Apache Nifi

1 - Introduction

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.

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.

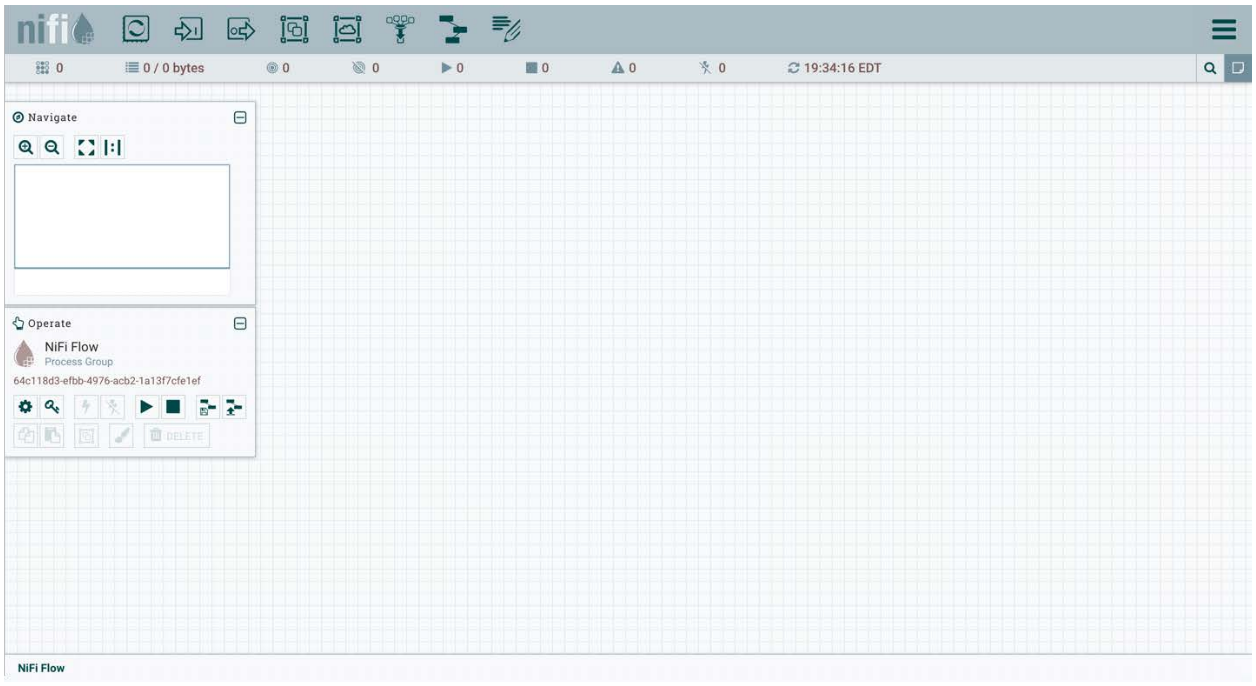
Download & install Nifi

You can download apache nifi from this link - https://nifi.apache.org/download.html
Which is free and last two version you can see.

Supposing you have a Java runtime installed, you can get NiFi running by using the *bin/nifi.sh* script (on Linux or Mac) or *bin/run-nifi.bat* for windows.

The service is a bit slow to start, so do not hurry before reaching http://localhost:8080/nifi/.

Now start you see this window.



Component of Nifi

NiFi’s fundamental design concepts closely relate to the main ideas of Flow Based Programming [fbp]. 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

high level overview of key NiFi features

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 to 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.

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.

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.

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.

Designed for 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

Processors, Controller Services, Reporting Tasks, Prioritizers, Customer User Interfaces

Classloader Isolation

For any component-based system, dependency nightmares 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 will be discussed in greater detail in the developer's guide.

Nifi Processores for our Application

1- GenrateFlowfile

This processor creates FlowFiles with random data or custom content. GenerateFlowFile is useful for load testing, configuration, and simulation.

Remember Point-

- In Scheduling menu set **Run scheduling** property if not set it generate more than 1000 flowfile in second.
- if you not set customeText it generate null flow-file.

- It have only one relationship succes **.**

Name	Default Value	Allowable Values	Description
File Size	0B		The size of the file that will be used
Batch Size	1		The number of FlowFiles to be transferred in each invocation
Data Format	Text	Binary Text	Specifies whether the data should be Text or Binary
Unique FlowFiles	false	true false	If true, each FlowFile that is generated will be unique. If false, a random value will be generated and all FlowFiles will get the same content but this offers much higher throughput
Custom Text			If Data Format is text and if Unique FlowFiles is false, then this custom text will be used as content of the generated FlowFiles and the File Size will be ignored. Finally, if Expression Language is used, evaluation will be performed only once per batch of generated FlowFiles **Supports Expression Language: true (will be evaluated using variable registry only)**
Character Set	UTF-8		Specifies the character set to use when writing the bytes of Custom Text to a flow file.

2- **GetFile –** This processor use when we want to upload file from our local directory as flowfile .

- Set this attribute must be which in bold capture.

Name	Default Value	Allowable Values	Description
Input Directory			The input directory from which to pull files **Supports Expression Language: true** **(will be evaluated using variable registry only)**
File Filter	[^\.].*		Only files whose names match the given regular expression will be picked up
Path Filter			When Recurse Subdirectories is true, then only subdirectories whose path matches the given regular expression will be scanned
Batch Size	10		The maximum number of files to pull in each iteration
Keep Source File	false	true false	If true, the file is not deleted after it has been copied to the Content Repository; this causes the file to be picked up continually and is useful for testing purposes. If not keeping original NiFi will need write permissions on the directory it is pulling from otherwise it will ignore the file.
Recurse Subdirectories	true	true false	Indicates whether or not to pull files from subdirectories
Polling Interval	0 sec		Indicates how long to wait before performing a directory listing
Ignore Hidden Files	true	true false	Indicates whether or not hidden files should be ignored
Minimum File Age	0 sec		The minimum age that a file must be in order to be pulled; any file younger than this amount of time

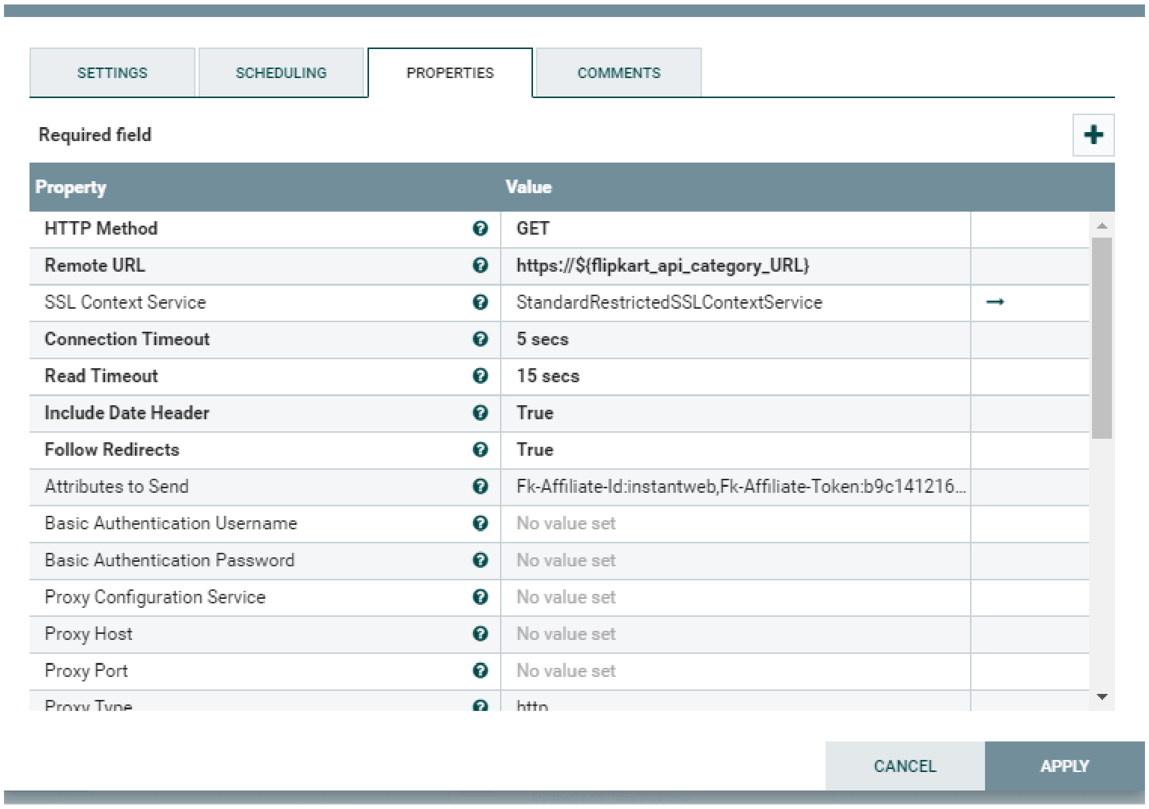
			(according to last modification date) will be ignored
Maximum File Age			The maximum age that a file must be in order to be pulled; any file older than this amount of time (according to last modification date) will be ignored
Minimum File Size	0 B		The minimum size that a file must be in order to be pulled
Maximum File Size			The maximum size that a file can be in order to be pulled

Relationships:

Name	**Description**
success	All files are routed to success

3- **invokeHTTP –** using this processor we can fetch data from APIs . for this you can use GET or SET method.

- For this you create and enable SSL Context Service.
- You can also set new Attribute using click on + icon
- If Data in json form then you fetch data using **${name_of_key}.**



Property	Value
HTTP Method	GET
Remote URL	https://${flipkart_api_category_URL}
SSL Context Service	StandardRestrictedSSLContextService
Connection Timeout	5 secs
Read Timeout	15 secs
Include Date Header	True
Follow Redirects	True
Attributes to Send	Fk-Affiliate-Id:instantweb,Fk-Affiliate-Token:b9c141216...
Basic Authentication Username	No value set
Basic Authentication Password	No value set
Proxy Configuration Service	No value set
Proxy Host	No value set
Proxy Port	No value set
Proxy Type	http

4- **EvaluateJsonPath** - Evaluates one or more JsonPath expressions against the content of a FlowFile. The results of those expressions are assigned to FlowFile Attributes or are written to the content of the FlowFile itself, depending on configuration of the Processor. JsonPaths are entered by adding user-defined properties; the name of the property maps to the Attribute Name into which the result will be placed (if the Destination is flowfile-attribute; otherwise, the property name is ignored). The value of the property must be a valid JsonPath expression. A Return Type of 'auto-detect' will make a determination based off the configured destination. When 'Destination' is set to 'flowfile-attribute,' a return type of 'scalar' will be used. When 'Destination' is set to 'flowfile-content,' a return type of 'JSON' will be used.If the JsonPath evaluates to a JSON array or JSON object and the Return Type is set to 'scalar' the FlowFile will be unmodified and will be routed to failure. A Return Type of JSON can return scalar values if the provided JsonPath evaluates to the specified value and will be routed as a match.If Destination is 'flowfile-content' and the JsonPath does not evaluate to a defined path, the FlowFile will be routed to 'unmatched' without having its contents modified. If Destination is flowfile-attribute and the expression matches nothing, attributes will be created with empty strings as the value, and the FlowFile will always be routed to 'matched.'

➔ **This Processor manly use for create new attribute .**

➔ **Note that Return_Type Property.**

Name	Default Value	Allowable Values	Description
Destination	flowfile-content	flowfile-content flowfile-attribute	Indicates whether the results of the JsonPath evaluation are written to the FlowFile content or a FlowFile attribute; if using attribute, must specify the Attribute Name property. If set to flowfile-content, only one JsonPath may be specified, and the property name is ignored.
Return Type	auto-detect	auto-detect json scalar	Indicates the desired return type of the JSON Path expressions. Selecting 'auto-detect' will set the return type to 'json' for a Destination of 'flowfile-content', and 'scalar' for a Destination of 'flowfile-attribute'.
Path Not Found Behavior	ignore	warn ignore	Indicates how to handle missing JSON path expressions when destination is set to 'flowfile-attribute'. Selecting 'warn' will generate a warning when a JSON path expression is not found.
Null Value Representation	empty string	empty string the string 'null'	Indicates the desired representation of JSON Path expressions resulting in a null value.

Relationships:

Name	Description
failure	FlowFiles are routed to this relationship when the JsonPath cannot be evaluated against the content of the FlowFile; for instance, if the FlowFile is not valid JSON
unmatched	FlowFiles are routed to this relationship when the JsonPath does not match the content of the FlowFile and the Destination is set to flowfile-content
matched	FlowFiles are routed to this relationship when the JsonPath is successfully evaluated and the FlowFile is modified as a result

5- **JoltTransformJSON 1.7.1** - This Processor use for convert incoming jsonflow file data into our requirement json flow file. Like some filed skip ,some are add etc.

→ The Jolt utilities processing JSON are not not stream based therefore large JSON document transformation may consume large amounts of memory. Currently UTF-8 FlowFile content and Jolt specifications are supported. A specification can be defined using Expression Language where attributes can be referred either on the left or right hand side within the specification syntax. Custom Jolt Transformations (that implement the Transform interface) are supported. Modules containing custom libraries which do not existing on the current class path can be included via the custom module directory property. **Note:**When configuring a processor if user selects of the Default transformation yet provides a Chain specification the system does not alert that the specification is invalid and and will produce failed flow files. This is a known issue identified within the Jolt library.

Name	Default Value	Allowable Values	Description
Jolt Transformation DSL	jolt-transform-chain	Cardinality ? Chain ? Default ? Modify - Default ? Modify - Define ? Modify - Overwrite ? Remove ? Shift ? Sort ? Custom ?	Specifies the Jolt Transformation that should be used with the provided specification.
Custom Transformation Class Name			Fully Qualified Class Name for Custom Transformation
Custom Module Directory			Comma-separated list of paths to files and/or directories which contain modules containing custom transformations (that are not included on NiFi's classpath).
Jolt Specification			Jolt Specification for transform of JSON data. This value is ignored if the Jolt Sort Transformation is selected. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Transform Cache Size	1		Compiling a Jolt Transform can be fairly expensive. Ideally, this will be done only once. However, if the Expression Language is used in the transform, we may need a new Transform for each FlowFile. This value controls how many of those Transforms we cache in memory in order to avoid having to compile the Transform each time.

Relationships:

Name	Description
success	The FlowFile with transformed content will be routed to this relationship
failure	If a FlowFile fails processing for any reason (for example, the FlowFile is not valid JSON), it will be routed to this relationship

6- SplitJson - Splits a JSON File into multiple, separate FlowFiles for an array element specified by a JsonPath expression. Each generated FlowFile is comprised of an element of the specified array and transferred to relationship 'split,' with the original file transferred to the 'original' relationship. If the specified JsonPath is not found or does not evaluate to an array element, the original file is routed to 'failure' and no files are generated.

Name	Default Value	Allowable Values	Description
JsonPath Expression			A JsonPath expression that indicates the array element to split into JSON/scalar fragments.
Null Value Representation	empty string	empty string the string 'null'	Indicates the desired representation of JSON Path expressions resulting in a null value.

Set property JsonPath Expression = **$.[0:]** split array into sepratefile . here $.. means hierarchy of "{".

Relationships:

Name	Description
failure	If a FlowFile fails processing for any reason (for example, the FlowFile is not valid JSON or the specified path does not exist), it will be routed to this relationship
original	The original FlowFile that was split into segments. If the FlowFile fails processing, nothing will be sent to this relationship
split	All segments of the original FlowFile will be routed to this relationship

7- **ReplaceText -** Updates the content of a FlowFile by evaluating a Regular Expression (regex) against it and replacing the section of the content that matches the Regular Expression with some alternate value.

➔ Using this we can add extra character , Remove , find and Remove operation done. It also use in SQL Query write in flowfile which is input as putSql processor.

Name	Default Value	Allowable Values	Description
Search Value	(?s)(^.*$)		The Search Value to search for in the FlowFile content. Only used for 'Literal Replace' and 'Regex Replace' matching strategies **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Replacement Value	$1		The value to insert using the 'Replacement Strategy'. Using "Regex Replace" back-references to Regular Expression capturing groups are supported, but back-references that reference capturing groups that do not exist in the regular expression will be treated as literal value. Back References may also be referenced using the Expression Language, as '$1', '$2', etc. The single-tick marks MUST be included, as these variables are not "Standard" attribute names (attribute names must be quoted unless they contain only numbers, letters, and _).

			Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)
Character Set	UTF-8		The Character Set in which the file is encoded
Maximum Buffer Size	1 MB		Specifies the maximum amount of data to buffer (per file or per line, depending on the Evaluation Mode) in order to apply the replacement. If 'Entire Text' (in Evaluation Mode) is selected and the FlowFile is larger than this value, the FlowFile will be routed to 'failure'. In 'Line-by-Line' Mode, if a single line is larger than this value, the FlowFile will be routed to 'failure'. A default value of 1 MB is provided, primarily for 'Entire Text' mode. In 'Line-by-Line' Mode, a value such as 8 KB or 16 KB is suggested. This value is ignored if the <Replacement Strategy> property is set to one of: Append, Prepend, Always Replace
Replacement Strategy	Regex Replace	Prepend ❓ Append ❓ Regex Replace ❓ Literal Replace ❓ Always Replace ❓	The strategy for how and what to replace within the FlowFile's text content.
Evaluation Mode	Entire text	Line-by-Line Entire text	Run the 'Replacement Strategy' against each line separately (Line-by-Line) or buffer the entire file into memory (Entire Text) and run against that.

Relationships:

Name	Description
success	FlowFiles that have been successfully processed are routed to this relationship. This includes both FlowFiles that had text replaced and those that did not.

failure	FlowFiles that could not be updated are routed to this relationship

8- **AttributesToJSON --** Generates a JSON representation of the input FlowFile Attributes. The resulting JSON can be written to either a new Attribute 'JSONAttributes' or written to the FlowFile as content.

Attributes List = {a1,a2,a3}

Name	Default Value	Allowable Values	Description
Attributes List			**Comma separated list of attributes to be included in the resulting JSON. If this value is left empty then all existing Attributes will be included. This list of attributes is case sensitive. If an attribute specified in the list is not found it will be be emitted to the resulting JSON with an empty string or NULL value.**
Attributes Regular Expression			Regular expression that will be evaluated against the flow file attributes to select the matching attributes. This property can be used in combination with the attributes list property. **Supports Expression Language: true (will be evaluated using variable registry only)**
Destination	flowfile-attribute	flowfile-attribute flowfile-content	Control if JSON value is written as a new flowfile attribute 'JSONAttributes' or written in the flowfile content. Writing to flowfile content will overwrite any existing flowfile content.
Include Core Attributes	true	true false	Determines if the FlowFile org.apache.nifi.flowfile.attributes.CoreAttributes which are contained in every FlowFile should be included in the final JSON value generated.
Null Value	false	true	If true a non existing or empty attribute will be NULL in the resulting JSON. If false an empty string will be

		false	placed in the JSON

Relationships:

Name	Description
success	Successfully converted attributes to JSON
failure	Failed to convert attributes to JSON

9- ExtractText - get flowfile contain into variable. Evaluates one or more Regular Expressions against the content of a FlowFile. The results of those Regular Expressions are assigned to FlowFile Attributes.

Name	Default Value	Allowable Values	Description
Character Set	UTF-8		The Character Set in which the file is encoded
Maximum Buffer Size	1 MB		Specifies the maximum amount of data to buffer (per file) in order to apply the regular expressions. Files larger than the specified maximum will not be fully evaluated.
Maximum Capture Group Length	1024		Specifies the maximum number of characters a given capture group value can have. Any characters beyond the max will be truncated.
Enable Canonical Equivalence	false	true false	Indicates that two characters match only when their full canonical decompositions match.
Enable Case-insensitive	false	true	Indicates that two characters match even if

Matching		false	they are in a different case. Can also be specified via the embedded flag (?i).
Permit Whitespace and Comments in Pattern	false	true false	In this mode, whitespace is ignored, and embedded comments starting with # are ignored until the end of a line. Can also be specified via the embedded flag (?x).
Enable DOTALL Mode	false	true false	Indicates that the expression '.' should match any character, including a line terminator. Can also be specified via the embedded flag (?s).
Enable Literal Parsing of the Pattern	false	true false	Indicates that Metacharacters and escape characters should be given no special meaning.
Enable Multiline Mode	false	true false	Indicates that '^' and '$' should match just after and just before a line terminator or end of sequence, instead of only the beginning or end of the entire input. Can also be specified via the embeded flag (?m).
Enable Unicode-aware Case Folding	false	true false	When used with 'Enable Case-insensitive Matching', matches in a manner consistent with the Unicode Standard. Can also be specified via the embedded flag (?u).
Enable Unicode Predefined Character Classes	false	true false	Specifies conformance with the Unicode Technical Standard #18: Unicode Regular Expression Annex C: Compatibility Properties. Can also be specified via the embedded flag (?U).
Enable Unix Lines Mode	false	true false	Indicates that only the ' ' line terminator is recognized in the behavior of '.', '^', and '$'. Can also be specified via the embedded flag (?d).

Include Capture Group 0	true	true false	Indicates that Capture Group 0 should be included as an attribute. Capture Group 0 represents the entirety of the regular expression match, is typically not used, and could have considerable length.
Enable repeating capture group	false	true false	If set to true, every string matching the capture groups will be extracted. Otherwise, if the Regular Expression matches more than once, only the first match will be extracted.

10 ExecuteSQL -- Executes provided SQL select query. Query result will be converted to Avro format.

Name	Default Value	Allowable Values	Description
Database Connection Pooling Service		**Controller Service API:** DBCPService **Implementations:** DBCPConnectionPool DBCPConnectionPoolLookup HiveConnectionPool	The Controller Service that is used to obtain connection to database
SQL Pre-Query			A semicolon-delimited list of queries executed before the main SQL query is executed. For example, set session properties before main query. Results/outputs from these queries will be suppressed if there are no errors. **Supports Expression Language: true (will be**

			evaluated using flow file attributes and variable registry)
SQL select query			The SQL select query to execute. The query can be empty, a constant value, or built from attributes using Expression Language. If this property is specified, it will be used regardless of the content of incoming flowfiles. If this property is empty, the content of the incoming flow file is expected to contain a valid SQL select query, to be issued by the processor to the database. Note that Expression Language is not evaluated for flow file contents. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
SQL Post-Query			A semicolon-delimited list of queries executed after the main SQL query is executed. Example like setting session properties after main query. Results/outputs from these queries will be suppressed if there are no errors. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Max Wait Time	0 seconds		The maximum amount of time allowed for a running SQL select query , zero means there is no limit. Max time less than 1 second will be equal to zero.
Normalize Table/Column Names	false	true false	Whether to change non-Avro-compatible characters in column names to Avro-compatible characters. For example, colons and periods will be changed to underscores in order to build a valid Avro record.
Use Avro	fals	true	Whether to use Avro Logical Types for

Logical Types	e	false	DECIMAL/NUMBER, DATE, TIME and TIMESTAMP columns. If disabled, written as string. If enabled, Logical types are used and written as its underlying type, specifically, DECIMAL/NUMBER as logical 'decimal': written as bytes with additional precision and scale meta data, DATE as logical 'date-millis': written as int denoting days since Unix epoch (1970-01-01), TIME as logical 'time-millis': written as int denoting milliseconds since Unix epoch, and TIMESTAMP as logical 'timestamp-millis': written as long denoting milliseconds since Unix epoch. If a reader of written Avro records also knows these logical types, then these values can be deserialized with more context depending on reader implementation.
Compression Format	NO NE	BZIP2 DEFLATE NONE SNAPPY LZO	Compression type to use when writing Avro files. Default is None.
Default Decimal Precision	10		When a DECIMAL/NUMBER value is written as a 'decimal' Avro logical type, a specific 'precision' denoting number of available digits is required. Generally, precision is defined by column data type definition or database engines default. However undefined precision (0) can be returned from some database engines. 'Default Decimal Precision' is used when writing those undefined precision numbers. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Default Decimal Scale	0		When a DECIMAL/NUMBER value is written as a 'decimal' Avro logical type, a specific 'scale' denoting number of available decimal digits is required. Generally, scale is defined by column data type

			definition or database engines default. However when undefined precision (0) is returned, scale can also be uncertain with some database engines. 'Default Decimal Scale' is used when writing those undefined numbers. If a value has more decimals than specified scale, then the value will be rounded-up, e.g. 1.53 becomes 2 with scale 0, and 1.5 with scale 1. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Max Rows Per Flow File	0		The maximum number of result rows that will be included in a single FlowFile. This will allow you to break up very large result sets into multiple FlowFiles. If the value specified is zero, then all rows are returned in a single FlowFile. **Supports Expression Language: true (will be evaluated using variable registry only)**
Output Batch Size	0		The number of output FlowFiles to queue before committing the process session. When set to zero, the session will be committed when all result set rows have been processed and the output FlowFiles are ready for transfer to the downstream relationship. For large result sets, this can cause a large burst of FlowFiles to be transferred at the end of processor execution. If this property is set, then when the specified number of FlowFiles are ready for transfer, then the session will be committed, thus releasing the FlowFiles to the downstream relationship. NOTE: The fragment.count attribute will not be set on FlowFiles when this property is set. **Supports Expression Language: true (will be evaluated using variable registry only)**

Relationships:

Name	Description
success	Successfully created FlowFile from SQL query result set.
failure	SQL query execution failed. Incoming FlowFile will be penalized and routed to this relationship

Reads Attributes:

Name	Description
sql.args.N.type	Incoming FlowFiles are expected to be parametrized SQL statements. The type of each Parameter is specified as an integer that represents the JDBC Type of the parameter.
sql.args.N.value	Incoming FlowFiles are expected to be parametrized SQL statements. The value of the Parameters are specified as sql.args.1.value, sql.args.2.value, sql.args.3.value, and so on. The type of the sql.args.1.value Parameter is specified by the sql.args.1.type attribute.
sql.args.N.format	This attribute is always optional, but default options may not always work for your data. Incoming FlowFiles are expected to be parametrized SQL statements. In some cases a format option needs to be specified, currently this is only applicable for binary data types, dates, times and timestamps. Binary Data Types (defaults to 'ascii') - ascii: each string character in your attribute value represents a single byte. This is the format provided by Avro Processors. base64: the string is a Base64 encoded string that can be decoded to bytes. hex: the string is hex encoded with all letters in upper case and no '0x' at the beginning. Dates/Times/Timestamps - Date, Time and Timestamp formats all support both custom formats or named format ('yyyy-MM-dd','ISO_OFFSET_DATE_TIME') as specified according to java.time.format.DateTimeFormatter. If not specified, a long value input is expected to be an unix epoch (milli seconds from 1970/1/1), or a string value in 'yyyy-MM-dd' format for Date, 'HH:mm:ss.SSS' for Time (some database engines e.g. Derby or

	MySQL do not support milliseconds and will truncate milliseconds), 'yyyy-MM-dd HH:mm:ss.SSS' for Timestamp is used.

Writes Attributes:

Name	Description
executesql.row.count	Contains the number of rows returned by the query. If 'Max Rows Per Flow File' is set, then this number will reflect the number of rows in the Flow File instead of the entire result set.
executesql.query.duration	Combined duration of the query execution time and fetch time in milliseconds. If 'Max Rows Per Flow File' is set, then this number will reflect only the fetch time for the rows in the Flow File instead of the entire result set.
executesql.query.executiontime	Duration of the query execution time in milliseconds. This number will reflect the query execution time regardless of the 'Max Rows Per Flow File' setting.
executesql.query.fetchtime	Duration of the result set fetch time in milliseconds. If 'Max Rows Per Flow File' is set, then this number will reflect only the fetch time for the rows in the Flow File instead of the entire result set.
executesql.resultset.index	Assuming multiple result sets are returned, the zero based index of this result set.
executesql.error.message	If processing an incoming flow file causes an Exception, the Flow File is routed to failure and this attribute is set to the exception message.
fragment.identifier	If 'Max Rows Per Flow File' is set then all FlowFiles from the same

	query result set will have the same value for the fragment.identifier attribute. This can then be used to correlate the results.
fragment.count	If 'Max Rows Per Flow File' is set then this is the total number of FlowFiles produced by a single ResultSet. This can be used in conjunction with the fragment.identifier attribute in order to know how many FlowFiles belonged to the same incoming ResultSet. If Output Batch Size is set, then this attribute will not be populated.
fragment.index	If 'Max Rows Per Flow File' is set then the position of this FlowFile in the list of outgoing FlowFiles that were all derived from the same result set FlowFile. This can be used in conjunction with the fragment.identifier attribute to know which FlowFiles originated from the same query result set and in what order FlowFiles were produced

10- ConvertAvroToJSON - Converts a Binary Avro record into a JSON object. This processor provides a direct mapping of an Avro field to a JSON field, such that the resulting JSON will have the same hierarchical structure as the Avro document. Note that the Avro schema information will be lost, as this is not a translation from binary Avro to JSON formatted Avro. The output JSON is encoded the UTF-8 encoding. If an incoming FlowFile contains a stream of multiple Avro records, the resultant FlowFile will contain a JSON Array containing all of the Avro records or a sequence of JSON Objects. If an incoming FlowFile does not contain any records, an empty JSON object is the output. Empty/Single Avro record FlowFile inputs are optionally wrapped in a container as dictated by 'Wrap Single Record'

Name	Default Value	Allowable Values	Description
JSON container options	array	none array	Determines how stream of records is exposed: either as a sequence of single Objects (none) (i.e. writing every Object to a new line), or as an array of Objects (array).
Wrap Single	false	true	Determines if the resulting output for empty

Record		false	records or a single record should be wrapped in a container array as specified by 'JSON container options'
Avro schema			If the Avro records do not contain the schema (datum only), it must be specified here.

11- ConvertJSONToSQL - Converts a JSON-formatted FlowFile into an UPDATE, INSERT, or DELETE SQL statement. The incoming FlowFile is expected to be "flat" JSON message, meaning that it consists of a single JSON element and each field maps to a simple type. If a field maps to a JSON object, that JSON object will be interpreted as Text. If the input is an array of JSON elements, each element in the array is output as a separate FlowFile to the 'sql' relationship. Upon successful conversion, the original FlowFile is routed to the 'original' relationship and the SQL is routed to the 'sql' relationship.

Properties:

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the NiFi Expression Language.

Name	Default Value	Allowable Values	Description
JDBC Connection Pool		**Controller Service API:** DBCPService **Implementations:** DBCPConnectionPool DBCPConnectionPoolLookup HiveConnectionPool	Specifies the JDBC Connection Pool to use in order to convert the JSON message to a SQL statement. The Connection Pool is necessary in order to determine the appropriate database column types.

Statement Type		UPDATE
		INSERT
		DELETE
			Specifies the type of SQL Statement to generate

Table Name			The name of the table that the statement should update **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Catalog Name			The name of the catalog that the statement should update. This may not apply for the database that you are updating. In this case, leave the field empty **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Schema Name			The name of the schema that the table belongs to. This may not apply for the database that you are updating. In this case, leave the field empty **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Translate Field Names	true	true false	If true, the Processor will attempt to translate JSON field names into the appropriate column names for the table specified. If false, the JSON field names must match the column names exactly, or the column will not be updated
Unmatched	Ignore Unmatched	Ignore Unmatched	If an incoming JSON element has a field that does not map to any of the database

Field Behavior	Fields	Fields Fail	table's columns, this property specifies how to handle the situation
Unmatched Column Behavior	Fail on Unmatched Columns	Ignore Unmatched Columns Warn on Unmatched Columns Fail on Unmatched Columns	If an incoming JSON element does not have a field mapping for all of the database table's columns, this property specifies how to handle the situation
Update Keys			A comma-separated list of column names that uniquely identifies a row in the database for UPDATE statements. If the Statement Type is UPDATE and this property is not set, the table's Primary Keys are used. In this case, if no Primary Key exists, the conversion to SQL will fail if Unmatched Column Behaviour is set to FAIL. This property is ignored if the Statement Type is INSERT **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Quote Column Identifiers	false	true false	Enabling this option will cause all column names to be quoted, allowing you to use reserved words as column names in your tables.

Quote Table Identifiers	false	true false	Enabling this option will cause the table name to be quoted to support the use of special characters in the table name
SQL Parameter Attribute Prefix	sql		The string to be prepended to the outgoing flow file attributes, such as <sql>.args.1.value, where <sql> is replaced with the specified value **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Table Schema Cache Size	100		Specifies how many Table Schemas should be cached

Relationships:

Name	Description
sql	A FlowFile is routed to this relationship when its contents have successfully been converted into a SQL statement
failure	A FlowFile is routed to this relationship if it cannot be converted into a SQL statement. Common causes include invalid JSON content or the JSON content missing a required field (if using an INSERT statement type).
original	When a FlowFile is converted to SQL, the original JSON FlowFile is routed to this relationship

12- PutSQL - Executes a SQL UPDATE or INSERT command. The content of an incoming FlowFile is expected to be the SQL command to execute.

Name	Default Value	Allowable Values	Description
JDBC Connection Pool		**Controller Service API:** DBCPService **Implementations:** DBCPConnectionPool DBCPConnectionPoolLookup HiveConnectionPool	Specifies the JDBC Connection Pool to use in order to convert the JSON message to a SQL statement. The Connection Pool is necessary in order to determine the appropriate database column types.
SQL Statement			The SQL statement to execute. The statement can be empty, a constant value, or built from attributes using Expression Language. If this property is specified, it will be used regardless of the content of incoming flowfiles. If this property is empty, the content of the incoming flow file is expected to contain a valid SQL statement, to be issued by the processor to the database. **Supports Expression Language: true (will be**

			evaluated using flow file attributes and variable registry)
Support Fragmented Transactions	true	true false	If true, when a FlowFile is consumed by this Processor, the Processor will first check the fragment.identifier and fragment.count attributes of that FlowFile. If the fragment.count value is greater than 1, the Processor will not process any FlowFile with that fragment.identifier until all are available; at that point, it will process all FlowFiles with that fragment.identifier as a single transaction, in the order specified by the FlowFiles' fragment.index attributes. This Provides atomicity of those SQL statements. If this value is false, these attributes will be ignored and the updates will occur independent of one another.
Database Session AutoCommit	false	true false	The autocommit mode to set on the database connection being used.
Transaction Timeout			If the <Support Fragmented Transactions> property is set to true, specifies how long to

			wait for all FlowFiles for a particular fragment.identifier attribute to arrive before just transferring all of the FlowFiles with that identifier to the 'failure' relationship
Batch Size	100		The preferred number of FlowFiles to put to the database in a single transaction
Obtain Generated Keys	false	true false	If true, any key that is automatically generated by the database will be added to the FlowFile that generated it using the sql.generate.key attribute. This may result in slightly slower performance and is not supported by all databases.
Rollback On Failure	false	true • false	Specify how to handle error. By default (false), if an error occurs while processing a FlowFile, the FlowFile will be routed to 'failure' or 'retry' relationship based on error type, and processor can continue with next FlowFile. Instead, you may want to rollback currently processed FlowFiles and stop further processing immediately. In that case, you can do so by enabling this 'Rollback On

			Failure' property. If enabled, failed FlowFiles will stay in the input relationship without penalizing it and being processed repeatedly until it gets processed successfully or removed by other means. It is important to set adequate 'Yield Duration' to avoid retrying too frequently.

Relationships:

Name	Description
retry	A FlowFile is routed to this relationship if the database cannot be updated but attempting the operation again may succeed
success	A FlowFile is routed to this relationship after the database is successfully updated
failure	A FlowFile is routed to this relationship if the database cannot be updated and retrying the operation will also fail, such as an invalid query or an integrity constraint violation

13- PutFile - Writes the contents of a FlowFile to the local file system .

Name	Default Value	Allowable Values	Description
Directory			The directory to which files should be written. You may use expression language such as /aa/bb/${path} **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Conflict Resolution Strategy	fail	replace ignore fail	Indicates what should happen when a file with the same name already exists in the output directory
Create Missing Directories	true	true false	If true, then missing destination directories will be created. If false, flowfiles are penalized and sent to failure.
Maximum File Count			Specifies the maximum number of files that can exist in the output directory
Last Modified Time			Sets the lastModifiedTime on the output file to the value of this attribute. Format must be yyyy-MM-dd'T'HH:mm:ssZ. You may also use expression language such as ${file.lastModifiedTime}. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Permissions			Sets the permissions on the output file to the value of this attribute. Format must be either UNIX rwxrwxrwx with a - in place of denied permissions (e.g. rw-r--r--) or an octal number (e.g. 644). You may also use expression language such as ${file.permissions}. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**

Owner			Sets the owner on the output file to the value of this attribute. You may also use expression language such as ${file.owner}. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Group			Sets the group on the output file to the value of this attribute. You may also use expression language such as ${file.group}. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**

Relationships:

Name	**Description**
success	Files that have been successfully written to the output directory are transferred to this relationship
failure	Files that could not be written to the output directory for some reason are transferred to this relationship

15 – GetHBase -This Processor polls HBase for any records in the specified table. The processor keeps track of the timestamp of the cells that it receives, so that as new records are pushed to HBase, they will automatically be pulled. Each record is output in JSON format, as {"row": "<row key>", "cells": { "<column 1 family>:<column 1 qualifier>": "<cell 1 value>", "<column 2 family>:<column 2 qualifier>": "<cell 2 value>", ... }}. For each record received, a Provenance RECEIVE event is emitted with the format hbase://<table name>/<row key>, where <row key> is the UTF-8 encoded value of the row's key.
There is a GetHBase processor that is made to incrementally extract data from an HBase table by keeping track of the last timestamp seen, and finding cells where the timestamp is greater than the last time seen.

Name	Default Value	Allowable Values	Description
HBase Client Service		**Controller Service API:** HBaseClientService **Implementations:** HBase_1_1_2_ClientService HBase_2_ClientService	Specifies the Controller Service to use for accessing HBase.
Distributed Cache Service		**Controller Service API:** DistributedMapCacheClient **Implementations:** HBase_1_1_2_ClientMapCacheService RedisDistributedMapCacheClientService CouchbaseMapCacheClient HBase_2_ClientMapCacheService DistributedMapCacheClientService	Specifies the Controller Service that should be used to maintain state about what has been pulled from HBase so that if a new node begins pulling data, it won't duplicate all of the work that has been done.
Table Name			The name of the HBase Table to put data into
Columns			A comma-separated list of "<colFamily>:<colQualifier>" pairs to return when scanning. To return all columns for a given family, leave off the qualifier such as "<colFamily1>,<colFamily2>".
Authorizations			The list of authorizations to pass to the scanner. This will be ignored if cell visibility labels are not in use.

			Supports Expression Language: true (will be evaluated using variable registry only)
Filter Expression			An HBase filter expression that will be applied to the scan. This property can not be used when also using the Columns property.
Initial Time Range	None	None Current Time	The time range to use on the first scan of a table. None will pull the entire table on the first scan, Current Time will pull entries from that point forward.
Character Set	UTF-8		Specifies which character set is used to encode the data in HBase

Relationships:

Name	Description
success	All FlowFiles are routed to this relationship

16- FetchHBaseRow -Fetches a row from an HBase table. The Destination property controls whether the cells are added as flow file attributes, or the row is written to the flow file content as JSON. This processor may be used to fetch a fixed row on a interval by specifying the table and row id directly in the processor, or it may be used to dynamically fetch rows by referencing the table and row id from incoming flow files.

Name	Default Value	Allowable Values	Description
HBase Client Service		**Controller Service API:** HBaseClientService **Implementations:** HBase_1_1_2_ClientService HBase_2_ClientService	Specifies the Controller Service to use for accessing HBase.
Table Name			The name of the HBase Table to fetch from. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Row Identifier			The identifier of the row to fetch. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Columns			An optional comma-separated list of "<colFamily>:<colQualifier>" pairs to fetch. To return all columns for a given family, leave off the qualifier such as "<colFamily1>,<colFamily2>". **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Authorizations			The list of authorizations to pass to the scanner. This will be ignored if cell visibility labels are not in use.

			Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)
Destination	flowfile-attributes	flowfile-attributes flowfile-content	Indicates whether the row fetched from HBase is written to FlowFile content or FlowFile Attributes.
JSON Format	full-row	full-row col-qual-and-val	Specifies how to represent the HBase row as a JSON document.
JSON Value Encoding	none	none base64	Specifies how to represent row ids, column families, column qualifiers, and values when stored in FlowFile attributes, or written to JSON.
Encode Character Set	UTF-8		The character set used to encode the JSON representation of the row.
Decode Character Set	UTF-8		The character set used to decode data from HBase.

Relationships:

Name	Description
success	All successful fetches are routed to this relationship.

failure	All failed fetches are routed to this relationship.
not found	All fetches where the row id is not found are routed to this relationship.

18- PutHBaseCell – Adds the Contents of a FlowFile to HBase as the value of a single cell

This processor provides the ability to attach visibility labels to HBase Puts that it generates, if visibility labels are enabled on the HBase cluster. There are two ways to enable this:

- Attributes on the flowfile.
- Dynamic properties added to the processor.

When the dynamic properties are defined on the processor, they will be the default value, but can be overridden by attributes set on the flowfile. The naming convention for both (property name and attribute name) is:

- visibility.COLUMN_FAMILY - every column qualifier under the column family will get this.
- visibility.COLUMN_FAMILY.COLUMN_VISIBILITY - the qualified column qualifier will be assigned this value.

Name	Default Value	Allowable Values	Description
HBase Client Service		**Controller Service API:** HBaseClientService **Implementations:** HBase_1_1_2_ClientServic	Specifies the Controller Service to use for accessing HBase.

		e HBase_2_ClientService	
Table Name			The name of the HBase Table to put data into **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Row Identifier			Specifies the Row ID to use when inserting data into HBase **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Row Identifier Encoding Strategy	String	String Binary	Specifies the data type of Row ID used when inserting data into HBase. The default behavior is to convert the row id to a UTF-8 byte array. Choosing Binary will convert a binary formatted string to the correct byte[] representation. The Binary option should be used if you are using Binary row keys in HBase
Column Family			The Column Family to use when inserting data into HBase **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Column Qualifier			The Column Qualifier to use when inserting data into HBase **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**

Timestamp			The timestamp for the cells being created in HBase. This field can be left blank and HBase will use the current time. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Batch Size	25		The maximum number of FlowFiles to process in a single execution. The FlowFiles will be grouped by table, and a single Put per table will be performed.

Dynamic Properties:

Dynamic Properties allow the user to specify both the name and value of a property.

Name	Value	Description
visibility.<COLUMN FAMILY>	visibility label for <COLUMN FAMILY>	Visibility label for everything under that column family when a specific label for a particular column qualifier is not available. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
visibility.<COLUMN FAMILY>.<COLUMN QUALIFIER>	visibility label for <COLUMN FAMILY>:<COLUMN QUALIFIER>.	Visibility label for the specified column qualifier qualified by a configured column family. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**

Relationships:

Name	Description
success	A FlowFile is routed to this relationship after it has been successfully stored in HBase
failure	A FlowFile is routed to this relationship if it cannot be sent to HBase

19- RouteOnAttribute - Routes FlowFiles based on their Attributes using the Attribute Expression Language

Here you can add new attribute and give some condition . you can also rout strategy. like name

For example – attribute **name--${retry.counter:le('${retry.maxcount}')}**

Name	Default Value	Allowable Values	Description
Routing Strategy	Route to Property name	Route to Property name Route to 'matched' if all **match** Route to 'matched' if any matches	Specifies how to determine which relationship to use when evaluating the Expression Language

20- RouteOnContent -Applies Regular Expressions to the content of a FlowFile and routes a copy of the FlowFile to each destination whose Regular Expression matches. Regular Expressions are added as User-Defined Properties where the name of the property is the name of the relationship and the value is a Regular Expression to match against the FlowFile content. User-Defined properties do support the Attribute Expression Language, but the results are interpreted as literal values, not Regular Expressions.

Name	Default Value	Allowable Values	Description
Match Requirement	content must match exactly	content must match exactly content must contain match	Specifies whether the entire content of the file must match the regular expression exactly, or if any part of the file (up to Content Buffer Size) can contain the regular expression in order to be considered a match
Character Set	UTF-8		The Character Set in which the file is encoded
Content Buffer Size	1 MB		Specifies the maximum amount of data to buffer in order to apply the regular expressions. If the size of the FlowFile exceeds this value, any amount of this value will be ignored

Dynamic Properties:

Dynamic Properties allow the user to specify both the name and value of a property.

Name	Value	Description
Relationship Name	A Regular Expression	Routes FlowFiles whose content matches the regular expression defined by Dynamic Property's value to the Relationship defined by the Dynamic Property's key **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**

Relationships:

Name	Description
unmatched	FlowFiles that do not match any of the user-supplied regular expressions will be routed to this relationship

21 - ExecuteScript – This processor useful when we want apply some functionality or any operation perform on content file content and its attribute then it is very helpful . you can write code in groovy, python, jython, jruby, ruby, javascript, js, lua, luaj, clojure language.

For more understand this processor refer link – Part-1,2,3

https://community.hortonworks.com/articles/75032/executescript-cookbook-part-1.html

Name	Default Value	Allowable Values	Description
Script Engine	Clojure	Clojure ECMAScript Groovy lua python ruby	The engine to execute scripts
Script File			Path to script file to execute. Only one of Script File or Script Body may be used **Supports Expression Language: true (will be evaluated using variable registry only)**
Script Body			Body of script to execute. Only one of Script File or Script Body may be used
Module Directory			Comma-separated list of paths to files and/or directories which contain modules required by the script. **Supports Expression Language: true (will be evaluated using variable registry only)**

Dynamic Properties:

Dynamic Properties allow the user to specify both the name and value of a property.

Name	Value	Description
A script engine property to update	The value to set it to	Updates a script engine property specified by the Dynamic Property's key with the value specified by

		the Dynamic Property's value **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**

Relationships:

Name	Description
success	FlowFiles that were successfully processed
failure	FlowFiles that failed to be processed

23 - LogMessage - Emits a log message at the specified log level.

Name	Default Value	Allowable Values	Description
Log Level	info		The Log Level to use when logging the message: [trace, debug, info, warn, error] **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Log prefix			Log prefix appended to the log lines. It helps to distinguish the output of multiple LogMessage processors. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Log message			The log message to emit **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**

Relationships:

Name	Description
success	All FlowFiles are routed to this relationship

24- UpdateAttribute - The properties in this processor are added by the user. The expression language is supported in user-added properties for this processor. See the NiFi Expression Language Guide to learn how to formulate proper expression language statements to perform the desired functions.

If an Attribute is added with the name **alternate.identifier** and that attribute's value is a URI, an ADD_INFO Provenance Event will be registered, correlating the FlowFile with the given alternate identifier.

Relationships:

- success
  - If the processor successfully updates the specified attribute(s), then the FlowFile follows this relationship.
- set state fail
  - If the processor is running statefully, and fails to set the state after adding attributes to the FlowFile, then the FlowFile will be routed to this relationship.

Basic Usage

For basic usage, changes are made by adding a new processor property and referencing as its name the attribute you want to change. Then enter the desired attribute value as the Value. The Value can be as simple as any text string or it can be a NiFi Expression Language statement that specifies how to formulate the value. (See the NiFi Expression Language Usage Guide for details on crafting NiFi Expression Language statements.)

As an example, to alter the standard "filename" attribute so that it has ".txt" appended to the end of it, add a new property and make the property name "filename" (to reference the desired attribute), and as the value, use the NiFi Expression Language statement shown below:

- **Property**: filename
- **Value**: ${filename}.txt

The preceding example illustrates how to modify an existing attribute. If an attribute does not already exist, this processor can also be used to add a new attribute. For example, the following property could be added to create a new attribute called myAttribute that has the value myValue:

- **Property**: myAttribute
- **Value**: myValue

In this example, all FlowFiles passing through this processor will receive an additional FlowFile attribute called myAttribute with the value myValue. This type of configuration might be used in a flow where you want to tag every FlowFile with an attribute so that it can be used later in the flow, such as for routing in a RouteOnAttribute processor.

23--MergeContent - Merges a Group of FlowFiles together based on a user-defined strategy and packages them into a single FlowFile. It is recommended that the Processor be configured with only a single incoming connection, as Group of FlowFiles will not be created from FlowFiles in different connections. This processor updates the mime.type attribute as appropriate**.**

Name	Default Value	Allowable Values	Description
Merge Strategy	Bin-Packing Algorithm	Bin-Packing Algorithm ❓ Defragment ❓	Specifies the algorithm used to merge content. The 'Defragment' algorithm combines fragments that are associated by attributes back into a single cohesive FlowFile. The 'Bin-Packing Algorithm' generates a FlowFile populated by arbitrarily chosen FlowFiles
Merge Format	Binary Concatenation	TAR ❓ ZIP ❓ FlowFile Stream, v3 ❓ FlowFile Stream, v2 ❓ FlowFile Tar, v1 ❓ Binary Concatenation ❓ Avro ❓	Determines the format that will be used to merge the content.
Attribute Strategy	Keep Only Common Attributes	Keep Only Common Attributes ❓ Keep All Unique Attributes ❓	Determines which FlowFile attributes should be added to the bundle. If 'Keep All Unique Attributes' is selected, any attribute on any FlowFile that gets bundled will be kept unless its value conflicts with the value from another FlowFile. If 'Keep Only Common Attributes' is selected, only the attributes that exist on all FlowFiles in the bundle, with the same value, will be preserved.
Correlation Attribute			If specified, like FlowFiles will be binned together, where 'like FlowFiles' means

Name			FlowFiles that have the same value for this Attribute. If not specified, FlowFiles are bundled by the order in which they are pulled from the queue. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Metadata Strategy	Do Not Merge Uncommon Metadata	Use First Metadata ? Keep Only Common Metadata ? Do Not Merge Uncommon Metadata ? Ignore Metadata ?	For FlowFiles whose input format supports metadata (Avro, e.g.), this property determines which metadata should be added to the bundle. If 'Use First Metadata' is selected, the metadata keys/values from the first FlowFile to be bundled will be used. If 'Keep Only Common Metadata' is selected, only the metadata that exists on all FlowFiles in the bundle, with the same value, will be preserved. If 'Ignore Metadata' is selected, no metadata is transferred to the outgoing bundled FlowFile. If 'Do Not Merge Uncommon Metadata' is selected, any FlowFile whose metadata values do not match those of the first bundled FlowFile will not be merged.
Minimum Number of Entries	1		The minimum number of files to include in a bundle
Maximum Number of Entries	1000		The maximum number of files to include in a bundle
Minimum Group Size	0 B		The minimum size of for the bundle
Maximum Group Size			The maximum size for the bundle. If not specified, there is no maximum.
Max Bin Age			The maximum age of a Bin that will trigger a Bin to be complete. Expected format is <duration> <time unit> where <duration> is a positive integer and time unit is one of seconds,

			minutes, hours
Maximum number of Bins	5		Specifies the maximum number of bins that can be held in memory at any one time
Delimiter Strategy	Filename	Filename Text	Determines if Header, Footer, and Demarcator should point to files containing the respective content, or if the values of the properties should be used as the content.
Header			Filename specifying the header to use. If not specified, no header is supplied. This property is valid only when using the binary-concatenation merge strategy; otherwise, it is ignored. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Footer			Filename specifying the footer to use. If not specified, no footer is supplied. This property is valid only when using the binary-concatenation merge strategy; otherwise, it is ignored. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Demarcator			Filename specifying the demarcator to use. If not specified, no demarcator is supplied. This property is valid only when using the binary-concatenation merge strategy; otherwise, it is ignored. **Supports Expression Language: true (will be evaluated using flow file attributes and variable registry)**
Compression Level	1	0 1 2 3 4	Specifies the compression level to use when using the Zip Merge Format; if not using the Zip Merge Format, this value is ignored

		5 6 7 8 9	
Keep Path	false	true false	If using the Zip or Tar Merge Format, specifies whether or not the FlowFiles' paths should be included in their entry names; if using other merge strategy, this value is ignored
Tar Modified Time	${file.lastModifiedTime}		re the modified timestamp either by expression match the ISO8601 format 'yyyy-MM- value is ignored **low file attributes and variable registry)**

Relationships:

Name	Description
failure	If the bundle cannot be created, all FlowFiles that would have been used to created the bundle will be transferred to failure
original	The FlowFiles that were used to create the bundle
merged	The FlowFile containing the merged content