Table of Contents

Introduction

This page documents the tools we use for configuring Nagios headnodes. There are two specific usecases

- Configuring Nagios for a single VO (ie those instances managed by IT-SDC-MI).
- Configuring Nagios at a grid site (quite possibly for more than one VO).

Vision

Here's how we think this will all work in the future.

- If a machine is going to use the following approach to installing Nagios, it must have the following:
  - The necessary (vo-specific?) `ncg-metric-config` and `grid-monitoring-probes` RPMs installed. My understanding is that there RPMs only exist for SL5, so it may be easier for us to include the files in our own RPM (see below). For testing purposes, these have so far just been lifted from a working nagios instance (e.g. samnag042).
  - A network connection to reach the POEM and VO feed URLs
- We (will) ship an RPM which contains the `nagconf.rb` script (see below) along with a few Ruby template (`.erb`) files.
- The `nagconf.rb` script itself needs to be run every n-hours. In our usecase, we'll probably use Puppet to manage a cronjob to do this.

Instructions for RPM building

So, ultimately, we'll want NagConf to be contained within an RPM (currently the installation at CERN is managed with puppet). Here's a list of the files we want in that rpm, mapping where they are on the AI today, and where they should be installed by the RPM.

- puppet:///modules/hg_dashboard/nagios/cgi.cfg => /etc/nagios/cgi.cfg
- puppet:///modules/hg_dashboard/nagios/nagios.conf => /etc/httpd/conf.d/nagios.conf
- puppet:///modules/hg_dashboard/nagios/nagios.cfg => /etc/nagios/nagios.cfg
- puppet:///modules/hg_dashboard/nagios/php.conf => /etc/httpd/conf.d/php.conf
- puppet:///modules/hg_dashboard/nagios/ssl.conf => /etc/httpd/conf.d/ssl.conf
- puppet:///modules/hg_dashboard/nagios/nagconf/nagconf.rb => /usr/local/bin/nagconf.rb
- puppet:///modules/hg_dashboard/nagios/nagconf/erb_templates/* => /etc/nagconf/
- puppet:///modules/hg_dashboard/nagios/nagconf/host.templates.cfg => /etc/nagios/nagconf/host.templates.cfg
- puppet:///modules/hg_dashboard/nagios/nagconf/service.templates.cfg => /etc/nagios/nagconf/service.templates.cfg
- puppet:///modules/hg_dashboard/nagios/nagconf/commands.cfg => /etc/nagios/nagconf/commands.cfg

The directory `/var/nagios/spool/checkresults` needs creating with ownership nagios:nagios and mode 755

Puppet-specific notes

- In addition to the RPMs described above, you will need to set the `nagconf_vo` parameter to the VO of your choice in Foreman to trigger the initial installation of NagConf. Currently it's installed simply by copying files from the Puppet files directory, but in the future this will be an RPM install. `nagconf_vo` is either a single VO or a comma-separated list of VOs. Note that the latter of these scenarios is not as well tested, and will almost certainly result in duplicate Nagios objects being created (which need fixing before nagios will start).
- It's possible to configure nagconf to only perfom "hostcheck" tests against a node (ie, is this node up) rather than all of the grid-specific tests, by setting the Foreman parameter `nagconf_hostcheck` to `true`. This has the effect of creating a single Nagios configuration file (`hosts.cfg`) which contains only hostgroup and host information (i.e. no services).

Components

nagconf.rb

This script does the main work of configuring a Nagios instance. In production environments, this will most likely be executed by a cronjob, but can be called on the command line with the following parameters.

- --vo: (Mandatory). Specify a single VO or list of comma separated VOs (e.g. atlas,cms)
- --poem: (Optional). The POEM URL is hardcoded into `nagconf.rb`, but can be overridden with this command line option
- --site: (Optional). If provided, Nagios will only be configured for the specified site's nodes. Use the GOCDB-registered name for the site.
- --confdir: (Optional). Location to store Nagios configuration. Defaults to /etc/nagios/nagconf

Execution logic

1. From `getPOEM()`, get the POEM XML data and store it in a nested hash (`@poem_hash`). An md5 checksum is calculated of the XML data (and the XML itself stored in a temporary file) so we know if it has changed since the last execution.
2. For each VO that we want to configure
    1. With `getVOfeed()`, get the VO feed (as before, calculating an md5 checksum and storing the XML in a file) and create two lists and a hash:
        - ◊ The list `@all_sites` contains a simple list of all sites in the VO feed or, if the **`--site`** parameter was passed on the command line, the single site that we're configuring Nagios for. This is only used by `template_host.groups.erb` file to create the Nagios `host.groups.cfg` file.
        - ◊ The list `@all_flavours` contains all service flavours for the site(s) we're interested in. It is used to create hostgroups and servicegroups (ie by `template_host.groups.erb` and `template_service.groups.erb`).
        - ◊ `@atp_hash` is a nested hash. Each top-level key is a site name, which points to a sub-hash containing mappings between hostnames and their corresponding service flavours. This is used by all `.erb` files in one way or another.
    2. Run `buildNagios()` which is responsible for reading in the configuration files provided by the ncg-metric-config and grid-monitoring-probes rpms. This function is also responsible for running the `.erb` template files to generate the Nagios config.
3. Once all config files have been generated, the temporary files containing XML are renamed, so that their md5 checksums can be calculated the next time `nagconf.rb` runs. This allows us to decide whether the POEM or VO feed data have changed since last execution.

*.erb files

These are Ruby template files. They're basically Ruby (with some addittional, ugly, markup) that is executed to produce (many) Nagios configuration stanzas. It's a very efficient, though not pretty, way of doing this.

*.templates.cfg files

These are template configuration files, used by Nagios (not `nagconf.rb`) and stolen from an existing WLCG Nagios installation. They should be included in our own RPM.

To do

- Hard-code the cms.conf etc locations, rather than use current directory
- Write to nagconf.new and then validate before restarting nagios - not sure if this is possible (nagios.conf would need to have nagios.new entry?) How about: move old config to nagconf.old; write new config in nagconf; if nagios -v succeeds great! else move nagconf -> broken and move nagconf.old -> nagconf...this should probably be handled outside of the nagconf.rb script, though..
- Recently implemented multiple VOs. This is probably going to result in duplicated configuration objects in the *.host.cfg files, but will likely only affect site-specific installations.
- Ability to force a re-generation of the config; this can currently be achieved by deleting the files `ncg_poem.tmp` and `ncg_vofeed*.tmp`; in fact, these files should probably also be renamed to `nagconf_poem` etc
- There are some cases where the same node is defined to exist at two sites (with the same service). This causes problems because we end up creating duplicate host definitions in Nagios (i.e. multiple SITENAME directories contain the host definition in their host.cfg files). Nagios does not allow this. The current workaround is to only add the host into a host.cfg file once (the first time it's seen). Whether this will result in an unstable nagios config is unclear at the time of writing. Either way, we need to fix this in the long term. Oh, this also means we may end up with some empty host.cfg files (such as for CERN-PROD-AI, CERN-PROD-HLT and CERN-PROD).
- At the moment we have to manually clear out /etc/nagios/nagconf before rerunning nagconf.rb if we want to switch between "host-only checking" and the full, WLCG sites checking. This could be incorporated into the script, or puppet.

This topic: LCG > NagConf
Topic revision: r7 - 2013-10-29 - MikeKenyon