Apache NiFi 3

NiFi State Management

Date of Publish: 2019-03-15

https://docs.hortonworks.com/

Contents

State Management

NiFi provides a mechanism for Processors, Reporting Tasks, Controller Services, and the framework itself to persist state. This allows a Processor, for example, to resume from the place where it left off after NiFi is restarted. Additionally, it allows for a Processor to store some piece of information so that the Processor can access that information from all of the different nodes in the cluster. This allows one node to pick up where another node left off, or to coordinate across all of the nodes in a cluster.

Configuring State Providers

When a component decides to store or retrieve state, it does so by providing a "Scope" - either Node-local or Cluster-wide. The mechanism that is used to store and retrieve this state is then determined based on this Scope, as well as the configured State Providers. The nifi.properties file contains three different properties that are relevant to configuring these State Providers.

Property	Description
nifi.state.management.configuration.file	The first is the property that specifies an external XML file that is used for configuring the local and/or cluster-wide State Providers. This XML file may contain configurations for multiple providers
nifi.state.management.provider.local	The property that provides the identifier of the local State Provider configured in this XML file
nifi.state.management.provider.cluster	Similarly, the property provides the identifier of the cluster-wide State Provider configured in this XML file.

This XML file consists of a top-level state-management element, which has one or more local-provider and zero or more cluster-provider elements. Each of these elements then contains an id element that is used to specify the identifier that can be referenced in the nifi.properties file, as well as a class element that specifies the fully-qualified class name to use in order to instantiate the State Provider. Finally, each of these elements may have zero or more property elements. Each property element has an attribute, name that is the name of the property that the State Provider supports. The textual content of the property element is the value of the property.

Once these State Providers have been configured in the state-management.xml file (or whatever file is configured), those Providers may be referenced by their identifiers.

By default, the Local State Provider is configured to be a WriteAheadLocalStateProvider that persists the data to the $NIFI_HOME/state/local directory. The default Cluster State Provider is configured to be a ZooKeeperStateProvider. The default ZooKeeper-based provider must have its Connect String property populated before it can be used. It is also advisable, if multiple NiFi instances will use the same ZooKeeper instance, that the value of the Root Node property be changed. For instance, one might set the value to /nifi/<team name>/production. A Connect String takes the form of comma separated <host>:<port> tuples, such as my-zk-server1:2181,my-zk-server2:2181,my-zk-server3:2181. In the event a port is not specified for any of the hosts, the ZooKeeper default of 2181 is assumed.

When adding data to ZooKeeper, there are two options for Access Control: Open and CreatorOnly. If the Access Control property is set to Open, then anyone is allowed to log into ZooKeeper and have full permissions to see, change, delete, or administer the data. If CreatorOnly is specified, then only the user that created the data is allowed to read, change, delete, or administer the data. In order to use the CreatorOnly option, NiFi must provide some form of authentication.

If NiFi is configured to run in a standalone mode, the cluster-provider element need not be populated in the state-management.xml file and will actually be ignored if they are populated. However, the local-provider element must always be present and populated. Additionally, if NiFi is run in a cluster, each node must also have the cluster-provider element present and properly configured. Otherwise, NiFi will fail to startup.

While there are not many properties that need to be configured for these providers, they were externalized into a separate state-management.xml file, rather than being configured via the nifi.properties file, simply because different implementations may require different properties, and it is easier to maintain and understand the configuration in an XML-based file such as this, than to mix the properties of the Provider in with all of the other NiFi framework-specific properties.

It should be noted that if Processors and other components save state using the Clustered scope, the Local State Provider will be used if the instance is a standalone instance (not in a cluster) or is disconnected from the cluster. This also means that if a standalone instance is migrated to become a cluster, then that state will no longer be available, as the component will begin using the Clustered State Provider instead of the Local State Provider.

Embedded ZooKeeper Server

As mentioned above, the default State Provider for cluster-wide state is the ZooKeeperStateProvider. At the time of this writing, this is the only State Provider that exists for handling cluster-wide state. What this means is that NiFi has dependencies on ZooKeeper in order to behave as a cluster. However, there are many environments in which NiFi is deployed where there is no existing ZooKeeper ensemble being maintained. In order to avoid the burden of forcing administrators to also maintain a separate ZooKeeper instance, NiFi provides the option of starting an embedded ZooKeeper server.

Property	Description
nifi.state.management.embedded.zookeeper.start	Specifies whether or not this instance of NiFi should run an embedded ZooKeeper server
nifi.state.management.embedded.zookeeper.properties	Properties file that provides the ZooKeeper properties to use if nifi.state.management.embedded.zookeeper.start is set to true

This can be accomplished by setting the nifi.state.management.embedded.zookeeper.start property in nifi.properties to true on those nodes that should run the embedded ZooKeeper server. Generally, it is advisable to run ZooKeeper on either 3 or 5 nodes. Running on fewer than 3 nodes provides less durability in the face of failure. Running on more than 5 nodes generally produces more network traffic than is necessary. Additionally, running ZooKeeper on 4 nodes provides no more benefit than running on 3 nodes, ZooKeeper requires a majority of nodes be active in order to function. However, it is up to the administrator to determine the number of nodes most appropriate to the particular deployment of NiFi.

If the nifi.state.management.embedded.zookeeper.start property is set to true, the nifi.state.management.embedded.zookeeper.properties property in nifi.properties also becomes relevant. This specifies the ZooKeeper properties file to use. At a minimum, this properties file needs to be populated with the list of ZooKeeper servers. The servers are specified as properties in the form of server.1, server.2, to server.n. Each of these servers is configured as <hostname>:<quorum port>[:<leader election port>]. For example, myhost:2888:3888. This list of nodes should be the same nodes in the NiFi cluster that have the nifi.state.management.embedded.zookeeper.start property set to true. Also note that because ZooKeeper will be listening on these ports, the firewall may need to be configured to open these ports for incoming traffic, at least between nodes in the cluster. Additionally, the port to listen on for client connections must be opened in the firewall. The default value for this is 2181 but can be configured via the clientPort property in the zookeeper.properties file.

When using an embedded ZooKeeper, the ./conf/zookeeper.properties file has a property named dataDir. By default, this value is set to ./state/zookeeper. If more than one NiFi node is running an embedded ZooKeeper, it is important to tell the server which one it is. This is accomplished by creating a file named myid and placing it in ZooKeeper's data directory. The contents of this file should be the index of the server as specific by the server.<number>. So for one of the ZooKeeper servers, we will accomplish this by performing the following commands:

```
         cd $NIFI_HOME
mkdir state
mkdir state/zookeeper
echo 1 > state/zookeeper/myid
```

For the next NiFi Node that will run ZooKeeper, we can accomplish this by performing the following commands:

```
cd $NIFI_HOME
mkdir state
mkdir state/zookeeper
echo 2 > state/zookeeper/myid
```

And so on.

For more information on the properties used to administer ZooKeeper, see the https://zookeeper.apache.org/doc/current/zookeeperAdmin.html.

ZooKeeper Access Control

ZooKeeper provides Access Control to its data via an Access Control List (ACL) mechanism. When data is written to ZooKeeper, NiFi will provide an ACL that indicates that any user is allowed to have full permissions to the data, or an ACL that indicates that only the user that created the data is allowed to access the data. Which ACL is used depends on the value of the Access Control property for the ZooKeeperStateProvider.

In order to use an ACL that indicates that only the Creator is allowed to access the data, we need to tell ZooKeeper who the Creator is. There are two mechanisms for accomplishing this. The first mechanism is to provide authentication using Kerberos.

The second option is to use a user name and password. This is configured by specifying a value for the Username and a value for the Password properties for the ZooKeeperStateProvider. The important thing to keep in mind here, though, is that ZooKeeper will pass around the password in plain text. This means that using a user name and password should not be used unless ZooKeeper is running on localhost as a one-instance cluster, or if communications with ZooKeeper occur only over encrypted communications, such as a VPN or an SSL connection. ZooKeeper will be providing support for SSL connections in version 3.5.0.

Securing ZooKeeper

When NiFi communicates with ZooKeeper, all communications, by default, are non-secure, and anyone who logs into ZooKeeper is able to view and manipulate all of the NiFi state that is stored in ZooKeeper. To prevent this, we can use Kerberos to manage the authentication. At this time, ZooKeeper does not provide support for encryption via SSL. Support for SSL in ZooKeeper is being actively developed and is expected to be available in the 3.5.x release version.

In order to secure the communications, we need to ensure that both the client and the server support the same configuration. Instructions for configuring the NiFi ZooKeeper client and embedded ZooKeeper server to use Kerberos are provided below.

If Kerberos is not already setup in your environment, you can find information on installing and setting up a Kerberos Server at https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Managing_Smart_Cards/Configuring_a_Kerberos_5_Server.html. This guide assumes that Kerberos already has been installed in the environment in which NiFi is running.

Note, the following procedures for kerberizing an Embedded ZooKeeper server in your NiFi Node and kerberizing a ZooKeeper NiFi client will require that Kerberos client libraries be installed. This is accomplished in Fedora-based Linux distributions via:

```
yum install krb5-workstation
```

Once this is complete, the /etc/krb5.conf will need to be configured appropriately for your organization's Kerberos environment.

Kerberizing Embedded ZooKeeper Server

The krb5.conf file on the systems with the embedded zookeeper servers should be identical to the one on the system where the krb5kdc service is running. When using the embedded ZooKeeper server, we may choose to secure the server by using Kerberos. All nodes configured to launch an embedded ZooKeeper and using Kerberos should follow these steps. When using the embedded ZooKeeper server, we may choose to secure the server by using Kerberos. All nodes configured to launch an embedded ZooKeeper and using Kerberos should follow these steps.

In order to use Kerberos, we first need to generate a Kerberos Principal for our ZooKeeper servers. The following command is run on the server where the krb5kdc service is running. This is accomplished via the kadmin tool:

```
        kadmin: addprinc "zookeeper/myHost.example.com@EXAMPLE.COM"
```

Here, we are creating a Principal with the primary zookeeper/myHost.example.com, using the realm EXAMPLE.COM. We need to use a Principal whose name is <service name>/<instance name>. In this case, the service is zookeeper and the instance name is myHost.example.com (the fully qualified name of our host).

Next, we will need to create a KeyTab for this Principal, this command is run on the server with the NiFi instance with an embedded zookeeper server:

```
        kadmin: xst -k zookeeper-server.keytab zookeeper/
myHost.example.com@EXAMPLE.COM
```

This will create a file in the current directory named zookeeper-server.keytab. We can now copy that file into the $NIFI_HOME/conf/ directory. We should ensure that only the user that will be running NiFi is allowed to read this file.

We will need to repeat the above steps for each of the instances of NiFi that will be running the embedded ZooKeeper server, being sure to replace myHost.example.com with myHost2.example.com, or whatever fully qualified hostname the ZooKeeper server will be run on.

Now that we have our KeyTab for each of the servers that will be running NiFi, we will need to configure NiFi's embedded ZooKeeper server to use this configuration. ZooKeeper uses the Java Authentication and Authorization Service (JAAS), so we need to create a JAAS-compatible file In the $NIFI_HOME/conf/ directory, create a file named zookeeper-jaas.conf (this file will already exist if the Client has already been configured to authenticate via Kerberos. That's okay, just add to the file). We will add to this file, the following snippet:

```
        Server {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="./conf/zookeeper-server.keytab"
  storeKey=true
  useTicketCache=false
  principal="zookeeper/myHost.example.com@EXAMPLE.COM";
};
```

Be sure to replace the value of principal above with the appropriate Principal, including the fully qualified domain name of the server.

Next, we need to tell NiFi to use this as our JAAS configuration. This is done by setting a JVM System Property, so we will edit the conf/bootstrap.conf file. If the Client has already been configured to use Kerberos, this is not necessary, as it was done above. Otherwise, we will add the following line to our bootstrap.conf file:

```
        java.arg.15=-Djava.security.auth.login.config=./conf/zookeeper-
jaas.conf
```

Note: This additional line in the file doesn't have to be number 15, it just has to be added to the bootstrap.conf file. Use whatever number is appropriate for your configuration.

We will want to initialize our Kerberos ticket by running the following command:

```
        kinit -kt zookeeper-server.keytab "zookeeper/
myHost.example.com@EXAMPLE.COM"
```

Again, be sure to replace the Principal with the appropriate value, including your realm and your fully qualified hostname.

Finally, we need to tell the Kerberos server to use the SASL Authentication Provider. To do this, we edit the $NIFI_HOME/conf/zookeeper.properties file and add the following lines:

```
  authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
kerberos.removeHostFromPrincipal=true
kerberos.removeRealmFromPrincipal=true
jaasLoginRenew=3600000
requireClientAuthScheme=sasl
```

The kerberos.removeHostFromPrincipal and the kerberos.removeRealmFromPrincipal properties are used to normalize the user principal name before comparing an identity to acls applied on a Znode. By default the full principal is used however setting the kerberos.removeHostFromPrincipal and the kerberos.removeRealmFromPrincipal properties to true will instruct Zookeeper to remove the host and the realm from the logged in user's identity for comparison. In cases where NiFi nodes (within the same cluster) use principals that have different host(s)/realm(s) values, these kerberos properties can be configured to ensure that the nodes' identity will be normalized and that the nodes will have appropriate access to shared Znodes in Zookeeper.

The last line is optional but specifies that clients MUST use Kerberos to communicate with our ZooKeeper instance.

Now, we can start NiFi, and the embedded ZooKeeper server will use Kerberos as the authentication mechanism.

Kerberizing NiFi's ZooKeeper Client

Note: The NiFi nodes running the embedded zookeeper server will also need to follow the below procedure since they will also be acting as a client at the same time.

The preferred mechanism for authenticating users with ZooKeeper is to use Kerberos. In order to use Kerberos to authenticate, we must configure a few system properties, so that the ZooKeeper client knows who the user is and where the KeyTab file is. All nodes configured to store cluster-wide state using ZooKeeperStateProvider and using Kerberos should follow these steps.

First, we must create the Principal that we will use when communicating with ZooKeeper. This is generally done via the kadmin tool:

```
```

```
kadmin: addprinc "nifi@EXAMPLE.COM"
```

A Kerberos Principal is made up of three parts: the primary, the instance, and the realm. Here, we are creating a Principal with the primary nifi, no instance, and the realm EXAMPLE.COM. The primary (nifi, in this case) is the identifier that will be used to identify the user when authenticating via Kerberos.

After we have created our Principal, we will need to create a KeyTab for the Principal:

```
kadmin: xst -k nifi.keytab nifi@EXAMPLE.COM
```

This keytab file can be copied to the other NiFi nodes with embedded zookeeper servers.

This will create a file in the current directory named nifi.keytab. We can now copy that file into the $NIFI_HOME/conf/ directory. We should ensure that only the user that will be running NiFi is allowed to read this file.

Next, we need to configure NiFi to use this KeyTab for authentication. Since ZooKeeper uses the Java Authentication and Authorization Service (JAAS), we need to create a JAAS-compatible file. In the $NIFI_HOME/conf/ directory, create a file named zookeeper-jaas.conf and add to it the following snippet:

```
Client {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="./conf/nifi.keytab"
  storeKey=true
  useTicketCache=false
  principal="nifi@EXAMPLE.COM";
};
```

We then need to tell NiFi to use this as our JAAS configuration. This is done by setting a JVM System Property, so we will edit the conf/bootstrap.conf file. We add the following line anywhere in this file in order to tell the NiFi JVM to use this configuration:

```
java.arg.15=-Djava.security.auth.login.config=./conf/zookeeper-jaas.conf
```

Finally we need to update nifi.properties to ensure that NiFi knows to apply SASL specific ACLs for the Znodes it will create in Zookeeper for cluster management. To enable this, in the $NIFI_HOME/conf/nifi.properties file and edit the following properties as shown below:

```
nifi.zookeeper.auth.type=sasl
nifi.zookeeper.kerberos.removeHostFromPrincipal=true
nifi.zookeeper.kerberos.removeRealmFromPrincipal=true
```

Note: The kerberos.removeHostFromPrincipal and kerberos.removeRealmFromPrincipal should be consistent with what is set in Zookeeper configuration.

We can initialize our Kerberos ticket by running the following command:

```
kinit -kt nifi.keytab nifi@EXAMPLE.COM
```

Now, when we start NiFi, it will use Kerberos to authentication as the nifi user when communicating with ZooKeeper.

Troubleshooting Kerberos Configuration

When using Kerberos, it is import to use fully-qualified domain names and not use localhost. Please ensure that the fully qualified hostname of each server is used in the following locations:

- conf/zookeeper.properties file should use FQDN for server.1, server.2, …, server.N values.
- The Connect String property of the ZooKeeperStateProvider
- The /etc/hosts file should also resolve the FQDN to an IP address that is not 127.0.0.1.

Failure to do so, may result in errors similar to the following:

```
        2016-01-08 16:08:57,888 ERROR [pool-26-thread-1-
SendThread(localhost:2181)] o.a.zookeeper.client.ZooKeeperSaslClient
 An error: (java.security.PrivilegedActionException:
 javax.security.sasl.SaslException: GSS initiate failed [Caused by
 GSSException: No valid credentials provided (Mechanism level: Server
 not found in Kerberos database (7) - LOOKING_UP_SERVER)]) occurred when
 evaluating Zookeeper Quorum Member's  received SASL token. Zookeeper Client
 will go to AUTH_FAILED state.
```

If there are problems communicating or authenticating with Kerberos, this http://docs.oracle.com/javase/7/docs/technotes/guides/security/jgss/tutorials/Troubleshooting.html may be of value.

One of the most important notes in the above Troubleshooting guide is the mechanism for turning on Debug output for Kerberos. This is done by setting the sun.security.krb5.debug environment variable. In NiFi, this is accomplished by adding the following line to the $NIFI_HOME/conf/bootstrap.conf file:

```
        java.arg.16=-Dsun.security.krb5.debug=true
```

This will cause the debug output to be written to the NiFi Bootstrap log file. By default, this is located at $NIFI_HOME/logs/nifi-bootstrap.log. This output can be rather verbose but provides extremely valuable information for troubleshooting Kerberos failures.