Service Monitoring Examples

This page is a companion to the Managing Servers and Services with NMIS8 page, and provides some examples of how various services can be monitored with NMIS.

Web

Remote, port only

NMIS can monitor the accessibility of TCP ports (using the NMAP tool), and for a Web service that would tell you whether the server is reachable (but not whether it's fully working). This kind of monitoring does not require any software running on the target server, however.

Here is a configuration snippet for this level of monitoring, for the standard web ports 443 and 80, which you would activate for the server that you want to test:

Services Wed 14:10

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
port443	HTTPS	port	tcp:443	5m					view edit delete
port80	HTTP	port	tcp:80	5m					view edit delete

Server Process

If SNMP is enabled for the system in question, if NMIS is polling that system and if the system and its model supports the Host Resources MIB, then NMIS can check process statuses and verify the existence of a specific process. The Service Type must be "`service`", the name of the process must be given as Service Name, and you need to activate the service for the node that you want to test.

For a CentOS box with Apache 2.2.x we'd be using the following service definition, which checks for processes named "`httpd`":

Services Wed 13:57

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
HTTP_Daemon	httpd	service		5m					view edit delete

SAPI-script based

NMIS can also do a limited amount of interaction with a TCP-based service using SAPI scripts. Example scripts for POP3 and basic HTTP are shipped with NMIS in `/usr/local/nmis8/conf/scripts`. The default `http` script connects to the Web server in question and attempts to download the root index URL "/"; if this request succeeds or returns an HTTP redirect, then the service is considered to be ok.

To enable this kind of monitoring, you need to define the service with the Name matching the script file name. The Service Name can be a text of your choice, but the Service Type must be "`script`", and you must activate that service for the node that you want to communicate with:

Services Wed 14:17

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
http	Basic HTTP Interaction	script	80	5m					view edit delete

end-to-end using a custom program

If you need more precise interaction with your web service than the SAPI scripts can provide (e.g. SSL/TLS or cookies or the like), then you'll need to use a custom script. NMIS 8.5.4g ships with an example script of that type in `/usr/local/nmis8/install/scripts/webtest`, which should to be moved to a directory meant for binaries (e.g. `/usr/local/nmis8/bin` or `/usr/local/bin/`) if you want to use it.

NOTE - NMIS9 ships this script in /usr/local/nmis9/conf-default/scripts/webtest.

The example script downloads a web page (optionally following a number of redirections) using http or https, and optionally checks that the document content matches a given regular expression. You need to define this service with Service Type "`program`", provide suitable Program settings for the program and activate the service for the server that you want to test (but please note: the custom program will always be run *locally* on your NMIS server!)

Here is how we verify that the Opmantek website is up and running: this downloads the page using https, then looks for the phrase "Opmantek Products":

Services Wed 14:27

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
opmantek-site-ssl	opmantek-site-ssl	program		5m	/usr/local/bin/webtest	-c 'opmantek.*products' https://opmantek.com/	10	false	view edit delete

DNS

remote, port only

NMIS can monitor the accessibility of TCP and UDP ports (using the NMAP tool), which in the case of DNS would give only a rough indication of whether the DNS server is reachable at all.

Here is a configuration snippet for this level of monitoring:

Services Wed 13:13

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
port53	DNS port reachability	port	udp:53	5m					view edit delete

remote, protocol only

To verify the general operation of a remote DNS server, you can use the service 'dns' that's built into NMIS. This service will make a DNS request to the server in question and then triggers outage alerts based on getting a DNS record back or not (and also captures the response time).

Here is how our own internal monitoring is set up to check our own domain, which involves servers outside of our control: We've defined nodes with the model set statically to "PingOnly" for the external DNS servers in question, and activated service "opmantek-dns" for them, which looks like this:

Services Wed 13:17

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
opmantek-dns	opmantek.com	dns		5m					view edit delete

Please note that model "PingOnly" by itself is *not* sufficient to disable SNMP (or WMI) accesses; you also have to change the node configuration option `collect` to false.

local, custom script

On a system that is under your control, and which runs NMIS you can execute arbitrary scripts to collect service statuses. The example script below checks that the local NMIS server itself has a running BIND DNS server process:

```
#!/bin/sh
# small script that tests that a local bind is up and communicating
if /sbin/pidof named >/dev/null 2>&1 && /usr/sbin/rndc status | grep -q 'up and running'; then
        exit 100
else
        exit 0
fi
```

To use this, save the script somewhere NMIS can access it (as `/usr/local/bin/bindpresent` for example), then configure NMIS with this service of type "program" and activate the service for the NMIS server itself:

Services Wed 13:27

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
local-bind	local-bind	program		5m	/usr/local /bin/bindpresent		10	false	view edit delete

MySQL Database

remote, port only

To verify that your MySQL database server is reachable you could define a service to check TCP port 3306, similar to the examples above. Naturally that's not an end-to-end test.

remote, server process status

In addition to the port reachability you can define a service for checking the existence of the "`mysqld`" process, if you are polling the server in question with SNMP:

Services Wed 14:43

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
mysqld_daemon	mysqld	service		5m					view edit delete
port_3306	MySQL Port	port	tcp:3306	5m			10	false	view edit delete

remote, custom script

The third, and most comprehensive end-to-end monitoring setup would require a small custom script that actually connects to the server and performs a query on said server. Here is an example of such a script, which would have to be adjusted for your environment (or be changed to accept more command line arguments) and saved as `/usr/local/bin/mysqltest`:

```
#!/bin/sh
# a small wrapper around the mysql client, which connects to a test database
# and runs show tables; if successful (and there are tables) we call it good
NODE=$1                                                  # passed in, comes from node.host
DBUSER="mytest"
DBPASSWORD="something secret"
DBNAME="testdb"
OUTPUT=`mysql -u$DBUSER -p$DBPASSWORD -h$NODE $DBNAME -e "show tables;"`
if [ $? != 0 ]; then
        exit 0                                            # service bad
elif ! echo "$OUTPUT" | grep -q "Tables_in_"; then
        exit 50;                                        # service not fully ok
else
        exit 100;                                        # service good
fi
```

To use this service test, you'd define a service of Service Type "`program`", with an appropriate Program path, and with the Program Args being set to "`node.host`", which would be replaced by the address or hostname of the node in question:

Services Wed 15:01

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
mysql-tables	MySQL Tables	program		5m	/usr/local /bin/mysqltest	node.host	10	false	view edit delete

UPS Status

custom scripts

Cheaper UPS systems that don't have builtin networking or SNMP capabilities can be monitored by NMIS as well, as long as there is some sort of management infrastructure that supports querying the UPS status. In this example we're checking two UPS systems that are connected to our NMIS server via USB cables, where the NUT (Network UPS Tools) suite takes care of the interfacing.

The `upstest.pl` script below uses the NUT tools to query the named UPS and reports whether it's working and at what charge level it is. (NMIS does not yet graph extra variables like the charge level here as of version 8.5.4G, but this feature will be added soon.)

```
#!/usr/bin/perl
# a tiny wrapper around upsc to integrate with nmis
# exits with 100 if ups online, charge otherwise
# this means the service is down only when the ups is dead,
# NOT while its discharging.
# also reports battery charge as charge=NNN
use strict;
# args: name of the ups
my $upsname = $ARGV[0];
die "usage: $0 <upsname>\n" if (!@ARGV);
my @knownones = `upsc -L 2>/dev/null`;
die "unknown ups $upsname\n" if !grep (/^$upsname:/, @knownones);
my ($status,$charge);
for my $line (`upsc $upsname 2>/dev/null`)
{
                chomp $line;
                my ($varname,$value) = split(/\s*:\s*/, $line);
                if ($varname eq "ups.status")
                {
                                $status = $value;
                }
                elsif ($varname eq "battery.charge")
                {
                                $charge = $value;
                }
}
print "charge=$charge\n" if (defined $charge);
exit ($status =~ /^OL/? 100 : $charge);
```

For our UPS systems we first make use of NMIS' builtin SNMP-based process status monitoring, which checks that there is at least one active process with a given name (here 'upsd'), and then added the per-UPS status checks with the UPS names passed to the upstest script. This example setup requires that the UPSs are connected to the NMIS server itself, but NUT could of course be accessed over the network.

Here is our service definition:

Services Wed 13:32

Table Services

Name	Service Name	Service Type	Port	Poll Interval	Program Path	Program Args	Max Program Runtime	Collect Program Output	Action > add
ups-cyberpower	ups-cyberpower	program		5m	/usr/local/nmis8 /bin/upstest.pl	cyberpower	10	true	view edit delete
ups-eaton	ups-eaton	program		5m	/usr/local/nmis8 /bin/upstest.pl	eaton	10	true	view edit delete
upsd	upsd	service		5m			10	false	view edit delete