Apache NiFi 3

NiFi System Properties

Date of Publish: 2018-11-15

https://docs.hortonworks.com/

Contents

System Properties

The nifi.properties file in the conf directory is the main configuration file for controlling how NiFi runs. This section provides an overview of the properties in this file and includes some notes on how to configure it in a way that will make upgrading easier. After making changes to this file, restart NiFi in order for the changes to take effect.

Note: The contents of this file are relatively stable but do change from time to time. It is always a good idea to review this file when upgrading and pay attention for any changes. Consider configuring items below marked with an asterisk (*) in such a way that upgrading will be easier. For details, see a full discussion on upgrading at the end of this section. Note that values for periods of time and data sizes must include the unit of measure, for example "10 secs" or "10 MB", not simply "10".

Core Properties

The first section of the nifi.properties file is for the Core Properties. These properties apply to the core framework as a whole.

Property	Description
nifi.flow.configuration.file*	The location of the flow configuration file (i.e., the file that contains what is currently displayed on the NiFi graph). The default value is ./conf/flow.xml.gz.
nifi.flow.configuration.archive.enabled*	Specifies whether NiFi creates a backup copy of the flow automatically when the flow is updated. The default value is true.
nifi.flow.configuration.archive.dir*	The location of the archive directory where backup copies of the flow.xml are saved. The default value is ./conf/archive. NiFi removes old archive files to limit disk usage based on archived file lifespan, total size, and number of files, as specified with nifi.flow.configuration.archive.max.time, max.storage and max.count properties respectively. If none of these limitation for archiving is specified, NiFi uses default conditions, that is 30 days for max.time and 500 MB for max.storage. This cleanup mechanism takes into account only automatically created archived flow.xml files. If there are other files or directories in this archive directory, NiFi will ignore them. Automatically created archives have filename with ISO 8601 format timestamp prefix followed by <original-filename>. That is <year><month><day>T<hour><minute><second>+<timezone offset>_<original filename>. For example, 20160706T160719+0900_flow.xml.gz. NiFi checks filenames when it cleans archive directory. If you would like to keep a particular archive in this directory without worrying about NiFi deleting it, you can do so by copying it with a different filename pattern.
nifi.flow.configuration.archive.max.time*	The lifespan of archived flow.xml files. NiFi will delete expired archive files when it updates flow.xml if this property is specified. Expiration is determined based on current system time and the last modified timestamp of an archived flow.xml. If no archive limitation is specified in nifi.properties, NiFi removes archives older than 30 days.
nifi.flow.configuration.archive.max.storage*	The total data size allowed for the archived flow.xml files. NiFi will delete the oldest archive files until the total archived file size becomes less than this configuration value, if this property is specified. If no archive limitation is specified in nifi.properties, NiFi uses 500 MB for this.
nifi.flow.configuration.archive.max.count*	The number of archive files allowed. NiFi will delete the oldest archive files so that only N latest archives can be kept, if this property is specified.

nifi.flowcontroller.autoResumeState	Indicates whether -upon restart- the components on the NiFi graph should return to their last state. The default value is true.
nifi.flowcontroller.graceful.shutdown.period	Indicates the shutdown period. The default value is 10 secs.
nifi.flowservice.writedelay.interval	When many changes are made to the flow.xml, this property specifies how long to wait before writing out the changes, so as to batch the changes into a single write. The default value is 500 ms.
nifi.administrative.yield.duration	If a component allows an unexpected exception to escape, it is considered a bug. As a result, the framework will pause (or administratively yield) the component for this amount of time. This is done so that the component does not use up massive amounts of system resources, since it is known to have problems in the existing state. The default value is 30 secs.
nifi.bored.yield.duration	When a component has no work to do (i.e., is "bored"), this is the amount of time it will wait before checking to see if it has new data to work on. This way, it does not use up CPU resources by checking for new work too often. When setting this property, be aware that it could add extra latency for components that do not constantly have work to do, as once they go into this "bored" state, they will wait this amount of time before checking for more work. The default value is 10 ms.
nifi.queue.backpressure.count	When drawing a new connection between two components, this is the default value for that connection's back pressure object threshold. The default is 10000 and the value must be an integer.
nifi.queue.backpressure.size	When drawing a new connection between two components, this is the default value for that connection's back pressure data size threshold. The default is 1 GB and the value must be a data size including the unit of measure.
nifi.authorizer.configuration.file*	This is the location of the file that specifies how authorizers are defined. The default value is ./conf/authorizers.xml.
nifi.login.identity.provider.configuration.file*	This is the location of the file that specifies how username/password authentication is performed. This file is only considered if nifi.security.user.login.identity.provider is configured with a provider identifier. The default value is ./conf/login-identity-providers.xml.
nifi.templates.directory*	This is the location of the directory where flow templates are saved (for backward compatibility only). Templates are stored in the flow.xml.gz starting with NiFi 1.0. The template directory can be used to (bulk) import templates into the flow.xml.gz automatically on NiFi startup. The default value is ./conf/templates.
nifi.ui.banner.text	This is banner text that may be configured to display at the top of the User Interface. It is blank by default.
nifi.ui.autorefresh.interval	The interval at which the User Interface auto-refreshes. The default value is 30 secs.
nifi.nar.library.directory	The location of the nar library. The default value is ./lib and probably should be left as is.NOTE: Additional library directories can be specified by using the nifi.nar.library.directory. prefix with unique suffixes and separate paths as values. For example, to provide two additional library locations, a user could also specify additional properties with keys of:nifi.nar.library.directory.lib1=/nars/lib1 nifi.nar.library.directory.lib2=/nars/lib2 Providing three total locations, including nifi.nar.library.directory.
nifi.nar.working.directory	The location of the nar working directory. The default value is ./work/nar and probably should be left as is.
nifi.documentation.working.directory	The documentation working directory. The default value is ./work/docs/components and probably should be left as is.

nifi.processor.scheduling.timeout	Time to wait for a Processor's life-cycle operation (@OnScheduled and @OnUnscheduled) to finish before other life-cycle operation (e.g., stop) could be invoked. The default value is 1 min.

State Management

The State Management section of the Properties file provides a mechanism for configuring local and cluster-wide mechanisms for components to persist state.

Property	Description
nifi.state.management.configuration.file	The XML file that contains configuration for the local and cluster-wide State Providers. The default value is ./conf/state-management.xml.
nifi.state.management.provider.local	The ID of the Local State Provider to use. This value must match the value of the id element of one of the local-provider elements in the state-management.xml file.
nifi.state.management.provider.cluster	The ID of the Cluster State Provider to use. This value must match the value of the id element of one of the cluster-provider elements in the state-management.xml file. This value is ignored if not clustered but is required for nodes in a cluster.
nifi.state.management.embedded.zookeeper.start	Specifies whether or not this instance of NiFi should start an embedded ZooKeeper Server. This is used in conjunction with the ZooKeeperStateProvider.
nifi.state.management.embedded.zookeeper.properties	Specifies a properties file that contains the configuration for the embedded ZooKeeper Server that is started (if the nifi.state.management.embedded.zookeeper.start property is set to true)

H2 Settings

The H2 Settings section defines the settings for the H2 database, which keeps track of user access and flow controller history.

Property	Description
nifi.database.directory*	The location of the H2 database directory. The default value is ./database_repository.
nifi.h2.url.append	This property specifies additional arguments to add to the connection string for the H2 database. The default value should be used and should not be changed. It is: ;LOCK_TIMEOUT=25000;WRITE_DELAY=0;AUTO_SERVER=FALSE.

FlowFile Repository

The FlowFile repository keeps track of the attributes and current state of each FlowFile in the system. By default, this repository is installed in the same root installation directory as all the other repositories; however, it is advisable to configure it on a separate drive if available.

Property	Description

nifi.flowfile.repository.implementation	The FlowFile Repository implementation. The default value is org.apache.nifi.controller.repository.WriteAheadFlowFileRepository and should only be changed with caution. To store flowfiles in memory instead of on disk (accepting data loss in the event of power/machine failure or a restart of NiFi), set this property to org.apache.nifi.controller.repository.VolatileFlowFileRepository.
nifi.flowfile.repository.wal.implementation	If the repository implementation is configured to use the WriteAheadFlowFileRepository, this property can be used to specify which implementation of the Write-Ahead Log should be used. The default value is org.apache.nifi.wali.SequentialAccessWriteAheadLog. This version of the write-ahead log was added in version 1.6.0 of Apache NiFi and was developed in order to address an issue that exists in the older implementation. In the event of power loss or an operating system crash, the old implementation was susceptible to recovering FlowFiles incorrectly. This could potentially lead to the wrong attributes or content being assigned to a FlowFile upon restart, following the power loss or OS crash. However, one can still choose to opt into using the previous implementation and accept that risk, if desired (for example, if the new implementation were to exhibit some unexpected error). To do so, set the value of this property to org.wali.MinimalLockingWriteAheadLog. If the value of this property is changed, upon restart, NiFi will still recover the records written using the previously configured repository and delete the files written by the previously configured implementation.
nifi.flowfile.repository.directory*	The location of the FlowFile Repository. The default value is ./flowfile_repository.
nifi.flowfile.repository.partitions	The number of partitions. The default value is 256.
nifi.flowfile.repository.checkpoint.interval	The FlowFile Repository checkpoint interval. The default value is 2 mins.
nifi.flowfile.repository.always.sync	If set to true, any change to the repository will be synchronized to the disk, meaning that NiFi will ask the operating system not to cache the information. This is very expensive and can significantly reduce NiFi performance. However, if it is false, there could be the potential for data loss if either there is a sudden power loss or the operating system crashes. The default value is false.

Swap Management

NiFi keeps FlowFile information in memory (the JVM) but during surges of incoming data, the FlowFile information can start to take up so much of the JVM that system performance suffers. To counteract this effect, NiFi "swaps" the FlowFile information to disk temporarily until more JVM space becomes available again. These properties govern how that process occurs.

Property	Description
nifi.swap.manager.implementation	The Swap Manager implementation. The default value is org.apache.nifi.controller.FileSystemSwapManager and should not be changed.
nifi.queue.swap.threshold	The queue threshold at which NiFi starts to swap FlowFile information to disk. The default value is 20000.
nifi.swap.in.period	The swap in period. The default value is 5 sec.
nifi.swap.in.threads	The number of threads to use for swapping in. The default value is 1.
nifi.swap.out.period	The swap out period. The default value is 5 sec.
nifi.swap.out.threads	The number of threads to use for swapping out. The default value is 4.

Content Repository

The Content Repository holds the content for all the FlowFiles in the system. By default, it is installed in the same root installation directory as all the other repositories; however, administrators will likely want to configure it on a separate drive if available. If nothing else, it is best if the Content Repository is not on the same drive as the FlowFile Repository. In dataflows that handle a large amount of data, the Content Repository could fill up a disk and the FlowFile Repository, if also on that disk, could become corrupt. To avoid this situation, configure these repositories on different drives.

Property	Description
nifi.content.repository.implementation	The Content Repository implementation. The default value is org.apache.nifi.controller.repository.FileSystemRepository and should only be changed with caution. To store flowfile content in memory instead of on disk (at the risk of data loss in the event of power/machine failure), set this property to org.apache.nifi.controller.repository.VolatileContentRepository.

File System Content Repository Properties

Property	Description
nifi.content.repository.implementation	The Content Repository implementation. The default value is org.apache.nifi.controller.repository.FileSystemRepository and should only be changed with caution. To store flowfile content in memory instead of on disk (at the risk of data loss in the event of power/machine failure), set this property to org.apache.nifi.controller.repository.VolatileContentRepository.
nifi.content.claim.max.appendable.size	The maximum size for a content claim. The default value is 1 MB.
nifi.content.claim.max.flow.files	The maximum number of FlowFiles to assign to one content claim. The default value is 100.
nifi.content.repository.directory.default*	The location of the Content Repository. The default value is ./ content_repository.NOTE: Multiple content repositories can be specified by using the nifi.content.repository.directory. prefix with unique suffixes and separate paths as values. For example, to provide two additional locations to act as part of the content repository, a user could also specify additional properties with keys of:nifi.content.repository.directory.content1=/repos/content1 nifi.content.repository.directory.content2=/repos/content2 Providing three total locations, including nifi.content.repository.directory.default.
nifi.content.repository.archive.max.retention.period	If archiving is enabled (see nifi.content.repository.archive.enabled below), then this property specifies the maximum amount of time to keep the archived data. The default value is 12 hours.
nifi.content.repository.archive.max.usage.percentage	If archiving is enabled (see nifi.content.repository.archive.enabled below), then this property must have a value that indicates the content repository disk usage percentage at which archived data begins to be removed. If the archive is empty and content repository disk usage is above this percentage, then archiving is temporarily disabled. Archiving will resume when disk usage is below this percentage. The default value is 50%.
nifi.content.repository.archive.enabled	To enable content archiving, set this to true and specify a value for the nifi.content.repository.archive.max.usage.percentage property above. Content archiving enables the provenance UI to view or replay content that is no longer in a dataflow queue. By default, archiving is enabled.

nifi.content.repository.always.sync	If set to true, any change to the repository will be synchronized to the disk, meaning that NiFi will ask the operating system not to cache the information. This is very expensive and can significantly reduce NiFi performance. However, if it is false, there could be the potential for data loss if either there is a sudden power loss or the operating system crashes. The default value is false.
nifi.content.viewer.url	The URL for a web-based content viewer if one is available. It is blank by default.

Volatile Content Repository Properties

Property	Description
nifi.volatile.content.repository.max.size	The Content Repository maximum size in memory. The default value is 100 MB.
nifi.volatile.content.repository.block.size	The Content Repository block size. The default value is 32 KB.

Provenance Repository

The Provenance Repository contains the information related to Data Provenance. The next four sections are for Provenance Repository properties.

Property	Description

nifi.provenance.repository.implementation	The Provenance Repository implementation. The default value is org.apache.nifi.provenance.WriteAheadProvenanceRepository. Three additional repositories are available as well. To store provenance events in memory instead of on disk (in which case all events will be lost on restart, and events will be evicted in a first-in-first-out order), set this property to org.apache.nifi.provenance.VolatileProvenanceRepository. This leaves a configurable number of Provenance Events in the Java heap, so the number of events that can be retained is very limited. A third and fourth option are available: org.apache.nifi.provenance.PersistentProvenanceRepository and org.apache.nifi.provenance.EncryptedWriteAheadProvenanceRepository. The PersistentProvenanceRepository was originally written with the simple goal of persisting Provenance Events as they are generated and providing the ability to iterate over those events sequentially. Later, it was desired to be able to compress the data so that more data could be stored. After that, the ability to index and query the data was added. As requirements evolved over time, the repository kept changing without any major redesigns. When used in a NiFi instance that is responsible for processing large volumes of small FlowFiles, the PersistentProvenanceRepository can quickly become a bottleneck. The WriteAheadProvenanceRepository was then written to provide the same capabilities as the PersistentProvenanceRepository while providing far better performance. The WriteAheadProvenanceRepository was added in version 1.2.0 of NiFi. Since then, it has proven to be very stable and robust and as such was made the default implementation. The PersistentProvenanceRepository is now considered deprecated and should no longer be used. If administering an instance of NiFi that is currently using the PersistentProvenanceRepository, it is highly recommended to upgrade to the WriteAheadProvenanceRepository. Doing so is as simple as changing the implementation property value from org.apache.nifi.provenance.PersistentProvenanceRepository to org.apache.nifi.provenance.WriteAheadProvenanceRepository. Because the Provenance Repository is backward compatible, there will be no loss of data or functionality. The EncryptedWriteAheadProvenanceRepository builds upon the WriteAheadProvenanceRepository and ensures that data is encrypted at rest. NOTE: The WriteAheadProvenanceRepository will make use of the Provenance data stored by the PersistentProvenanceRepository. However, the PersistentProvenanceRepository may not be able to read the data written by the WriteAheadProvenanceRepository. Therefore, once the Provenance Repository is changed to use the WriteAheadProvenanceRepository, it cannot be changed back to the PersistentProvenanceRepository without deleting the data in the Provenance Repository.

Write Ahead Provenance Repository Properties

Property	Description
nifi.provenance.repository.directory.default*	The location of the Provenance Repository. The default value is ./ provenance_repository.NOTE: Multiple provenance repositories can be specified by using the nifi.provenance.repository.directory. prefix with unique suffixes and separate paths as values. For example, to provide two additional locations to act as part of the provenance repository, a user could also specify additional properties with keys of:nifi.provenance.repository.directory.provenance1=/repos/ provenance1 nifi.provenance.repository.directory.provenance2=/ repos/provenance2 Providing three total locations, including nifi.provenance.repository.directory.default.

nifi.provenance.repository.max.storage.time	The maximum amount of time to keep data provenance information. The default value is 24 hours.
nifi.provenance.repository.max.storage.size	The maximum amount of data provenance information to store at a time. The default value is 1 GB. The Data Provenance capability can consume a great deal of storage space because so much data is kept. For production environments, values of 1-2 TB or more is not uncommon. The repository will write to a single "event file" (or set of "event files" if multiple storage locations are defined, as described above) for some period of time (defined by the nifi.provenance.repository.rollover.time and nifi.provenance.repository.rollover.size properties). Data is always aged off one file at a time, so it is not advisable to write to a single "event file" for a tremendous amount of time, as it will prevent old data from aging off as smoothly.
nifi.provenance.repository.rollover.time	The amount of time to wait before rolling over the "event file" that the repository is writing to.
nifi.provenance.repository.rollover.size	The amount of data to write to a single "event file." The default value is 100 MB. For production environments where a very large amount of Data Provenance is generated, a value of 1 GB is also very reasonable.
nifi.provenance.repository.query.threads	The number of threads to use for Provenance Repository queries. The default value is 2.
nifi.provenance.repository.index.threads	The number of threads to use for indexing Provenance events so that they are searchable. The default value is 2. For flows that operate on a very high number of FlowFiles, the indexing of Provenance events could become a bottleneck. If this happens, increasing the value of this property may increase the rate at which the Provenance Repository is able to process these records, resulting in better overall throughput. It is advisable to use at least 1 thread per storage location (i.e., if there are 3 storage locations, at least 3 threads should be used). For high throughput environments, where more CPU and disk I/O is available, it may make sense to increase this value significantly. Typically going beyond 2-4 threads per storage location is not valuable. However, this can be tuned depending on the CPU resources available compared to the I/O resources.
nifi.provenance.repository.compress.on.rollover	Indicates whether to compress the provenance information when an "event file" is rolled over. The default value is true.
nifi.provenance.repository.always.sync	If set to true, any change to the repository will be synchronized to the disk, meaning that NiFi will ask the operating system not to cache the information. This is very expensive and can significantly reduce NiFi performance. However, if it is false, there could be the potential for data loss if either there is a sudden power loss or the operating system crashes. The default value is false.
nifi.provenance.repository.indexed.fields	This is a comma-separated list of the fields that should be indexed and made searchable. Fields that are not indexed will not be searchable. Valid fields are: EventType, FlowFileUUID, Filename, TransitURI, ProcessorID, AlternateIdentifierURI, Relationship, Details. The default value is: EventType, FlowFileUUID, Filename, ProcessorID.
nifi.provenance.repository.indexed.attributes	This is a comma-separated list of FlowFile Attributes that should be indexed and made searchable. It is blank by default. But some good examples to consider are filename and mime.type as well as any custom attributes you might use which are valuable for your use case.

nifi.provenance.repository.index.shard.size	The repository uses Apache Lucene to performing indexing and searching capabilities. This value indicates how large a Lucene Index should become before the Repository starts writing to a new Index. Large values for the shard size will result in more Java heap usage when searching the Provenance Repository but should provide better performance. The default value is 500 MB. However, this is due to the fact that defaults are tuned for very small environments where most users begin to use NiFi. For production environments, it is advisable to change this value to 4 to 8 GB. Once all Provenance Events in the index have been aged off from the "event files," the index will be destroyed as well. Note: this value should be smaller than (no more than half of) the nifi.provenance.repository.max.storage.size property.
nifi.provenance.repository.max.attribute.length	Indicates the maximum length that a FlowFile attribute can be when retrieving a Provenance Event from the repository. If the length of any attribute exceeds this value, it will be truncated when the event is retrieved. The default value is 65536.
nifi.provenance.repository.concurrent.merge.threads	Apache Lucene creates several "segments" in an Index. These segments are periodically merged together in order to provide faster querying. This property specifies the maximum number of threads that are allowed to be used for each of the storage directories. The default value is 2. For high throughput environments, it is advisable to set the number of index threads larger than the number of merge threads * the number of storage locations. For example, if there are 2 storage locations and the number of index threads is set to 8, then the number of merge threads should likely be less than 4. While it is not critical that this be done, setting the number of merge threads larger than this can result in all index threads being used to merge, which would cause the NiFi flow to periodically pause while indexing is happening, resulting in some data being processed with much higher latency than other data.
nifi.provenance.repository.warm.cache.frequency	Each time that a Provenance query is run, the query must first search the Apache Lucene indices (at least, in most cases - there are some queries that are run often and the results are cached to avoid searching the Lucene indices). When a Lucene index is opened for the first time, it can be very expensive and take several seconds. This is compounded by having many different indices, and can result in a Provenance query taking much longer. After the index has been opened, the Operating System's disk cache will typically hold onto enough data to make re-opening the index much faster - at least for a period of time, until the disk cache evicts this data. If this value is set, NiFi will periodically open each Lucene index and then close it, in order to "warm" the cache. This will result in far faster queries when the Provenance Repository is large. As with all great things, though, it comes with a cost. Warming the cache does take some CPU resources, but more importantly it will evict other data from the Operating System disk cache and will result in reading (potentially a great deal of) data from the disk. This can result in lower NiFi performance. However, if NiFi is running in an environment where CPU and disk are not fully utilized, this feature can result in far faster Provenance queries. The default value for this property is blank (i.e. disabled).

Encrypted Write Ahead Provenance Repository Properties

All of the properties defined above still apply. Only encryption-specific properties are listed here.

Property	Description
nifi.provenance.repository.debug.frequency	Controls the number of events processed between DEBUG statements documenting the performance metrics of the repository. This value is only used when DEBUG level statements are enabled in the log configuration.

nifi.provenance.repository.encryption.key.provider.implementation	This is the fully-qualified class name of the key provider. A key provider is the datastore interface for accessing the encryption key to protect the provenance events. There are currently two implementations - StaticKeyProvider which reads a key directly from nifi.properties, and FileBasedKeyProvider which reads n many keys from an encrypted file. The interface is extensible, and HSM-backed or other providers are expected in the future.
nifi.provenance.repository.encryption.key.provider.location	The path to the key definition resource (empty for StaticKeyProvider, ./keys.nkp or similar path for FileBasedKeyProvider). For future providers like an HSM, this may be a connection string or URL.
nifi.provenance.repository.encryption.key.id	The active key ID to use for encryption (e.g. Key1).
nifi.provenance.repository.encryption.key	The key to use for StaticKeyProvider. The key format is hex-encoded (0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFEDCBA9876543210) but can also be encrypted using the encrypt-config tool in NiFi Toolkit.
nifi.provenance.repository.encryption.key.id.*	Allows for additional keys to be specified for the StaticKeyProvider. For example, the line nifi.provenance.repository.encryption.key.id.Key2=012…210 would provide an available key Key2.

The simplest configuration is below:

```
nifi.provenance.repository.implementation=org.apache.nifi.provenance.EncryptedWriteAhead
nifi.provenance.repository.debug.frequency=100
nifi.provenance.repository.encryption.key.provider.implementation=org.apache.nifi.securi
nifi.provenance.repository.encryption.key.provider.location=
nifi.provenance.repository.encryption.key.id=Key1
nifi.provenance.repository.encryption.key=0123456789ABCDEFFEDCBA98765432100123456789ABCD
```

Persistent Provenance Repository Properties

Property	Description
nifi.provenance.repository.directory.default*	The location of the Provenance Repository. The default value is ./provenance_repository.NOTE: Multiple provenance repositories can be specified by using the nifi.provenance.repository.directory. prefix with unique suffixes and separate paths as values. For example, to provide two additional locations to act as part of the provenance repository, a user could also specify additional properties with keys of:nifi.provenance.repository.directory.provenance1=/repos/provenance1 nifi.provenance.repository.directory.provenance2=/repos/provenance2 Providing three total locations, including nifi.provenance.repository.directory.default.
nifi.provenance.repository.max.storage.time	The maximum amount of time to keep data provenance information. The default value is 24 hours.
nifi.provenance.repository.max.storage.size	The maximum amount of data provenance information to store at a time. The default value is 1 GB.
nifi.provenance.repository.rollover.time	The amount of time to wait before rolling over the latest data provenance information so that it is available in the User Interface. The default value is 30 secs.
nifi.provenance.repository.rollover.size	The amount of information to roll over at a time. The default value is 100 MB.
nifi.provenance.repository.query.threads	The number of threads to use for Provenance Repository queries. The default value is 2.

nifi.provenance.repository.index.threads	The number of threads to use for indexing Provenance events so that they are searchable. The default value is 2. For flows that operate on a very high number of FlowFiles, the indexing of Provenance events could become a bottleneck. If this is the case, a bulletin will appear, indicating that "The rate of the dataflow is exceeding the provenance recording rate. Slowing down flow to accommodate." If this happens, increasing the value of this property may increase the rate at which the Provenance Repository is able to process these records, resulting in better overall throughput.
nifi.provenance.repository.compress.on.rollover	Indicates whether to compress the provenance information when rolling it over. The default value is true.
nifi.provenance.repository.always.sync	If set to true, any change to the repository will be synchronized to the disk, meaning that NiFi will ask the operating system not to cache the information. This is very expensive and can significantly reduce NiFi performance. However, if it is false, there could be the potential for data loss if either there is a sudden power loss or the operating system crashes. The default value is false.
nifi.provenance.repository.journal.count	The number of journal files that should be used to serialize Provenance Event data. Increasing this value will allow more tasks to simultaneously update the repository but will result in more expensive merging of the journal files later. This value should ideally be equal to the number of threads that are expected to update the repository simultaneously, but 16 tends to work well in must environments. The default value is 16.
nifi.provenance.repository.indexed.fields	This is a comma-separated list of the fields that should be indexed and made searchable. Fields that are not indexed will not be searchable. Valid fields are: EventType, FlowFileUUID, Filename, TransitURI, ProcessorID, AlternateIdentifierURI, Relationship, Details. The default value is: EventType, FlowFileUUID, Filename, ProcessorID.
nifi.provenance.repository.indexed.attributes	This is a comma-separated list of FlowFile Attributes that should be indexed and made searchable. It is blank by default. But some good examples to consider are filename, uuid, and mime.type as well as any custom attritubes you might use which are valuable for your use case.
nifi.provenance.repository.index.shard.size	Large values for the shard size will result in more Java heap usage when searching the Provenance Repository but should provide better performance. The default value is 500 MB.
nifi.provenance.repository.max.attribute.length	Indicates the maximum length that a FlowFile attribute can be when retrieving a Provenance Event from the repository. If the length of any attribute exceeds this value, it will be truncated when the event is retrieved. The default value is 65536.

Volatile Provenance Repository Properties

Property	Description
nifi.provenance.repository.buffer.size	The Provenance Repository buffer size. The default value is 100000 provenance events.

Component Status Repository

The Component Status Repository contains the information for the Component Status History tool in the User Interface. These properties govern how that tool works.

The buffer.size and snapshot.frequency work together to determine the amount of historical data to retain. As an example to configure two days worth of historical data with a data point snapshot occurring every 5 minutes you

would configure snapshot.frequency to be "5 mins" and the buffer.size to be "576". To further explain this example for every 60 minutes there are 12 (60 / 5) snapshot windows for that time period. To keep that data for 48 hours (12 * 48) you end up with a buffer size of 576.

Property	Description
nifi.components.status.repository.implementation	The Component Status Repository implementation. The default value is org.apache.nifi.controller.status.history.VolatileComponentStatusRepository and should not be changed.
nifi.components.status.repository.buffer.size	Specifies the buffer size for the Component Status Repository. The default value is 1440.
nifi.components.status.snapshot.frequency	This value indicates how often to present a snapshot of the components' status history. The default value is 1 min.

Site to Site Properties

These properties govern how this instance of NiFi communicates with remote instances of NiFi when Remote Process Groups are configured in the dataflow. Remote Process Groups can choose transport protocol from RAW and HTTP. Properties named with nifi.remote.input.socket.* are RAW transport protocol specific. Similarly, nifi.remote.input.http.* are HTTP transport protocol specific properties.

Property	Description
nifi.remote.input.host	The host name that will be given out to clients to connect to this NiFi instance for Site-to-Site communication. By default, it is the value from InetAddress.getLocalHost().getHostName(). On UNIX-like operating systems, this is typically the output from the hostname command.
nifi.remote.input.secure	This indicates whether communication between this instance of NiFi and remote NiFi instances should be secure. By default, it is set to false. In order for secure site-to-site to work, set the property to true.
nifi.remote.input.socket.port	The remote input socket port for Site-to-Site communication. By default, it is blank, but it must have a value in order to use RAW socket as transport protocol for Site-to-Site.
nifi.remote.input.http.enabled	Specifies whether HTTP Site-to-Site should be enabled on this host. By default, it is set to true. Whether a Site-to-Site client uses HTTP or HTTPS is determined by nifi.remote.input.secure. If it is set to true, then requests are sent as HTTPS to nifi.web.https.port. If set to false, HTTP requests are sent to nifi.web.http.port.
nifi.remote.input.http.transaction.ttl	Specifies how long a transaction can stay alive on the server. By default, it is set to 30 secs. If a Site-to-Site client hasn't proceeded to the next action after this period of time, the transaction is discarded from the remote NiFi instance. For example, when a client creates a transaction but doesn't send or receive flow files, or when a client sends or receives flow files but doesn't confirm that transaction.
nifi.remote.contents.cache.expiration	Specifies how long NiFi should cache information about a remote NiFi instance when communicating via Site-to-Site. By default, NiFi will cache the responses from the remote system for 30 secs. This allows NiFi to avoid constantly making HTTP requests to the remote system, which is particularly important when this instance of NiFi has many instances of Remote Process Groups.

Site to Site Routing Properties for Reverse Proxies

Site-to-Site requires peer-to-peer communication between a client and a remote NiFi node. E.g. if a remote NiFi cluster has 3 nodes (nifi0, nifi1 and nifi2) then client requests have to be reachable to each of those remote nodes.

If a NiFi cluster is planned to receive/transfer data from/to Site-to-Site clients over the internet or a company firewall, a reverse proxy server can be deployed in front of the NiFi cluster nodes as a gateway to route client requests to upstream NiFi nodes, to reduce number of servers and ports those have to be exposed.

In such environment, the same NiFi cluster would also be expected to be accessed by Site-to-Site clients within the same network. Sending FlowFiles to itself for load distribution among NiFi cluster nodes can be a typical example. In this case, client requests should be routed directly to a node without going through the reverse proxy.

In order to support such deployments, remote NiFi clusters need to expose its Site-to-Site endpoints dynamically based on client request contexts. Following properties configure how peers should be exposed to clients. A routing definition consists of 4 properties, when, hostname, port, and secure, grouped by protocol and name. Multiple routing definitions can be configured. protocol represents Site-to-Site transport protocol, i.e. RAW or HTTP.

Property	Description
nifi.remote.route.{protocol}.{name}.when	Boolean value, true or false. Controls whether the routing definition for this name should be used.
nifi.remote.route.{protocol}.{name}.hostname	Specify hostname that will be introduced to Site-to-Site clients for further communications.
nifi.remote.route.{protocol}.{name}.port	Specify port number that will be introduced to Site-to-Site clients for further communications.
nifi.remote.route.{protocol}.{name}.secure	Boolean value, true or false. Specify whether the remote peer should be accessed via secure protocol. Defaults to false.

All of above routing properties can use NiFi Expression Language to compute target peer description from request context. Available variables are:

Variable name	Description
s2s.{source|target}.hostname	Hostname of the source where the request came from, and the original target.
s2s.{source|target}.port	Same as above, for ports. Source port may not be useful as it is just a client side TCP port.
s2s.{source|target}.secure	Same as above, for secure or not.
s2s.protocol	The name of Site-to-Site protocol being used, RAW or HTTP.
s2s.request	The name of current request type, SiteToSiteDetail or Peers. See Site-to-Site protocol sequence below for detail.
HTTP request headers	HTTP request header values can be referred by its name.

Site to Site protocol sequence

Configuring these properties correctly would require some understandings on Site-to-Site protocol sequence.

1. A client initiates Site-to-Site protocol by sending a HTTP(S) request to the specified remote URL to get remote cluster Site-to-Site information. Specifically, to '/nifi-api/site-to-site'. This request is called SiteToSiteDetail.
2. A remote NiFi node responds with its input and output ports, and TCP port numbers for RAW and TCP transport protocols.
3. The client sends another request to get remote peers using the TCP port number returned at #2. From this request, raw socket communication is used for RAW transport protocol, while HTTP keeps using HTTP(S). This request is called Peers.

4. A remote NiFi node responds with list of available remote peers containing hostname, port, secure and workload such as the number of queued FlowFiles. From this point, further communication is done between the client and the remote NiFi node.
5. The client decides which peer to transfer data from/to, based on workload information.
6. The client sends a request to create a transaction to a remote NiFi node.
7. The remote NiFi node accepts the transaction.
8. Data is sent to the target peer. Multiple Data packets can be sent in batch manner.
9. When there is no more data to send, or reached to batch limit, the transaction is confirmed on both end by calculating CRC32 hash of sent data.
10. The transaction is committed on both end.

Reverse Proxy Configurations

Most reverse proxy software implement HTTP and TCP proxy mode. For NiFi RAW Site-to-Site protocol, both HTTP and TCP proxy configurations are required, and at least 2 ports needed to be opened. NiFi HTTP Site-to-Site protocol can minimize the required number of open ports at the reverse proxy to 1.

Setting correct HTTP headers at reverse proxies are crucial for NiFi to work correctly, not only routing requests but also authorize client requests.

There are two types of requests-to-NiFi-node mapping techniques those can be applied at reverse proxy servers. One is 'Server name to Node' and the other is 'Port number to Node'.

With 'Server name to Node', the same port can be used to route requests to different upstream NiFi nodes based on the requested server name (e.g. nifi0.example.com, nifi1.example.com). Host name resolution should be configured to map different host names to the same reverse proxy address, that can be done by adding /etc/hosts file or DNS server entries. Also, if clients to reverse proxy uses HTTPS, reverse proxy server certificate should have wildcard common name or SAN to be accessed by different host names.

Some reverse proxy technologies do not support server name routing rules, in such case, use 'Port number to Node' technique. 'Port number to Node' mapping requires N open port at a reverse proxy for a NiFi cluster consists of N nodes.

Refer to the following examples for actual configurations.

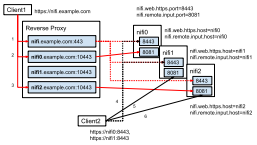
Site to Site and Reverse Proxy Examples

Here are some example reverse proxy and NiFi setups to illustrate what configuration files look like.

Client1 in the following diagrams represents a client that does not have direct access to NiFi nodes, and it accesses through the reverse proxy, while Client2 has direct access.

In this example, Nginx is used as a reverse proxy.

Example 1: RAW - Server name to Node mapping



1. Client1 initiates Site-to-Site protocol, the request is routed to one of upstream NiFi nodes. The NiFi node computes Site-to-Site port for RAW. By the routing rule example1 in nifi.properties shown below, port 10443 is returned.

2. Client1 asks peers to nifi.example.com:10443, the request is routed to nifi0:8081. The NiFi node computes available peers, by example1 routing rule, nifi0:8081 is converted to nifi0.example.com:10443, so are nifi1 and nifi2. As a result, nifi0.example.com:10443, nifi1.example.com:10443 and nifi2.example.com:10443 are returned.
3. Client1 decides to use nifi2.example.com:10443 for further communication.
4. On the other hand, Client2 has two URIs for Site-to-Site bootstrap URIs, and initiates the protocol using one of them. The example1 routing does not match this for this request, and port 8081 is returned.
5. Client2 asks peers from nifi1:8081. The example1 does not match, so the original nifi0:8081, nifi1:8081 and nifi2:8081 are returned as they are.
6. Client2 decides to use nifi2:8081 for further communication.

Routing rule example1 defined in nifi.properties (all nodes have the same routing configuration):

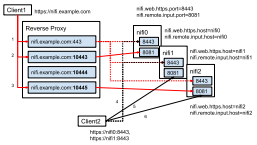
```
# S2S Routing for RAW, using server name to node
nifi.remote.route.raw.example1.when=\
${X-ProxyHost:equals('nifi.example.com'):or(\
${s2s.source.hostname:equals('nifi.example.com'):or(\
${s2s.source.hostname:equals('192.168.99.100')})})}
nifi.remote.route.raw.example1.hostname=${s2s.target.hostname}.example.com
nifi.remote.route.raw.example1.port=10443
nifi.remote.route.raw.example1.secure=true
```

nginx.conf :

```
http {

    upstream nifi {
        server nifi0:8443;
        server nifi1:8443;
        server nifi2:8443;
    }

    # Use dnsmasq so that hostnames such as 'nifi0' can be resolved by /etc/
hosts
    resolver 127.0.0.1;

    server {
        listen 443 ssl;
        server_name nifi.example.com;
        ssl_certificate /etc/nginx/nginx.crt;
        ssl_certificate_key /etc/nginx/nginx.key;

        proxy_ssl_certificate /etc/nginx/nginx.crt;
        proxy_ssl_certificate_key /etc/nginx/nginx.key;
        proxy_ssl_trusted_certificate /etc/nginx/nifi-cert.pem;

        location / {
            proxy_pass https://nifi;
            proxy_set_header X-ProxyScheme https;
            proxy_set_header X-ProxyHost nginx.example.com;
            proxy_set_header X-ProxyPort 17590;
            proxy_set_header X-ProxyContextPath /;
            proxy_set_header X-ProxiedEntitiesChain $ssl_client_s_dn;
        }
    }
}

stream {

    map $ssl_preread_server_name $nifi {
        nifi0.example.com nifi0;
        nifi1.example.com nifi1;
        nifi2.example.com nifi2;
```

```
        default nifi0;
    }

    resolver 127.0.0.1;

    server {
        listen 10443;
        proxy_pass $nifi:8081;
    }
}
```

Example 2: RAW - Port number to Node mapping



The example2 routing maps original host names (nifi0, nifi1 and nifi2) to different proxy ports (10443, 10444 and 10445) using equals and ifElse expressions.

Routing rule example2 defined in nifi.properties (all nodes have the same routing configuration):

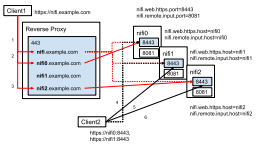
```
# S2S Routing for RAW, using port number to node
nifi.remote.route.raw.example2.when=\
${X-ProxyHost:equals('nifi.example.com'):or(\
${s2s.source.hostname:equals('nifi.example.com'):or(\
${s2s.source.hostname:equals('192.168.99.100')})})}
nifi.remote.route.raw.example2.hostname=nifi.example.com
nifi.remote.route.raw.example2.port=\
${s2s.target.hostname:equals('nifi0'):ifElse('10443',\
${s2s.target.hostname:equals('nifi1'):ifElse('10444',\
${s2s.target.hostname:equals('nifi2'):ifElse('10445',\
'undefined')})})}
nifi.remote.route.raw.example2.secure=true
```

nginx.conf :

```
http {
    # Same as example 1.
}

stream {

    map $ssl_preread_server_name $nifi {
        nifi0.example.com nifi0;
        nifi1.example.com nifi1;
        nifi2.example.com nifi2;
        default nifi0;
    }

    resolver 127.0.0.1;

    server {
        listen 10443;
        proxy_pass nifi0:8081;
    }
```

```
    server {
        listen 10444;
        proxy_pass nifi1:8081;
    }
    server {
        listen 10445;
        proxy_pass nifi2:8081;
    }
}
```

Example 3: HTTP - Server name to Node mapping



Routing rule example3 defined in nifi.properties (all nodes have the same routing configuration):

```
# S2S Routing for HTTP
nifi.remote.route.http.example3.when=${X-ProxyHost:contains('.example.com')}
nifi.remote.route.http.example3.hostname=${s2s.target.hostname}.example.com
nifi.remote.route.http.example3.port=443
nifi.remote.route.http.example3.secure=true
```

nginx.conf :

```
http {
    upstream nifi_cluster {
        server nifi0:8443;
        server nifi1:8443;
        server nifi2:8443;
    }

    # If target node is not specified, use one from cluster.
    map $http_host $nifi {
        nifi0.example.com:443 "nifi0:8443";
        nifi1.example.com:443 "nifi1:8443";
        nifi2.example.com:443 "nifi2:8443";
        default "nifi_cluster";
    }

    resolver 127.0.0.1;

    server {
        listen 443 ssl;
        server_name ~^(.+\.example\.com)$;
        ssl_certificate /etc/nginx/nginx.crt;
        ssl_certificate_key /etc/nginx/nginx.key;

        proxy_ssl_certificate /etc/nginx/nginx.crt;
        proxy_ssl_certificate_key /etc/nginx/nginx.key;
        proxy_ssl_trusted_certificate /etc/nginx/nifi-cert.pem;

        location / {
            proxy_pass https://$nifi;
            proxy_set_header X-ProxyScheme https;
```

```
            proxy_set_header X-ProxyHost $1;
            proxy_set_header X-ProxyPort 443;
            proxy_set_header X-ProxyContextPath /;
            proxy_set_header X-ProxiedEntitiesChain $ssl_client_s_dn;
        }
    }
}
```

Web Properties

These properties pertain to the web-based User Interface.

Property	Description
nifi.web.war.directory	This is the location of the web war directory. The default value is ./lib.
nifi.web.http.host	The HTTP host. It is blank by default.
nifi.web.http.port	The HTTP port. The default value is 8080.
nifi.web.http.port.forwarding	The port which forwards incoming HTTP requests to nifi.web.http.host. This property is designed to be used with 'port forwarding', when NiFi has to be started by a non-root user for better security, yet it needs to be accessed via low port to go through a firewall. For example, to expose NiFi via HTTP protocol on port 80, but actually listening on port 8080, you need to configure OS level port forwarding such as iptables (Linux/Unix) or pfctl (OS X) that redirects requests from 80 to 8080. Then set nifi.web.http.port as 8080, and nifi.web.http.port.forwarding as 80. It is blank by default.
nifi.web.http.network.interface*	The name of the network interface to which NiFi should bind for HTTP requests. It is blank by default.NOTE: Multiple network interfaces can be specified by using the nifi.web.http.network.interface. prefix with unique suffixes and separate network interface names as values. For example, to provide two additional network interfaces, a user could also specify additional properties with keys of:nifi.web.http.network.interface.eth0=eth0 nifi.web.http.network.interface.eth1=eth1 Providing three total network interfaces, including nifi.web.http.network.interface.default.
nifi.web.https.host	The HTTPS host. It is blank by default.
nifi.web.https.port	The HTTPS port. It is blank by default. When configuring NiFi to run securely, this port should be configured.
nifi.web.https.port.forwarding	Same as nifi.web.http.port.forwarding, but with HTTPS for secure communication. It is blank by default.
nifi.web.https.network.interface*	The name of the network interface to which NiFi should bind for HTTPS requests. It is blank by default.NOTE: Multiple network interfaces can be specified by using the nifi.web.https.network.interface. prefix with unique suffixes and separate network interface names as values. For example, to provide two additional network interfaces, a user could also specify additional properties with keys of:nifi.web.https.network.interface.eth0=eth0 nifi.web.https.network.interface.eth1=eth1 Providing three total network interfaces, including nifi.web.https.network.interface.default.
nifi.web.jetty.working.directory	The location of the Jetty working directory. The default value is ./work/jetty.
nifi.web.jetty.threads	The number of Jetty threads. The default value is 200.

nifi.web.max.header.size	The maximum size allowed for request and response headers. The default value is 16 KB.
nifi.web.proxy.host	A comma separated list of allowed HTTP Host header values to consider when NiFi is running securely and will be receiving requests to a different host[:port] than it is bound to. For example, when running in a Docker container or behind a proxy (e.g. localhost:18443, proxyhost:443). By default, this value is blank meaning NiFi should only allow requests sent to the host[:port] that NiFi is bound to.
nifi.web.proxy.context.path	A comma separated list of allowed HTTP X-ProxyContextPath or X-Forwarded-Context header values to consider. By default, this value is blank meaning all requests containing a proxy context path are rejected. Configuring this property would allow requests where the proxy path is contained in this listing.

Security Properties

These properties pertain to various security features in NiFi.

Property	Description
nifi.sensitive.props.key	This is the password used to encrypt any sensitive property values that are configured in processors. By default, it is blank, but the system administrator should provide a value for it. It can be a string of any length, although the recommended minimum length is 10 characters. Be aware that once this password is set and one or more sensitive processor properties have been configured, this password should not be changed.
nifi.sensitive.props.algorithm	The algorithm used to encrypt sensitive properties. The default value is PBEWITHMD5AND256BITAES-CBC-OPENSSL.
nifi.sensitive.props.provider	The sensitive property provider. The default value is BC.
nifi.sensitive.props.additional.keys	The comma separated list of properties in nifi.properties to encrypt in addition to the default sensitive properties.
nifi.security.keystore*	The full path and name of the keystore. It is blank by default.
nifi.security.keystoreType	The keystore type. It is blank by default.
nifi.security.keystorePasswd	The keystore password. It is blank by default.
nifi.security.keyPasswd	The key password. It is blank by default.
nifi.security.truststore*	The full path and name of the truststore. It is blank by default.
nifi.security.truststoreType	The truststore type. It is blank by default.
nifi.security.truststorePasswd	The truststore password. It is blank by default.
nifi.security.user.authorizer	Specifies which of the configured Authorizers in the authorizers.xml file to use. By default, it is set to file-provider.
nifi.security.user.login.identity.provider	This indicates what type of login identity provider to use. The default value is blank, can be set to the identifier from a provider in the file specified in nifi.login.identity.provider.configuration.file. Setting this property will trigger NiFi to support username/password authentication.
nifi.security.ocsp.responder.url	This is the URL for the Online Certificate Status Protocol (OCSP) responder if one is being used. It is blank by default.

nifi.security.ocsp.responder.certificate	This is the location of the OCSP responder certificate if one is being used. It is blank by default.

Identity Mapping Properties

These properties can be utilized to normalize user identities. When implemented, identities authenticated by different identity providers (certificates, LDAP, Kerberos) are treated the same internally in NiFi. As a result, duplicate users are avoided and user-specific configurations such as authorizations only need to be setup once per user.

The following examples demonstrate normalizing DNs from certificates and principals from Kerberos:

```
nifi.security.identity.mapping.pattern.dn=^CN=(.*?), OU=(.*?), O=(.*?),
 L=(.*?), ST=(.*?), C=(.*?)$
nifi.security.identity.mapping.value.dn=$1@$2
nifi.security.identity.mapping.transform.dn=NONE
nifi.security.identity.mapping.pattern.kerb=^(.*?)/instance@(.*?)$
nifi.security.identity.mapping.value.kerb=$1@$2
nifi.security.identity.mapping.transform.kerb=NONE
```

The last segment of each property is an identifier used to associate the pattern with the replacement value. When a user makes a request to NiFi, their identity is checked to see if it matches each of those patterns in lexicographical order. For the first one that matches, the replacement specified in the nifi.security.identity.mapping.value.xxxx property is used. So a login with CN=localhost, OU=Apache NiFi, O=Apache, L=Santa Monica, ST=CA, C=US matches the DN mapping pattern above and the DN mapping value $1@$2 is applied. The user is normalized to localhost@Apache NiFi.

In addition to mapping, a transform may be applied. The supported versions are NONE (no transform applied), LOWER (identity lowercased), and UPPER (identity uppercased). If not specified, the default value is NONE.

Note: These mappings are also applied to the "Initial Admin Identity", "Cluster Node Identity", and any legacy users in the authorizers.xml file as well as users imported from LDAP (See Authorizers.xml Setup).

Group names can also be mapped. The following example will accept the existing group name but will lowercase it. This may be helpful when used in conjunction with an external authorizer.

```
nifi.security.group.mapping.pattern.anygroup=^(.*)$
nifi.security.group.mapping.value.anygroup=$1
nifi.security.group.mapping.transform.anygroup=LOWER
```

Note: These mappings are applied to any legacy groups referenced in the authorizers.xml as well as groups imported from LDAP.

Cluster Common Properties

When setting up a NiFi cluster, these properties should be configured the same way on all nodes.

Property	Description
nifi.cluster.protocol.heartbeat.interval	The interval at which nodes should emit heartbeats to the Cluster Coordinator. The default value is 5 sec.
nifi.cluster.protocol.is.secure	This indicates whether cluster communications are secure. The default value is false.

Cluster Node Properties

Configure these properties for cluster nodes.

Property	Description
nifi.cluster.is.node	Set this to true if the instance is a node in a cluster. The default value is false.
nifi.cluster.node.address	The fully qualified address of the node. It is blank by default.
nifi.cluster.node.protocol.port	The node's protocol port. It is blank by default.
nifi.cluster.node.protocol.threads	The number of threads that should be used to communicate with other nodes in the cluster. This property defaults to 10, but for large clusters, this value may need to be larger.
nifi.cluster.node.protocol.max.threads	The maximum number of threads that should be used to communicate with other nodes in the cluster. This property defaults to 50.
nifi.cluster.node.event.history.size	When the state of a node in the cluster is changed, an event is generated and can be viewed in the Cluster page. This value indicates how many events to keep in memory for each node. The default value is 25.
nifi.cluster.node.connection.timeout	When connecting to another node in the cluster, specifies how long this node should wait before considering the connection a failure. The default value is 5 secs.
nifi.cluster.node.read.timeout	When communicating with another node in the cluster, specifies how long this node should wait to receive information from the remote node before considering the communication with the node a failure. The default value is 5 secs.
nifi.cluster.node.max.concurrent.requests	The maximum number of outstanding web requests that can be replicated to nodes in the cluster. If this number of requests is exceeded, the embedded Jetty server will return a "409: Conflict" response. This property defaults to 100.
nifi.cluster.firewall.file	The location of the node firewall file. This is a file that may be used to list all the nodes that are allowed to connect to the cluster. It provides an additional layer of security. This value is blank by default, meaning that no firewall file is to be used.
nifi.cluster.flow.election.max.wait.time	Specifies the amount of time to wait before electing a Flow as the "correct" Flow. If the number of Nodes that have voted is equal to the number specified by the nifi.cluster.flow.election.max.candidates property, the cluster will not wait this long. The default value is 5 mins. Note that the time starts as soon as the first vote is cast.
nifi.cluster.flow.election.max.candidates	Specifies the number of Nodes required in the cluster to cause early election of Flows. This allows the Nodes in the cluster to avoid having to wait a long time before starting processing if we reach at least this number of nodes in the cluster.
nifi.cluster.flow.election.max.wait.time	Specifies the amount of time to wait before electing a Flow as the "correct" Flow. If the number of Nodes that have voted is equal to the number specified by the nifi.cluster.flow.election.max.candidates property, the cluster will not wait this long. The default value is 5 mins. Note that the time starts as soon as the first vote is cast.
nifi.cluster.flow.election.max.candidates	Specifies the number of Nodes required in the cluster to cause early election of Flows. This allows the Nodes in the cluster to avoid having to wait a long time before starting processing if we reach at least this number of nodes in the cluster.

nifi.cluster.load.balance.port	Specifies the port to listen on for incoming connections for load balancing data across the cluster. The default value is 6342.
nifi.cluster.load.balance.host	Specifies the hostname to listen on for incoming connections for load balancing data across the cluster. If not specified, will default to the value used by the nifi.cluster.node.address property.
nifi.cluster.load.balance.connections.per.node	The maximum number of connections to create between this node and each other node in the cluster. For example, if there are 5 nodes in the cluster and this value is set to 4, there will be up to 20 socket connections established for load-balancing purposes (5 x 4 = 20). The default value is 4.
nifi.cluster.load.balance.max.thread.count	The maximum number of threads to use for transferring data from this node to other nodes in the cluster. If this value is set to 8, for example, there will be up to 8 threads responsible for transferring data to other nodes, regardless of how many nodes are in the cluster. While a given thread can only write to a single socket at a time, a single thread is capable of servicing multiple connections simultaneously because a given connection may not be available for reading/writing at any given time. The default value is 8.
nifi.cluster.load.balance.comms.timeout	When communicating with another node, if this amount of time elapses without making any progress when reading from or writing to a socket, then a TimeoutException will be thrown. This will then result in the data either being retried or sent to another node in the cluster, depending on the configured Load Balancing Strategy. The default value is 30 sec.

Claim Management

Whenever a request is made to change the dataflow, it is important that all nodes in the NiFi cluster are kept in-sync. In order to allow for this, NiFi employs a two-phase commit. The request is first replicated to all nodes in the cluster, simply asking whether or not the request is allowed. Each node then determines whether or not it will allow the request and if so issues a "Claim" on the component(s) being modified. This claim can be thought of as a mutually-exclusive lock that is owned by the requestor. Once all nodes have voted on whether or not the request is allowed, the node from which the request originated must decide whether or not to complete the request. If any node voted 'NO' then the request is canceled and the Claim is canceled with an error message sent back to the user. However, if the nodes all vote 'YES' then the request is completed. In this sort of distributed environment, it is possible that the node that made the original request will fail after the voting has occurred and before the request was completed. This would leave the component locked indefinitely so that no more changes can be made to the component. In order to avoid this, the Claim will time out after some period of time. These properties determines how these locks are managed.

Property	Description
nifi.cluster.request.replication.claim.timeout	Specifies how long to wait before considering a lock 'expired' and automatically unlocking.

ZooKeeper Properties

NiFi depends on Apache ZooKeeper for determining which node in the cluster should play the role of Primary Node and which node should play the role of Cluster Coordinator. These properties must be configured in order for NiFi to join a cluster.

Property	Description

nifi.zookeeper.connect.string	The Connect String that is needed to connect to Apache ZooKeeper. This is a comma-separated list of hostname:port pairs. For example, localhost:2181,localhost:2182,localhost:2183. This should contain a list of all ZooKeeper instances in the ZooKeeper quorum. This property must be specified to join a cluster and has no default value.
nifi.zookeeper.connect.timeout	How long to wait when connecting to ZooKeeper before considering the connection a failure. The default value is 3 secs.
nifi.zookeeper.session.timeout	How long to wait after losing a connection to ZooKeeper before the session is expired. The default value is 3 secs.
nifi.zookeeper.root.node	The root ZNode that should be used in ZooKeeper. ZooKeeper provides a directory-like structure for storing data. Each 'directory' in this structure is referred to as a ZNode. This denotes the root ZNode, or 'directory', that should be used for storing data. The default value is /root. This is important to set correctly, as which cluster the NiFi instance attempts to join is determined by which ZooKeeper instance it connects to and the ZooKeeper Root Node that is specified.

Kerberos Properties

Property	Description
nifi.kerberos.krb5.file*	The location of the krb5 file, if used. It is blank by default. At this time, only a single krb5 file is allowed to be specified per NiFi instance, so this property is configured here to support SPNEGO and service principals rather than in individual Processors. If necessary the krb5 file can support multiple realms. Example: /etc/krb5.conf
nifi.kerberos.service.principal*	The name of the NiFi Kerberos service principal, if used. It is blank by default. Note that this property is for NiFi to authenticate as a client other systems. Example: nifi/nifi.example.com or nifi/nifi.example.com@EXAMPLE.COM
nifi.kerberos.service.keytab.location*	The file path of the NiFi Kerberos keytab, if used. It is blank by default. Note that this property is for NiFi to authenticate as a client other systems. Example: /etc/nifi.keytab
nifi.kerberos.spnego.principal*	The name of the NiFi Kerberos service principal, if used. It is blank by default. Note that this property is used to authenticate NiFi users. Example: HTTP/nifi.example.com or HTTP/nifi.example.com@EXAMPLE.COM
nifi.kerberos.spnego.keytab.location*	The file path of the NiFi Kerberos keytab, if used. It is blank by default. Note that this property is used to authenticate NiFi users. Example: /etc/http-nifi.keytab
nifi.kerberos.spengo.authentication.expiration*	The expiration duration of a successful Kerberos user authentication, if used. The default value is 12 hours.

Custom Properties

To configure custom properties for use with NiFi's Expression Language:

- Create the custom property. Ensure that:
  - Each custom property contains a distinct property value, so that it is not overridden by existing environment properties, system properties, or FlowFile attributes.
  - Each node in a clustered environment is configured with the same custom properties.
- Update nifi.variable.registry.properties with the location of the custom property file(s):

Property	Description
nifi.variable.registry.properties	This is a comma-separated list of file location paths for one or more custom property files.

- Restart your NiFi instance(s) for the updates to be picked up.

Custom properties can also be configured in the NiFi UI.