Install Guide

System Requirements

1. You can install Spin on most common operating systems including Microsoft Windows, and UNIX-like operating systems, such as Linux, Solaris, and Mac OSX.
2. Java Runtime Environment, Standard Edition (JRE) (A Java Virtual Machine (JVM) 1.6 or greater compatible virtual machine must be present)
3. Java Servlet Container. Spin has been tested with Apache Tomcat, versions 5.0 and 6.0, but any servlet container should work. Deployment instructions are written for Tomcat.

Note: This document uses Unix-like conventions for paths and environment variables that you can map to Windows equivalents.

Obtaining Spin

You can obtain the Spin WAR from the open.med Maven repository:

http://repo.open.med.harvard.edu/nexus/content/groups/public/org/spin/node-server/<version>/node-server-<version>.war

The location and filename corresponds to the latest released Spin version. For example, the latest released SPIN version is 1.16. The SPIN WAR would be located at the following URL with the following name:

http://repo.open.med.harvard.edu/nexus/content/groups/public/org/spin/node-server/1.16/node-server-1.16.war

Installation

Layout

SPIN is installed in two distinct places on the host operating system:

- Spin web application (webapp) is installed within the web servlet container
- Spin home directory is a place on the file system that contains folders for storing configuration files, log files and plugins.

Process Overview

Installing the SPIN software, configuring and deploying the network node consists of the following processes performed in this order:

1. Establish the SPIN home directory
2. Deploy the SPIN web application
3. Create and configure node.xml and routingtable.xml
4. Restart the web servlet container
5. Test the installation.

Step 1: Establish the SPIN Home Directory

SPIN requires a dedicated "home directory", $SPIN_HOME, which is the root of a hierarchy of folders that contain configuration files, query log files, and plugins. It doesn't matter where this folder is located, but it is recommended that it is configured outside the webapp structure.

The default location is the user's home directory ~/.spin, where ~ is the user. To override the default, set the Java system variable SPIN_HOME or the environment variable SPIN_HOME to the absolute path of the home directory.

The SPIN home directory consists of the following folders:

conf/---configuration files

db/---query log files

plugins/---JARs containing plugins (queries to be run by the Node)

On Microsoft Windows systems, the default directory is ~/spin. The process owner must have write permission for the directory.

Step 2: Deploy the Spin web application

Determine the Java servlet container's home directory, which is typically dictated by the host operating system. If you are using Tomcat, the directory is, $TOMCAT_HOME/webapps. $TOMCAT_HOME is the directory where Tomcat was installed. Note: The file must be copied into the correct directory or Tomcat won't find it.

1. Deploy the .war file. Copy <version>/node-server-<version>.war to the $TOMCAT_HOME/webapps directory.
2. Start the servlet container. This creates a keystore with a generated keypair and keystore.xml in the $SPIN_HOME/conf directory.

Two additional configuration files, node.xml and routingtable.xml are not created, but the node will start without them.

Step 3: Create node.xml and routingtable.xml

These two configuration files contain miscellaneous node-wide configuration information, values for threshholds and time outs, and information abou the node's place in the network topology. These files are generated automatically when you run a SPIN plug-in such as eagle-i or you can create them based on the examples provided.

1. Create node.xml and routingtable.xml. If you are deploying SPIN with a plug-in files are generated automatically and saved in the current directory. When the files are generated, you need to supply the following parameters based on the node's place in the network.

Institutional node (most common)

```
java -jar eagle-i-network-queryplugin-<version>.jar <url of parent node>
```

The parent node is the URL of the node that forms the entry point to the network that this new node is joining.

Central node

```
java -jar eagle-i-network-queryplugin-<version>.jar <url of child node 1> <url of child node 2> ... <url of
child node N>
```

2. Copy both files to the $SPIN_HOME/conf directory.

Examples of this files are located:

https://open.med.harvard.edu/spin/trunk/examples/config-files/src/test/resources/node.xml

https://open.med.harvard.edu/spin/trunk/examples/config-files/src/test/resources/routingtable.xml

Step 4: Restart the web servlet container

Step 5: Test the installation

You have two options for testing the installation, by viewing a WSDL as a high-level check, or using a node status utility. Both options are explained below:

WSDL

Nodes communicate using Simple Object Access Protocol (SOAP), a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It exposes a Web Service Definition Language (WSDL), an XML-based language that provides a model for describing Web services.

You can find the WSDL at: http:////node?wsdl (http://%3chostname_and_port%3e/%3cnode-webapp-context%3e/node?wsdl), for example, http://localhost:8080/node-server-1.15.2/node?wsdl

View the WSDL in a browser as a very high-level check.

Smoke tests using a node status utility

A node status utility can provide coarse-grained information about the health of the node.

To perform the smoke test

1. Obtain the node status utility at http://repo.open.med.harvard.edu/nexus/content/groups/public/org/spin/node-status/<version>/node-status-<version>-shaded.jar.
2. Run the node status utility with this command, $ java -jar node-status-<version>-shaded.jar <node-wsdl-url>. For example,

```
$ java -jar node-status-1.15.2-shaded.jar http://localhost:8080/node-server-1.15.2/node?wsdl
```

Example output for a healthy node:

> http://localhost:8080/node-server-1.15.2/node?wsdl
>
> [ONLINE] [All Queries OK]

Number of entries in result store: 2

Oldest queryID: 'e4fd2c71-2826-4f45-b895-66ea0674fa4d' - Age: 2636667ms

Any exceptions, or a non-zero return code, indicate a non-healthy node. Nodes, if possible, start in a state that can be examined by the node status utility.

Configuration

Node.xml configuration properties

Node.xml contains miscellaneous node-wide configuration information, such as the node's role in the network. The configuration identifies if the node performs the following tasks:

- Authenticates users
- Broadcasts queries
- Aggregates responses
- Performs queries
- Routes incoming and outgoing messages

The order of the elements in the configuration file is important. Properties are required or optional.

Required Node Configuration Properties	**Description**	**Default**
<version>	Describes the human-readable version of SPIN on which this node is running. (This field is deprecated and will go away in future SPIN versions.)	
<nodeName>	Describes the human-readable node nam	
Optional elements	**If these elements are omitted, the defaults apply.**	**Default:**
<isAuthenticator>	Can this node authenticate users?	False
<isBroadcaster>	Should this node propagate the queries it receives to other nodes.	True
<isAggregator>	Identifies if this node should aggregate responses from other nodes.	False, if the node is not the root of a heierarchy, otherwise true.
<isQueryable>	Identifies if this node should respond to queries.	True. Set to false if the node should only broadcast queries and aggregate results, for example.
<identityServiceClass>	Identifies the fully-qualified name of the JVM class used to authenticate users based on credentials. Only required if <isAuthenticator> is true. The supplied class, if present, must implement org.spin.query.message.identity.IdentityService.	Null
<queries>	dentifies the queries, other than the basic queries, to which nodes respond. Each maps a queryType string to the fully-qualified name of a JVM class that implements that query. For each <queries> element, the <queryType> and <className> sub-elements are required. May be present zero or more times. Note that all nodes respond to the following queries, at a minimum: Spin.Echo Spin.Audit Spin.Discovery	no additional queries
<certificationTTL>	When clients query a SPIN network, the queries must be digitally signed. Nodes examine queries when they are received and respond only if the digital signature is valid and not expired. The <certificationTTL> element describes, in milliseconds, how old a signature can be before being considered invalid due to expiration.	3600000, or one hour)
<cacheTTL>	In some circumstances, SPIN caches the results of queries to allow clients to retrieve them at a later time. However, results that are unclaimed after some period are purged automatically. Specified, in milliseconds, by the <cacheTTL> element.	3600000, or one hour

<resultStoreType>	The type of underlying result store used to cache query results. Allowed values are 'SimpleInMemory' and 'Ehcache'. The SimpleInMemory result store is generally faster, but fails un-gracefully when memory runs out. The EHCache result store is generally slower, but fails more gracefully.	SimpleInMemory
<queryActionMapClassName>	Defines the fully-qualified name of a JVM class that implements org.spin.node.QueryActionMap and provides QueryAction instances by name. Useful for creating SPIN using Spring, Guice, or another dependency-injection framework, or for instantiating QueryActions lazily, instead of eagerly at node-startup time, which is the default behavior.	Null. Eagerly instantate QueryActions defined by <queries> elements)
<broadcastTimeoutPeriod>	SPIN nodes attempt to broadcast queries they receive to other nodes defined as being *children of*, or *downstream from* the current node. A timeout period for these broadcast attempts is defined by the **<broadcastTimeoutPeriod>** element. The **<duration>** sub-element defines the amount of the timeout. The **<unit>** sub element defines the unit for the value of the <duration> field. enum constants in java.util.concurrent.TimeUnit define values for this field. Note the following examples:	

```
<broadcastTimeoutPeriod>

<duration>5</duration>

<unit>SECONDS</unit>

</broadcastTimeoutPeriod>
```

```
<broadcastTimeoutPeriod>

<duration>1</duration>

<unit>HOURS</unit>

</broadcastTimeoutPeriod>
```

```
<broadcastTimeoutPeriod>

<duration>500</duration>

<unit>MILLISECONDS</unit>

</broadcastTimeoutPeriod> | 5 seconds |
```

routingtable.xml configuration properties

A SPIN node's place in a network topology is defined in routingtable.xml. SPIN nodes may participate in different overlay networks, fulfilling different roles in a topology, by joining different peer groups. A peer group defines a group of nodes and how they fit into a network topology. In routingtable.xml, nodes specify their place in different network topologies for zero or more peer groups.

Nodes always belong to the peer group, LOCAL, which includes only the current node. This is useful for querying only one node, without triggering query propagation.

When clients query a SPIN network, they send, along with their query, the peer group for which that the query is intended. Nodes use that peer group to look up how to propagate the query and where to return the results.

For a given peer group, nodes may have zero or one parents, and zero or more children. If a parent node is specified, query results from the current node, and those downstream, will be sent to the parent for aggregation. If no parent is specified, results are stored at the current node until the client retrieves them. It is also possible for a client to specify an arbitrary node as the point at which results will be aggregated and stored for retrieval.

If any child nodes are specified, the current node broadcasts all queries that it receives to the child nodes. Note that node, node B, can be listed as the child of node A, but node B does not have to list node A as its parent.

routingtable.xml contains zero or more <peerGroup> elements. The structure of these elements is as follows:

Property	Description	Optional /Required
peerGroup	LOCAL peer group named with no parent or children added implicitly	optional
<groupName>	Unique name of the peer group	required
<parent>	The current node's parent (default null - no parent)	optional
<children>	Zero or more child nodes that to which the current node should broadcast queries, if it receives a query destined for this peer group. The structure of <parent> and <children> elements are identical.	
<endpointType>	Values: Only supports SOAP. (Local used only for testing.)	required
<address>	Identifies the URL of the referenced node's WSDL, such as http://my-node.med.harvard.edu:8080/node-server/node?wsdl	require

keystore.xml configuration properties

A keystore is a repository of security certificates, either Certification Authority Certificates or Public key certificates. Keystore.xml contains information about the keystore file that contains the node's cryptographic keypair. The keypair includes the keystore file password and the keystore private key alias that you should use, if more than one private key exists.

The serial number of the node private key identifies the SPIN node.

The serial number of their private key supports the PKCS12 and JKS keystore formats.

Nodes are definitively identified on a network by the serial number of their private key. This file contains information about the keystore file, which contains the Node's cryptographic keypair, including the password of the keystore file and the alias of the private key in the keystore to use---if there is more than one.

The keystore file referenced by keystore.xml will also contain the certificates of authenticating and querying agents that the Node trusts.

PKCS12 and JKS keystore formats are supported

You can locate keystore.xml at http://spin.org/xml/res/keystore. Note that the order of elements in this file is important.

Required Elements	Description	Default Value
<File>	The file containing the keystore---<file>/opt/spin/main/conf/spin.keystore</file>Path, either absolute or relative, to the keystore file containing cryptographic certificates.	
<Password>	Password for the file referenced by the <file> element.<password>spinkeystore</password>	
Optional elements	The alias of the node's public/private keypair is optional. If no keyAlias is specified but a single public/private keypair exists, the alias of the keypair is used. If no keyAlias is specified but multiple pub/priv keypairs exist, PKITool's constructor throws an exception.	
<keyAlias>	Identifies the alias of the certificate containing this node's private key. If more than one public/private keypair is contained in the referenced keystore, this element can be used to choose one. If only one public/private keypair exists in the referenced keystore, Spin choses that keypair, and this element is optional.	Null. Node tries to detect the public /private keypair.
<caPublicKeyAlias>	May be present 0 or more times. The aliases of the public keys of certificate authorities that this node trusts. If a node trusts a particular certificate authority by importing the CA's public cert into the node's keystore and referencing it with the <caPublicKeyAlias> element, the node will implicitly trust all queriers for which the CA vouches.	Null. No CAs are trusted.
<attachCertificateToSignature>	Relevant only if a <caPublicKeyAlias> element (or elements) is present. If set to true, then the public cert of the CA that vouched for a querier is attached to outbound queries.	False
<setSystemProperties>	Work around some failures with establishing SSL connections by modifying JVM-wide settings. Set to true as a last resort when troubleshooting SSL/HTTPS communication. This element is deprecated and will likely go away in a future release of SPIN.	False
<keystoreType>	The format of the referenced keystore file; See org.spin.tools.config.KeyStoreType for allowed values (currently 'JKS' and 'PKCS12') Example keystore.xml: https://open.med.harvard.edu/spin/trunk/examples/src/test/resources/keystore.xml	JKS (Default) or PKCS12.