NoHa

The Notification Handler – Design Document

Document version 0.03

Why?

Notifications in the configuration of any “nagios” based monitoring are complex to setup, handle and modify, where a lot of resources are wasted and locked in to make it work on a day to day basis. Make it so modular one can add new external applications to also submit notifications for filters.

NoMa – Notification Manager, a Netways project, is hard to maintain with the original authors no longer maintain it, and the code is not made to be agile to handle any input given, rather being locked towards a quite strict regime of notification filters.

How?

Python driven daemon based around the concept of event handling, using libraries to make it easier to maintain. A common task may be an event being called for notifications to be filtered for submission to other targets. Like a new notification from external daemons that want to notify something to someone.

A notification will have different inputs to be passed to the filter module, to return with groups and users to be notified. The notification rule is owned by the organization so anyone being a member of it, can modify it with the rights as manager+ in the organization.

Features

Daemon

Event based with safe threading so it can handle more than one notification at a time.

Support ANY input given, with a specified filter for the type of sender, i.e., if a “nagios” based sender.

Ex if nagios / icinga / centreon, input can be at least:

- Notification type (Problem, recovery, flapping*, downtime*, acknowledgement, custom)
- Notification state (OK, Warning, Critical, Unknown)
- Hostgroups filtering (include, exclude)
- Servicegroups filtering (include, exclude)

- Hosts (include, exclude)
- Services (include, exclude)
- Custom variables (include, exclude) (Note, both the name of the variable and the value itself must be given)

This must be processed fast with a query to the rules, and return valid rules for it, with more queries to get organizations, groups and users.

Frontend

Frontend to managing the notifications is important for ease of use, rather than configuration files only a few can manage, after all, that's why this idea is being written down.

If any rules, PHP should be avoided if possible.

Filter of notifications

Filters must be extremely flexible, to fit the needs of anyone, meaning that the user must be able to specify for each rule in what order it is to be handled.

Handling contact organizations, groups and users

Groups and Users:

- A user can be a member of a group
- *A group can be a member of a group?*

Time handling:

- Vacations set to timeframes
- Handle shifts, i.e. only notify while only contacts that are on shift
- Handle on-call in the allotted time, so only specified contacts get notifications, with optional escalation
- *Support time zones?*

Storing notifications for internal escalation + logs

Store in database, internal(python procedures), external (but local to box) or remote (remote host)?

Logging:

- Internal application logs: Log4J equivalent for Python
- Notification logs: ?
- Escalation Logs: ?
- Event logs: ?

Configuration

Configuration of filters for rules and how to notify must be flexible.

Propose by sni is to use a similar approach as of mod_rewrite, to use a set of conditions. With this approach, one can set the conditions as one wants to, while keeping in the good behavior of clear and readable filters as the examples below show.

Examples of filters(concept):

Ex 1:
*HOSTGROUPS;&;linux-**
HOSTGROUPS;!;windows-mssql
SERVICE;!;Memory, SSH
*HOST;!;lab**
_NRPE_AGENT;&;YES

Ex 2:
*HOSTGROUPS;&;windows-**
HOSTGROUPS;!;windows-mssql
SERVICE;!;Memory, CPU
*HOST;!;lab**

Ex 3:
SERVICEGROUPS;&;mysql-srv
*HOSTGROUPS;!; freebsd**
*SERVICE;!;Memory, SSH,_HOST_*
*HOST;&;test**

Basically, anything specified before = in the input is considered a variable and after is the content for filtering.

Syntax

Input

Input should have the following syntax for nagios based applications:

“<name>=<value>,<value>,<value>;<name>=<value>”

Ex 1: “NOTIFICATIONTYPE=PROBLEM;SERVICESTATE=CRITICAL;HOSTGROUPS=mysql-srv,ssh-srv;HOST=dbsrv03;SERVICE=Memory;OUTPUT=CRITICAL”

Ex 2: “NOTIFICATIONTYPE=PROBLEM;SERVICESTATE=OK;HOSTGROUPS=mysql-srv;HOST=dbsrv03;OUTPUT=OK”

Legend:

- <name> is case sensitive.

- = separates <name> from <value>, where everything after = where separated by , will be treated as a list in Python, similar to Array in Perl.
- <value> is case sensitive, it can have many entries if separated by comma, “,”.

Rules

One rule is made up of one or more conditions for it to pass. For a rule to be valid, all conditions must pass(be True)

Format of conditions:

<name>;<operator>;<value>,<value>,<value>

Any values given after the first in filter is optional.
The filter allows regex, ie asterisk if one wants to include everything starting with something, like linux-*

Legend:

- <name> is case sensitive, ex: HOSTGROUPS or SERVICEGROUPS
- <operator> can be &, | and !:
  - & = must contain
  - | = must contain
  - ! = must not contain
- <value> is case sensitive, can contain more, separated by comma, “,”.

Code structure

Libraries

Interface

Will contain basic functions:

- Compare strings
- Functions related to connecting to sockets and pipes
- Loading and saving to and from setting repositories

Ruleeval

Will contain functions related to evaluating rules and conditions

Messages

Will contain logging and message functions

Timehandling

Will contain functions related to time calculations, durations, if inside our outside timeranges, speciality functions like holidays, on-call and shift functions.