Apache NiFi 3

NiFi Configuration Best Practices

Date of Publish: 2018-08-13

http://docs.hortonworks.com

Contents

Configuration Best Practices

Note: If you are running on Linux, consider these best practices. Typical Linux defaults are not necessarily well tuned for the needs of an IO intensive application like NiFi. For all of these areas, your distribution's requirements may vary. Use these sections as advice, but consult your distribution-specific documentation for how best to achieve these recommendations.

Maximum File Handles

NiFi will at any one time potentially have a very large number of file handles open. Increase the limits by editing '/etc/security/limits.conf' to add something like

```
*  hard  nofile  50000
*  soft  nofile  50000
```

Maximum Forked Processes

NiFi may be configured to generate a significant number of threads. To increase the allowable number edit '/etc/security/limits.conf'

```
*  hard  nproc  10000
*  soft  nproc  10000
```

And your distribution may require an edit to /etc/security/limits.d/90-nproc.conf by adding

```
*  soft  nproc  10000
```

Increase the number of TCP socket ports available

This is particularly important if your flow will be setting up and tearing down a large number of sockets in small period of time.

```
sudo sysctl -w net.ipv4.ip_local_port_range="10000 65000"
```

Set how long sockets stay in a TIMED_WAIT state when closed

You don't want your sockets to sit and linger too long given that you want to be able to quickly setup and teardown new sockets. It is a good idea to read more about it but to adjust do something like

```
sudo sysctl -w net.ipv4.netfilter.ip_conntrack_tcp_timeout_time_wait="1"
```

Tell Linux you never want NiFi to swap

Swapping is fantastic for some applications. It isn't good for something like NiFi that always wants to be running. To tell Linux you'd like swapping off you can edit '/etc/sysctl.conf' to add the following line

```
vm.swappiness = 0
```

For the partitions handling the various NiFi repos turn off things like 'atime'. Doing so can cause a surprising bump in throughput. Edit the '/etc/fstab' file and for the partition(s) of interest add the 'noatime' option.

Security Configuration

NiFi provides several different configuration options for security purposes. The most important properties are those under the "security properties" heading in the nifi.properties file. In order to run securely, the following properties must be set:

Property Name	Description
nifi.security.needClientAuth	Set to true to specify that connecting clients must authenticate themselves. This property is used by the NiFi cluster protocol to indicate that nodes in the cluster will be authenticated and must have certificates that are trusted by the Truststores.
nifi.security.keystore	Filename of the Keystore that contains the server's private key.
nifi.security.keystoreType	The type of Keystore. Must be either PKCS12 or JKS. JKS is the preferred type, PKCS12 files will be loaded with BouncyCastle provider.
nifi.security.keystorePasswd	The password for the Keystore.
nifi.security.keyPasswd	The password for the certificate in the Keystore. If not set, the value of nifi.security.keystorePasswd will be used.
nifi.security.truststore	Filename of the Truststore that will be used to authorize those connecting to NiFi. A secured instance with no Truststore will refuse all incoming connections.
nifi.security.truststoreType	The type of the Truststore. Must be either PKCS12 or JKS. JKS is the preferred type, PKCS12 files will be loaded with BouncyCastle provider.
nifi.security.truststorePasswd	The password for the Truststore.

Once the above properties have been configured, we can enable the User Interface to be accessed over HTTPS instead of HTTP. This is accomplished by setting the nifi.web.https.host and nifi.web.https.port properties. The nifi.web.https.host property indicates which hostname the server should run on. If it is desired that the HTTPS interface be accessible from all network interfaces, a value of 0.0.0.0 should be used. To allow admins to configure the application to run only on specific network interfaces, nifi.web.http.network.interface* or nifi.web.https.network.interface* properties can be specified.

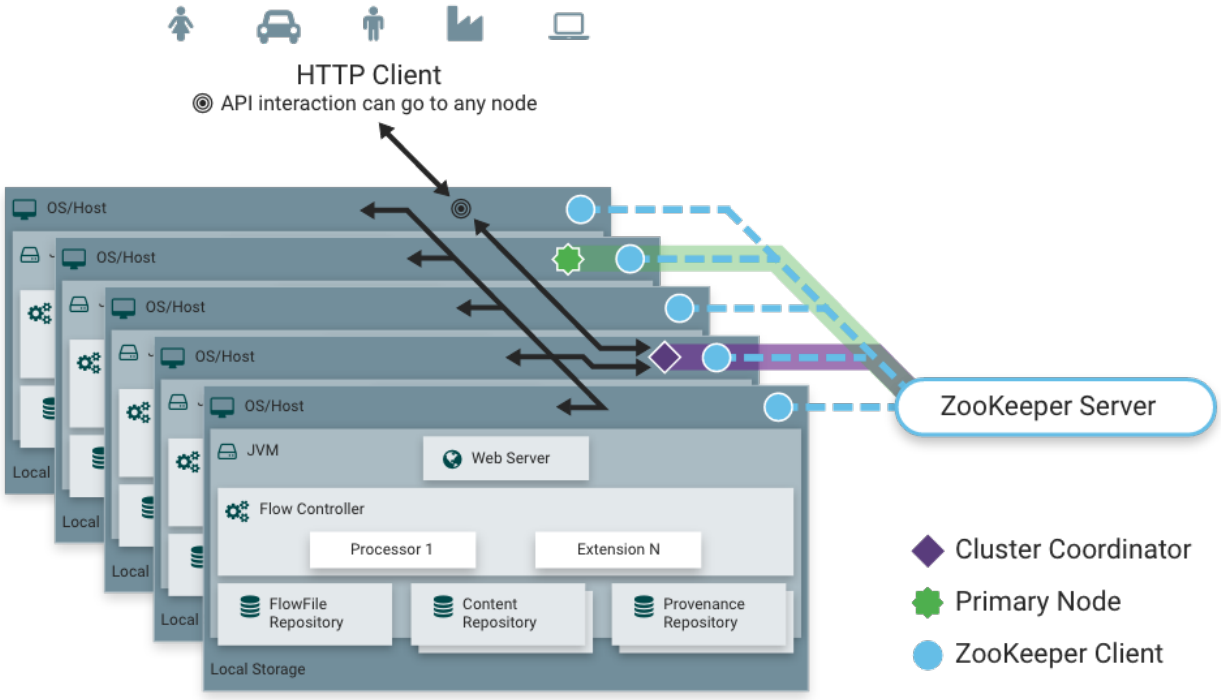
> **Note:** It is important when enabling HTTPS that the nifi.web.http.port property be unset. NiFi only supports running on HTTP or HTTPS, not both simultaneously.

Similar to nifi.security.needClientAuth, the web server can be configured to require certificate based client authentication for users accessing the User Interface. Either of these options will configure the web server to WANT certificate based client authentication. This will allow it to support users with certificates and those without that may be logging in with their credentials or those accessing anonymously. If username/password authentication and anonymous access are not configured, the web server will REQUIRE certificate based client authentication.

Now that the User Interface has been secured, we can easily secure Site-to-Site connections and inner-cluster communications, as well. This is accomplished by setting the nifi.remote.input.secure and nifi.cluster.protocol.is.secure properties, respectively, to true.

Clustering Configuration

This section provides a quick overview of NiFi Clustering and instructions on how to set up a basic cluster. In the future, we hope to provide supplemental documentation that covers the NiFi Cluster Architecture in depth.



NiFi employs a Zero-Master Clustering paradigm. Each node in the cluster performs the same tasks on the data, but each operates on a different set of data. One of the nodes is automatically elected (via Apache ZooKeeper) as the Cluster Coordinator. All nodes in the cluster will then send heartbeat/status information to this node, and this node is responsible for disconnecting nodes that do not report any heartbeat status for some amount of time. Additionally, when a new node elects to join the cluster, the new node must first connect to the currently-elected Cluster Coordinator in order to obtain the most up-to-date flow. If the Cluster Coordinator determines that the node is allowed to join (based on its configured Firewall file), the current flow is provided to that node, and that node is able to join the cluster, assuming that the node's copy of the flow matches the copy provided by the Cluster Coordinator. If the node's version of the flow configuration differs from that of the Cluster Coordinator's, the node will not join the cluster.

Why Cluster?

NiFi Administrators or Dataflow Managers (DFMs) may find that using one instance of NiFi on a single server is not enough to process the amount of data they have. So, one solution is to run the same dataflow on multiple NiFi servers. However, this creates a management problem, because each time DFMs want to change or update the dataflow, they must make those changes on each server and then monitor each server individually. By clustering the NiFi servers, it's possible to have that increased processing capability along with a single interface through which to make dataflow changes and monitor the dataflow. Clustering allows the DFM to make each change only once, and that change is then replicated to all the nodes of the cluster. Through the single interface, the DFM may also monitor the health and status of all the nodes.

NiFi Clustering is unique and has its own terminology. It's important to understand the following terms before setting up a cluster.

Terminology

NiFi Cluster Coordinator: A NiFi Cluster Cluster Coordinator is the node in a NiFi cluster that is responsible for carrying out tasks to manage which nodes are allowed in the cluster and providing the most up-to-date flow to newly joining nodes. When a DataFlow Manager manages a dataflow in a cluster, they are able to do so through the User Interface of any node in the cluster. Any change made is then replicated to all nodes in the cluster.

Nodes: Each cluster is made up of one or more nodes. The nodes do the actual data processing.

Primary Node: Every cluster has one Primary Node. On this node, it is possible to run "Isolated Processors" (see below). ZooKeeper is used to automatically elect a Primary Node. If that node disconnects from the cluster for any reason, a new Primary Node will automatically be elected. Users can determine which node is currently elected as the Primary Node by looking at the Cluster Management page of the User Interface.

Isolated Processors: In a NiFi cluster, the same dataflow runs on all the nodes. As a result, every component in the flow runs on every node. However, there may be cases when the DFM would not want every processor to run on every node. The most common case is when using a processor that communicates with an external service using a protocol that does not scale well. For example, the GetSFTP processor pulls from a remote directory, and if the GetSFTP Processor runs on every node in the cluster tries simultaneously to pull from the same remote directory, there could be race conditions. Therefore, the DFM could configure the GetSFTP on the Primary Node to run in isolation, meaning that it only runs on that node. It could pull in data and - with the proper dataflow configuration - load-balance it across the rest of the nodes in the cluster. Note that while this feature exists, it is also very common to simply use a standalone NiFi instance to pull data and feed it to the cluster. It just depends on the resources available and how the Administrator decides to configure the cluster.

Heartbeats: The nodes communicate their health and status to the currently elected Cluster Coordinator via "heartbeats", which let the Coordinator know they are still connected to the cluster and working properly. By default, the nodes emit heartbeats every 5 seconds, and if the Cluster Coordinator does not receive a heartbeat from a node within 40 seconds, it disconnects the node due to "lack of heartbeat". (The 5-second setting is configurable in the nifi.properties file.) The reason that the Cluster Coordinator disconnects the node is because the Coordinator needs to ensure that every node in the cluster is in sync, and if a node is not heard from regularly, the Coordinator cannot be sure it is still in sync with the rest of the cluster. If, after 40 seconds, the node does send a new heartbeat, the Coordinator will automatically request that the node re-join the cluster, to include the re-validation of the node's flow. Both the disconnection due to lack of heartbeat and the reconnection once a heartbeat is received are reported to the DFM in the User Interface.

Communication within the Cluster

As noted, the nodes communicate with the Cluster Coordinator via heartbeats. When a Cluster Coordinator is elected, it updates a well-known ZNode in Apache ZooKeeper with its connection information so that nodes understand where to send heartbeats. If one of the nodes goes down, the other nodes in the cluster will not automatically pick up the load of the missing node. It is possible for the DFM to configure the dataflow for failover contingencies; however, this is dependent on the dataflow design and does not happen automatically.

When the DFM makes changes to the dataflow, the node that receives the request to change the flow communicates those changes to all nodes and waits for each node to respond, indicating that it has made the change on its local flow.

Dealing with Disconnected Nodes

A DFM may manually disconnect a node from the cluster. But if a node becomes disconnected for any other reason (such as due to lack of heartbeat), the Cluster Coordinator will show a bulletin on the User Interface. The DFM will not be able to make any changes to the dataflow until the issue of the disconnected node is resolved. The DFM or the Administrator will need to troubleshoot the issue with the node and resolve it before any new changes may be made to the dataflow. However, it is worth noting that just because a node is disconnected does not mean that it is not working; this may happen for a few reasons, including that the node is unable to communicate with the Cluster Coordinator due to network problems.

There are cases where a DFM may wish to continue making changes to the flow, even though a node is not connected to the cluster. In this case, they DFM may elect to remove the node from the cluster entirely through the Cluster Management dialog. Once removed, the node cannot be rejoined to the cluster until it has been restarted.

Flow Election When a cluster first starts up, NiFi must determine which of the nodes have the "correct" version of the flow. This is done by voting on the flows that each of the nodes has. When a node attempts to connect to a cluster, it provides a copy of its local flow to the Cluster Coordinator. If no flow has yet been elected the "correct" flow, the node's flow is compared to each of the other Nodes' flows. If another Node's flow matches this one, a vote is cast for this flow. If no other Node has reported the same flow yet, this flow will be added to the pool of possibly elected flows with one vote. After some amount of time has elapsed (configured by setting the nifi.cluster.flow.election.max.wait.time property) or some number of Nodes have cast votes (configured by setting the

nifi.cluster.flow.election.max.candidates property), a flow is elected to be the "correct" copy of the flow. All nodes that have incompatible flows are then disconnected from the cluster while those with compatible flows inherit the cluster's flow. Election is performed according to the "popular vote" with the caveat that the winner will never be an "empty flow" unless all flows are empty. This allows an administrator to remove a node's flow.xml.gz file and restart the node, knowing that the node's flow will not be voted to be the "correct" flow unless no other flow is found.

Basic Cluster Setup

This section describes the setup for a simple three-node, non-secure cluster comprised of three instances of NiFi.

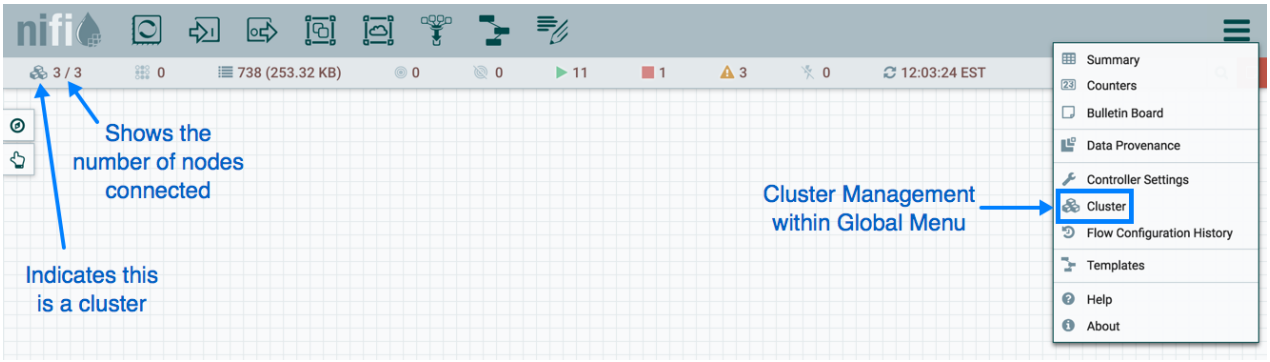
For each instance, certain properties in the nifi.properties file will need to be updated. In particular, the Web and Clustering properties should be evaluated for your situation and adjusted accordingly.

For all three instances, the Cluster Common Properties can be left with the default settings. Note, however, that if you change these settings, they must be set the same on every instance in the cluster.

For each Node, the minimum properties to configure are as follows:

- Under the Web Properties section, set either the HTTP or HTTPS port that you want the Node to run on. Also, consider whether you need to set the HTTP or HTTPS host property. All nodes in the cluster should use the same protocol setting.
- Under the State Management section, set the nifi.state.management.provider.cluster property to the identifier of the Cluster State Provider. Ensure that the Cluster State Provider has been configured in the state-management.xml file.
- Under Cluster Node Properties, set the following:
    - nifi.cluster.is.node - Set this to true.
    - nifi.cluster.node.address - Set this to the fully qualified hostname of the node. If left blank, it defaults to "localhost".
    - nifi.cluster.node.protocol.port - Set this to an open port that is higher than 1024 (anything lower requires root).
    - 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. A thread pool is used for replicating requests to all nodes, and the thread pool will never have fewer than this number of threads. It will grow as needed up to the maximum value set by the nifi.cluster.node.protocol.max.threads property.
    - 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. A thread pool is used for replication requests to all nodes, and the thread pool will have a "core" size that is configured by the nifi.cluster.node.protocol.threads property. However, if necessary, the thread pool will increase the number of active threads to the limit set by this property.
    - nifi.zookeeper.connect.string - The Connect String that is needed to connect to Apache ZooKeeper. This is a comma-separted 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.
    - 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.
    - 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.

Now, it is possible to start up the cluster. It does not matter which order the instances start up. Navigate to the URL for one of the nodes, and the User Interface should look similar to the following:



Troubleshooting

If you encounter issues and your cluster does not work as described, investigate the nifi-app.log and nifi-user.log files on the nodes. If needed, you can change the logging level to DEBUG by editing the conf/logback.xml file. Specifically, set the level="DEBUG" in the following line (instead of "INFO"):

```
   <logger name="org.apache.nifi.web.api.config" level="INFO"
additivity="false">
       <appender-ref ref="USER_FILE"/>
   </logger>
```

Bootstrap Properties

The bootstrap.conf file in the conf directory allows users to configure settings for how NiFi should be started. This includes parameters, such as the size of the Java Heap, what Java command to run, and Java System Properties.

Here, we will address the different properties that are made available in the file. Any changes to this file will take effect only after NiFi has been stopped and restarted.

Property	Description
java	Specifies the fully qualified java command to run. By default, it is simply java but could be changed to an absolute path or a reference an environment variable, such as $JAVA_HOME/bin/java
run.as	The username to run NiFi as. For instance, if NiFi should be run as the 'nifi' user, setting this value to 'nifi' will cause the NiFi Process to be run as the 'nifi' user. This property is ignored on Windows. For Linux, the specified user may require sudo permissions.
lib.dir	The lib directory to use for NiFi. By default, this is set to ./lib
conf.dir	The conf directory to use for NiFi. By default, this is set to ./conf
graceful.shutdown.seconds	When NiFi is instructed to shutdown, the Bootstrap will wait this number of seconds for the process to shutdown cleanly. At this amount of time, if the service is still running, the Bootstrap will "kill" the process, or terminate it abruptly.
java.arg.N	Any number of JVM arguments can be passed to the NiFi JVM when the process is started. These arguments are defined by adding properties to bootstrap.conf that begin with java.arg.. The rest of the property name is not relevant, other than to different property names, and will be ignored. The default includes properties for minimum and maximum Java Heap size, the garbage collector to use, etc.

notification.services.file	When NiFi is started, or stopped, or when the Bootstrap detects that NiFi has died, the Bootstrap is able to send notifications of these events to interested parties. This is configured by specifying an XML file that defines which notification services can be used.
notification.max.attempts	If a notification service is configured but is unable to perform its function, it will try again up to a maximum number of attempts. This property configures what that maximum number of attempts is. The default value is 5.
nifi.start.notification.services	This property is a comma-separated list of Notification Service identifiers that correspond to the Notification Services defined in the notification.services.file property. The services with the specified identifiers will be used to notify their configured recipients whenever NiFi is started.
nifi.stop.notification.services	This property is a comma-separated list of Notification Service identifiers that correspond to the Notification Services defined in the notification.services.file property. The services with the specified identifiers will be used to notify their configured recipients whenever NiFi is stopped.
nifi.died.notification.services	This property is a comma-separated list of Notification Service identifiers that correspond to the Notification Services defined in the notification.services.file property. The services with the specified identifiers will be used to notify their configured recipients if the bootstrap determines that NiFi has unexpectedly died.

Notification Services

When the NiFi bootstrap starts or stops NiFi, or detects that it has died unexpectedly, it is able to notify configured recipients. Currently, the only mechanisms supplied are to send an e-mail or HTTP POST notification. The notification services configuration file is an XML file where the notification capabilities are configured.

The default location of the XML file is conf/bootstrap-notification-services.xml, but this value can be changed in the conf/bootstrap.conf file.

The syntax of the XML file is as follows:

```
<services>
    <!-- any number of service elements can be defined. -->
    <service>
        <id>some-identifier</id>
        <!-- The fully-qualified class name of the Notification Service. -->

 <class>org.apache.nifi.bootstrap.notification.email.EmailNotificationService</
class>

        <!-- Any number of properties can be set using this syntax.
             The properties available depend on the Notification Service. --
>
        <property name="Property Name 1">Property Value</property>
        <property name="Another Property Name">Property Value 2</property>
    </service>
</services>
```

Once the desired services have been configured, they can then be referenced in the bootstrap.conf file.

Email Notification Service

The first Notifier is to send emails and the implementation is org.apache.nifi.bootstrap.notification.email.EmailNotificationService. It has the following properties available:

Property	Required	Description
SMTP Hostname	true	The hostname of the SMTP Server that is used to send Email Notifications
SMTP Port	true	The Port used for SMTP communications
SMTP Username	true	Username for the SMTP account
SMTP Password		Password for the SMTP account
SMTP Auth		Flag indicating whether authentication should be used
SMTP TLS		Flag indicating whether TLS should be enabled
SMTP Socket Factory		javax.net.ssl.SSLSocketFactory
SMTP X-Mailer Header		X-Mailer used in the header of the outgoing email
Content Type		Mime Type used to interpret the contents of the email, such as text/plain or text/html
From	true	Specifies the Email address to use as the sender. Otherwise, a "friendly name" can be used as the From address, but the value must be enclosed in double-quotes.
To		The recipients to include in the To-Line of the email
CC		The recipients to include in the CC-Line of the email
BCC		The recipients to include in the BCC-Line of the email

In addition to the properties above that are marked as required, at least one of the To, CC, or BCC properties must be set.

A complete example of configuring the Email service would look like the following:

```
     <service>
        <id>email-notification</id>

  <class>org.apache.nifi.bootstrap.notification.email.EmailNotificationService</
class>
        <property name="SMTP Hostname">smtp.gmail.com</property>
        <property name="SMTP Port">587</property>
        <property name="SMTP Username">username@gmail.com</property>
        <property name="SMTP Password">super-secret-password</property>
        <property name="SMTP TLS">true</property>
        <property name="From">"NiFi Service Notifier"</property>
        <property name="To">username@gmail.com</property>
     </service>
```

HTTP Notification Service

The second Notifier is to send HTTP POST requests and the implementation is org.apache.nifi.bootstrap.notification.http.HttpNotificationService. It has the following properties available:

Property	Required	Description
URL	true	The URL to send the notification to. Expression language is supported.
Connection timeout		Max wait time for connection to remote service. Expression language is supported. This defaults to 10s.
Write timeout		Max wait time for remote service to read the request sent. Expression language is supported. This defaults to 10s.
Truststore Filename		The fully-qualified filename of the Truststore
Truststore Type		The Type of the Truststore. Either JKS or PKCS12
Truststore Password		The password for the Truststore
Keystore Filename		The fully-qualified filename of the Keystore
Keystore Type		The password for the Keystore
Keystore Password		The password for the key. If this is not specified, but the Keystore Filename, Password, and Type are specified, then the Keystore Password will be assumed to be the same as the Key Password.
SSL Protocol		The algorithm to use for this SSL context. This can either be "SSL" or "TLS".

In addition to the properties above, dynamic properties can be added. They will be added as headers to the HTTP request. Expression language is supported.

The notification message is in the body of the POST request. The type of notification is in the header "notification.type" and the subject uses the header "notification.subject".

A complete example of configuring the HTTP service could look like the following:

```
     <service>
        <id>http-notification</id>

 <class>org.apache.nifi.bootstrap.notification.http.HttpNotificationService</
class>
        <property name="URL">https://testServer.com:8080/</property>
        <property name="Truststore Filename">localhost-ts.jks</property>
        <property name="Truststore Type">JKS</property>
        <property name="Truststore Password">localtest<property>
        <property name="Keystore Filename">localhost-ts.jks</property>
        <property name="Keystore Type">JKS</property>
        <property name="Keystore Password">localtest</property>
        <property name="notification.timestamp">${now()}</property>
     </service>
```

Proxy Configuration

When running Apache NiFi behind a proxy there are a couple of key items to be aware of during deployment.

- NiFi is comprised of a number of web applications (web UI, web API, documentation, custom UIs, data viewers, etc), so the mapping needs to be configured for the root path. That way all context paths are passed through accordingly. For instance, if only the /nifi context path was mapped, the custom UI for UpdateAttribute will not work, since it is available at /update-attribute-ui-<version>.
- NiFi's REST API will generate URIs for each component on the graph. Since requests are coming through a proxy, certain elements of the URIs being generated need to be overridden. Without overriding, the users will be able to view the dataflow on the canvas but will be unable to modify existing components. Requests will be attempting to call back directly to NiFi, not through the proxy. The elements of the URI can be overridden by adding the following HTTP headers when the proxy generates the HTTP request to the NiFi instance:

```
X-ProxyScheme - the scheme to use to connect to the proxy
X-ProxyHost - the host of the proxy
X-ProxyPort - the port the proxy is listening on
X-ProxyContextPath - the path configured to map to the NiFi instance
```

- If NiFi is running securely, any proxy needs to be authorized to proxy user requests. These can be configured in the NiFi UI through the Global Menu. Once these permissions are in place, proxies can begin proxying user requests. The end user identity must be relayed in a HTTP header. For example, if the end user sent a request to the proxy, the proxy must authenticate the user. Following this the proxy can send the request to NiFi. In this request an HTTP header should be added as follows.

```
X-ProxiedEntitiesChain: <end-user-identity>
```

If the proxy is configured to send to another proxy, the request to NiFi from the second proxy should contain a header as follows.

```
X-ProxiedEntitiesChain: <end-user-identity><proxy-1-identity>
```

An example Apache proxy configuration that sets the required properties may look like the following. Complete proxy configuration is outside of the scope of this document. Please refer the documentation of the proxy for guidance for your deployment environment and use case.

```
...
<Location "/my-nifi">
    ...
        SSLEngine On
        SSLCertificateFile /path/to/proxy/certificate.crt
        SSLCertificateKeyFile /path/to/proxy/key.key
        SSLCACertificateFile /path/to/ca/certificate.crt
        SSLVerifyClient require
        RequestHeader add X-ProxyScheme "https"
        RequestHeader add X-ProxyHost "proxy-host"
        RequestHeader add X-ProxyPort "443"
        RequestHeader add X-ProxyContextPath "/my-nifi"
        RequestHeader add X-ProxiedEntitiesChain "<%{SSL_CLIENT_S_DN}>"
        ProxyPass https://nifi-host:8443
        ProxyPassReverse https://nifi-host:8443
        ...
</Location>
...
```

- Additional NiFi proxy configuration must be updated to allow expected Host and context paths HTTP headers.
    - By default, if NiFi is running securely it will only accept HTTP requests with a Host header matching the host[:port] that it is bound to. If NiFi is to accept requests directed to a different host[:port] the expected values need to be configured. This may be required when running behind a proxy or in a containerized environment. This is configured in a comma separated list in nifi.properties using the nifi.web.proxy.host property (e.g.

localhost:18443, proxyhost:443). IPv6 addresses are accepted. Please refer to RFC 5952 Sections https://tools.ietf.org/html/rfc5952#section-4 and https://tools.ietf.org/html/rfc5952#section-6 for additional details.

- NiFi will only accept HTTP requests with a X-ProxyContextPath or X-Forwarded-Context header if the value is whitelisted in the nifi.web.proxy.context.path property in nifi.properties. This property accepts a comma separated list of expected values. In the event an incoming request has an X-ProxyContextPath or X-Forwarded-Context header value that is not present in the whitelist, the "An unexpected error has occurred" page will be shown and an error will be written to the nifi-app.log.

- Additional configurations at both proxy server and NiFi cluster are required to make NiFi Site-to-Site work behind reverse proxies.
  - In order to transfer data via Site-to-Site protocol through reverse proxies, both proxy and Site-to-Site client NiFi users need to have following policies, 'retrieve site-to-site details', 'receive data via site-to-site' for input ports, and 'send data via site-to-site' for output ports.