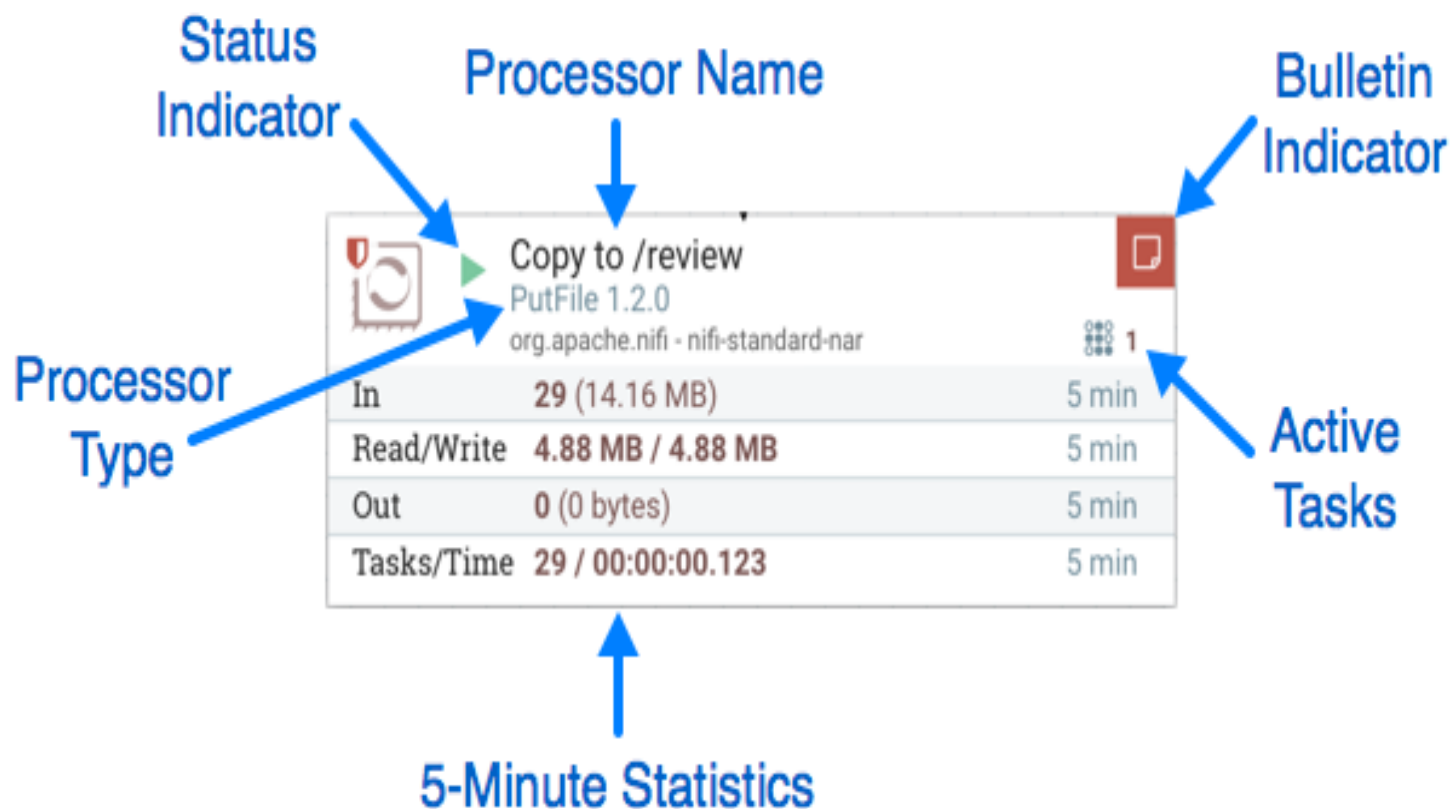
COMMENTS

Required field



Property		Value	
Release Signal Identifier	?	No value set	
Signal Counter Name	?	default	
Signal Counter Delta	?	1	
Signal Buffer Count	?	1	
Distributed Cache Service	?	No value set	
Attribute Cache Regex	?	No value set	

NiFi – Monitoring

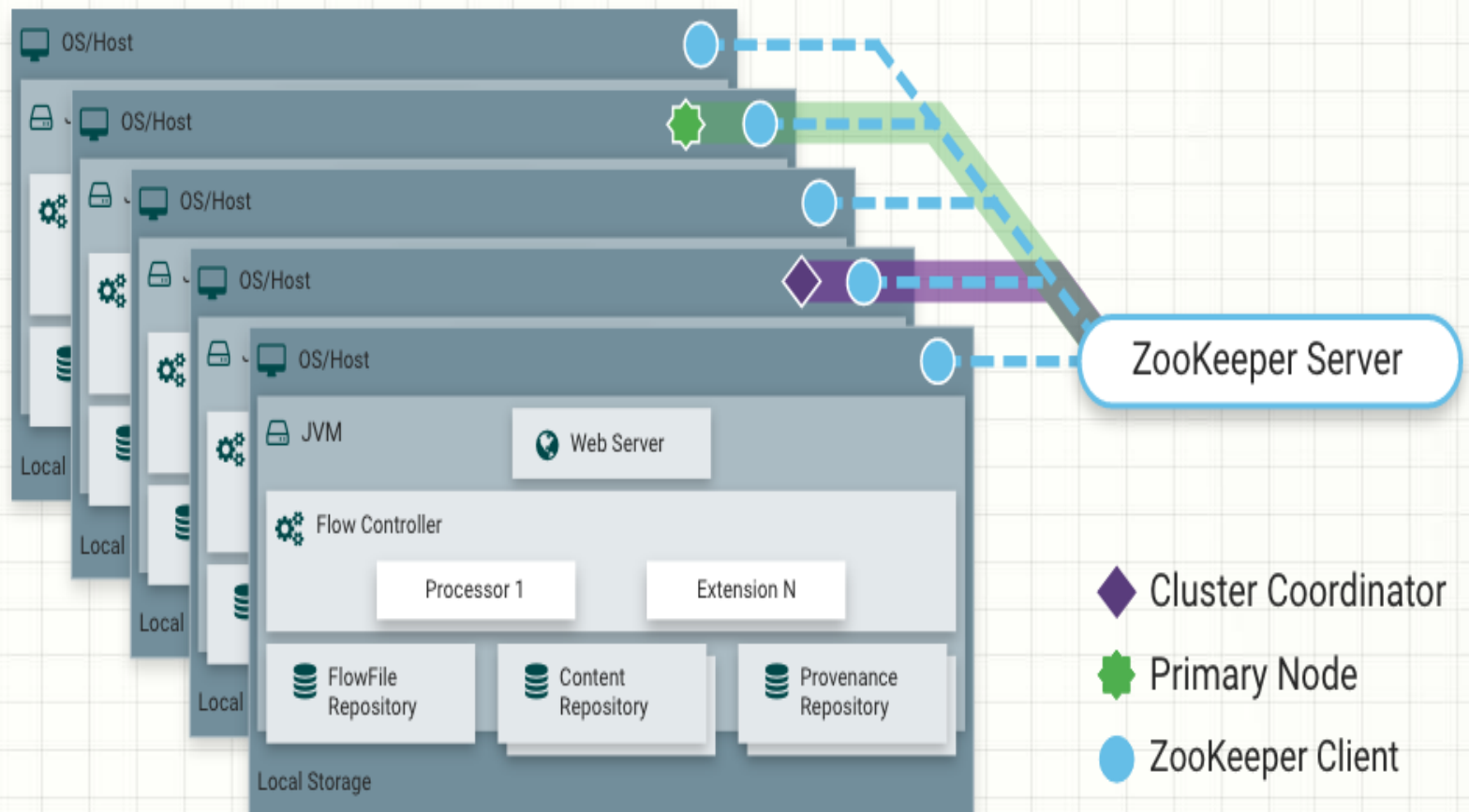


Nifi Architecture

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

- Web Server
- Flow Controller
- Extensions
- FlowFile Repository
- Content Repository
- Provenance Repository

Nifi Architecture



Nifi Architecture

- 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

Nifi Architecture

- 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.

Nifi Architecture

- 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 Performance Consideration

- For IO
- For CPU
- For RAM

Nifi - Ease of Use

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

Nifi - Flow Management

- Guaranteed Delivery
- Data Buffering w/ Back Pressure and Pressure Release
- Prioritized Queuing
- Flow Specific QoS (latency v throughput, loss tolerance, etc.)

Nifi - Security

- System to System
- User to System
- Multi-tenant Authorization

Nifi - Extensible Architecture

- Extension
- Classloader Isolation
- Site-to-Site Communication Protocol

Nifi – Working with Processor

NiFi contains many different Processors out of the box. These Processors provide capabilities to ingest data from numerous different systems, route, transform, process, split, and aggregate data, and distribute data to many systems

Nifi – ETL Design Pattern

Data Enrichment

- LookupRecord Processor and the following control service
 - SimpleCsvFileLookupService
 - ScriptedLookupService
 - IPLookupService (MaxMind Database)
 - SimpleKeyValueLookupService
 - MongoDBLookupService
 - XMLFileLookupService
 - PropertiesFileLookupService

Nifi – ETL Design Pattern

Data Enrichment

- ISPEnrichIP
- LookupAttribute
- ModifyHTMLElement
- ModifyBytes

Nifi – ETL Design Pattern

Data Transformation

- **CompressContent**: Compress or Decompress Content
- **ConvertCharacterSet**: Convert the character set used to encode the content from one character set to another
- **EncryptContent**: Encrypt or Decrypt Content
- **ReplaceText**: Use Regular Expressions to modify textual Content
- **TransformXml**: Apply an XSLT transform to XML Content
- **JoltTransformJSON**: Apply a JOLT specification to transform JSON Content

Nifi – ETL Design Pattern

Data Transformation

- Split Data Set
 - SegmentContent
 - SplitText
 - SplitXml
 - SplitJson
 - SplitContent
 - SplitAvro
 - SplitRecord

Nifi – ETL Design Pattern

Data Transformation

- Data integration (Join)
 - MergeContent
 - MergeRecord
 - QueryRecord
 - ExecuteSQL

Nifi – ETL Design Pattern

Data Transformation

- Format and Schema Conversion
 - ConvertAvroSchema
 - ConvertAvroToJSON
 - ConvertAvroToORC
 - ConvertCharacterSet
 - ConvertCSVToAvro
 - ConvertExcelToCSVProcessor
 - ConvertJSONToAvro
 - ConvertJSONToSQL
 - ConvertRecord
 - AttributesToJSON

Nifi – ETL Design Pattern

Ordering, Partitioning and Filtering

- EnforceOrder
- PartitionRecord
- ScanAttribute
- ScanContent

Nifi – ETL Design Pattern

Routing and Mediation

- **ControlRate**: Throttle the rate at which data can flow through one part of the flow
- **DetectDuplicate**: Monitor for duplicate FlowFiles, based on some user-defined criteria. Often used in conjunction with HashContent
- **DistributeLoad**: Load balance or sample data by distributing only a portion of data to each user-defined Relationship
- **MonitorActivity**: Sends a notification when a user-defined period of time elapses without any data coming through a particular point in the flow. Optionally send a notification when dataflow resumes.

Nifi – ETL Design Pattern

Routing and Mediation

- **RouteOnAttribute**: Route FlowFile based on the attributes that it contains.
- **ScanAttribute**: Scans the user-defined set of Attributes on a FlowFile, checking to see if any of the Attributes match the terms found in a user-defined dictionary.
- **RouteOnContent**: Search Content of a FlowFile to see if it matches any user-defined Regular Expression. If so, the FlowFile is routed to the configured Relationship.

Nifi – ETL Design Pattern

Routing and Mediation

- **ScanContent**: Search Content of a FlowFile for terms that are present in a user-defined dictionary and route based on the presence or absence of those terms. The dictionary can consist of either textual entries or binary entries.
- **ValidateXml**: Validation XML Content against an XML Schema; routes FlowFile based on whether or not the Content of the FlowFile is valid according to the user-defined XML Schema.

Nifi – ETL Design Pattern

Database Access

- **ConvertJSONToSQL**: Convert a JSON document into a SQL INSERT or UPDATE command that can then be passed to the PutSQL Processor
- **ExecuteSQL**: Executes a user-defined SQL SELECT command, writing the results to a FlowFile in Avro format
- **PutSQL**: Updates a database by executing the SQL DDM statement defined by the FlowFile's content

Nifi – ETL Design Pattern

Database Access

- ExecuteSQL
- GenerateTableFetch
- GetMongo
- GetRethinkDB
- PutCassandraQL
- PutDatabaseRecord
- PutMongo
- PutMongoRecord
- PutRethinkDB
- PutSQL
- QueryCassandra
- QueryDatabaseTable
- CaptureChangeMySQL
- DeleteHDFS
- DeleteRethinkDB

Nifi – ETL Design Pattern

Big Data Access

- **SelectHiveQL**: Executes a user-defined HiveQL SELECT command against an Apache Hive database, writing the results to a FlowFile in Avro or CSV format
- **PutHiveQL**: Updates a Hive database by executing the HiveQL DDM statement defined by the FlowFile's content

Nifi – ETL Design Pattern

Big Data Access

- FetchHBaseRow
- FetchHDFS
- GetHDFS
- GetHDFSEvents
- GetHDFSSequenceFile
- GetHBase
- PutHBaseCell
- PutHBaseJSON
- PutHBaseRecord
- PutHDFS
- PutHiveQL
- PutHiveStreaming

Nifi – ETL Design Pattern

Attribute Extraction

- **EvaluateJsonPath**: User supplies JSONPath Expressions (Similar to XPath, which is used for XML parsing/extraction), and these Expressions are then evaluated against the JSON Content to either replace the FlowFile Content or extract the value into the user-named Attribute.
- **EvaluateXPath**: User supplies XPath Expressions, and these Expressions are then evaluated against the XML Content to either replace the FlowFile Content or extract the value into the user-named Attribute.

Nifi – ETL Design Pattern

Attribute Extraction

- **EvaluateXQuery**: User supplies an XQuery query, and this query is then evaluated against the XML Content to either replace the FlowFile Content or extract the value into the user-named Attribute.
- **ExtractText**: User supplies one or more Regular Expressions that are then evaluated against the textual content of the FlowFile, and the values that are extracted are then added as user-named Attributes.

Nifi – ETL Design Pattern

System Interaction

- **ExecuteProcess**: Runs the user-defined Operating System command. The Process's StdOut is redirected such that the content that is written to StdOut becomes the content of the outbound FlowFile. This Processor is a Source Processor - its output is expected to generate a new FlowFile, and the system call is expected to receive no input. In order to provide input to the process, use the ExecuteStreamCommand Processor.

Nifi – ETL Design Pattern

System Interaction

- **ExecuteStreamCommand**: Runs the user-defined Operating System command. The contents of the FlowFile are optionally streamed to the StdIn of the process. The content that is written to StdOut becomes the content of the outbound FlowFile. This Processor cannot be used as a Source Processor - it must be fed incoming FlowFiles in order to perform its work. To perform the same type of functionality with a Source Processor, see the ExecuteProcess Processor.

Nifi – ETL Design Pattern

Data Ingestion

- GetFile
- GetFTP
- GetSFTP
- GetJMSQueue
- GetJMSTopic
- GetHTTP
- ListenHTTP
- ListenUDP
- GetHDFS
- ListHDFS / FetchHDFS
- FetchS3Object
- GetKafka
- GetMongo
- GetTwitter

Nifi – ETL Design Pattern

Data Egress / Sending Data

- PutEmail
- PutFile
- PutFTP
- PutSFTP
- PutJMS
- PutSQL
- PutKafka
- PutMongo

Nifi – ETL Design Pattern

Splitting and Aggregation

- SplitText
- SplitJson
- SplitXml
- UnpackContent
- MergeContent
- SegmentContent
- SplitContent

Nifi – ETL Design Pattern

HTTP

- GetHTTP
- ListenHTTP
- InvokeHTTP
- PostHTTP
- HandleHttpRequest / HandleHttpResponse

Nifi – ETL Design Pattern

Data Validation

- ValidateCsv
- ValidateRecord
- ValidateXml

Nifi – ETL Design Pattern

Data Cleaning

- EvaluateJsonPath
- EvaluateXPath
- EvaluateXQuery

Nifi – ETL Design Pattern

Data Profiling

- EvaluateJsonPath
- EvaluateXPath
- EvaluateXQuery
- ExtractText
- DuplicateFlowFile
- DetectDuplicate
- FuzzyHashContent
- CompareFuzzyHash
- HashAttribute
- HashContent
- IdentifyMimeType

Nifi – ETL Design Pattern

Data Reduction

- CompressContent
- Base64EncodeContent
- EncryptContent

Nifi – Working with Attributes

FlowFile Common Attributes:

- filename
- path
- uuid
- entryDate
- lineageStartDate
- fileSize

Nifi – Working with Attributes

- A FlowFile is comprised of two major pieces: content and attributes
- The attributes portion of the FlowFile represents information about the data itself, or metadata
- key-value pairs

Nifi – Working with Attributes

- Extracting Attributes
- Adding User-Defined Attributes
- Routing on Attributes
- Expression Language

Nifi – Error Handling

Exceptions within the Processor

- **ProcessException**

- Roll back the session that was being processed
- Penalize the FlowFiles that were being processed

- **Other Exceptions**

- Roll back the session that was being processed
- Penalize the FlowFiles that were being processed
- Processor may be in a bad state
- Depleting system resources
- Administratively Yield" the Processor

Nifi – Error Handling

Exceptions within a callback

- **IOException**
 - Exception will be wrapped within a `ProcessException`
- **RuntimeException**
 - Exception will propagate back to the `onTrigger` method

it is recommended that Processors that use callbacks do so within a try/catch block and catch `ProcessException` as well as any other `RuntimeException` that they expect their callback to throw

Nifi – Error Handling

Penalization vs. Yielding

When an issue occurs during processing, the framework exposes two methods to allow Processor developers to avoid performing unnecessary work: "penalization" and "yielding."

- Penalizing means the FlowFile itself to be inaccessible to downstream Processors for a period of time
- Yielding allows a Processor developer to indicate to the framework that it will not be able to perform any useful function for some period of time

Nifi – Error Handling

Session Rollback

- ProcessSession not only to access the flowfiles but also provide transactionality
- calling commit() or by calling rollback()

Nifi – General Design Considerations

- Consider the User - Simplicity is crucial
- Cohesion and Reusability - Do one thing and do it well
- Naming Conventions - Get<Service> or Get<Protocol> or Put<Service> or Put<Protocol> or lower-cased relation or Property names
- Processor Behavior Annotations
- Data Buffering - It should be avoided if at all possible

Nifi – Controlling Service

- DistributedMapCacheServer and DistributedMapCacheClient
- DBCPConnectionPool
- HiveConnectionPool
- JsonPathReader
- AvroSchemaRegistry
- CSVReader
- JettyWebSocketServer and JettyWebSocketClient

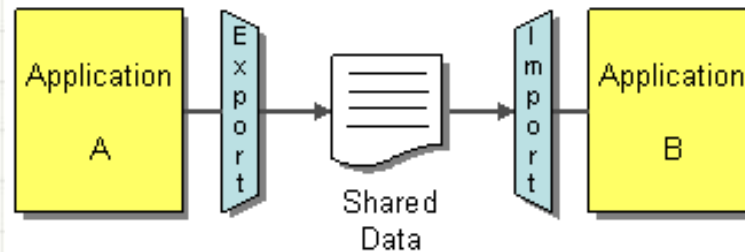
Nifi – RecordPath

Apache NiFi offers a very robust set of Processors that are capable of ingesting, processing, routing, transforming, and delivering data of any format

Nifi – Enterprise Integration Pattern

- **EIP - File Transfer**

Have each application produce files containing information that other applications need to consume

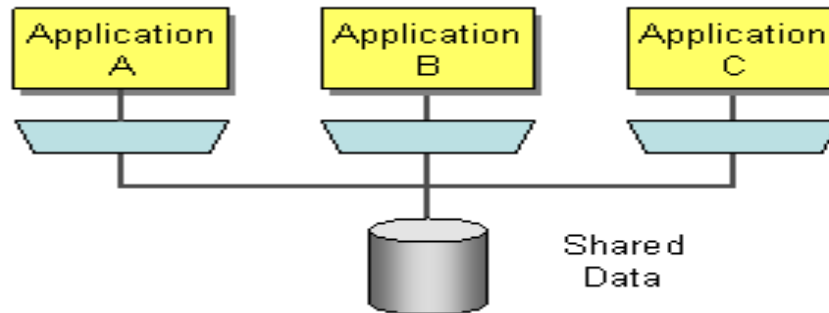


- **Nifi – JSON, XML, TEXT, CSV, FlowFile**

Nifi – Enterprise Integration Pattern

- **EIP - Shared Database**

Integrate applications by having them store their data in a single Shared Database

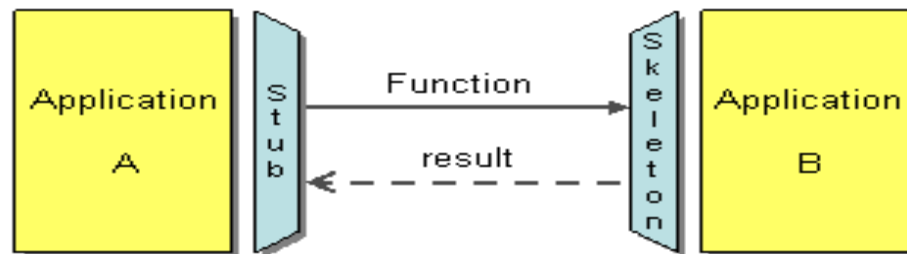


- **Nifi – Content Repository, Flow Repository, Provenance Repository**

Nifi – Enterprise Integration Pattern

- **EIP - Remote Procedure Invocation**

Provide an interface to allow other applications to interact with the running application

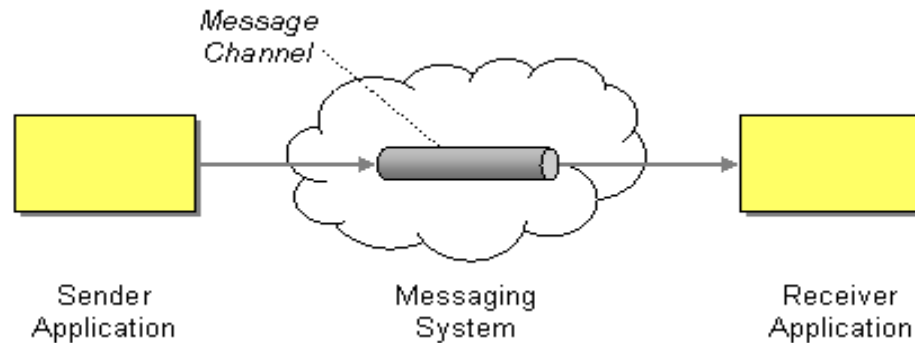


- **Nifi – Site-to-Site Remote Processor Group**

Nifi – Enterprise Integration Pattern

- **EIP - Message Channel**

Connect the applications using a Message Channel, where one application writes information to the channel and the other one reads that information from the channel

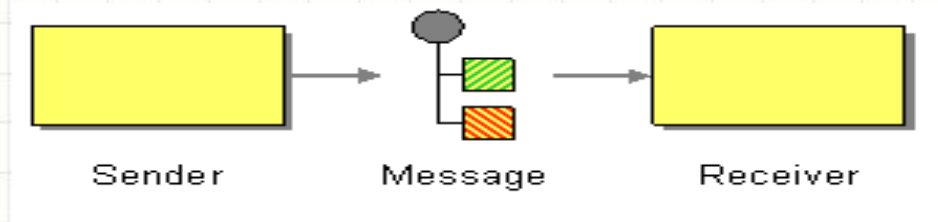


- **Nifi – Connection Queue**

Nifi – Enterprise Integration Pattern

- **EIP - Message**

Package the information into a Message, a data record that the messaging system can transmit through a message channel

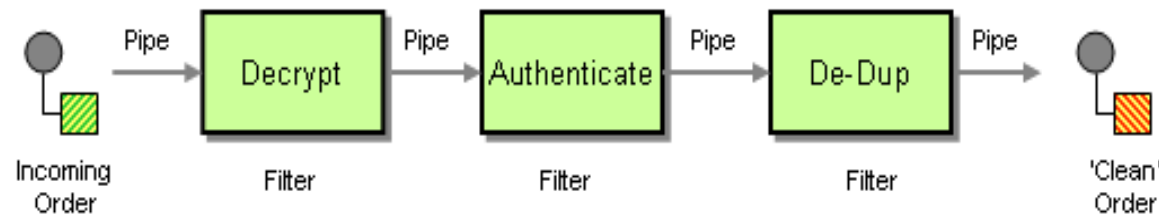


- **Nifi – FlowFile**

Nifi – Enterprise Integration Pattern

- **EIP - Pipes and Filters**

Use the Pipes and Filters architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (Filters) that are connected by channels (Pipes)

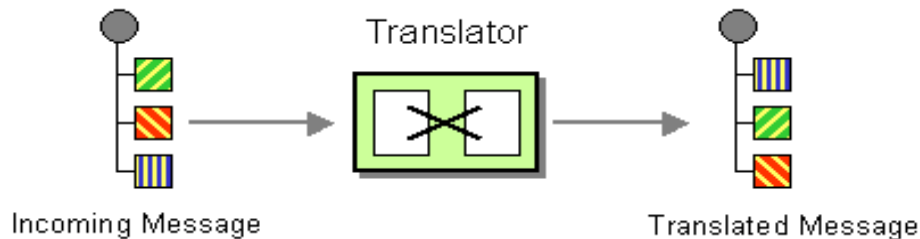


- **Nifi – CompressContent, EncriptContent and UnpackContent,**

Nifi – Enterprise Integration Pattern

- **EIP - Message Translator**

Use a special filter, a Message Translator, between other filters or applications to translate one data format into another

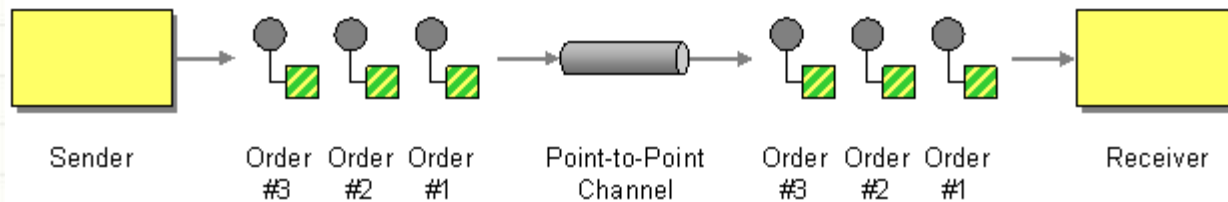


- **Nifi – ConvertAvroToJSON, ConvertAvroToORC, ReplaceText, TransformXml**

Nifi – Enterprise Integration Pattern

- **EIP - Point-to-Point Channel**

Send the message on a Point-to-Point Channel, which ensures that only one receiver will receive a particular message

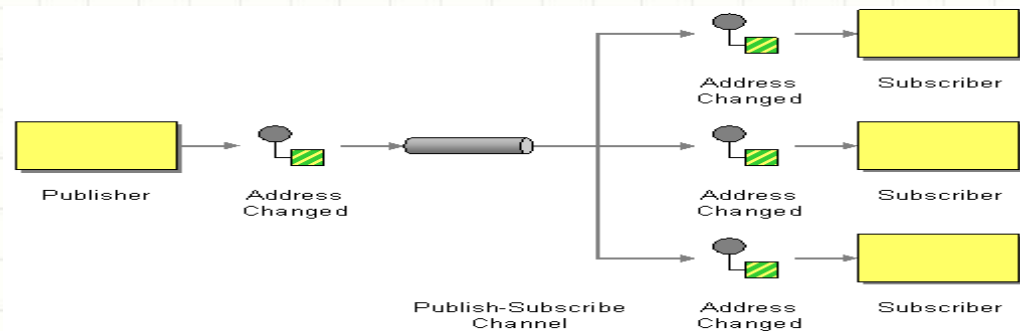


- **Nifi – Connection Queue**

Nifi – Enterprise Integration Pattern

- **EIP - Publish-Subscribe Channel**

Send the event on a Publish-Subscribe Channel, which delivers a copy of a particular event to each receiver

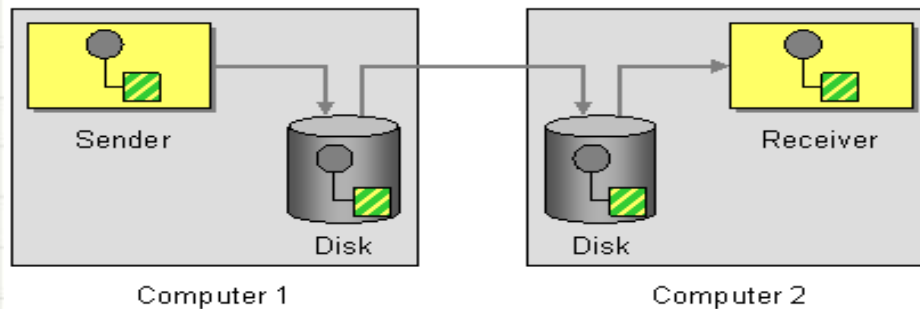


- **Nifi – Connection Queue, DistributeLoad**

Nifi – Enterprise Integration Pattern

- **EIP - Guaranteed Delivery**

Use Guaranteed Delivery to make messages persistent so that they are not lost even if the messaging system crashes

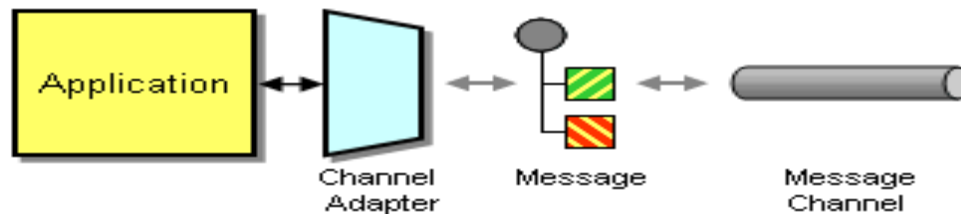


- **Nifi – GetJMSQueue, Connection Queue**

Nifi – Enterprise Integration Pattern

- **EIP - Channel Adapter**

Use a Channel Adapter that can access the application's API or data and publish messages on a channel based on this data, and that likewise can receive messages and invoke functionality inside the application

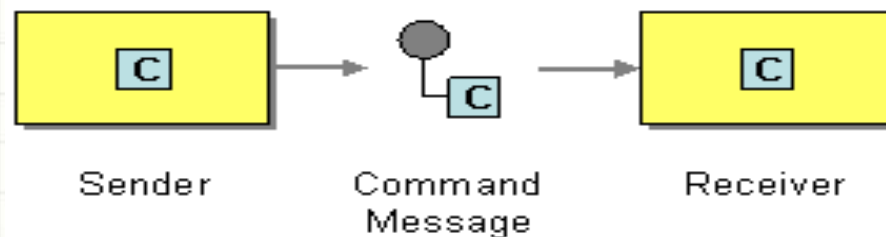


- **Nifi – GetXXX and PutXXX Processor**

Nifi – Enterprise Integration Pattern

- **EIP - Command Message**

Use a Command Message to reliably invoke a procedure in another application

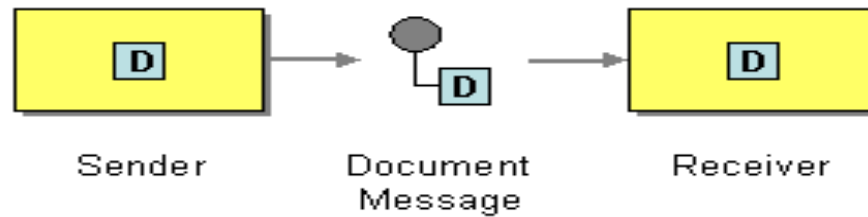


- **Nifi – ExecuteScript**

Nifi – Enterprise Integration Pattern

- **EIP - Document Message**

Use a Document Message to reliably transfer a data structure between applications

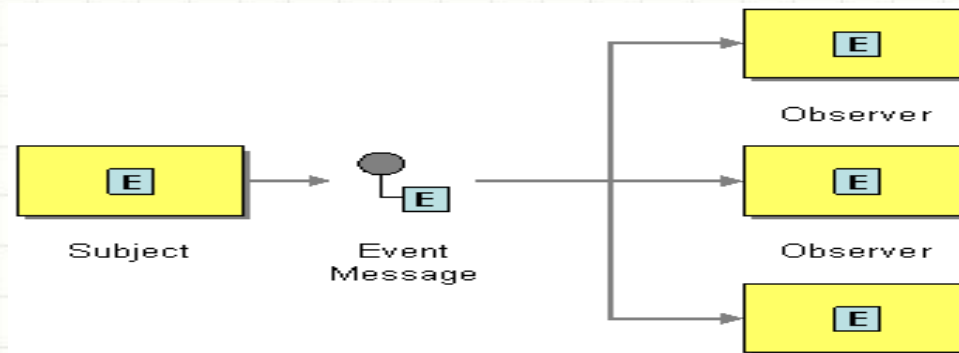


- **Nifi – FlowFile**

Nifi – Enterprise Integration Pattern

- **EIP - Event Message**

Use an Event Message for reliable, asynchronous event notification between applications

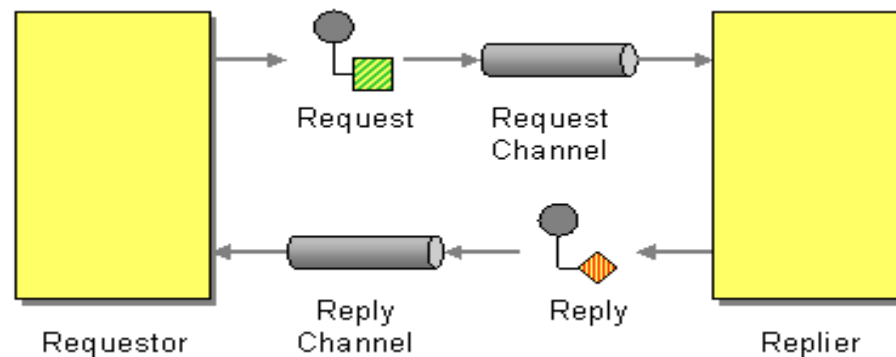


- **Nifi – Provenance Event and Component Lifecycle**

Nifi – Enterprise Integration Pattern

- **EIP - Request-Reply**

Send a pair of Request-Reply messages, each on its own channel

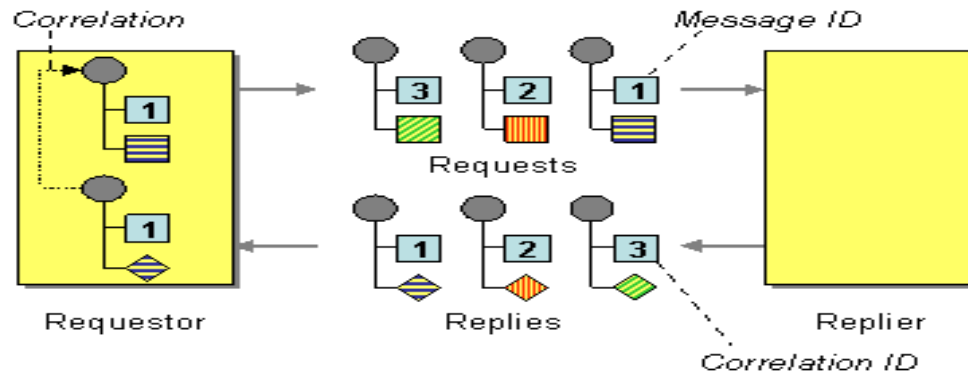


- **Nifi – HandleHttpRequest and HandleHttpResponse**

Nifi – Enterprise Integration Pattern

- **EIP - Correlation Identifier**

Each reply message should contain a Correlation Identifier, a unique identifier that indicates which request message this reply is for

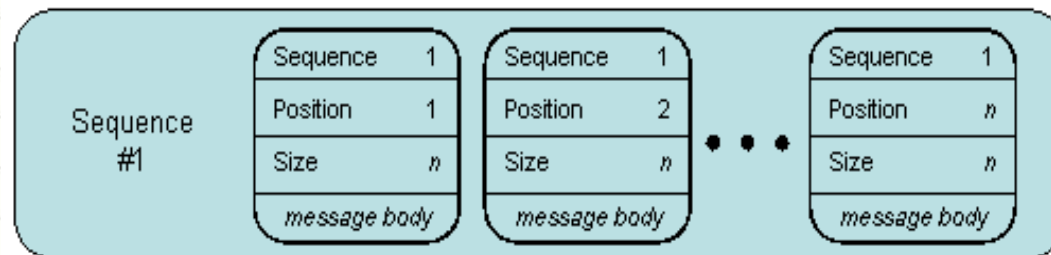


- **Nifi – FlowFile UUID attribute**

Nifi – Enterprise Integration Pattern

- **EIP - Message Sequence**

Whenever a large set of data may need to be broken into message-size chunks, send the data as a Message Sequence and mark each message with sequence identification fields

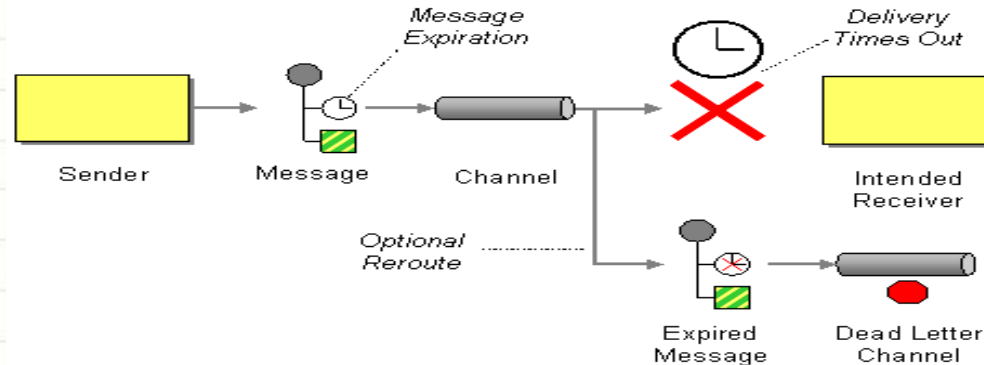


- **Nifi – SegmentContent**

Nifi – Enterprise Integration Pattern

- EIP - Message Expiration

Set the Message Expiration to specify a time limit how long the message is viable



- Nifi – FlowFile and Connection Queue

Nifi – Enterprise Integration Pattern

- **EIP - Format Indicator**

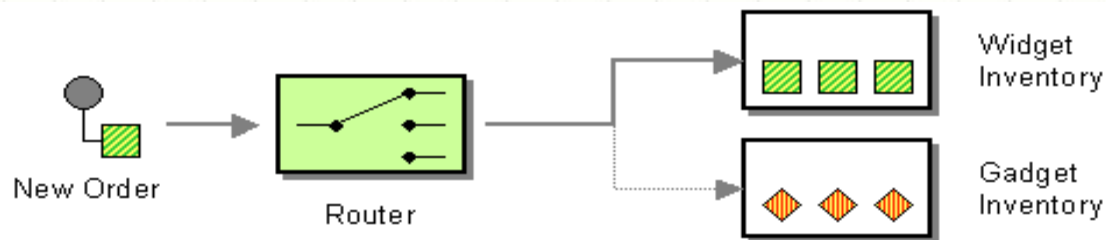
Design a data format that includes a Format Indicator, so that the message specifies what format it is using

- **Nifi – IdentifyMimeType**

Nifi – Enterprise Integration Pattern

- **EIP - Content-Based Router**

Use a Content-Based Router to route each message to the correct recipient based on message content

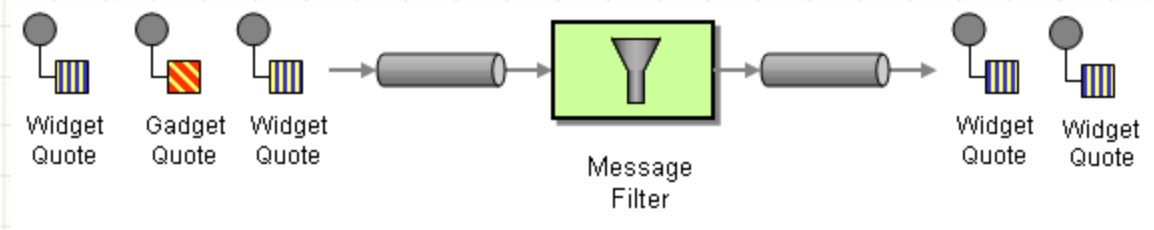


- **Nifi – RouteOnContent, RouteText**

Nifi – Enterprise Integration Pattern

- **EIP - Message Filter**

Use a special kind of Message Router, a Message Filter, to eliminate undesired messages from a channel based on a set of criteria

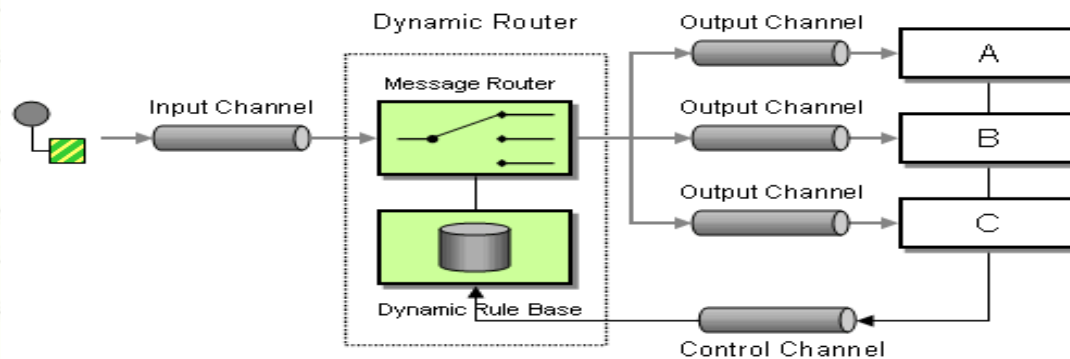


- **Nifi – ScanContent, ScanAttribute**

Nifi – Enterprise Integration Pattern

- **EIP - Dynamic Router**

Use a Dynamic Router, a Router that can self-configure based on special configuration messages from participating destinations

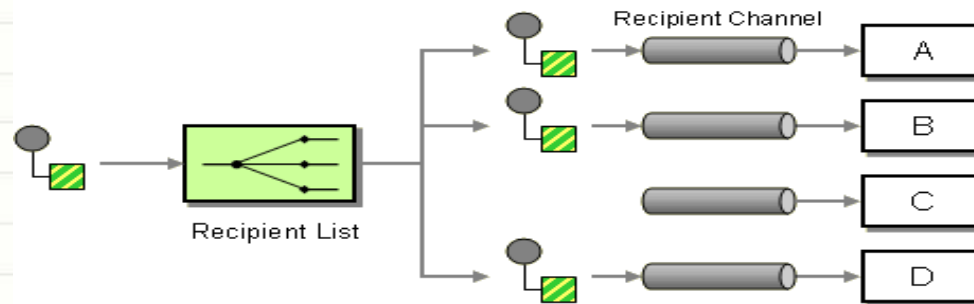


- **Nifi – RouteOnAttribute, RouteOnContent, RouteText**

Nifi – Enterprise Integration Pattern

- **EIP - Recipient List**

Define a channel for each recipient. Then use a Recipient List to inspect an incoming message, determine the list of desired recipients, and forward the message to all channels associated with the recipients in the list

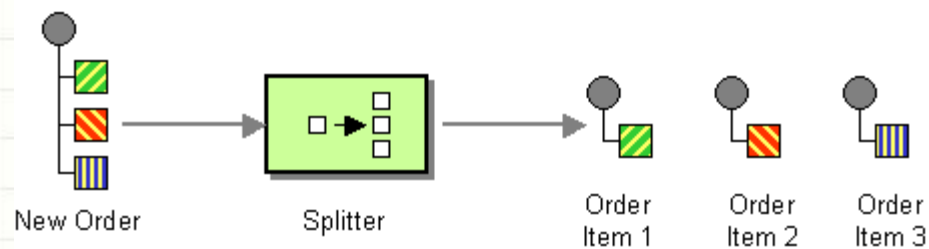


- **Nifi – RouteText**

Nifi – Enterprise Integration Pattern

- **EIP - Splitter**

Use a Splitter to break out the composite message into a series of individual messages, each containing data related to one item

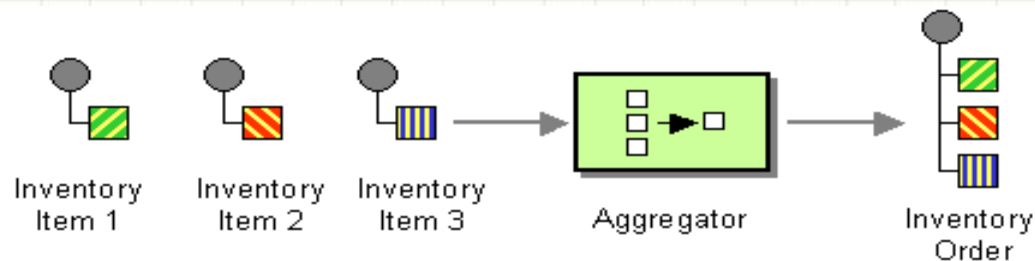


- **Nifi – SplitText, SplitXml, SplitJson, SplitContent and SplitAvro**

Nifi – Enterprise Integration Pattern

- **EIP - Aggregator**

Use a stateful filter, an Aggregator, to collect and store individual messages until a complete set of related messages has been received. Then, the Aggregator publishes a single message distilled from the individual messages

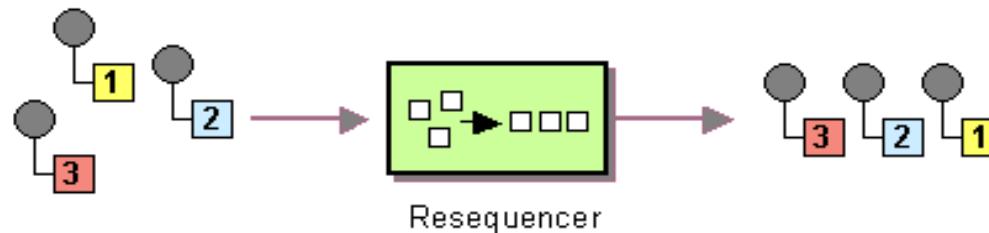


- **Nifi – MergeContent**

Nifi – Enterprise Integration Pattern

- **EIP - Resequencer**

Use a stateful filter, an Aggregator, to collect and store individual messages until a complete set of related messages has been received. Then, the Aggregator publishes a single message distilled from the individual messages. Use a stateful filter, a Resequencer, to collect and re-order messages so that they can be published to the output channel in a specified order.

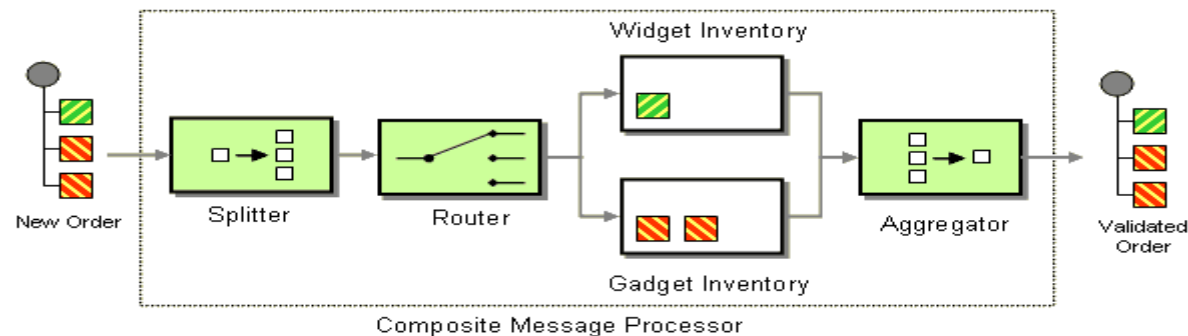


- **Nifi – MergeContent, EnforceOrder**

Nifi – Enterprise Integration Pattern

- **EIP - Composed Message Processor**

Use Composed Message Processor to process a composite message. The Composed Message Processor splits the message up, routes the sub-messages to the appropriate destinations and re-aggregates the responses back into a single message

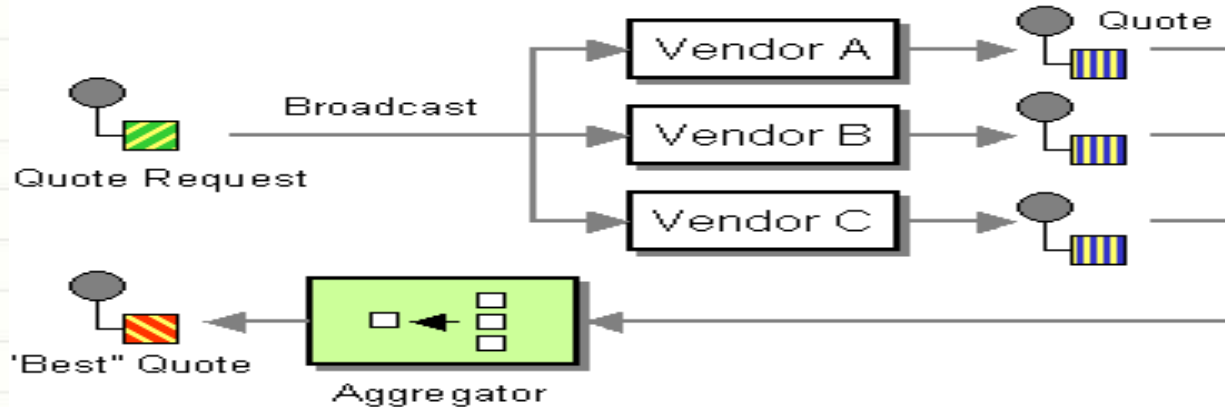


- **Nifi – SegmentContent and MergeContent**

Nifi – Enterprise Integration Pattern

- **EIP - Scatter-Gather**

Use a Scatter-Gather that broadcasts a message to multiple recipients and re-aggregates the responses back into a single message

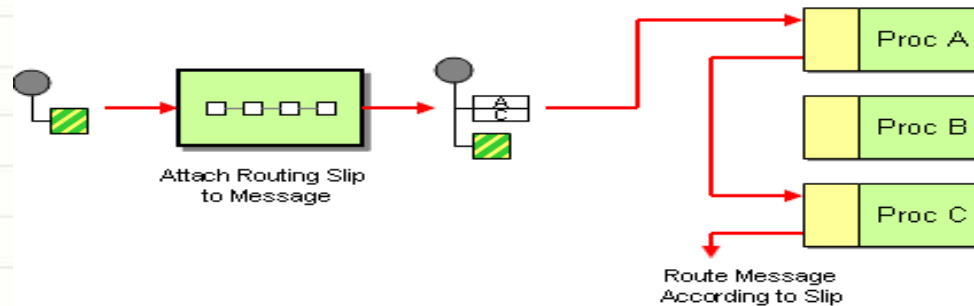


- **Nifi – SegmentContent and MergeContent**

Nifi – Enterprise Integration Pattern

- **EIP - Routing Slip**

Attach a Routing Slip to each message, specifying the sequence of processing steps. Wrap each component with a special message router that reads the Routing Slip and routes the message to the next component in the list

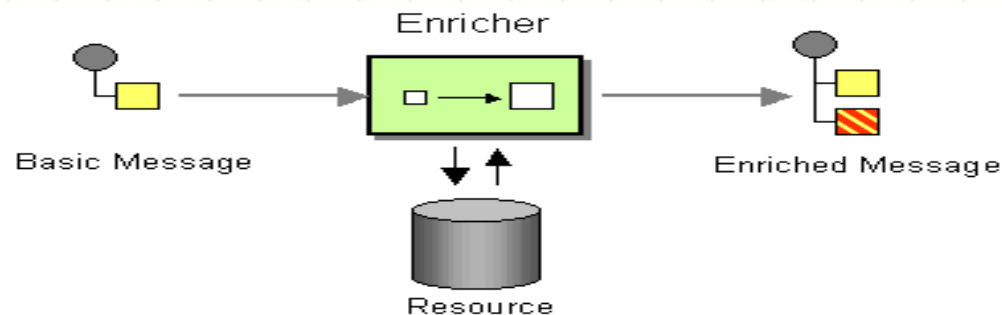


- **Nifi – ScanContent and ScanAttribute**

Nifi – Enterprise Integration Pattern

- **EIP - Content Enricher**

Use a specialized transformer, a Content Enricher, to access an external data source in order to augment a message with missing information

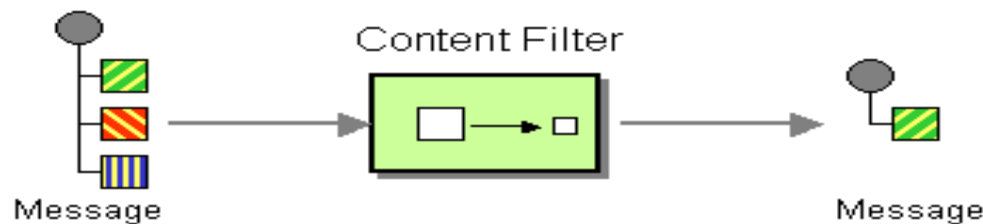


- **Nifi – GepEnrichIP, ModifyHTMLElement, ModifyBytes**

Nifi – Enterprise Integration Pattern

- **EIP - Content Filter**

Use a Content Filter to remove unimportant data items from a message leaving only important items

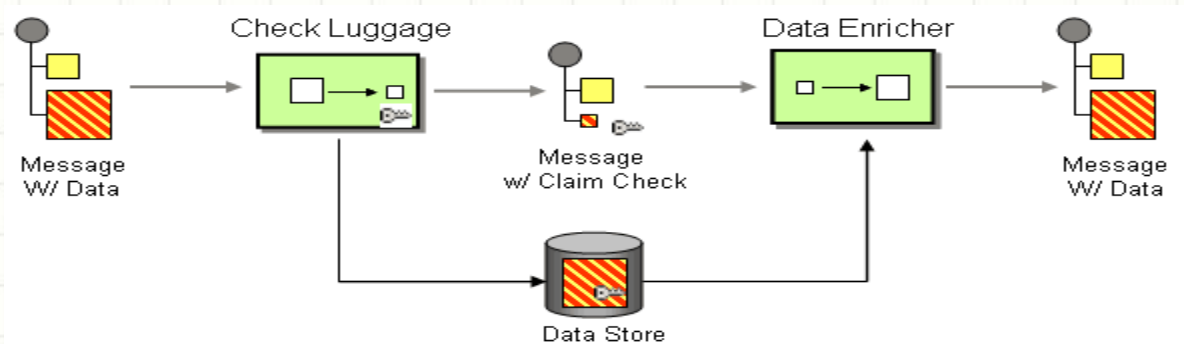


- **Nifi – ReplaceText, ReplaceTextWithMapping**

Nifi – Enterprise Integration Pattern

- **EIP - Claim Check**

Store message data in a persistent store and pass a Claim Check to subsequent components. These components can use the Claim Check to retrieve the stored information

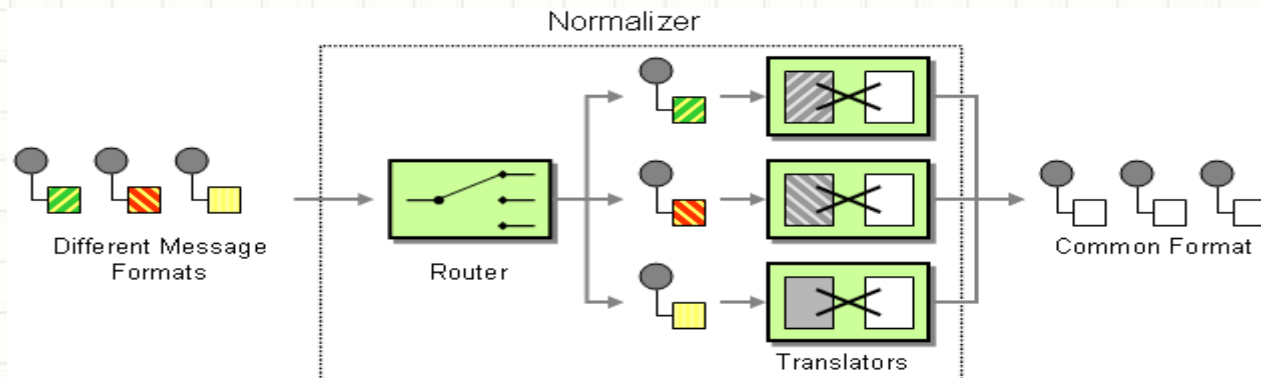


- **Nifi – ValidateXML and ValidateCSV**

Nifi – Enterprise Integration Pattern

- **EIP - Normalizer**

Use a Normalizer to route each message type through a custom Message Translator so that the resulting messages match a common format

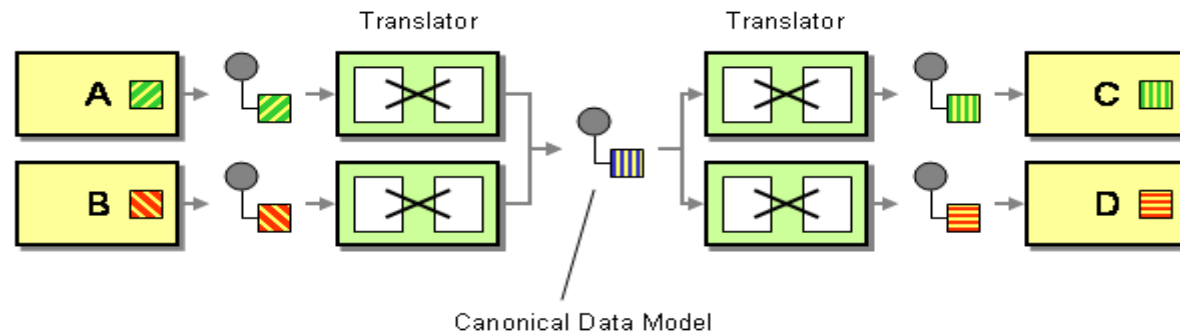


- **Nifi – FlowFile**

Nifi – Enterprise Integration Pattern

- **EIP - Canonical Data Model**

Design a Canonical Data Model that is independent from any specific application. Require each application to produce and consume messages in this common format

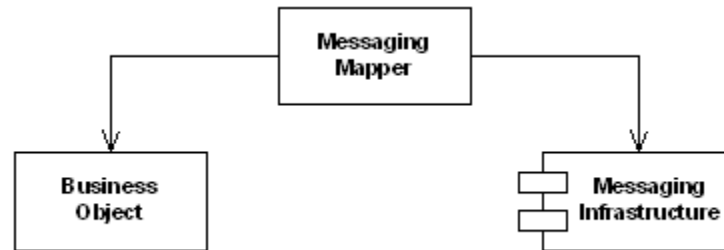


- **Nifi – JSON, XML**

Nifi – Enterprise Integration Pattern

- **EIP - Messaging Mapper**

Create a separate Messaging Mapper that contains the mapping logic between the messaging infrastructure and the domain objects. Neither the objects nor the infrastructure have knowledge of the Messaging Mapper's existence

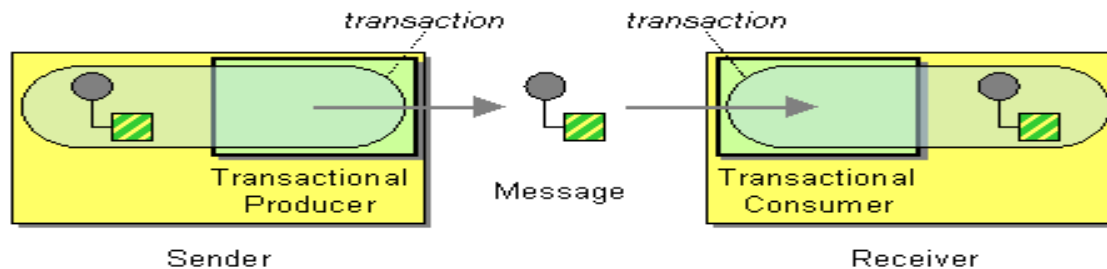


- **Nifi – TransformXML and JoltTransformJSON**

Nifi – Enterprise Integration Pattern

- **EIP - Transactional Client**

Use a Transactional Client—make the client's session with the messaging system transactional so that the client can specify transaction boundaries

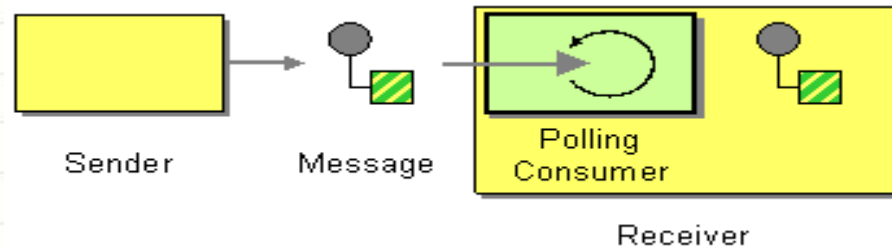


- **Nifi – ProcessorSession**

Nifi – Enterprise Integration Pattern

- **EIP - Polling Consumer**

The application should use a Polling Consumer, one that explicitly makes a call when it wants to receive a message

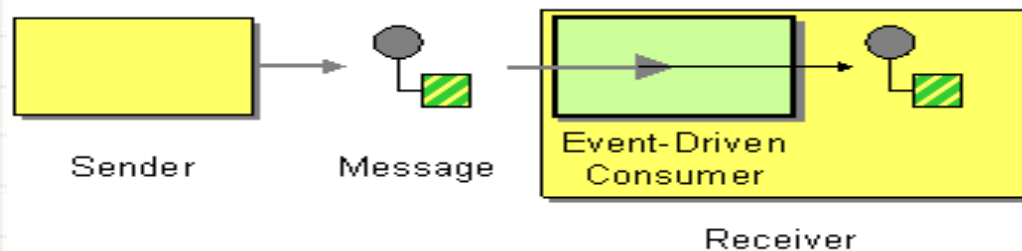


- **Nifi – Site To Site (Pull)**

Nifi – Enterprise Integration Pattern

- **EIP - Event-Driven Consumer**

The application should use an Event-Driven Consumer, one that is automatically handed messages as they're delivered on the channel

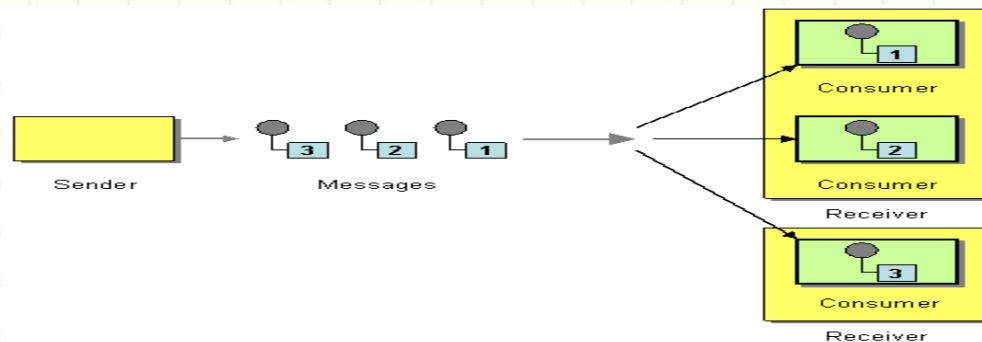


- **Nifi – Most of Processors**

Nifi – Enterprise Integration Pattern

- **EIP - Competing Consumers**

Create multiple Competing Consumers on a single channel so that the consumers can process multiple messages concurrently

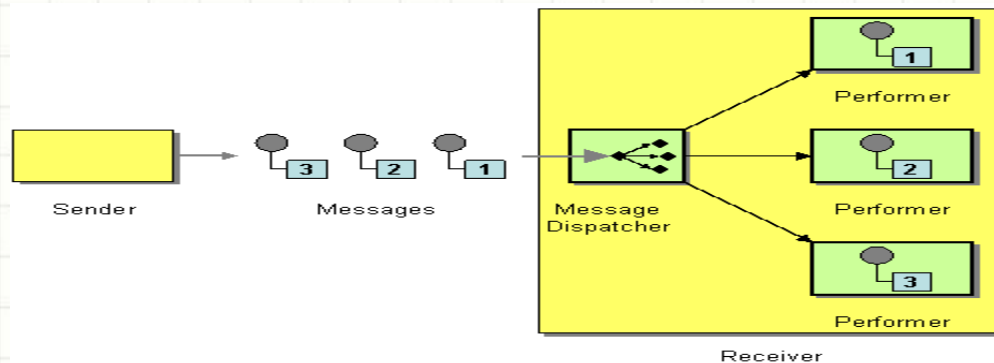


- **Nifi – GetKafka and ConsumeKafka**

Nifi – Enterprise Integration Pattern

- **EIP - Message Dispatcher**

Create a Message Dispatcher on a channel that will consume messages from a channel and distribute them to performers

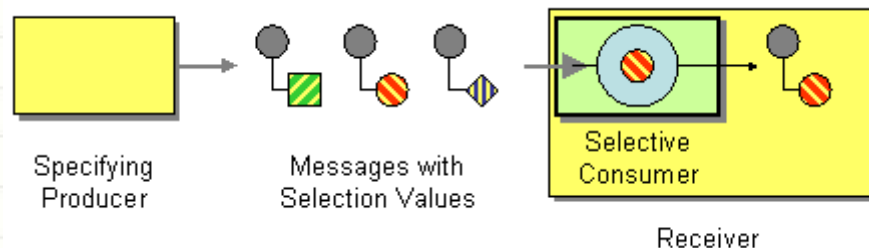


- **Nifi – GetJMSQueue and GetJMSTopic and DistributeLoad**

Nifi – Enterprise Integration Pattern

- **EIP - Selective Consumer**

Make the consumer a Selective Consumer, one that filters the messages delivered by its channel so that it only receives the ones that match its criteria

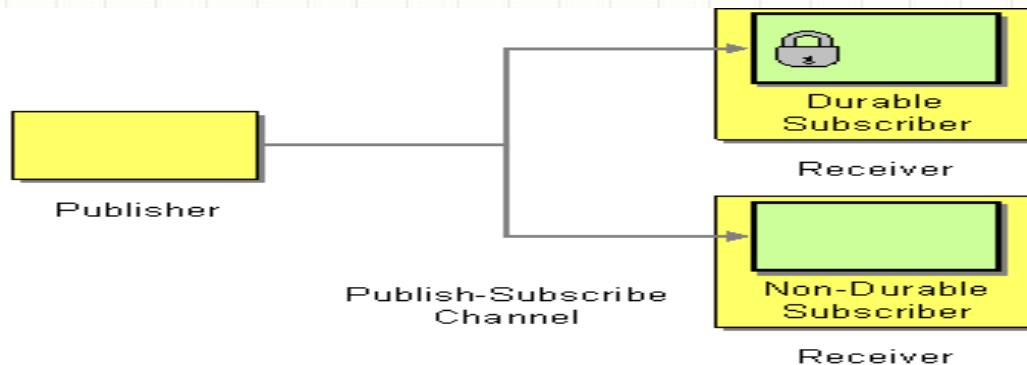


- **Nifi – GetJMSQueue and GetJMSTopic**

Nifi – Enterprise Integration Pattern

- **EIP - Durable Subscriber**

Use a Durable Subscriber to make the messaging system save messages published while the subscriber is disconnected

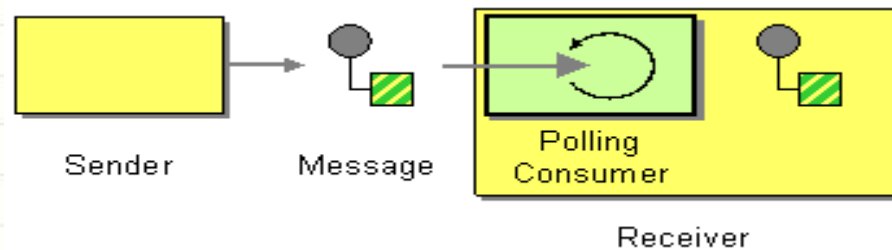


- **Nifi – GetKafka**

Nifi – Enterprise Integration Pattern

- **EIP - Idempotent Receiver**

Design a receiver to be an Idempotent Receiver--one that can safely receive the same message multiple times

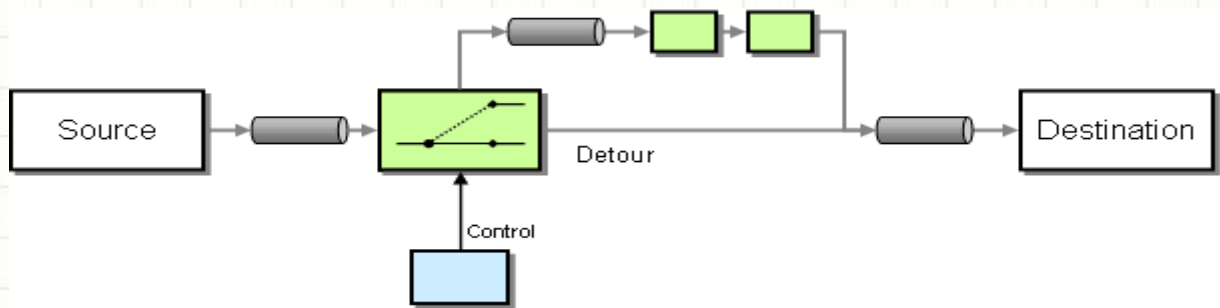


- **Nifi – DetectDuplicate**

Nifi – Enterprise Integration Pattern

- **EIP - Detour**

Construct a Detour with a context-based router controlled via the Control Bus. In one state the router routes incoming messages through additional steps while in the other it routes messages directly to the destination channel

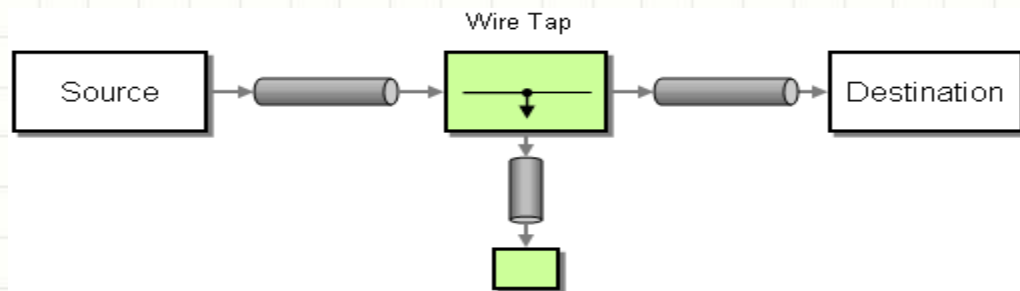


- **Nifi – RouteOnAttribute and RouteOnContent**

Nifi – Enterprise Integration Pattern

- **EIP - Wire Tap**

Insert a simple Recipient List into the channel that publishes each incoming message to the main channel and a secondary channel

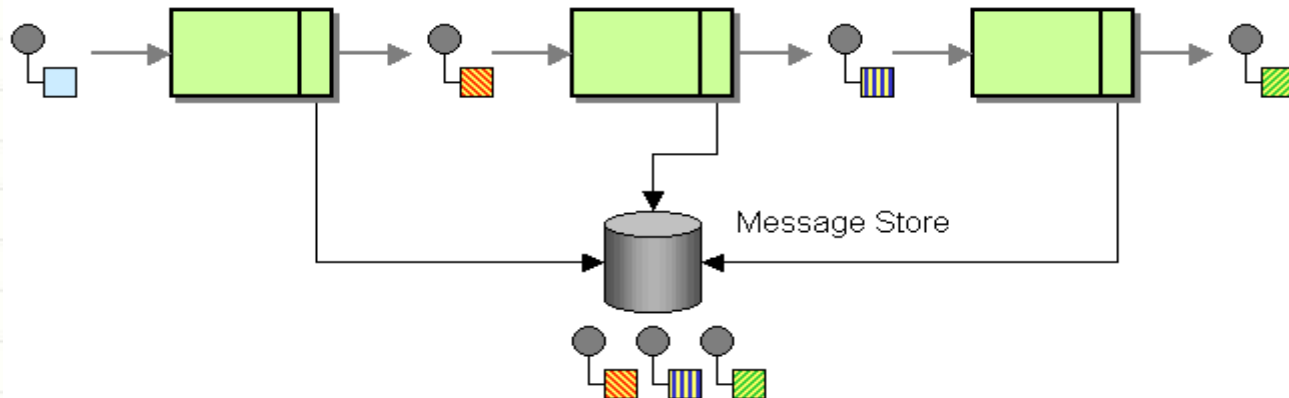


- **Nifi – DistributeLoad**

Nifi – Enterprise Integration Pattern

- **EIP - Message Store**

Use a Message Store to capture information about each message in a central location

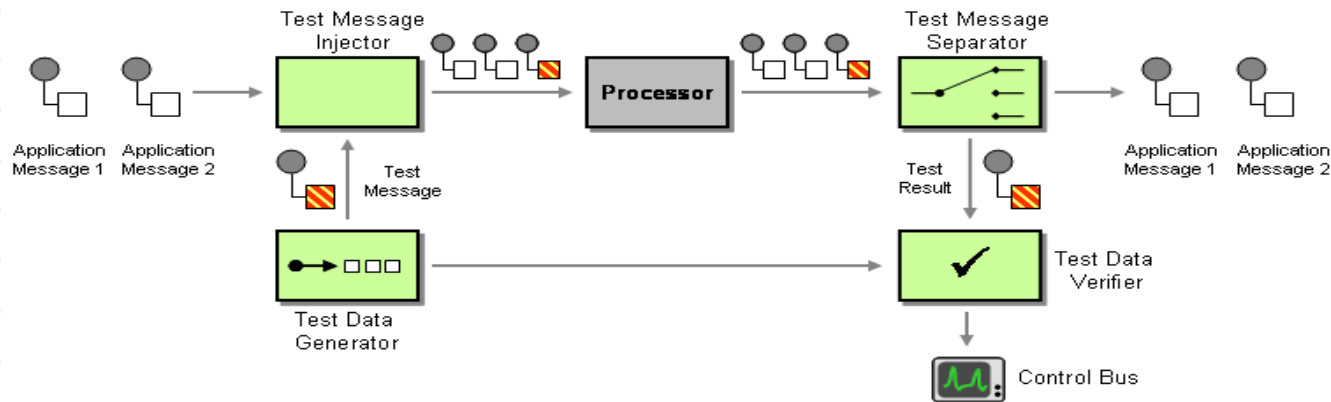


- **Nifi – Content Repository, Provenance Repository**

Nifi – Enterprise Integration Pattern

- EIP - Test Message

Use Test Message to assure the health of message processing components



- Nifi – JSON, XML, TEXT, CSV, FlowFile

Nifi – Enterprise Integration Pattern

- **EIP - Channel Purger**

Use a Channel Purger to remove unwanted messages from a channel



- **Nifi – Set FlowFiles Expiration**

Remove specific FlowFile from a queue

Use ExecuteScript to filter incoming FlowFiles

```
def flowFile = session.get()
if(!flowFile) return

if(flowFile.getAttribute("uuid")) {
    session.transfer(flowFile, REL_SUCCESS)
    return
}

session.transfer(flowFile, REL_FAILURE)
```

How to wait until notified to move to the next step

Use wait and notify processor



How to Put/Get to/from a DistributedCache

Use FetchDistributedMapCache and PutDistributedMapCache



How to sort out-of-order FlowFiles

**Use EnforceOrder Processor and FIFO Prioritizer
Connection**

How to lookup a configure path from an external xml file

Use ExecuteScript

```
FlowFile flowFile = session.get()
```

```
if (!flowFile) return
```

```
def filename = flowFile.getAttribute('filename')
```

```
def configuration = new XmlSlurper().parse("/tmp/config.xml")
```

```
def dataFlow = configuration.dataFlows.dataFlow.find {  
Pattern.compile(it.filePattern.text()).matcher(filename).matches() }
```

How to join two data sources

- ExecuteSQL for one data source and one for another data source, then use MergeContent to join via correlation attribute name
- Write Custom Java code with two JDBC connections and join the data
- Ingest one data source into HDFS with that ExecuteSQL query, Ingest the another data source into HDFS with ExecuteSQL query, then write a Hive query on top of that and run a third ExecuteSQL query
- Push them all to HBase and do Phoenix queries
- Add the column to specify attribute, then use MergeContent and QueryRecord

Nifi – Expression Language

- **Overview**

Placing FlowFile attributes on a FlowFile do not provide much benefit if the user is unable to make use of them. The NiFi Expression Language provides the ability to reference these attributes, compare them to other values, and manipulate their values

Nifi – Expression Language

- **Structure of a NiFi Expression**

- Start delimiter `${` and ends with the end delimiter `}`
- Any FlowFile attribute can be referenced using the Expression Language
- If the attribute name contains a “special character,” the attribute name must be escaped by quoting it
- Example:
 - ✓ `${filename}`
 - ✓ `${filename:toUpper()}`
 - ✓ `${filename:toUpper():equals('HELLO.TXT')}`
 - ✓ `${"my attribute"}`
 - ✓ `${hostname():toUpper()}`
 - ✓ `{filename:equals(${uuid})}`

Nifi – Expression Language

- **Data Type**

- String
- Number
- Decimal
- Date
- Boolean

Nifi – Expression Language

- **Boolean Logic**

- isNull
- notNull
- isEmpty
- equals
- equalsIgnoreCase
- gt
- ge
- lt
- le
- and
- or
- not

Nifi – Expression Language

- **String Manipulation**

- toUpper
- toLower
- trim
- substring
- substringBefore
- substringBeforeLast
- substringAfter
- substringAfterLast
- getDelimitedField

Nifi – Expression Language

- **String Manipulation**

- append
- prepend
- replace
- replaceFirst
- replaceAll
- replaceNull
- replaceEmpty
- length

Nifi – Expression Language

- **Encode/Decode Functions**

- escapeJson
- escapeXml
- escapeCsv
- escapeHtml3
- escapeHtml4
- unescapeJson
- unescapeXml
- unescapeCsv
- unescapeHtml3
- unescapeHtml4
- urlEncode
- urlDecode
- base64Encode
- base64Decode

Nifi – Expression Language

- **Searching**

- `startsWith`
- `endsWith`
- `contains`
- `in`
- `find`
- `matches`
- `indexOf`
- `lastIndexOf`
- `jsonPath`

Nifi – Expression Language

- **Mathematical Operations and Numeric Manipulation**

- plus
- minus
- multiply
- divide
- mod
- toRadix
- fromRadix
- random
- math

Nifi – Expression Language

- **Date Manipulation**

- format
- toDate
- now

Nifi – Expression Language

- **Type Coercion**

- toString
- toNumber
- toDecimal

Nifi – Expression Language

- **Subjectless Functions**

- ip
- hostname
- UUID
- nextInt
- literal

Nifi – Expression Language

- **Evaluating Multiple Attributes**

- anyAttribute
- anyAttribute
- anyMatchingAttribute
- allMatchingAttributes
- anyDelineatedValue
- allDelineatedValues
- join
- count

Nifi – NiFi Archives (NARs)

A NAR allows several components and their dependencies to be packaged together into a single package. The NAR package is then provided ClassLoader isolation from other NAR packages. Developers should always deploy their NiFi components as NAR packages.

Nifi – Repository

- content_repository
- database_repository
- flowfile_repository
- provenance_repository
- work directory
- logs directory

Nifi – Configuration Best Practices

- Maximum File Handles - `/etc/security/limits.conf` (50000 for both hard and soft)
- Maximum Forked Processes - `/etc/security/limits.conf` (10000 for both hard and soft)
- Increase the number of TCP socket ports available
- Never want NiFi to swap `/etc/sysctl.conf`

Nifi – Security

- "Security properties" heading for certificate, truststore in the nifi.properties file
- Enable the User Interface to be accessed over HTTPS instead of HTTP
- The web server can be configured to require certificate based client authentication for users accessing the User Interface
- Secure Site-to-Site connections and inner-cluster communications
- Supports user authentication via client certificates or via username/password (LDAP or Kerberos)
- Multi-Tenant Authorization via User, Group and Access Policies

Nifi – Data Protection

- EncryptContent processor
- Key Derivation Functions
 - NiFi Legacy KDF
 - OpenSSL PKCS#5 v1.5 EVP_BytesToKey
 - Bcrypt
 - Scrypt
 - PBKDF2
- Salt and IV Encoding
- Encrypted Passwords in Configuration Files
- Password Key Derivation

Nifi – Clustering

- NiFi Cluster Coordinator
- Primary Node - Isolated Processors
- Nodes
- Heartbeats
- Zookeeper
- Dealing with Disconnected Nodes
- Flow Election - Vote correct" version of the flow
- Basic Cluster Setup

Advanced NiFi - FlowFile Repository

- FlowFiles are held in a hash map in the JVM memory
- FlowFile Repository is Write-Ahead Log to provide durability of data
- FlowFile Repository stores the metadata such as all the attributes associated with the FlowFile, a pointer to the actual content of the FlowFile, state of the FlowFile
- NiFi recovers a FlowFile by restoring a snapshot of the FlowFile, A snapshot is automatically taken periodically by the system
- "swapping" FlowFiles in the queue
- FlowFile transactions never modify the original content

Advanced NiFi - Content Repository

- Immutability and copy-on-write
- Hold the FlowFile's content on disk and only read it into JVM memory when it's needed
- FlowFile expiration is handled by dedicated thread in NiFi

Advanced NiFi – Best Practice

- The option for Provenance and content repos to stripe the information across multiple physical partitions
- Analyze the contents of a FlowFile as few times as possible and instead extract key information from the contents into the attributes of the FlowFile
- Attributes are kept in-memory and updating the FlowFile repository is much faster than updating the Content repository

Advanced NiFi - Provenance Repository

- Provide the Data Lineage
- Provenance event
- It copies all the FlowFile's attributes and the pointer to provenance repo
- Replay the data
- Not copying the content in the Content Repo

Nifi – Open Sources

- **NiFi Audio bundle** - <https://github.com/simonellistonball/nifi-audio-bundle>
- **NiFi Flow Deployment Automation** - <https://github.com/aperepel/nifi-api-deploy>
- **NIFI - CoreNLP - Processor** - <https://github.com/tspannhw/nifi-corenlp-processor>
- **Social Media NiFi Templates** - <https://github.com/tspannhw/ApacheNiFiTemplates>
- **Nifi TCPDump Processor** - <https://github.com/abajwa-hw/nifi-network-processor>
- **NiFi for Neo4J** - <https://github.com/jonathantelfer/nifi-neo4j>
- **Launching Spark Jobs from NiFi** - <https://github.com/diegobaez/PUBLIC/tree/master/NiFi-SnapSpark>
- **Dynamic List Filtering with Apache NiFi** - Dynamic List Filtering with Apache NiFi