SPLUNK for JMX

by Damien Dallimore
damien@dtdsoftware.com

Version 1.2.6

Collaborate at GitHub
https://github.com/damiendallimore/SPLUNK4JMX

Contents

Overview

SPLUNK for JMX

This SPLUNK application provides the means to :

1) connect to any local or remote JVM's JMX server, either via the remote JMX interface, attaching directly to a local JVM process, or using an MX4J HTTP based connector.
2) query any MBean registered on that JMX server
3) extract any MBean attributes (simple, composite or tabular)
4) invoke MBean operations and extract the result
5) write attributes/operation results out in a default “key=value” format, or alternatively your own custom format, for SPLUNK indexing and searching
6) declare clusters of JMX Servers for large scale JVM deployments

This application is basically a custom scripted input comprising of a bash/bat script and a Java program. I have bundled it up as an application for convenience and included some demo dashboards that graph up a selection of MBean attributes from the *java.lang* domain.

JMX

Browse here for an overview of JMX :
http://en.wikipedia.org/wiki/Java_Management_Extensions

Supported platforms

*nix and Windows

Currently supported JVMs

SPLUNK for JMX has been tested against these JVM variants :

1. Hotspot
2. JRockit
3. IBM J9

Currently supported Connectors

SPLUNK for JMX has support for :

1. **rmi** (JSR160 Standard Implementation and MX4J's JSR160 Implementation)
2. **iiop** (JSR160 Standard Implementation and MX4J's JSR160 Implementation)
3. **local** (MX4J only)
4. **soap** (MX4J only , http & https)
5. **hessian** (MX4J only, http & https)
6. **burlap** (MX4J only, http & https)

Installation

Prerequisites

1) SPLUNK version 4+ installed
2) JRE 1.6+ installed

Environment Variables

1) set the JAVA_HOME environment variable to the location of your JRE install
2) set the SPLUNK_HOME environment variable to the install root of SPLUNK

On Windows you can set these variables at *System Properties → Advanced → Environment Variables*
On Linux you can set and export these variables in the *~/.bashrc* file for the user account that you are running SPLUNK under

Download

1) Download application from http://splunk-base.splunk.com/apps/25505/download/
2) Uncompress the tarball to SPLUNK_HOME/etc/apps
3) Restart SPLUNK

Installed Components

The main components you need to know about are in the SPLUNK_HOME/etc/apps/SPLUNK4JMX/bin directory

bin/poll_jmx

This is the script that SPLUNK fires at scheduled intervals, this script invokes a Java program that has the logic for querying the JMX Server(s).The standard JRE JMX implementation will be used.

bin/poll_mx4j_jmx

Same as the above script, however the MX4J JMX implementation will be used. You must use this script should you want to use the MX4J specific JMX connectors.

bin/config/config.xml

This is the config file where you specify JMX server(s), MBeans and MBean attributes & operations.

You can create as many config files,with any name, as you want and set them up as the argument to the *poll_jmx* script on the SPLUNK Manager Data Inputs page.

Included in the application release are several example xml files showing various configuration scenarios that you can use as examples to craft your own configuration xml files.

XML config files must always live in the SPLUNK_HOME/etc/apps/SPLUNK4JMX/bin/config directory.

Additional documentation is installed in the SPLUNK_HOME/etc/apps/SPLUNK4JMX/docs directory.

The SPLUNK4JMX app sets up a custom index(index=jms) and a custom script input (source=jmx).

By default, the input script is disabled and the schedule time is 60 seconds

After you configure your config XML file, you should then enable the input script for the platform you are running on.

There is a "sh" script for *nix and a "bat" script for Windows.

Configuration XML

Overview

The configuration xml file is where you declare JMX Servers, MBeans ,MBean attributes to extract and MBean operations to invoke.

The file can be named whatever you like and must reside in the SPLUNK_HOME/etc/apps/SPLUNK4JMX/bin/config directory

Included in the SPLUNK4JMX app distribution is a comprehensive sample *config.xml*
This sample file queries various java.lang MBeans and obtains attributes

You can use this as a basis for creating your own config files for whatever MBeans and attributes you need to poll across your environment

To try out this sample xml file , just set the values for your environment in the ***jmxserver*** element , and you should be good to go.

There are also sample config files that show other ways of setting up connectivity to your JMX Servers using local process ID attachment and clustering for large scale JVM deployments.

Examples of invoking operations can be found in *config_operations_example.xml*

XSD

The configuration xml file is strictly validated against an XSD.

XSD documentation is at *docs/configuration_schema/config.pdf*

Additional Schema Notes

1 or more ***jmxserver*** elements can be included in this XML defintion ie: if you need to access multiple different JMX sources within the same scheduled execution
Each ***jmxserver*** element you declare will be run in parallel.

jmxuser and ***jmxpass*** are optional , if not specified, they will be ignored

host can be a hostname, dns alias or ip address.

jmxport is the JMX server port to connect to.

protocol(default is rmi***)*** is the jmx service protocol to use : rmi, iiop and mx4j specific protocols soap, hessian etc..

For more advanced finer grained control over the format of the jmx service URL you can also specify the ***stubSource(***default is jndi***), encodedStub*** and url ***lookupPath(***default is jmxrmi***)***. Refer to the schema docs for more details.

Or, to facilitate complete user flexibility, you can enter the full jmx service url as a raw string using the ***jmxServiceURL*** attribute.

This is raw jmx service url in standard format "service:jmx:protocol:sap"
service:jmx:rmi:///jndi/rmi://myhost:9909/jmxrmi

If this attribute is set, it will override any other set connection related attributes(***host, jmxport, protocol, stubSource, encodedStub*** and ***lookupPath***)

At index time the ***host*** field is extracted and transformed into the SPLUNK internal host value.

jvmDescription is just meta data, useful for searching on in SPLUNK where you might have multiple JVMs on the same host

By default, no timestamp is added , instead relying on the SPLUNK index time as the event time.

For MBean definitions , standard JMX object name wildcard patterns ***** and **?** are supported for the ***domain*** and ***properties*** attributes
http://download.oracle.com/javase/1,5.0/docs/api/javax/management/ObjectName.html

If no values are specified for the ***domain*** and ***properties*** attributes , the value will default to the ***** wildcard.
The MBean's canonical objectname will be written out to SPLUNK
Simple, Composite and Tabular MBean data structures are supported.

The value obtained from the Attribute or as the return type of an Operation can be :

1. Any primitive type(Object wrapped)
2. String
3. Array
4. List
5. Map
6. Set

Collection types will be recursively deeply inspected, so you can have Collections of Collections etc..

For attributes that are multi level ie: composite and tabular attributes , then you can use a "**:**" delimited notation for specifying the attribute name.

MBean operation invocation is supported.
You can declare operations that return a result or no result & take arguments or no arguments.
Operation overloading is supported for methods of the same name with different signatures.

The following parameter types are supported :

1. int
2. float
3. double
4. long
5. short
6. byte
7. boolean
8. char
9. string

Internally these get autoboxed into their respective Object counterparts.

Creating additional configuration files

You can create as many configuration xml files as you need, with any name.
In SPLUNK Manager → Data Inputs , you can specify the xml file that you want to be executed as an argument to the *poll_jmx* script.
Therefore you can setup as many xml files to run at whatever scheduled you require simply by :

1) cloning the *poll_jmx* script input
2) set the scheduled execution frequency
3) specifying the name of the xml file to run as an argument to the *poll_jmx* script

Discovering MBeans,Attributes & Operations

I recommend using either JConsole or JRockit Mission Control to connect to a JMX Server, browse what platform and custom application MBeans are available and the associated attributes and operations. This will not only test that JMX connectivity works correctly, but you can also then obtain the MBean domain/properties values , attribute names and operation signatures for using in the SPLUNK4JMX configuration xml file.

Connecting to a local or remote JVM via remote JMX interface

You can connect to any local or remote JVMs by enabling the remote JMX Interface(rmi and iiop) and then declaring the ***host, jmxport, jmxuser,jmxpass*** in the ***jmxserver*** element of the configuration xml file. There are also several other connection attributes that can be set for much finer grained control over the format of the JMX Service URL. Refer to the schema documentation.

Enabling Remote JMX Connector on your JVM

You enable the remote JMX connector by providing JVM system properties to the java executable.

To set up a JMX server for remote access via JVM system properties at startup, see this :
http://download.oracle.com/javase/1.5.0/docs/guide/management/agent.html#remote

If that is to much reading, then at the very simplest level, just add these 3 JVM system properties :

-Dcom.sun.management.jmxremote.port=9001
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false

This will enable unauthenticated/unencrypted remote JMX connectivity on port 9001. You can use any port.

When setting this up on a local JVM,you can just specify “localhost” or “127.0.0.1” as the ***host*** value in the configuration xml file. Alternatively, you can attach directly using the JVM's process ID.

MX4J Support

If you wish to use MX4J as your JMX implementation for remote connectors(rmi and iiop) or use any of the MX4J specific JMX connectors(soap, burlap, hessian, local), then it is simply a matter of using the *poll_mx4j_jmx* script rather than the standard *poll_jmx* script.

For more details about MX4J browse here : http://mx4j.sourceforge.net

Connecting to a local JVM via Process ID

You can attach directly to a locally running JVM process by specifying the Process ID in the ***pid*** attribute.
In this case, the ***host, jmxport, jmxuser,jmxpass*** attributes will not be used.
The host value will default to the localhost's name.

For more dynamic flexibility,rather than specifying a static Process ID value , you can also specify the name of a PID file that contains the JVM's Process ID.
You set this in the ***pidFile*** attribute.
Many long running Java processes output the PID value to a file, particularly if using a JVM Service Wrapper such as Tanuki.
Each time SPLUNK4JMX runs, it will dynamically inspect the value of the PID from this file.
This way, you don't have to alter the configuration xml each time you do a JVM restart and the PID

changes.

You can also specify the path to a command to execute to output the JVM's PID.
You set this in the ***pidCommand*** attribute.
This command might be a custom script that looks for the JVM process and extracts the PID to STD OUT.
On Linux you could do this with a simple script that uses ps, grep and awk.

Example : ps -eafH | grep "java" | grep -v "grep" | awk '{print $2}'

See the example xml configuration file, *config_PID_example.xml*

When connecting directly to local JVM using a process ID, native librarys are used. You shouldn't need to do anything, they are loaded automatically.

Windows : JRE_HOME/bin/attach.dll
Linux : JDK_HOME/jre/lib/i386/libattach.so

NOTE : For some reason the Linux "attach" library is only packaged in the JRE that is part of a JDK install. Weird.

Creating a custom output Formatter

The default output format is a comma delimited list of key/value pairs :

host=damien-laptop,jvmDescription="My test JVM",mbean="java.lang:type=Threading",count="47"

Alternatively a custom "formatter" can be implemented and declared in the configuration xml.
Only 1 formatter may be specified for the entire configuration file

This custom formatter is a Java implementation of the
"*com.dtdsoftware.splunk.formatter.Formatter*" interface.

The customer formatter class should then be placed on the java classpath.
If a formatter declaration is omitted, then the default system formatter will be used.

Refer to the Javadoc API in the docs directory for creating a custom Formatter implementation.

Adding java classes to the application classpath

You can dump a jar file in the "SPLUNK_HOME/etc/apps/SPLUNK4JMX/bin/lib/ext" directory and it will be automatically loaded.
Why would you want to do this ? Well perhaps you have created a custom formatter implementation as described above.

Adding java classes to the boot classpath

Any classes that you need prepended to the java bootclasspath should be put in "SPLUNK_HOME/etc/apps/SPLUNK4JMX/bin/lib/boot".
They **won't** be automatically loaded, you will need to specify each jar in the poll_jmx/poll_mx4j_jmx

script.
Why would you want to do this ? Well perhaps you are targeting a JVM with some proprietary JMX logic that requires additional libraries on the JMX client side ie: using the IIOP connector with IBM Websphere products is one such scenario I've encountered.

Support for large scale JVM infrastructures

Large groups of JMX Servers that share that same MBeans can be grouped together in a cluster element, so that you only have to declare the MBeans/MBean attributes/operations once ie: a cluster of JEE appservers, a cluster of Hadoop or Cassandra nodes etc...This will be useful for enterprises with very large scale JVM deployments.

MBean inheritance is supported, so you can have MBeans defined that are common to the cluster, and MBeans that are specific to a cluster member.
See the example xml configuration file, *config_cluster_example.xml*

Troubleshooting & Logging

- check your firewall setup
- check the correct spelling/case of your MBeans and MBean attribute/operations definitions
- check your hostname and port
- check your user and pass
- check that your JVM's JMX remote connector has been correctly setup, can you connect using JConsole or JRockit Mission Control ?
- check that the SPLUNK_HOME and JAVA_HOME environment variables have been correctly setup and exported
- ensure you have enabled the poll_jmx.sh / poll_jmx.bat script input
- check your process ID if trying to attach to a local JVM
- ensure your XML configuration file adheres to the XSD
- look for errors in the SPLUNK4JMX log files and the SPLUNK Server log files

By default , "error" level log events will be written to SPLUNK_HOME/etc/apps/SPLUNK4JMX/logs
The log file will roll daily and get timestamped.
If you need to tweak log settings, then the *log4j.xml* config file resides in *bin/lib/jmxpoller.jar*