1. Nagios 3

Nagios

Nagios is an opensource monitoring and alertingtool. It consists of several CGI and HTML-Templates. The actual work is done by Nagios plugins. The only requirements for Nagios plugins are:

- The Nagios host has to be able to execute them. So as far as the environment is working properly Plugins can be anything: bash-scripts, c/c++ executables, java, perl, python...
- The plugin's output has to conform the Nagios Plugin Guidelines, see this for more information.
- The plugin has to return exitcodes.

This makes Nagios pretty flexible and extensible. Furthermore Nagios can be extended by Addons which enrich its functionality. A nice extension is PNP, it uses Nagios' performance data to build Round Robin Databases (RRD) to aggregate and store the information and is able to display them in graphs.

Setups

As mentioned before Nagios itself uses plugins to acquire information for alerting/monitoring. The question is now where these plugins get their information from about remote hosts.

Public Services

One possibility is to get information which is publically available like icmp pings and tcp socket connections. Icmp pings are used by default when hosts are defined. Tcp connections can be monitored with the shipped plugin **`check_tcp`** or **`check_http`** respectively.

Private Services

Due to security restrictions more detailed information are not available from the outside of a remote host. Therefore some plugins have to be deployed on the remote hosts and invoked locally on that remote machine. For the process of invoking there are several possible ways to go:

SSH

The most secure and most available way to invoke plugins on remote machines is ssh. It is possible to write a plugin for invoking the remote plugin by ssh. It can look like:

```
#!/bin/bash

ssh $1@$2 "$3"
exit $?
```

It is possible to invoke this script by **`./script.sh user remote_machine command`** whereas user, remote_machine and command have to be replaced by their actual values respectively. Furthermore I suggest using "Public Key Authentication" instead of "Password Authentication" (see here) because it is more secure and needed here.

The Contra for using ssh for invoking commands on remote hosts is the overhead imposed on the nagios server. Due to this overhead it will scale bad for many monitored machines.

NRPE

NRPE is the Nagios Remote Plugin Executer. Its a deamon which runs on the remote hosts and is able to execute the plugins there and redirect the output and exit code to the nagios server.

The Contra for this is the fact that NRPE is unlikely to find in Linux Distribution Repositories and has to be downloaded as source and built for each architecture which is required. Furthermore there is no real way of security or authorization in the whole communication process between the nagios server and the distributed NRPE deamons. One attempt is to wrap it in xinetd and define IPs which are allowed to contact the NRPE deamon, but this is no real security because IPs can be spoofed.

SNMPD

SNMP is the Simple Network Management Protocol and snmpd is the deamon for this. it is possible to define executeables as information sources for snmp and thats the trick which is used here. The requirements for using snmpd as the deamon for invoking the plugins is only an installed snmpd which is available in Linux Distribution Repositories, a configuration file which defines the location of the plugins to be executed and their names and locations and a net-snmp v3 user on the remote machine. We need v3 here because that is the version net-snmp learned authentication with. It is possible to automate these steps with few ressources needed.

The Contra for this is that there is no way to pass arguments to invoking the plugins, so everything has to be managed on the remote hosts themselves.

NSCA and Cron

A different approach is to use the NSCA tools. They consist of the NSCA server and send_nsca. The NSCA server runs on the Nagios server and send_nsca is used on the remote machines to transmit the plugin data to the NSCA server. For this setup it might be appropriate to use cron for regular checks on the remote machines and to trigger send_nsca when needed.

2. The CCMS-Plugin 0.7.3

The Plugin developed by Jan Dostert und Wolfgang Rosenauer uses the RFCSDK 46B. The current official release of SAP ERP is 7.10/7.11. Due to these discrepancies there is a problem at the registration between the plugin and the SAP System. Because passwords were used to be uppercase in older ERP releases, the old RFCSDK 46B transforms passwords to uppercase automatically during transmission but passwords of current releases of ERP are case-sensitive. A known workaround was to set up an uppercase password on the remote SAP System.

3. Changes

Since Nagios 3.0b1 there is a config-option in nagios/etc/cgi.cfg which enables/disables html escape.

An Excerpt from the cgi.cfg:

# ESCAPE HTML TAGS
# This option determines whether HTML tags in host and service
# status output is escaped in the web interface. If enabled,
# your plugin output will not be able to contain clickable links.

escape_html_tags=1

Since the Plugin in Version 0.7.3 is used to print links and other html structures its recommended to adjust this setting in your configuration.

Because of the new Plugin-Output-Guidelines, which should be implemented if one wants his performancedata to be recognized by nagios and therefor pnp, its not possible to produce some html structures like `table` or `div`. These were used in version 0.7.3. More information are available on http://nagios.sourceforge.net/docs/3_0/pluginapi.html. So as of the output of the CCMS-Plugins is guideline compliant and can be recorded by addons.

To solve the password problem from section 2, the shipped sap_moni.so was statically linked against the RFCSDK 7.11 library. Its appropriate to note here that the RFCSDK 7.11 will be the last one without an API-Change. Newer versions of the RFCSDK will be called NetWeaver RFCSDK and wont be compatible with older versions. Furthermore librfc.a was removed instead librfccm.so was used.

The `Makefile` of version 0.7.3 has produced "Position independant Executables" which is unnecessary and impacts the execution runtime in a negative way. This was corrected. For more information about `gcc -fPIC` I suggest http://blog.flameeyes.eu/2008/12/07/again-pic-and-executables-this-time.

Furthermore an issue with a missing prototype header was fixed in several plugins causing strange Segmentation Faults on 64Bit architectures.

The CCMS-Plugin is compileable and tested on x86, x86_64, ia64, ppc64 and s390.

Last but not leased a reported bug concerning unexpected returncodes was fixed, compare https://sourceforge.net/tracker/index.php?func=detail&aid=2548272&group_id=114459&atid=668390.

4. How to get started

In this section I will show you how to perform after the download of the CCMS-Plugin to get it working with Nagios.

Unpacking and Installing

```
tar xzf sap-ccms-plugin-0.7.x
cd sap-ccms-plugin-0.7.x/src
make
```

By now everything should be compiled. The executables link dynamically against librfccm.so and sap_moni.so. These files are located in `src` should be placed somewhere the system can find them. I suggest copying them to `/usr/lib`.

Furthermore there are several configuration files in `sap-ccms-plugin-0.7.x/config`:

- agent.cfg

- login.cfg
- moni_tr.cfg
- ssh_config

These should be copied to `/etc/sapmon` and made accessible for nagios.

The Configurationfiles

For the beginning only `/etc/sapmon/login.cfg` and `/etc/sapmon/agent.cfg` are interesting. Lets have a look at the first one.

```
cat /etc/sapmon/login.cfg
```

```
[LOGIN_LNX]
LOGIN=-d <SID> -u <username> -p <password> -h <hostname|ip> -s <instance-number> -
c <client>
```

This configuration file defines a handle called "LNX" (due to [LOGIN_LNX]) and the parameters needed to perform a RFC-Call into that SAP-System. Fill in the values of the System that you want to monitor. Whereas `instance-number` is a two digit number which will be used to concatenate the port where the rfc-call will be issued to. `client` is a three digit number. Feel free to adjust the handle (LNX) to match your System-ID.

```
cat /etc/sapmon/agent.cfg
```

```
[EMPLATE_0]
DESCRIPTION="Free Swap"
MONI_SET_NAME=SAP CCMS Admin Workplace
MONI_NAME="Operating System"
MAX_TREE_DEPTH=0
PATTERN_0="*\*\Swap_Space\Freespace""
```

This entry defines a template with the handle "0" (due to [TEMPLATE_0]). Please notice that you should start handle-names with a digit. The description is just for documentation. The `MONI_` variables are important. These describe which data you want to receive. To get an impression of how the data is organized log in the remote SAP System and use the transaction `RZ20`. You will see a list of nodes, these are the Monitor Sets ("MONI_SET_NAME"). Once you expand one of these nodes, you will see the actual Monitor ("MONI_NAME"). You can navigate inside a Monitor by double clicking it. Having entered a Monitor there is again a tree with nodes. These are Monitorobjects and Monitorattributes. Back to the config file. MAX_TREE_DEPTH describes how many subnodes you want to receive by a given Monitor Name, "0" indicates all entries.
With `PATTERN_0` you can submit a regular expression which will be used to filter the results. You can find these patterns in the `RZ20` when you navigate in the monitor down to a leaf, mark it and click on `Properties`. In the first line of the appearing window there will be written the pattern used to access this leaf/entry.

First RFC Call with a plugin

As of now you should have understood the principles of the configuration and should be ready for a first plugin-run. These plugins are - as you might have realized yet - simple executables written in C. So you dont need Nagios to test them, its possible to run them in a shell.

Open a shell and navigate to the `src` subdirectory. There should be several executables starting with `check_sap*`

Issue
./check_sap

The output will be:

```
Agent

Syntax: ./check_sap <Template> <RFC-Template>

       <Template> defined in /etc/sapmon/agent.cfg
       <RFC-Template> defined in /etc/sapmon/login.cfg
```

We will use the Template "0" and the RFC-Template you have just created - assuming its still called "lnx".
Issue:

```
./check_sap 0 lnx
```

If everything went well you will see something like:

```
Freespace=5149MB;50.000000;25.000000;0;
```

Creating a Nagios User in the System

You were using a SAP_ALL account to get these data by now. For a productive usage this is not appropriate.

Creating User

So navigate to `SU01` and create a new User without any Roles or Profiles so far.

Analyzing Authority Objects

It's good practice to use as less privileges as needed for security reasons. The first thing to do for this is to find out which privileges the Plugin needs to perform its task. One possible approach is to activate Auth-Tracing in the SAP-System by navigating to Transaction `ST01`. On this GUI you select

- Authorization Check
- RFC Call

And afterwards click on `Trace On`. Rerun the plugin and deactivate the Trace again by clicking on `Trace Off` in the Transaction `ST01`.
To get to your results click on `Analysis` in the same Transaction, fill in the appropriate user and time-interval and start this report. Having done this you will get a list with tables. Search for green lines which have a matching timestamp on them and have `S_RFC` as `Object`. Those other values in the `Object` column are the Authorization Objects you will need to perform that specific RFC-Call - including `S_RFC`.

Creating the needed Role

Now that you have the authorization objects located its time to set up a SAP Role with these.

- Navigate to Transaction `PFCG` enter a new role-name and click on the button `Role` right of the input-field.
- In the new screen which appears now, you can add some descriptive text and jump to tabs like `Authorization` and `User`. There are more tabs but only the latter two are of interest. Lets first have a look at the `Authorization` tab.
- Having the `Authorization` tab open there is a button on the bottom labeled with `Expert Mode for Profile Generation`, click it.
- In the next screen click on `Manually` which should be in the first row of the screen.
- Fill in the authorization objects recorded by the Auth-Trace. And press the check.
- You will see some new rows on the screen with a yellow triangle next to their title, click on that triangle to set all attributes to "*". It will happen automatically after you clicked and confirmed. This step might take a while.
- Once this is done click on `Save`, the Disc-Symbol in the menubar, and on `Generate`, the red-white circle beneath the screen title. Make sure the the Status says `generated`.
- Hit the `Back` button to go back to the Profilescreen. The `Authorization` tab should have a green icon by now.
- The next step is to add the nagios-user you have created previously to this profile by clicking on the `User` tab and entering his name in the table.
- After hitting "Enter" the icon next to the `User` tab turns yellow and the icon next to the button `User comparison` turns red.
- Click this button to synchronize the just altered data in the user's database.
- After some confirming the `Authorization` and the `User` tab should have green icons. If so you successfully created a nagios-user, if not revise the steps above.

Testing the newly created User with the Plugin

To confirm that the newly created user performs well with the plugin, you should test it now. Before doing this, log in as the new user to change your initial password to something meaningful. Having logged in you can also verify your role with transaction `SU01` by entering the new user name and navigating to the role/profile tab on the next screen.

As long as everything is fine go on with this document, if you encounter any problems revise the steps above.

Assuming you could log in as the new nagios user and verify your profile/role, open `/etc/sapmon/login.cfg` and alter your username and password to match the newly created credentials.
Rerun the Plugin. There should be a similar output as before.

Setting up Nagios 3

In the following I assume you have installed Nagios to `/usr/local`. As of now the plugins work and are ready to be imported in Nagios 3. Therefore copy the plugins you want to use to `/usr/local/nagios/libexec`

The plugins are:

- check_sap
- check_sap_cons
- check_sap_cpu_load
- check_sap_instance
- check_sap_instance_cons
- check_sap_multiple
- check_sap_mult_no_thr
- check_sap_system
- check_sap_system_cons

In the following I will explain to you how to set up the check_sap_cpu_load plugin in Nagios 3. The other plugins have a similar configuration.

Command Definitions

First of all Nagios needs to know how to access the plugin, so add this definition to `/usr/local/nagios/etc/objects/commands.cfg`

```
define command{
        command_name    check_sap_cpu_load
        command_line    $USER1$/check_sap_cpu_load $ARG1$
}
```

Here we declare the nagios-internal-command check_sap_cpu_load which is accessible by $USER1$/check_sap_cpu_load $ARG1$, whereas $USER1$ is /usr/local/nagios/libexec or respectively. As you see we pass the RFC-Handle dynamically by $ARG1$ which is the first argument that will be passed to the internal-nagios-command. You can address each argument by the digit in $ARGx$.

Host and Service Definitions

To structure the configuration files create the directory `/usr/local/nagios/etc/servers` and add the following line to your `/usr/local/nagios/etc/nagios.cfg`

```
cfg_dir=/usr/local/nagios/etc/servers
```

This tells Nagios to parse every cfg in this directory. Thats where we are going put the host- and servicedefinitions to.

/usr/local/nagios/etc/servers/hosts.cfg

```
define host {
        use linux-server
        host_name lsXXXX
        address XXX.XXX.XXX.XXX
}
```

Whereas the `address` directive is not necessary. Nagios can resolve the hostname. At this stage the host lsXXXX is known to Nagios. In the following step we are going to set up a service check depending on the check_sap_cpu_load plugin.

/usr/local/nagios/etc/servers/service.cfg

```
define service {
        use local-service
        host_name lsXXXX
        service_description     CPU Load
        check_command   check_sap_cpu_load!lnx
}
```

This definition tells Nagios to inherit configurations from `local-service` (which in turn can be found in `/usr/local/nagios/etc/objects/templates.cfg`) add this service to the host lsXXXX, call this service "CPU Load" and check thsi service with the intern-command `check_sap_cpu_load!lnx`.
I hope the context between the `commands.cfg` and this cfg gets more clear now. According to `commands.cfg`, `check_sap_cpu_load!lnx` gets mapped to `$USER1$/check_sap_cpu_load 0 lnx` whereas $USER1$ is `/usr/local/nagios/libexec`. Furthermore $ARG1$ is "lnx" in this case, so the delimiter for arguments is "!".

You can now navigate to the webfrontend of your nagios-monitor and there should be the host and the service we just defined.