NSI-RI documentation

This document describes configuration parameters for NSI Reference Implementation (NSI-RI). Additionally provides built-in command line description, Network Resource Manager interface description and examples on how to run NSI-RIs with multiple topologies.

NSI-RI implements NSI Connection Service protocol version 2 [1].

# Prerequisites

Machine with Java 1.5 or greater installed.

Optionally RabbitMQ can be installed - this is a NSI-CONTEST specific feature.

IP address accessible from Internet if NSI-RI is to serve requests received from remote agents.

# Configuration parameters

Upon calling starting script, the first thing NSI-RI does is to read and validate configuration parameters. These parameters are used to initialize different components of NSI-RI. Configuration can be reloaded at any time with the "reload" command available in command line mode.

Logging can be configured with the help of \etc\log4j.properties file.

The NSI-RI configuration file is located at \etc\nsi.properties. This file is organized in a key-value fashion, meaning that only values can be altered. The following property entries are defined:

|  |  |
| --- | --- |
| Key | Value |
| ip | ip address in either canonical or domain form. "localhost" or "127.0.0.1" can be assigned. If this instance is to be accessible from Internet, an ip address assigned to an interface on deployment machine should be set |
| port | [1024-65535] port number on which this instance will listen on |
| nsa\_id | literal with identifier of this instance. The value of this property is internally used as providerNSA field in CommonHeaderType. Also it determines ownership of topology (topology with nsaId that matches this entry will be assigned to this instance) |
| service\_type | describes what type of requests are supported by this NSI-RI. This should always be <http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-GOLE> |
| reserve\_held\_timeout | [0-360] number of seconds reservation in state HELD will wait for either commit or abort message |
| fail\_if\_response\_not\_delivered | [true | false] true to fail operation in case asynchronous response could not be delivered. False to proceed to next state despite confirmation/failure message not being delivered |
| set\_ero | [true | false] true to always set ERO in reserve requests. This only works for multi-domain requests |
| requester\_timeout | [0-360] number of seconds requester will be awaiting asynchronous response before timing out |
| log\_header | [true | false] true to log common header verification output |
| log\_soap\_messages | [true | false] true to capture soap messages and write them to a file |
| log\_responder | [true | false] true to log responder output. Responder sends confirmation/failure and notification messages |
| num\_threads | [0-1024] size of internal thread pool where task jobs are submitted. Higher value allows more requests to be processed simultaneously |
| nrm | literal with fully qualified name of the class implementing the Nrm interface. If not set default SimpleNrm implementation will be used |
| topologies | comma separated file names. These files should be of nml/nsi topology type that will be added NSI-RI global topology |
| cli | [true | false] true to enable command line running in foreground (if you logout instance will be shut down), or false to run in background |
| telnet\_port | [0 | 1024-65535] port number for accepting telnet connections. 0 disables telnet server |
| disable\_nsi\_contest | [true | false] true to disable functionality related to the NSI-CONTEST project - instance will behave as generic NSI agent. False to enable NSI-CONTEST features |
| enable\_rabbitmq | [true | false] true to allow some events to be posted on rabbit message queue. This will work only if disable\_nsi\_contest is set to false |
| rabbitmq\_host | location of rabbitmq server where message can be posted |

**Important**

The urls for Provider and Requester services are created according to the following rule:

Provider - http://ip:port/nsicontest/provider

Requester - http://ip:port/nsicontest/requester

Provider service url and nsa\_id should match nsi/nml tags <nsi:link> and <nsi:NSA id> respectively.

Once configuration file has been set, NSI-RI can be started. To do that navigate to main NSI-RI directory and launch start.sh script. After a couple of seconds instance will be ready, or an error message will be displayed in case of failure. To stop running instance call stop.sh script.

# Command line

NSI-RI is supplied with command line that can be used to inspect current state of reservations, making new requests, topology and nrm management. Command line can be accessed in two ways:

* as a regular command line interface available once NSI-RI initialized - this can be enabled by setting the cli property of nsi.properties to true
* as a regular command line interface available by connecting with telnet client - this can be enabled by setting the telnet\_port property of nsi.properties to a number from 1024-65535 range. Note that only localhost connections are allowed (if you connect from a different host it will be rejected).

NSI-RI is a simple command line interface (with no support for tab) with commands that fall into five categories:

* utility - allows to reload configuration, show command line help or close NSI-RI
* provider - allows to interact with internal reservations directly (with omission of web service stack). This type of commands always start with the "provider" or "prov" word
* requester - allows to interact with Requester (Provider client). This command can be used to send soap requests to self or other domains. Taking into consideration a large number of parameters required for reserve operation, instead of typing them all, they are read from /etc/request.properties file. This type of commands always start with the "requester" or "req" word
* topology - user can view global topology (topologies uploaded to his NSI-RI) as well add new topologies from file (must be in nsi/nml format). This type of commands always start with the "topology" or "topo" word
* nrm - mainly used to export nrm topology into nsi/nml topology file. This type of commands always start with the "nrm" word

If disabling command line is desired, set cli=false and telnet\_port=0 in nsi.properties file.

**Important**

If command line is running in foreground (cli=true) and the ssh session is interrupted or quit, NSI-RI will be shut down.

# Attaching custom NRM

NRM stands for Network Resource Manager and is responsible for handling and managing data plane state. NSI-RI sits on top of NRM and delegates certain requests to it via the Nrm interface.

This interface can be found by inspecting \lib\nsi.jar file - net.geant.nsicontest.nrm.Nrm. By default simple implementation of the Nrm interface is provided - SimpleNrm. It does not do much however, all requests passed to it ends successfully. For more real scenarios one might want to provide his own implementation, probably with tracking resource utilization, configuring network devices and so on. To have NSI-RI redirecting requests to custom implementation, set the nrm property of nsi.properties to point to your class. The following table summarizes the Nrm interface functionality:

|  |  |
| --- | --- |
| Method | Description |
| getTopology |  |
|  |  |
| reserve |  |
| commit |  |
| abort |  |
|  |  |
|  |  |

**Exporting NRM topology in nsi/nml format**

When reserve message is processed by Provider, it parses source and destination STPs and passes them down to NRM. Consequently, it is up to NRM how to map abstract STPs to meaningful

# Sending requests with Requester

Requester can be created and used for sending NSI Connection Service v2 soap messages with command line interface. However given a large number of parameters that operation reserve takes, it would be uncomfortable to specify them by typing. Therefore a special file with reservation parameters is used - \etc\request.properties. It is a key-value text file that is read every time reserve operation is called, meaning it can be altered between reserve(modify) calls:

|  |  |
| --- | --- |
| Key | Value |
| endpoint | url where requests will be sent. Endpoints can be found in topology as <nsi:link> |
| provider\_nsa | nsaId of receiving agent. This can be found in topology as <nsi:NSA id> |
| reply\_to | url where asynchronous responses from remote Provider should be sent. This should be the Requester service url of this agent, a combination of ip and port values from nsi.properties - http://ip:port/nsicontest/ConnectionRequester |
| requester\_nsa | nsaId of agent sending requests. Should equal to <nsi:NSA id> associated with this Requester |
| reservation\_id | optional reservation identifier |
| description | literal with optional description of reservation |
| start\_time | reservation start time in seconds, blank for immediate start |
| end\_time | reservation end time in seconds, blank for infinite reservation |
| version | a nonnegative number with initial connection version |
| service\_type | literal with requested service type. Only <http://services.ogf.org/nsi/2013/07/descriptions/EVTS.A-GOLE> is supported |
| source\_stp | source STP (optionally with vlans) |
| dest\_stp | destination STP (optionally with vlans) |
| ero | even number of STPs that requested path should go through |
| capacity | a nonnegative number of requested capacity (bps) |
| bidirectional | [true | false] true if requesting bidirectional path |
| symmetric\_path | [true | false] true if requesting symmetric path |

**Example - one domain, one agent**

This example shows how to send a single domain request, in this case to self.

Setup new instance of NSI-RI with the following configuration parameters (nsi.properties):

ip = localhost

port = 8080

nsa\_id = urn:ogf:network:aruba.example:2013:nsa

topologies = \docs\aruba.xml

cli = true

The rest of entries need not be altered.

Edit \etc\request.properties:

endpoint = <http://localhost:8080/nsicontest/ConnectionProvider>

provider\_nsa = urn:ogf:network:aruba.example:2013:nsa

reply\_to = <http://localhost:8080/nsicontest/ConnectionRequester>

requester\_nsa = urn:ogf:network:aruba.example:2013:nsa

source\_stp = urn:ogf:network:aruba.example:2013:bi-ps

dest\_stp = urn:ogf:network:aruba.example:2013:bi-aruba-bonaire

The rest of entries need not be altered.

Go back to NSI-RI and type the following command:

requester reserve (or shorter form req reserve)

It should succeed (go to HELD state), type another command:

requester commit (or req commit)

Reservation should be committed (START state).

The example above showed how to use built-in Requester in order to send messages to Provider (to self in this case). As a result both Requester and Provider log their output to the same console making log analysis a bit difficult. They can be easily separated thus making log tracing more comfortable, as shown in next example.

**Example - one domain, one agent**

To show separated Provider and Requester to NSI-RIs must be started. Start NSI-RI with properties set as in previous example. Create new NSI-RI and set its port property value to 8081 (since you cannot have two services running on the same port). Second NSI-RI will serve as Requester so set its request.properties as in example above except reply\_to value - it should read <http://localhost:8081/nsicontest/ConnectionRequester>. Use second NSI-RI command line to type commands as in previous example. It should result in successful reservation but with logs displayed in two windows.

**Example - two domains, two agents**

This example shows multi-domain requests in action (multi-domain involves Aggregator and triggers path computation). Two NSI-RI instances will be needed with topology files - aruba.xml and bonaire.xml found under \docs directory.

First instance will be named as Aruba, with the following nsi.properties:

ip = localhost

port = 8080

nsa\_id = urn:ogf:network:aruba.example:2013:nsa

topologies = \docs\aruba.xml, \docs\bonaire.xml

cli = true

\docs\aruba.xml should contain correct <nsi:link> and <nsi:NSA id> values.

Second instance will be named as Bonaire, with the following nsi.properties:

ip = localhost

port = 8081

nsa\_id = urn:ogf:network:bonaire.example:2013:nsa

topologies = \docs\aruba.xml, \docs\bonaire.xml

\docs\bonaire.xml should contain correct <nsi:link> and <nsi:NSA id> values.

Start both instances by calling their starting scripts.

To order reservation spanning from Aruba to Bonaire, provide the following \etc\request.properties for Aruba:

endpoint = <http://localhost:8081/nsicontest/ConnectionProvider>

provider\_nsa = urn:ogf:network:bonaire.example:2013:nsa

reply\_to = <http://localhost:8080/nsicontest/ConnectionRequester>

requester\_nsa = urn:ogf:network:aruba.example:2013:nsa

source\_stp = urn:ogf:network:aruba.example:2013:bi-ps

dest\_stp = urn:ogf:network:bonaire.example:2013:bi-ps

**Important**

Care should be taken when specifying source and destination STPs for multi-domain requests. According to the NSI CS v2 protocol, two STPs are required for reservation per domain. If source STP is specified as egress STP, there would not second STP for that domain. Similar, ingress STP should not become destination STP.

# References

[1] NSI - <http://redmine.ogf.org/projects/nsi-wg>

[2] Topology tutorial - <https://github.com/jeroenh/NSI-Tutorial-Topologies>