talend*
*open integration solutions

Talend ESB System Management Integration

User Guide

5.2.1

Publication date 12 November 2012
Copyright © 2011-2012 Talend Inc.

Copyleft

This documentation is provided under the terms of the Creative Commons Public License (CCPL). For more information about what you can and cannot do with this documentation in accordance with the CCPL, please read: http://creativecommons.org/licenses/by-nc-sa/2.0/

This document may include documentation produced at The Apache Software Foundation which is licensed under The Apache License 2.0.

Notices

Talend and Talend ESB are trademarks of Talend, Inc.

Apache CXF, CXF, Apache Karaf, Karaf, Apache Cellar, Cellar, Apache Camel, Camel, Apache Maven, Maven, Apache Archiva, Archiva are trademarks of The Apache Foundation. Eclipse Equinox is a trademark of the Eclipse Foundation, Inc. SoapUI is a trademark of SmartBear Software. Hyperic is a trademark of VMware, Inc. Nagios is a trademark of Nagios Enterprises, LLC.

All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

Table of Contents

Chapter 1. Introduction

This document looks at System Management Integration tools and their use within Talend ESB.

The smooth running of the computer infrastructure is a critical part of any business. This requires constant system monitoring of network resources, to be aware of what is happening and of any problems that may arise, for example, in services being unavailable, or generating faults.

In the event of emergencies, the system can be configured to notify key personnel about the problem and can help resolve it.

These tools (such as Hyperic and Nagios) are generally based on the model of agents gathering information locally and sending it to monitoring server. This can then raise alarms, and also facilitates intervention by administrators.

- Hyperic is a cross platform monitoring system which is designed to monitor and control server resources. We look at Hyperic HQ integration support in Chapter 2, *Hyperic HQ Integration*
- Using JMX, a given application is instrumented by one or more Java objects known as Managed Beans, or MBeans, and this technology facilitates the monitoring process. JMX configuration is described in Chapter 3, *JMX configuration and using JConsole*, and this includes the monitoring JConsole tool.
- Nagios is an Open Source monitoring application, and Chapter 4, *Nagios Integration* deals with Nagios integration support.

Chapter 2. Hyperic HQ Integration

These sections describe the Hyperic HQ integration support in Talend ESB. This consists of two plugins files, one for Camel support and one for CXF support, which ship with the release and corresponding examples.

This consists of a number of stages:

1. Configure Tomcat and Karaf for JMX configuration, in Section 2.2.1, *"Pre-requisites for deploying the plugins"* . This enables the gathering of the basic metrics information.
2. Deploy the Hyperic HQ plugins and use them to autodiscover resources, in Section 2.2.2, *"How to deploy Hyperic plugins"*. This also shows how to display the metrics, and this output is updated as the examples run.
3. Build and run the examples to generate metrics, in README files for examples
4. Look at the updated metrics
5. To see the complete list of available CXF and Camel metrics, we also look at using JConsole - (which is additional information to that of the Hyperic HQ plugins), in Section 3.2, *"Viewing Camel route and CXF service metrics using jconsole"*.

2.1. Introduction to Hyperic HQ

2.1.1. Hyperic HQ overview

The smooth running of the computer infrastructure is a critical part of any business. This requires constant system monitoring of network resources, to be aware of what is happening and of any problems that may arise, for example, in services being unavailable, or generating faults. In the event of emergencies, the system can be configured to notify key personnel about the problem and can help resolve it.

Hyperic is a cross platform monitoring system which is designed to monitor and control server resources. The system implements four general functions:

- **Discovery**: HQ Agents that run on the machines in your environment automatically detect, or *auto-discover*, the software resources running on the machine. When HQ discovers a software resource, it collects key facts about it, including its type, vendor, version, and location
- **Monitoring**: HQ agents track the current state of services and servers in real time, and automatic detect abnormalities
- **Alerts**: this subsystem will notify you about problems at the resources that are monitored. Alerts can be sent to administrator using e-mail, a mobile phone or pager
- **Control**: You can use HQ for remote control and administration of your software resources. Available control actions vary by resource type

You can find further information about Hyperic at the "Hyperic" site http://www.hyperic.com/. For more information about installing Hyperic, please see : http://support.hyperic.com/display/DOC/QuickStart+Installation.

2.1.2. System Requirements

Camel and CXF management demo examples can be installed, built, and run on Windows or Linux.

There are a number of software and hardware prerequisites you should be aware of, prior to starting the installation of Talend ESB software.

For a complete list of compatible software and software versions:

- If you are using Talend ESB Studio, Talend Data Services Studio or Talend ESB, see the corresponding *Talend Installation Guide*.
- If you are using Talend Open Studio for ESB or Talend ESB Standard Edition, see the link to the Installation procedure on the Talend ESB download page (http://www.talend.com/download/esb).
- Some of the ESB components use Apache software components (for example, Apache CXF, Apache Camel). For details on the exact software versions involved, see the section on Apache software in the product release notes.

In addition, the following software is required for installing, building, and running the samples:

- Maven 3.0.3 or later from Apache should be installed, and the **mvn** executable should be in your `PATH`. When running **mvn**, HTTP access to the internet is required. The local Maven repository is expected to be created in its default location, i.e. the Maven configuration should not have been modified.
- The Talend OSGi container is included in the release installation in sub-directory `container`.
- Hyperic HQ 4.5.1 (CE or EE) Server and Agent from SpringSource should be installed. Later versions may work but have not been tested.

2.1.3. Release directory structure

First, you need to download and extract the Talend ESB software (see *Talend ESB Getting Started Guide* for more details). After you unpack the Talend ESB release, the plugin files are in a subdirectory `adapters`, which is structured as follows:

```
adapters/
    plugins/
       camel-plugin.xml
       cxf-plugin.xml
    LICENSE.txt
    README
```

2.2. Deploying the Hyperic HQ plugins

2.2.1. Pre-requisites for deploying the plugins

In order to provide metrics for monitoring, the Talend OSGi container or Tomcat web application container must be configured for remote JMX access.

The Karaf container of the Talend ESB installation is already configured.

For a Tomcat web application server, you must set an environment variable when starting it:

```
CATALINA_OPTS="-Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=6969 \
-Dcom.sun.management.jmxremote.ssl=false \
-Dcom.sun.management.jmxremote.authenticate=false"

export CATALINA_OPTS
```

This can be defined in a file:

```
set CATALINA_OPTS=...
```

On Linux, this is set at the beginning of `catalina.sh`

On Windows, it is set in `catalina.bat`.

To create MBeans that collect statistics about CXF services you first need to make an invocation on the CXF service, which registers the MBeans. Without that step Hyperic HQ won`t find metrics as MBeans don't exist.

2.2.2. How to deploy Hyperic plugins

Copy the Hyperic plugins to these folders:

- `<HypericServer>/hq-engine/hq-server/webapps/ROOT/WEB-INF/hq-plugins`
- `<HypericAgent>/bundles/agent-<version>/pdk/plugins`

The complete information about deploying Hyperic plugins is at http://support.hyperic.com/display/EVO/Deploy+Plugin

Tomcat and Karaf must be running to enable Hyperic to discover the servers.

1. Start the Hyperic server, if it wasn't running.
2. Start the Hyperic agent, if it wasn't running. Otherwise, restart it.

Information how to start/stop/restart Hyperic agent and Hyperic server is in the Hyperic documentation http://support.hyperic.com

Open your browser and login to Hyperic server (default http://localhost:7080/). You will see new discovered servers in the **Autodiscovery**.

Sometimes servers are not detected correctly. In this case you need to restart Hyperic HQ Agent. From the `<HyericAgent>/bin` directory restart agent:

`hq-agent.bat restart` (on Windows)

`sh hq-agent.sh restart` (on Linux)

1. Click on **Add to Inventory**.
2. Choose **Resources** Tab
3. Find and choose **Apache Camel(CXF) [Tomcat]([Karaf]) 2.x** from the list of servers, depending on which combination you wish to monitor. For example, you may wish to monitor Camel/Tomcat and CXF/Tomcat.
4. Choose a service group.

   For Camel you should see the group of the routes specified by context and name.

   For CXF you should see the group of the services specified by port and operation.
5. Hyperic will collect the metrics for these groups, and the initial results will be displayed.

These will subsequently updated as the examples are run.

Chapter 3. JMX configuration and using JConsole

JMX configuration facilitates monitoring and management of Java applications. This chapter looks at:

- overriding the standard defaults to perform more advanced configuration for Camel routes and CXF services.
- advanced configuration of the CXF and Camel plugins themselves
- it also shows how to use the jconsole tool to view Camel route and CXF service metrics

3.1. Configuration

You can take the default JMX configuration, which facilitates monitoring and management of Java applications. This section discusses how you can override these defaults, and perform more advanced configuration.

3.1.1. How to make advanced JMX configuration for Camel routes and CXF services

3.1.1.1. JMX configuration

Plugin section, in `camel-plugin.xml` and `cxf-plugin.xml`:

For the Karaf server, by default, in the plugin, there is the following JMX configuration properties:

Property Name	Description	Default
jmx.url	the JMX Service URL	service:jmx:rmi://localhost:44444/jndi/rmi://localhost:1099/karaf-trun
jmx.username	Username authentication	tadmin
jmx.password	Password authentication	tadmin

For the Tomcat server, by default, in the plugin, there is the following JMX configuration properties:

Property Name	Description	Default
jmx.url	the JMX Service URL	service:jmx:rmi:///jndi/rmi://localhost:6969/jmxrmi
jmx.username	Username authentication	""
jmx.password	Password authentication	""

3.1.1.2. JMX configuration for Camel routes

Apache Camel has support for JMX and allows you to monitor a Camel managed object (for example, routes). By default, a JMX agent is enabled in Camel which means that the Camel runtime creates and registers MBean management objects with a MBeanServer instance in the VM. But if you would like to configure a JMX agent (for example to use a custom port in JMX URL) the best way to do it is adding a `jmxAgent` element inside the camelContext element in Spring configuration:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <jmxAgent id="agent" mbeanObjectDomainName="your.domain.name">
    ...
<camelContext>
```

The default JMX configuration is used in both examples, but you can also configure it:

```
<jmxAgent id="agent" registryPort="port number" createConnector="true">
```

`createConnector` means that we should create a JMX connector (to allow remote management) for the MBeanServer.*`registryPort`* is the port for JMX.

You can set *`hostName`* and *`domainName`* for `DefaultManagementNamingStrategy`. As a default, *`hostName`* is the computer name and *`domainName`* is `org.apache.camel`

```
<bean id="naming"
        class="org.apache.camel.management.DefaultManagementNamingStrategy">
    <property name="hostName" value="localhost"/>
    <property name="domainName" value="org.apache.camel">
</bean>
```

To configure specific definitions for the Camel route object use the properties:

Property Name	Description
org.apache.camel	domain name
routes	Camel routes type
context	Camel context name
name	route name

You can find further information about configuring Camel JMX agent at the "Camel" site http://camel.apache.org/camel-jmx.html.

3.1.1.3. JMX configuration for CXF services

Each server type defines several service types such as EJBs, Connection Pools, JMS Queues, and so on. The plugin defines additional service types to provide management of CXF via custom MBeans. The service element defines a service type, for example:

Plugin object section:

```
<service name="CXF all services monitoring">
    <property name="OBJECT_NAME"
  value='org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service=*,
  port=*,operation=*'/>
    <metrics include="cxf"/>
    <plugin type="autoinventory" />
</service>
```

In order to access custom MBeans, the plugin must define its JMX ObjectName to be used with various MBeanServer interface methods. Only one ObjectName is defined per service type using the property tag within the service tag.

To configure specific definitions for the CXF service object use properties:

Property Name	Description
org.apache.cxf:bus.id	id of specific CXF bus
service	service endpoint name
port	service port name
operation	service operation name

To enable JMX integration for CXF you need to declare the following bean in service Spring configuration:

```
<bean id="org.apache.cxf.management.InstrumentationManager"
  class="org.apache.cxf.management.jmx.InstrumentationManagerImpl">
  <property name="bus" ref="cxf" />
  <property name="usePlatformMBeanServer" value="true" />
  <property name="createMBServerConnectorFactory" value="false" />
  <property name="enabled" value="true" />
</bean>
```

To avoid any unnecessary runtime overhead, the performance counters measuring response time are disabled by default. To collect statistics for running sevices define the following bean:

```
<bean id="CounterRepository"
            class="org.apache.cxf.management.counters.CounterRepository">
        <property name="bus" ref="cxf" />
</bean>
```

For further information about configuring JMX in CXF you can find at Apache CXF.

3.1.2. Advanced configuration for CXF and Camel plugins

We use the term `<TalendRuntimePath>` for the directory where Talend Runtime is installed. This is typically the full path of either `Runtime_ESBSE` or `Talend-ESB-V5.2.x`, depending on the version of the software that is being used. Please substitute appropriately.

If you wish to make advanced configuration of the plugin, you need to modify the plugin files that are shipped with the release (all these configurations are optional):

1. Remove all the metrics you don't need to observe.

2. If you have several instances of Karaf or Tomcat, Hyperic will try to assign discovered Camel routes or CXF services to each Tomcat or Karaf instance. So you can set INSTALLPATH where the instance with Camel routes (CXF services) is actually situated. This will prevent assigning Camel routes (CXF services) to each Tomcat or Karaf running instance.

   For Tomcat, add to Apache Camel Tomcat 2.x server in the `camel-plugin.xml`

   ```
   <property name="TOMCAT_HOME" value="C:\tomcat"/>
   <property name="INSTALLPATH_MATCH" value="${TOMCAT_HOME}"/>
   ```

   Set INSTALLPATH to Tomcat instead of `C:\tomcat`

   For Tomcat, add to Apache CXF [Tomcat] 2.x server in the `cxf-plugin.xml`

   ```
   <property name="TOMCAT_HOME" value="C:\tomcat"/>
   <property name="INSTALLPATH_MATCH" value="${TOMCAT_HOME}"/>
   ```

   Set INSTALLPATH to Tomcat instead of `C:\tomcat`

   For Karaf, add to Apache Camel [Karaf] 2.x server in the `camel-plugin.xml`

   ```
   <property name="KARAF_HOME" value="<TalendRuntimePath>\container\bin"/>
   <property name="INSTALLPATH_MATCH" value="${KARAF_HOME}"/>
   ```

   Set INSTALLPATH to bin directory of [Karaf] container instead of `<TalendRuntimePath>\container\bin`

   For Karaf, add to Apache CXF [Karaf] 2.x server in the `cxf-plugin.xml`

   ```
   <property name="KARAF_HOME" value="<TalendRuntimePath>\container\bin"/>
   <property name="INSTALLPATH_MATCH" value="${KARAF_HOME}"/>
   ```

   Set INSTALLPATH to bin directory of Karaf container instead of `<TalendRuntimePath>\container\bin`

   Another variant, even easier, to solve this problem is to override `INSTALLPATH` property. As an example for Karaf and Tomcat:

   ```
   <property name="INSTALLPATH" value="Camel Karaf" />
   <property name="INSTALLPATH" value="Camel Tomcat" />
   <property name="INSTALLPATH" value="CXF Karaf" />
   <property name="INSTALLPATH" value="CXF Tomcat" />
   ```

3. Set `jmx.url` value, which is defined for Tomcat and Karaf. As a default, Tomcat uses: `service:jmx:rmi:///jndi/rmi://localhost:6969/jmxrmi/`

   and Karaf uses:

   ```
   service:jmx:rmi://localhost:44444/jndi/rmi://localhost:1099/karaf-trun
   ```

4. Change the property `AUTOINVENTORY_NAME` for the defined services in the plugin.

5. For Camel you can set `domain` option, (the default is `org.apache.camel`).

That's all you need. The plugin is ready to be used. Additional information about developing Hyperic JMX plugins can be found at http://support.hyperic.com/display/EVO/JMX+Plugin+Tutorial

3.2. Viewing Camel route and CXF service metrics using jconsole

3.2.1. Hyperic metrics

In the case of JMX plugins, Hyperic monitors attributes of MBeans. The main metric attribute is `name`, and the `name` or `alias` of the metric should be the same as MBean attribute name.

In addition to the Hyperic information, to see the complete list of available CXF and Camel metrics, let's see how we can find Camel routes MBean attribute names using **jconsole** tool

You can find more information about metrics at http://support.hyperic.com/display/EVO/Metric+Parameters

3.2.2. Running JConsole

1. Type **`jconsole`** from Command Line.

2. Set remote JMX URL by selecting remote process and click on **Connect**. This opens the Java Monitoring & Management console.

3. After connection, choose **MBeans** tab.

   For Camel you should see:

   The attributes provided by Camel routes include:

   - Exchanges Completed
   - Exchanges Failed
   - Exchanges Total
   - Last Processing Time
   - Max Processing Time
   - Mean Processing Time
   - Min Processing Time
   - Total Processing Time

   For CXF you should see:

   The attributes provided by CXF services include:

   - Availability
   - Number of Invocations
   - Total Handling Time

- Num Checked Application Faults
- Num Logical Runtime Faults
- Num Runtime Faults
- Num Unchecked Application Faults

Chapter 4. Nagios Integration

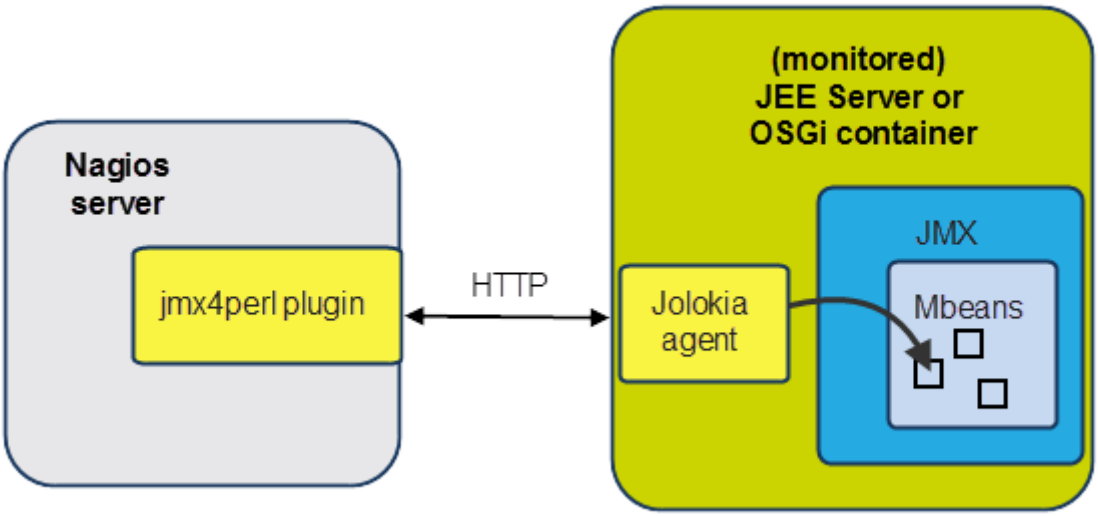
Nagios is an Open Source monitoring application which allows users to identify infrastructure problems before they impact on important business processes. Nagios monitors the entire IT infrastructure to ensure services, applications and business processes are working as expected.

In the case of critical problems in the infrastructure, Nagios can alert the IT department. That allows them to start fixing any issues as early as possible, before they affect the business processes.

In this chapter we describe how to monitor the Talend ESB infrastructure using Nagios.

4.1. Architecture overview of Nagios and Talend ESB

This is a diagram of how the Nagios functionality is integrated into Talend ESB - typically, the Nagios server is monitoring the OSGi container on the right, which may be a Talend Runtime container:



Jmx4Perl provides an alternative way of accessing JMX (Java Management Extensions) on JEE Servers or OSGi containers. It uses an agent-based approach where a small Java Web application (Jolokia), is deployed on the application server, and provides HTTP/JSON-based access to JMX MBeans registered within the application server.

In Talend ESB, for convenience, Jolokia has been integrated into the Talend Runtime container as an OSGi agent and also integrated into ActiveMQ using a JAR file.

This table is an overview of components in the jmx4perl distribution:

Component	Description
jmx4perl plugin	This installed on the same machine as Nagios; it communicates with a Jolokia agent which is integrated with the Talend ESB containers.
Jolokia	Jolokia is an HTTP/JSON bridge for efficent remote JMX access. It is a separate agent that resides on the monitored server, and works with the jmx4perl plugin. Jolokia is based on a set of Perl modules, and does not need a local Java installation. It is pre-installed into Talend ESB containers and ActiveMQ.
check_jmx4perl	A command utility on the Nagios server that can be used to get the monitored data - it is part of the jmx4perl distribution.
JMX::Jmx4Perl	a Perl library, for programmatic JMX access.

For more information, see:

- Nagios: http://www.nagios.com
- Jolokia: http://www.jolokia.org
- Jmx4perl: http://labs.consol.de/jmx4perl

4.2. Installing jmx4perl plugin to Nagios

In this section we describe how to download and install the jmx4perl plugin to a Nagios server.

4.2.1. Prerequisites

Nagios Open Source version or Nagios XI 2011 version should be installed into Linux platform (or VM). To download Nagios, go to http://www.nagios.org/download, and follow the installation instructions.

Note: Nagios Open Source 3.3.1 and Nagios XI 2011 version have been tested, but previous versions of Nagios may also work with Talend Runtime.

You also need an installation of the Talend Runtime, which may be on different host (for example, a Windows machine) to the Nagios installation.

4.2.2. Downloading the jmx4perl plugin

Download the jmx4perl plugin to the Nagios server machine, from http://search.cpan.org/CPAN/authors/id/R/RO/ROLAND/jmx4perl-1.05.tar.gz to, for example, `/tmp`.

4.2.3. Building the jmx4perl plugin

The main commands and scripts in this section (cpan, Build.PL, the Build perl script) need to be run with root permissions, for example, as root or using **sudo**.

We use the Perl CPAN (Comprehensive Perl Archive Network) shell here to download missing dependencies.

1. Extract the jmx4perl distribution:

```
$ cd /usr/local/src

# tar zxvf /tmp/jmx4perl-1.05.tar.gz

# ln -s -f jmx4perl-1.05 jmx4perl
```

2. Install the build module:

```
# cpan

cpan[1]> install Module::Build

cpan[1]> exit
```

3. This step installs missing dependencies for jmx4perl. There are two ways of doing this:

   - A Build perl script is created later in this section; if you have a Build script that was previously created (for example, during an similar installation on another machine), then it's quicker to use it here:

```
# cd /usr/local/src/jmx4perl
# ./Build installdeps
```

   - If you don't have a previously-created Build script, then explicitly install jmx4perl dependencies (and accept the default values to any questions):

```
# cpan

cpan[1]> install Config::General

cpan[2]> install Crypt::Blowfish_PP
```

```
cpan[3]> install File::SearchPath

cpan[4]> install JSON

cpan[5]> install Module::Find

cpan[6]> install Nagios::Plugin

cpan[7]> install Term::Clui

cpan[8]> install Term::ReadKey

cpan[9]> install Term::ReadLine::Perl

cpan[10]> install Term::ShellUI

cpan[11]> install Term::Size

cpan[12]> exit
```

4. Run the newly-created Build.PL script, which generates the ./Build perl script:

```
# cd /usr/local/src/jmx4perl

# perl Build.PL
```

Choose 'n' in response to "Install 'jolokia'" in the following script. Jolokia is not required directly on the Nagios server host, and only needs to be installed within the monitored containers (Talend containers have it pre-installed). Adding Jolokia would require installing additional modules and has not been tested.

When running this script give the following answers:

```
Install 'jmx4perl' ? (y/n) [y ]y
Install 'check_jmx4perl' ? (y/n) [y ]y
Install 'cacti_jmx4perl' ? (y/n) [y ]y
Install 'j4psh' ? (y/n) [y ]y
Install Term::ReadLine::Gnu ? (y/n) [n ]n
Install 'jolokia' ? (y/n) [y ]n * see note above
```

Run the ./Build script to recheck all dependencies are installed:

```
# ./Build installdeps
```

5. Run the Build comand:

```
# ./Build install
```

When all these steps finished successfully, the jmx4perl plugin should be installed onto Nagios.

- Check everything has been correctly installed by communicating with a Talend Runtime container (this already has an active Jolokia agent) (please replace `<jolokia_host>` with the host where the Talend Runtime container is running) :

```
$ check_jmx4perl -u http://<jolokia_host>:8040/jolokia --alias
   MEMORY_HEAP_USED --base MEMORY_HEAP_MAX --warning 80 --critical 90
```

If you have problems getting this running, try disabling the firewall on the Jolokia (Talend Runtime container) host.

4.3. The Jolokia agent and Talend software

Jolokia is a HTTP/JSON bridge for efficient remote JMX access, and is a separate agent which was created as part of the evolution of jmx4perl.

For more information about Jolokia, see http://www.jolokia.org/.

For convenience, Jolokia has been integrated into the Talend Runtime container as an OSGi agent and also integrated into ActiveMQ using a JAR file (see Section 4.1, *"Architecture overview of Nagios and Talend ESB"*).

4.3.1. Manually installing the Jolokia OSGi agent to a Talend Runtime container

The Jolokia OSGi agent has been installed by default in the Talend Runtime container. However, this section describes how to install the Jolokia OSGi agent manually, in case it was uninstalled from the container at some point.

To install a Jolokia agent to a Talend Runtime container, execute this command at the container console:

```
features:install tesb-jmx-http-agent
```

Then, Jolokia agent bundle will be installed to the container. Run the **`list`** command, and the output should look like this:

```
[ 191] [Active] [      ] [      ] [   60] Jolokia Agent (1.0.2)
```

In addition, if you access the URL http://localhost:8040/jolokia/version, you will see a JSON output line about version information, which indicates the Jolokia agent is running correctly.

4.3.2. The Jolokia agent and ActiveMQ

The Jolokia agent (JAR file) has been already integrated into the ActiveMQ distribution (included in Talend ESB). It's ready to use out of the box. For an example of monitoring an ActiveMQ broker, see Section 4.5.6, "*Monitoring an Apache ActiveMQ broker with Nagios*"

4.4. Using Nagios configuration templates

In this section we look at configuring Nagios to select the metrics you wish to monitor.

In Section 4.4.1, "*Syntax for adding metrics for monitoring*" we look in detail at the syntax in the configuration files.

In Section 4.4.2, "*Talend ESB Nagios configuration template files*" we look at the relevant configuration files that ship with Talend software.

Finally, in Section 4.4.3, "*Preparing the configuration files for running with Nagios*" we use these configuration templates to configure the Nagios software, and then run the Nagios server to monitor the specified functionality.

4.4.1. Syntax for adding metrics for monitoring

You can add the metrics for monitoring using the following steps and examples (these are already defined as much as possible for the shipped examples):

1. Edit the applicable configuration file or template in the Talend Runtime.
2. Define `Check` definition structures for the jmx4perl plugin, corresponding to the metrics that need to be monitored.
3. Define one or more commands for Nagios, that make use of the `check_jmx4perl` command from the plugin.
4. Describe a `host` and `service` definition for Nagios; the service definition needs to use the command defined in the previous step.

`jolokia_host` is the host where the Jolokia agent is installed, and is being monitored by the jmx4perl plugin.

You need to substitute this with a hostname or ip address for commands, or add it to `/etc/hosts` as described in Section 4.4.3, "*Preparing the configuration files for running with Nagios*" (recommended)

Here are some corresponding examples of using these steps in the shipped Talend software configuration files. In particular, we look at the structures needed to define a metric for monitoring Active MQ:

1. First we look at the configuration file `<TalendRuntimePath>/add-ons/adapters/nagios/template/activemq.cfg`
2. In particular, look at a `Check` definition for ActiveMQ:

```
# Define server connection parameters
<Server tesb_activemq>
   Url = http://jolokia_host:8161/jolokia
</Server>

# checks for ActiveMQ metrics
<Check Broker_TotalConsumerCount>
   MBean = org.apache.activemq:BrokerName=$0,Type=Broker
   Attribute =  TotalConsumerCount
   Name = TotalConsumerCount
   Warning 1000000
</Check>
```

   (See note about `jolokia_host`).

3. Here is an example of a command definition, which is in the file `<TalendRuntimePath>/add-ons/adapters/nagios/template/jmx_commands.cfg`:

```
# Define a command to monitor ActiveMQ using Nagios
# $USER5$ - user macros defining folder with check_jmx4perl
# $USER6$ - user macros defining folder with command configuration file
# $ARG1$ - check name which defined in activemq.cfg
# $ARG2$ - set broker name for activemq to be monitored
# $ARG3$ - set destination for queue to be monitored
# $ARG4$ - set destination for topic to be monitored
define command {
   command_name check_jmx4perl_activemq
   command_line $USER5$/check_jmx4perl \
                --config $USER6$/activemq.cfg \
                --server $HOSTNAME$ \
                --check  $ARG1$ $ARG2$ $ARG3$ $ARG4$
```

```
}
```

Note that the command definition specifies the configuration file `activemq.cfg` which contains all the check definitions defined earlier.

Several arguments are used in this command; setting their values is described in Section 4.4.3, "*Preparing the configuration files for running with Nagios*".

4. In the following configuration example (in the file `<TalendRuntimePath>/add-ons/adapters/nagios/sample/activemq_host.cfg`) you can see how to describe the host and service definition for Nagios:

```
# Define a host
define host{
        use     activemq-host     ; Name of host template to use.
                                  ; This host definition will inherit
                                  ; all the variables that are defined
                                  ; in (or inherited by) the linux-server
                                  ; host template definition.
        host_name        tesb_activemq
        alias            tesb_activemq
        }

define service {
    use                    generic-service
    service_description    Broker_TotalConsumerCount
    display_name           Broker_TotalConsumerCount:
    check_interval         1
    host_name              tesb_activemq
    check_command          check_jmx4perl_activemq!Broker_TotalConsumer
Count!localhost!example.A!ActiveMQ.Advisory.Consumer.Queue.example.A
    }
```

Note that you need to specify the values of the `check_command` properties in a strict order:

1. the name of command used to check the metric ("check_jmx4perl_activemq" from `jmx_commands.cfg`)
2. the name of check you use, from jmx4perl configuration ("Broker_TotalConsumerCount" from `activemq.cfg`)
3. the arguments for the command

4.4.2. Talend ESB Nagios configuration template files

Note that Talend configuration files for Nagios are only available in the Talend Open Studio for ESB and Talend ESB Studio download packages at the moment. They are in the `<TalendRuntimePath>/add-ons/adapters/nagios` directory.

There are four pre-defined configuration template files which ship with Talend ESB - these contain the Check definitions. These can be used for monitoring metrics of CXF, Camel and ActiveMQ resources.

File	Description
jmx_commands.cfg	defines three commands for Nagios monitoring, including **check_jmx4perl_cxf**, **check_jmx4perl_camel** and **check_jmx4perl_activemq** commands. Each command has several macros which need to be defined in the `etc/resources.cfg` of Nagios.

File	Description
cxf.cfg	check definition for cxf metrics to be monitored.
camel.cfg	check definition for camel metrics to be monitored.
activemq.cfg	check definition for activemq metrics to be monitored.

In addition, there are three sample xxx_host.cfg configuration files which provided most of the useful metrics monitoring for CXF, Camel and ActiveMQ - these contain the service and host definitions. You can define your own xxx_host.cfg for monitoring specific metrics and specific resources (CXF services, Camel routes, and so on):

File	Description
cxf_host.cfg	sample configuration of host and service definition for CXF monitoring.
camel_host.cfg	sample configuration of host and service definition for Camel monitoring.
activemq_host.cfg	sample configuration of host and service definition for ActiveMQ monitoring.

4.4.3. Preparing the configuration files for running with Nagios

In these examples, the Nagios installation directory may vary; typically it is in `/usr/local/nagios`, but it may not be this on all installations.

Similarily, the place to add configuration files Nagios directory is typically, but not always, `/usr/local/nagios/etc/objects`, which you may need to create if it hasn't been created by the installation.

You can use commands defined in `jmx_commands.cfg` file to monitor CXF services, Camel Context and Routes, ActiveMQ Broker, Topics and Queues.

In order to do it, you do not need to change template files `jmx_commands.cfg`, `cxf.cfg`, `camel.cfg`, `activemq.cfg` which already contain all check definitions and commands for these entities. For your own application, we suggest you add your own `new_host.cfg` to monitor your own cxf service, camel route, and so on, using `cxf_host.cfg`, `camel_host.cfg` and `activemq_host.cfg` as samples.

The process is as follows (see Section 4.5.2, *"Configure the Nagios plugin to monitor the sample applications"* for an example of following this process with the shipped examples):

1. Define `jolokia_host` in `/etc/hosts` - this name is used in subsequent files, rather than hard-coding in the ip address.

   For configuration templates `jolokia_host` means the host that has the Jolokia agent installed and would be monitored by the jmx4perl plugin. For example:

   ```
   192.168.1.101 jolokia_host
   ```

2. Put the configuration files into the configuration folder, for example, `/usr/local/nagios/etc/objects/` or `/etc/nagios3/etc/objects/`.

   In Talend ESB Studio or Talend ESB, the configuration files are in `<TalendRuntimePath>/add-ons/adapters/nagios`.

   Copy template and sample configuration files from this directory into, for example, `/usr/local/nagios/etc/objects/` or `/etc/nagios3/etc/objects/`, for example:

   ```
   # cp -f <TalendRuntimePath>/add-ons/adapters/nagios/template/*.cfg /usr/local/nagios/etc/objects/
   ```

```
# cp -f <TalendRuntimePath>/add-ons/adapters/nagios/sample/*.cfg /usr/
local/nagios/etc/objects/
```

3. Add the command configuration file to the existing `nagios.cfg`, here are some examples, which depend on where your installation puts config files:

   In `/usr/local/nagios/etc/nagios.cfg` add this line:
   `cfg_file=/usr/local/nagios/etc/objects/jmx_commands.cfg`

   Or in `/etc/nagios3/nagios.cfg` add this line:
   `cfg_file=/etc/nagios3/etc/objects/jmx_commands.cfg`

4. [Note: this step is not needed for samples, the shipped files are sufficient] Create host definitions file, for example `new_host.cfg`, by, for example, copying `jmx_host.cfg`. Note that you may need to edit it and add applications-specific service definitions.

5. [Note: this step is not needed for samples, the shipped files are sufficient]

   Add the file to the existing `nagios.cfg`, here are some examples, which depend on where your installation puts config files:

   In `/usr/local/nagios/etc/nagios.cfg` add this line:
   `cfg_file=/usr/local/nagios/etc/objects/new_host.cfg`

   Or in `/etc/nagios3/nagios.cfg` add this line:
   `cfg_file=/etc/nagios3/etc/objects/new_host.cfg`

6. Define macros which will be used by `jmx_commands.cfg` in the existing `resource.cfg`; here are some examples, which depend on where your installation puts config files:

   In `/usr/local/nagios/etc/resource.cfg` add these lines:

   ```
   # set the path which jmx4perl plugin installed
   $USER5$=/usr/local/src/jmx4perl/scripts
   # set the path to where to find configuration files
   $USER6$=/usr/local/nagios/etc/objects
   ```

   Or in `/etc/nagios3/resource.cfg` add these lines:

   ```
   # set the path which jmx4perl plugin installed
   $USER5$=/usr/local/src/jmx4perl/scripts
   # set the path to where to find configuration files
   $USER6$=/etc/nagios3/etc/objects
   ```

7. Then, restart Nagios for the changes to take effect.

   ```
   # service nagios restart
   ```

   Note: the name of this service may vary, depending on which package you used to install Nagios, so it may be called, for example, `nagios3`, instead of `nagios`.

4.5. Examples: monitoring camel-jmx, cxf-jmx and ActiveMQ samples

There are two sample applications that ship with the Talend Runtime: `camel-jmx` and `cxf-jmx`, in the `<TalendRuntimePath>/examples/talend/tesb` directory.

These can be built and installed on Windows or Linux. First we look at camel-jmx, and monitor the metrics in Nagios. Then we look at how to do the same with cxf-jmx.

Then we look at an example of monitoring an Apache ActiveMQ broker, using the shipped configuration files.

4.5.1. Build and install the camel-jmx sample applications

This sample consists of two parts:

1. service/ - This is the CXF service packaged as an OSGi bundle.
2. war/ - This module creates a WAR archive containing the service module. This is for Servlet container use only, not used in OSGi deployment.

1. From the example parent directory (`<TalendRuntimePath>/examples/talend/tesb`), run the following command to install the example parent `pom.xml` file into the local maven repo

   `mvn install --non-recursive`

2. From the base directory of this sample, the Maven pom.xml file can be used to build and run the demo

   `cd camel-jmx`

   `mvn clean install`

   Running this command will build the demo and create a WAR archive and an OSGi bundle for deploying the service either to servlet or OSGi containers

3. Start Talend Runtime container

   `trun.sh` (on Linux)

   `trun.bat` (on Windows)

4. Add camel-jmx example features URL. Type this command in Talend Runtime container:

   `features:addurl mvn:org.talend.esb.examples/camel-jmx-feature/5.2.1/xml`

5. To install the example feature, type this command in the Talend Runtime container:

   `features:install camel-jmx-service`

After deploying the samples you can see the Camel MBeans and their attributes which can be monitored using the JDK's JConsole (see Section 3.2.2, *"Running JConsole"*). These attributes are also included in the metrics that we will monitor with help of Nagios.

4.5.2. Configure the Nagios plugin to monitor the sample applications

Note: in Talend Open Studio for ESB or Talend ESB installations, the configuration files are in `<TalendRuntimePath>/add-ons/adapters/nagios`.

Note that there is an overview of this process in Section 4.4.3, "*Preparing the configuration files for running with Nagios*"

1. Create a local directory for the configuration files if it doesn't exist, for example: `/usr/local/nagios/etc/objects/` or `/etc/nagios3/etc/objects/`, depending on where your installation expects them.

2. Define jolokia_host (the host where the examples are running) in `/etc/hosts` - this name is used in subsequent files, rather than hard-coding in the ip address. For example, add the line (depending on the IP address):

   ```
   192.168.1.101 jolokia_host
   ```

3. Copy the template and sample configuration files into Nagios `etc/objects` subdirectory. Not all of these are needed for this example, but we will only reference the ones we need in the configuration files. For example:

   ```
   cp -f <TalendRuntimePath>/add-ons/adapters/nagios/template/*.cfg /usr/local/nagios/etc/objects/
   ```

   ```
   cp -f <TalendRuntimePath>/add-ons/adapters/nagios/sample/*.cfg /usr/local/nagios/etc/objects/
   ```

   This target directory may be elsewhere in your installation, for example: `/etc/nagios3/etc/objects/`.

4. Add the template configuration file to the existing `nagios.cfg`, here are some examples, which depend on where your installation puts config files:

   In `/usr/local/nagios/etc/nagios.cfg` add:

   ```
   cfg_file=/usr/local/nagios/etc/objects/jmx_commands.cfg
   cfg_file=/usr/local/nagios/etc/objects/camel_host.cfg
   ```

   Or in `/etc/nagios3/nagios.cfg` add:

   ```
   cfg_file=/etc/nagios3/etc/objects/jmx_commands.cfg
   cfg_file=/etc/nagios3/etc/objects/camel_host.cfg
   ```

5. Define macros which will be used by `jmx_commands.cfg` in `resource.cfg`, here are some examples, which depend on where your installation puts config files:

   In `/usr/local/nagios/etc/resource.cfg` add these lines:

   ```
   # set the path which jmx4perl plugin installed
   $USER5$=/usr/local/src/jmx4perl/scripts
   # set the path to where to find configuration files
   $USER6$=/usr/local/nagios/etc/objects
   ```

   Or in `/etc/nagios3/resource.cfg` add these lines:

   ```
   # set the path which jmx4perl plugin installed
   $USER5$=/usr/local/src/jmx4perl/scripts
   # set the path to where to find configuration files
   $USER6$=/etc/nagios3/etc/objects
   ```

6. Then, restart Nagios for the changes to take effect.

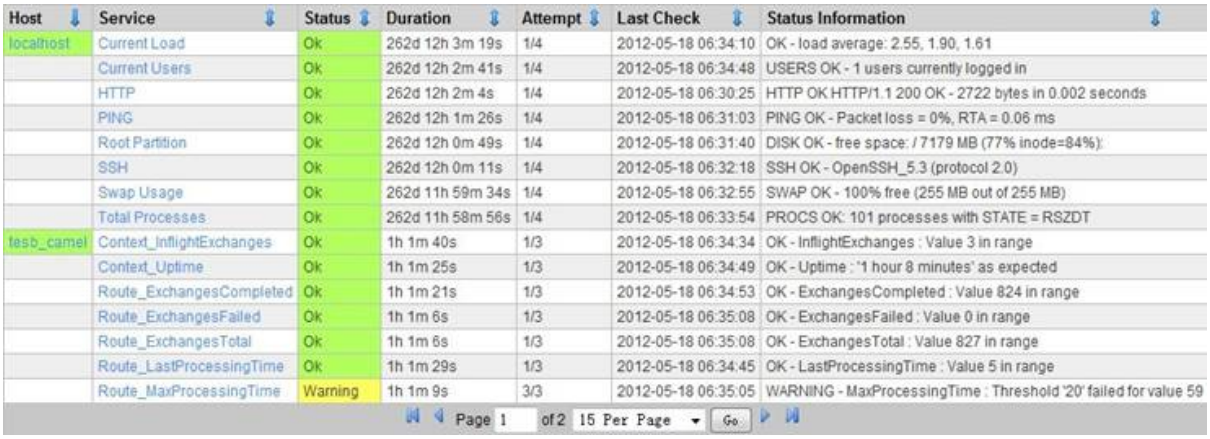
   ```
   # service nagios restart
   ```

   Note: the name of this service may vary, depending on which package you used to install Nagios, so it may be called, for example, `nagios3`, instead of `nagios`.

4.5.3. Monitoring with the Nagios web interface

Login to the Nagios Web Interface `http://<nagios_host>/<nagios_server>/` for example: `http://localhost/nagios/`, `http://192.168.1.10/nagios3/` or `http://192.168.198.5/nagios/`, and select **services** from the left hand menu.

There, you will find the status of metrics for camel-jmx example.



Host	Service	Status	Duration	Attempt	Last Check	Status Information
localhost	Current Load	Ok	262d 12h 3m 19s	1/4	2012-05-18 06:34:10	OK - load average: 2.55, 1.90, 1.61
	Current Users	Ok	262d 12h 2m 41s	1/4	2012-05-18 06:34:48	USERS OK - 1 users currently logged in
	HTTP	Ok	262d 12h 2m 4s	1/4	2012-05-18 06:30:25	HTTP OK HTTP/1.1 200 OK - 2722 bytes in 0.002 seconds
	PING	Ok	262d 12h 1m 26s	1/4	2012-05-18 06:31:03	PING OK - Packet loss = 0%, RTA = 0.06 ms
	Root Partition	Ok	262d 12h 0m 49s	1/4	2012-05-18 06:31:40	DISK OK - free space: / 7179 MB (77% inode=84%):
	SSH	Ok	262d 12h 0m 11s	1/4	2012-05-18 06:32:18	SSH OK - OpenSSH_5.3 (protocol 2.0)
	Swap Usage	Ok	262d 11h 59m 34s	1/4	2012-05-18 06:32:55	SWAP OK - 100% free (255 MB out of 255 MB)
	Total Processes	Ok	262d 11h 58m 56s	1/4	2012-05-18 06:33:54	PROCS OK: 101 processes with STATE = RSZDT
tesb_camel	Context_InflightExchanges	Ok	1h 1m 40s	1/3	2012-05-18 06:34:34	OK - InflightExchanges : Value 3 in range
	Context_Uptime	Ok	1h 1m 25s	1/3	2012-05-18 06:34:49	OK - Uptime : '1 hour 8 minutes' as expected
	Route_ExchangesCompleted	Ok	1h 1m 21s	1/3	2012-05-18 06:34:53	OK - ExchangesCompleted : Value 824 in range
	Route_ExchangesFailed	Ok	1h 1m 6s	1/3	2012-05-18 06:35:08	OK - ExchangesFailed : Value 0 in range
	Route_ExchangesTotal	Ok	1h 1m 6s	1/3	2012-05-18 06:35:08	OK - ExchangesTotal : Value 827 in range
	Route_LastProcessingTime	Ok	1h 1m 29s	1/3	2012-05-18 06:34:45	OK - LastProcessingTime : Value 5 in range
	Route_MaxProcessingTime	Warning	1h 1m 9s	3/3	2012-05-18 06:35:05	WARNING - MaxProcessingTime : Threshold '20' failed for value 59

Page 1 of 2 15 Per Page Go

4.5.4. Build and install the cxf-jmx sample applications

This sample is in `<TalendRuntimePath>/examples/talend/tesb/cxf-jmx` directory and consists of a number of parts:

Directory	Description
client/	a sample client application that uses the CXF JAX-WS API to create a SOAP client and make several calls with it.
common/	code that is common for both the client and the server.
features/	example features for use in OSGi deployment.
service/	the CXF web service provider packaged as an OSGi bundle.
war/	A WAR archive containing code from common and service modules. This is for Servlet container use only, not used in OSGi deployment.

1. From the base directory of the sample, the maven `pom.xml` file can be used to build and run the demo

   ```
   cd cxf-jmx
   mvn install
   ```

2. Start Talend Runtime container:

   `trun.sh` (on Linux)

   `trun.bat` (on Windows)

3. Add cxf-jmx example features URL. Type this command in Talend Runtime container:

   ```
   features:addurl mvn:org.talend.esb.examples/cxf-jmx-feature/5.2.1/xml
   ```

4. Install cxf-jmx example feature into the Talend Runtime container

   ```
   features:install cxf-jmx-service
   ```

5. You can find wsdl at http://localhost:8040/services/simpleService?wsdl

6. Now run the client; from cxf-jmx folder run:

   mvn exec:java -pl client

   Note: this will include some deliberate errors (to simulate failed requests), which you can ignore:

```
[...]
Hi Alex!
Hi Alex!
Hi Alex!
2
4
6
8
10
[WARNING]
java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
[...]
```

7. After the SOAP calls on the web service have completed, in JConsole, under `org.apache.cxf` you'll see the `Performance.Counter.Server` folder, where CXF MBeans with their attributes will be listed (see Section 3.2.2, "*Running JConsole*" for more details).

Now we do the Nagios monitoring:

1. Add the template configuration file to the existing `nagios.cfg`, here are some examples, which depend on where your installation puts config files:

   In `/usr/local/nagios/etc/nagios.cfg` add:

   `cfg_file=/usr/local/nagios/etc/objects/cxf_host.cfg`

   Or in `/etc/nagios3/nagios.cfg` add:

   `cfg_file=/etc/nagios3/etc/objects/cxf_host.cfg`

2. Then, restart Nagios for the changes to take effect.

   # service nagios restart

   Note: the name of this service may vary, depending on which package you used to install Nagios, so it may be called, for example, `nagios3`, instead of `nagios`.

3. Login to the Nagios Web Interface `http://<nagios_host>/<nagios_server>/` for example: `http://localhost/nagios/`, `http://192.168.1.10/nagios3/` or `http://192.168.198.5/nagios/`.

   There, you will find the status of metrics for cxf-jmx example.

4.5.5. Config file for Rent-a-car example

As a further example, here is the `new_host.cfg` file for the Rent-a-Car basic example (the host definitions are the same as the `cxf_host.cfg` file and have been omitted):

```
define service {
```

```
    use                     generic-service
    service_description     TESB_RAC_CRMService
    display_name            TESB_RAC_CRMService
    check_interval          1
    host_name               tesb_cxf
    check_command           check_jmx4perl_cxf!EndpointState!CRMServiceProvid
er!{http://services.talend.org/CRMService}CRMServiceProvider

    }

define service {
    use                     generic-service
    service_description     TESB_RAC_ReservationService
    display_name            TESB_RAC_CRMService
    check_interval          1
    host_name               tesb_cxf
    check_command           check_jmx4perl_cxf!EndpointState!ReservationServi
ceProvider!{http://services.talend.org/ReservationService}ReservationServic
eProvider

    }

 define service {
    use                     generic-service
    service_description     ReservationService_ANY_FAULTS
    display_name            ReservationService_ANY_FAULTS
    check_interval          1
    host_name               tesb_cxf
    check_command           check_jmx4perl_cxf!AnyFaults!ReservationServicePr
ovider!{http://services.talend.org/ReservationService}ReservationServicePro
vider

    }
```

You can run the example, add `new_host.cfg` to the `nagios.cfg` as before, restart Nagios, and view the metrics.

4.5.6. Monitoring an Apache ActiveMQ broker with Nagios

For more details on running and configuring Apache ActiveMQ, see *Talend ESB Infrastructure Services Configuration Guide*.

The process is to start an Apache ActiveMQ standalone broker on the `jolokia_host` machine, and then monitor it on the Nagios machine. The Jolokia agent (JAR file) has been already integrated into the ActiveMQ distribution which is included in Talend ESB.

1. On the `jolokia_host` machine, we start the Apache ActiveMQ standalone broker which is to be monitored; in a command console:

   `cd <TalendRuntimePath>/activemq/bin`

   `activemq console` (Linux*)
   `activemq` (Windows)

   The Apache ActiveMQ broker should now be running.

*Note the "console" option in Linux runs the broker in the foreground; the default is to run it in the background.

2. You can view this using the Web Console at http://localhost:8161/admin/; if you access the URL http://localhost:8161/jolokia/version, you will see a JSON output line about version information, which indicates the Jolokia agent is running correctly.

   We use this Web Console to define information to be monitored - these corresepond to pre-configured entries in the check command in the Talend ESB configuration `<TalendRuntimePath>/add-ons/adapters/nagios/template/activemq_host.cfg`.

3. Create a **Queue** with the name `example.A`.

4. Create a **Topic** with the name `ActiveMQ.Advisory.Consumer.Queue.example.A`.

Now we configure the Nagios machine to do the monitoring:

1. Check that the value of `jolokia_host` in `/etc/hosts` corresponds to the machine running the Apache ActiveMQ broker, for example:

   ```
   192.168.1.101 jolokia_host
   ```

2. Copy the configuration files `activemq.cfg`, `activemq_host.cfg` and `jmx_commands.cfg` from the `<TalendRuntimePath>/add-ons/adapters/nagios/template` directory to the Nagios configuration folder, for example: `/usr/local/nagios/etc/objects/` or `/etc/nagios3/etc/objects/`

   See Section 4.4.1, *"Syntax for adding metrics for monitoring"* for details of the check definitions, command definitions and other configuration details for ActiveMQ which are pre-defined in these files.

3. Edit `nagios.cfg` and add:

   ```
   cfg_file=/usr/local/nagios/etc/objects/activemq_host.cfg
   cfg_file=/usr/local/nagios/etc/objects/jmx_commands.cfg
   ```

   (these paths may be different, depending on your version of Nagios).

4. Define macros which will be used by `jmx_commands.cfg` in the existing `resource.cfg`; here are some examples, which depend on where your installation puts config files:

   In `/usr/local/nagios/etc/resource.cfg` add these lines:

   ```
   # set the path which jmx4perl plugin installed
   $USER5$=/usr/local/src/jmx4perl/scripts
   # set the path to where to find configuration files
   $USER6$=/usr/local/nagios/etc/objects
   ```

   Or in `/etc/nagios3/resource.cfg` add these lines:

   ```
   # set the path which jmx4perl plugin installed
   $USER5$=/usr/local/src/jmx4perl/scripts
   # set the path to where to find configuration files
   $USER6$=/etc/nagios3/etc/objects
   ```

5. Then, restart Nagios for the changes to take effect.

   ```
   # service nagios restart
   ```

   Note: the name of this service may vary, depending on which package you used to install Nagios, so it may be called, for example, `nagios3`, instead of `nagios`.

6. Finally, Login to the Nagios Web Interface `http://<nagios_host>/<nagios_server>/` for example: `http://localhost/nagios/`, `http://192.168.1.10/nagios3/` or `http://192.168.198.5/nagios/`, and select **services**.

   There, you will find the status of metrics for ActiveMQ.

4.6. Resources and metrics that are being monitored

Here is a complete list of the default metrics for CXF, Camel and Activemq that are being monitored in Talend ESB. More detailed definitions can be found in `<TalendRuntimePath>/add-ons/adapters/nagios/template/cxf.cfg`, `camel.cfg` and `activemq.cfg`.

4.6.1. CXF services metrics

Name	MBean	Attribute
NumInvocations	org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service="$1",port="$0"	NumInvocations
TotalHandlingTime	org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service="$1",port="$0"	TotalHandlingTime
NumCheckedApplicationFaults	org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service="$1",port="$0"	NumCheckedApplicationFaults
NumLogicalRuntimeFaults	org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service="$1",port="$0"	NumLogicalRuntimeFaults

Name	MBean	Attribute
NumRuntimeFaults	org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service="$1",port="$0"	NumRuntimeFaults
NumUnCheckedApplicationFaults	org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service="$1",port="$0"	NumUnCheckedApplicationFaults

4.6.2. Camel routes/contexts metrics

Name	MBean	Attribute
Context_InflightExchanges	org.apache.camel:context=*,type=context,name="$0"	InflightExchanges
Context_Uptime	org.apache.camel:context=*,type=context,name="$0"	Uptime
Route_ExchangesCompleted	org.apache.camel:context=*,type=routes,name="$1"	ExchangesCompleted
Route_ExchangesFailed	org.apache.camel:context=*,type=routes,name="$1"	ExchangesFailed
Route_ExchangesTotal	org.apache.camel:context=*,type=routes,name="$1"	ExchangesTotal
Route_LastProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	LastProcessingTime
Route_MaxProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	MaxProcessingTime
Route_MinProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	MinProcessingTime
Route_MeanProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	MeanProcessingTime
Route_TotalProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	TotalProcessingTime

4.6.3. ActiveMQ queues/topics metrics

Name	MBean	Attribute
Broker_TotalConsumerCount	org.apache.activemq:BrokerName=$0,Type=Broker	TotalConsumerCount
Broker_TotalDequeueCount	org.apache.activemq:BrokerName=$0,Type=Broker	TotalDequeueCount
Broker_TotalEnqueueCount	org.apache.activemq:BrokerName=$0,Type=Broker	TotalEnqueueCount
Broker_TotalMessageCount	org.apache.activemq:BrokerName=$0,Type=Broker	TotalMessageCount
Broker_MemoryPercentUsage	org.apache.activemq:BrokerName=$0,Type=Broker	MemoryPercentUsage
Broker_StorePercentUsage	org.apache.activemq:BrokerName=$0,Type=Broker	StorePercentUsage
Broker_TempPercentUsage	org.apache.activemq:BrokerName=$0,Type=Broker	TempPercentUsage
Queue_ConsumerCount	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	ConsumerCount
Queue_DequeueCount	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	DequeueCount

Name	MBean	Attribute
Queue_DispatchCount	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	DispatchCount
Queue_EnqueueCount	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	EnqueueCount
Queue_ExpiredCount	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	ExpiredCount
Queue_InFlightCount	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	InFlightCount
Queue_MaxEnqueueTime	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	MaxEnqueueTime
Queue_MemoryPercentUsage	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	MemoryPercentUsage
Queue_QueueSize	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	QueueSize
Queue_ProducerCount	org.apache.activemq:BrokerName=$0,Type=Queue,Destination=$1	ProducerCount
Topic_AverageEnqueueTime	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	AverageEnqueueTime
Topic_ConsumerCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	ConsumerCount
Topic_DequeueCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	DequeueCount
Topic_DispatchCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	DispatchCount
Topic_EnqueueCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	EnqueueCount
Topic_ExpiredCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	ExpiredCount
Topic_InFlightCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	InFlightCount
Topic_MaxEnqueueTime	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	MaxEnqueueTime
Topic_MemoryPercentUsage	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	MemoryPercentUsage
Topic_MinEnqueueTime	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	MinEnqueueTime
Topic_ProducerCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	ProducerCount
Topic_QueueSize	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	QueueSize

4.7. Metric criteria

This section describes the default metrics for CXF, Camel and ActiveMQ in Talend ESB, and the criteria for them signalling a state change in Nagios.

All states for Nagios checks are categorized as OK, WARNING, CRITICAL and UNKNOWN. For additional information about Nagios states see the State types in Nagios documentation.

Some examples of these states are:

- All Fault metrics indicate a warning state if 1 fault has occured and a critical state if 100 faults have occured.
- Multicheck AnyFaults is used for fault status indication.
- All countable metrics indicate a warning state if a count of 1,000,000 reached. It can be tuned for specific needs.
- All memory usage metrics indicate a warning state when 80% of memory is used and CRITICAL if 90%.

Here is a complete metric semantics table:

4.7.1. CXF services metrics criteria

Name	State change criteria
NumInvocations	Critical 6000, Warning 5000
TotalHandlingTime	Critical 6000000, Warning 5000000
NumCheckedApplicationFaults	Critical 100, Warning 1
NumLogicalRuntimeFaults	Critical 100, Warning 1
NumRuntimeFaults	Critical 100, Warning 1
NumUnCheckedApplicationFaults	Critical 100, Warning 1

4.7.2. Camel routes/contexts metrics criteria

Name	State change criteria
Context_InflightExchanges	Critical 20, Warning 10
Context_Uptime	Critical 6000000, Warning 5000000
Route_ExchangesCompleted	Critical 6000000, Warning 5000000
Route_ExchangesFailed	Critical 100, Warning 1
Route_ExchangesTotal	Critical 6000000, Warning 5000000
Route_LastProcessingTime	Critical 100, Warning 20
Route_MaxProcessingTime	Critical 100, Warning 20
Route_MinProcessingTime	Critical 20, Warning 10
Route_MeanProcessingTime	Critical 100, Warning 20
Route_TotalProcessingTime	Critical 6000000, Warning 5000000

4.7.3. ActiveMQ queues/topics metrics criteria

Name	State change criteria
Broker_TotalConsumerCount	Warning 1000000

Name	State change criteria
Broker_TotalDequeueCount	Warning 1000000
Broker_TotalEnqueueCount	Warning 1000000
Broker_Uptime	Warning 1000000
Broker_TotalMessageCount	Warning 1000000
Broker_MemoryPercentUsage	Critical 90, Warning 80
Broker_StorePercentUsage	Critical 90, Warning 80
Broker_TempPercentUsage	Critical 90, Warning 80
Queue_ConsumerCount	Warning 1000000
Queue_DequeueCount	Warning 1000000
Queue_DispatchCount	Warning 1000000
Queue_EnqueueCount	Warning 1000000
Queue_ExpiredCount	Critical 20, Warning 10
Queue_InFlightCount	Critical 20, Warning 10
Queue_MaxEnqueueTime	Critical 400, Warning 200
Queue_MemoryPercentUsage	Critical 100, Warning 80
Queue_QueueSize	Warning 80000
Queue_ProducerCount	Warning 1000000
Topic_AverageEnqueueTime	Warning 180, Critical 400
Topic_ConsumerCount	Warning 1000000
Topic_DequeueCount	Warning 1000000
Topic_DispatchCount	Warning 1000000
Topic_EnqueueCount	Warning 1000000
Topic_ExpiredCount	Critical 40, Warning 10
Topic_InFlightCount	Warning 1000
Topic_MaxEnqueueTime	Critical 100,Warning 40
Topic_MemoryPercentUsage	Critical 90, Warning 80
Topic_MinEnqueueTime	Warning 200
Topic_ProducerCount	Warning 1000000
Topic_QueueSize	Warning 1000