Nagios Messaging with ActiveMQ

Author : Julien Perrochet
Supervisor : James Casey
September 2009
Version 2.2
Distribution: Public

Abstract

This work is about service availability monitoring with Nagios and messaging with ActiveMQ in the context of WLCG.

Nagios needs to communicate with other nodes on the network. Not only because it is testing hosts and services, but also because it has to send and receive information about services states, changes and configurations. This information is handled via messages traveling through the messaging system.

The aim of this work is to replace the old local file-based messaging system with a local ActiveMQ message broker instance and to directly bridge this installation with the existing messaging network while conserving the easy way to customize message types with the help of message-handlers definitions.

Moving to the new messaging scheme will homogenize the architecture and centralize a part of the messaging configuration into the broker.

Introduction

This document will explain the practical changes in the infrastructure of nodes hosting an instance of Nagios caused by the project. The full project (see original description) could not be completed, as this required some much deeper knowledge of Nagios. One step has however been successfully accomplished.

This document can also serve as a basic documentation about the way Nagios communicates with the rest of the network via messaging for both the old and the new way, and how to install such a new infrastructure.

Part of my work also consisted in creating a Twiki page. It may still evolve and holds other informations related to the project, so I recommend reading it if you are somehow involved in Nagios and ActiveMQ.

Below you can find the description of the project as it was submitted to me.

1 Project description

Integration of Enterprise Messaging into an open-source monitoring system - Nagios

> *EGEE and WLCG are moving to using Nagios to measure reliability and availability of WLCG sites : https://twiki.cern.ch/twiki/bin/view/EGEE/OAT_EGEE_III. While messaging is currently used to communicate between Nagios instances on different sites, custom protocols and add-ons are used for components inside Nagios to communicate internally- We also have a complex queuing mechanism implemented to insulate the Nagios service from remote messaging clients.*
>
> *We propose to :*
>
> *1. Investigate messaging to perform this internal communication. Some similar work has been proposed but not yet carried out within the nagios community : http://community.nagios.org/wiki/index.php/Nagios_AMQP. We would build on these ideas and compare the performance and characteristics of a Nagios built on messaging against current standard implementation.*
>
> *2. Replace the custom queuing code on the Nagios server with a local ActiveMQ instance, and bridge this ActiveMQ to the global network of brokers.*

The first proposal could not be accomplished for the reason that it would imply a very deep knowledge of how Nagios works and is implemented. Furthermore, I also lack the programming skills and time required for such a deep inspection and modification.

The second proposal could however be fulfilled, and this document holds the information about the work that has been achieved.

2 Nagios' Messaging Needs

In order to work properly with the rest of WLCG monitoring, Nagios has to communicate with other nodes on the network. The following subjects are treated over messaging :

- **Service Checks :** Information about service checks results. These results have to be sent into the messaging system to be consumed by other Nagios instances and any client willing to get information about service checks, such as virtual organizations (VO) specific display tools.
- **Service Notifications :** Some changes in services availability need to be reported to other Nagios instances and to any client willing to be warned when such changes occur, such as monitoring dashboards or ticketing systems.
- **Configuration changes :** Notifications about Nagios configuration changes are sent via messaging.

This list is however not definitive, the messaging system could also be used for other purposes, for example to send comments to a Nagios instance. Both the current and the new messaging scheme allow to add such features easily by using specific message handlers. See corresponding section for more details.

3 Current Messaging Scheme

Outgoing messages

Current typical Nagios instances do the following when a message has to be sent : when a check has been accomplished, another script is ran, which will assemble the information it was given to in a message and drop this message in a specific directory that is serving as a message queue. Then, Nagios will regularly trigger another script that will be in charge of sending pending messages out to the messaging network.

Incoming messages

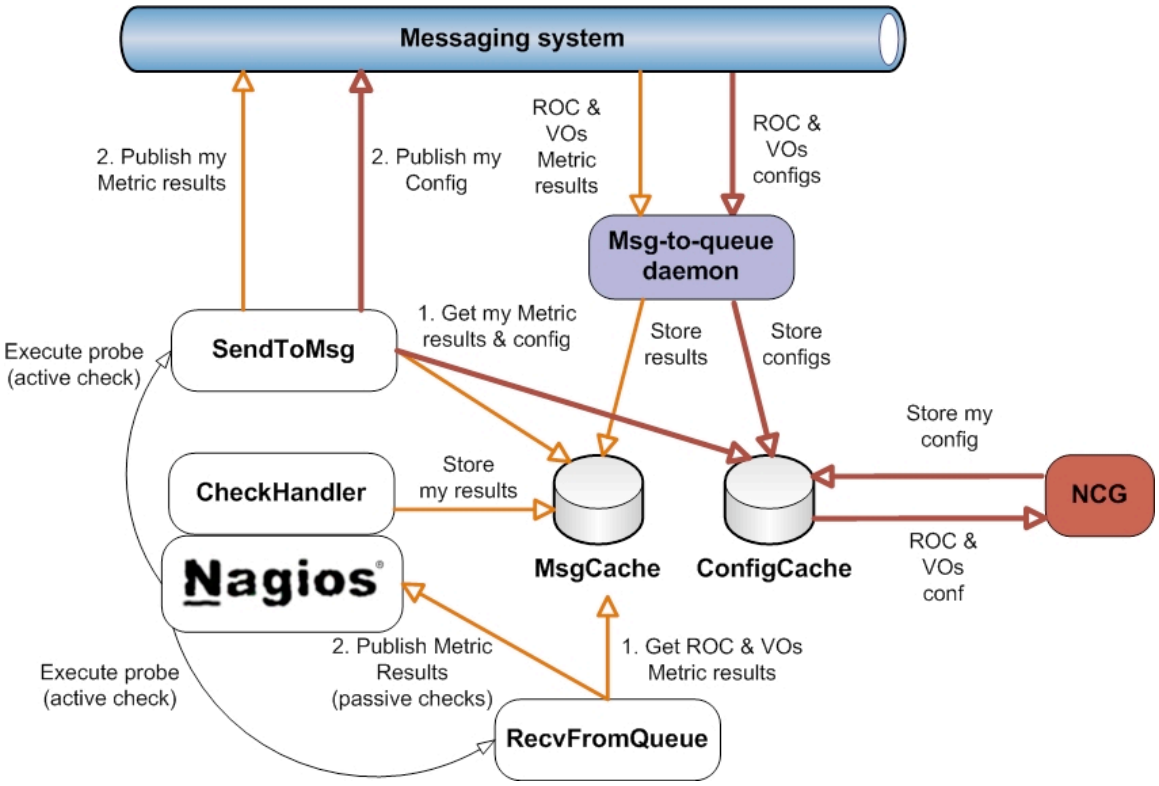
Besides Nagios, there is a daemon running that will continually import check and notification messages to the directory queue. A script will then regularly be triggered to import those pending messages as passive checks into Nagios.

Configuration messages

Verification for new configurations are actually only triggered by a Nagios check, but the configurations themselves are not generated by Nagios. On each run, NCG[1] will put its data into a database and a Nagios script will then send out this data. The daemon messaging script will handle incoming configuration messages and drop them into the local database.

[1] Nagios Configuration Generator : used to generate nagios configurations at installs and updates.

Getting the picture[2]



General scheme of the current architecture.

Script details

- `handle_service_check` is called to send out information relative to a specific check. It creates a message from the check data and drops it into the directory queue. `IPC::Dirqueue` is used to manage the directory queue.
- `handle_service_change` does the same as `handle_service_check` but for notification messages.
- `check_config` checks if the configuration database contains new or updated configuration confirmation. If so, it raises a WARNING.
- `recv_from_queue` imports messages from the directory queue into Nagios as passive checks.
- `send_to_msg` checks if some outgoing messages where cached in the directory queue and sends them to the messaging system.
- `msg_to_queue` is a daemon listening on specific queues from the messaging system. When messages are received, they are added to the directory queue.

[2] Picture kindly provided by Emir Imamagic.

4 New Messaging Scheme

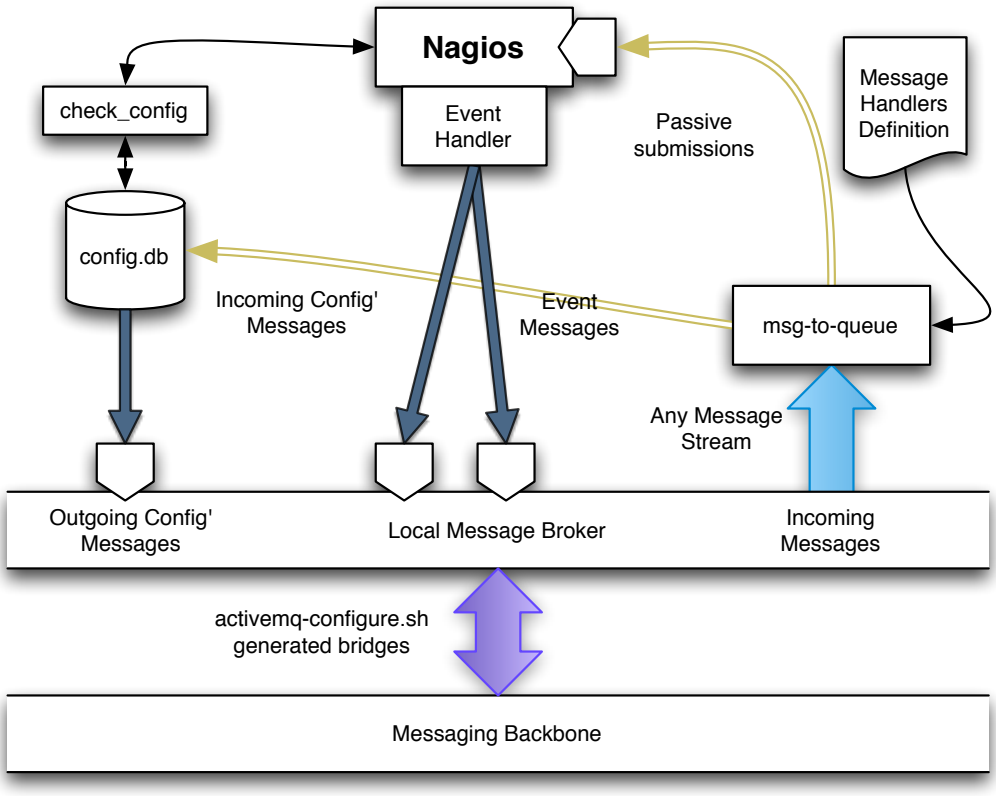
Changes

As the project proposal states, the new scheme replaces the custom queuing structure by an ActiveMQ message broker. Each interaction is directly done with the local message broker, except when it is unavailable ; in which case outgoing messages will be cached on the disk until the local broker is available again.

This new architecture is simpler as it homogenizes the messaging infrastructure and greatly improves end-to-end latency from several minutes down to milliseconds.

Basically, the `msg-to-queue` daemon has been modified to interact with the local broker and to directly write to Nagios' command pipe, while the `recv_from_queue` script was removed and the other scripts adapted.

Getting the picture



Detailed modifications

Scripts

To allow deploying of the new system in parallel with the current one through the same package and to avoid confusion, the new scripts have been renamed. They therefore have a *modified to* <new_name> following them in the description.

- `handle_service_check` *modified to* `handle_service_check_amq` - Now directly sends the MetricOutput to the local message broker (or any specified broker). Messages will be stored on disk if the broker is unavailable via the `IPC::Dirqueue` Perl module.
- `handle_service_change` *modified to* `handle_service_change_amq` - Now directly sends the Notifications to the local message broker. Messages will be stored on disk if the broker is unavailable.
- `check_config` *modified to* `check_config_amq` - Does the following :
  - Still checks if the local NCG SQLite database holds new or updated configurations. If so, it raises a warning.
  - Now checks if the local NCG SQLite database holds configurations to export and sends them to the local message broker.
- `recv_from_queue` Has been removed as modifications on the `msg-to-queue` daemon made it obsolete.
- `send_to_msg` *modified to* `flush_dirqueue_`msg - Checks if some outgoing messages where cached on the disk and sends them to the local message broker. Messages are directly cached on the disk only if the local broker is unavailable.
- `msg-to-queue` *modified to* `msg-to-handler` - Has been modified to listen to queues on the local broker. Calls to corresponding message handlers still work in the same way as before.
- `test_local_broker` New script that sends a message on a queue and tries to receive it. Used to check the local broker capacities.

Nagios Services

- `org.egee.CheckLocalBrokerMessaging` - *new* - Checks if the broker can receive and send a message to/from a queue by calling `test_local_broker`.
- `org.egee.RecvFromQueue` - *removed* - This service triggered the `recv_from_queue` script and is therefore useless in the new architecture.

5 Message Handlers

The actual messaging scheme uses so called *Message Handlers* to process incoming messages. When the `msg-to-queue` or the `msg-to-handler` daemon are started, they load a Message Handlers definition file (`/etc/msg-to-queue/msg-to-queue.conf`) which specifies queues to listen to and a specific handler for each of these queues. When a message is received, it is passed to the corresponding handler and processed by a custom method defined within the message handler.

The new architecture only requires little or no modification of the existing message handlers : only `MetricOuput.pm` had to be modified. This handler has also been renamed to `MetricOutputNagios.pm` . Furthermore, the definition file doesn't need any modification and can be used in the same way as before.

If new handlers have to be added, the procedure is exactly the same as before.

6 Setup a new Architecture

The described steps must be followed if you don't have access to a *bundle* install (via YUM or YAIM for example) and have to modify an existing Nagios installation. These steps are quite straight-forward.

ActiveMQ Setup

ActiveMQ can be installed very easily with the help of YUM if you use it. Typing the following to the command line should be enough :

```
$ yum install fuse-message-broker
```

You should need the `egee-SA1` repository in your `/etc/yum.repos.d/` directory to be able to install AtiveMQ.

If you can't install ActiveMQ via YUM, you can also install it by directly downloading the FUSE package from the following address, as there are currently no special configurations coming with the YUM install :

http://fusesource.com/products/enterprise-activemq/

ActiveMQ Configuration

I coded a configuration script that should do the biggest part of the work. `activemq-configure.sh` will read the following files to generate the `/etc/activemq/activemq.xml` configuration file :

- `configs/activemq_raw_xml.xml` – The model for the ActiveMQ configuration file.
- `configs/activemq-configurator.cfg` – The variables that will be written to the configuration file. If defaults should be used, only make sure the backbone broker address is correct.
- `/etc/msg-to-queue/msg-to-queue.conf` – The message handlers definitions. This information is used to create the appropriate bridges with the backbone.

The `activemq-configure.sh` script should be run from the same directory it lives in so it can correctly access the other configuration files.

Also keep in mind that you have to restart the broker each time you want to make a configuration change effective. This is achieved with the following command :

```
$ service activemq restart
```

Nagios services

The updates concerning Nagios have to be done by hand. First, you will have to copy the new scripts to the Nagios script directory, normally `/usr/libexec/grid-monitoring/plugins/nagios/`.

Then, update the Nagios services so they use the new scripts. You will have to edit the files that live in `/etc/nagios/wlcg.d`. If you don't know how to do edit Nagios object definition files, take a look at the documentation at http://nagios.sourceforge.net/docs/3_0/ .

Concerning the `msg-to-queue` daemon, you will have to remove it and get the `msg-to-handler` one running.

And at last, add the new message handlers. They generally live in a directory looking like this, but their exact directory should be given with them : `/usr/lib/perl5/vendor_perl/5.8.5/GridMon/MsgHandler/<Message_Handler>.pm`

If you only change the scripts, Nagios doesn't need to be restarted. But if you modified the object definition files, you will have to.

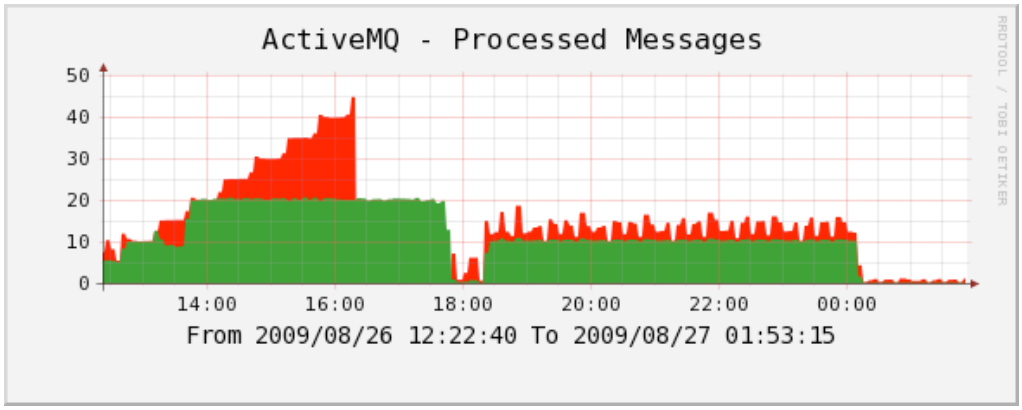
7 Performance

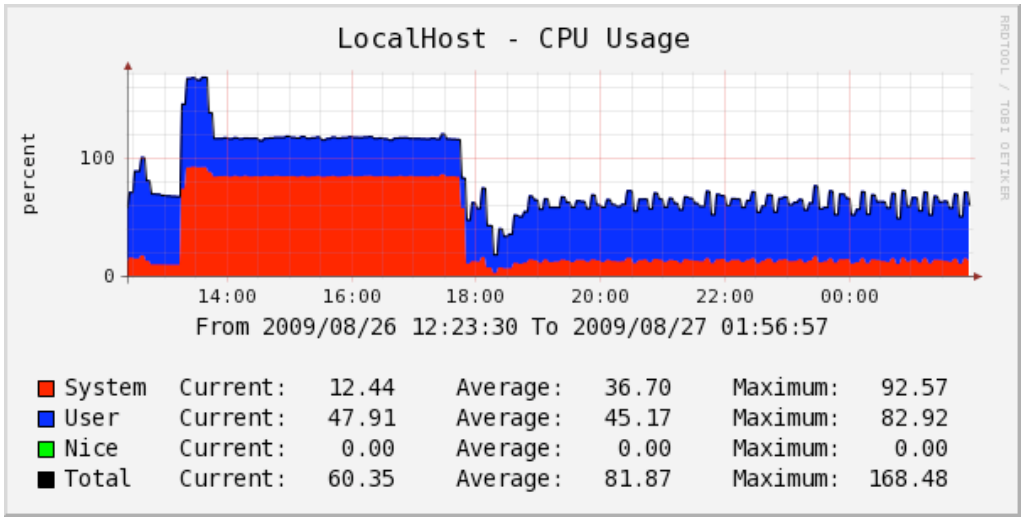
A stress test has been run against a machine hosting a new setup. This machine was an *Intel(R) Xeon(R) CPU E5410 @ 2.33GHz (2/8)* box running under Scientific Linux Cern 4.8 (name : lxbrf2711).
Up to 20 messages per second could be processed continuously. If the message rate goes higher, new messages simply queue up on the message broker.

The command pipe seems to represent the principal bottleneck, as each message has to be written to it. Messages that don't need any writing to pipe are processed much faster and allow rates higher than 1000 msg/sec.

Globally, testing shows that such a setup is able to handle relatively high constant message flows and correctly deal with consequent message bursts.

The two plots below show message enqueuing and dequeuing rates (red = enqueued, green = dequeued) and the corresponding CPU usage :





The plots describe two different phases : first, the incoming message rates is increasing and gets over the maximal consumption rate. This causes the queue to grow. When no more messages are enqueued, the dequeuing process can catch up. Then, a second message flow, this time limited to 10 msg/s, comes in and can be processed on-the-fly without notifiable CPU consumption overhead.

A possible hint to this high system CPU usage when the message rate gets over 20 msg/s might be several Nagios plugins and processes competing to write to the command pipe. This issue should be investigated if high message rates are to become common. It can however be left aside at the moment as typical message flows currently never even reach 1 msg/s.

8 Conclusion

A new functional architecture could be designed and has proven its usability. Furthermore, messages now directly go into the messaging system, homogenizing the global infrastructure and reducing overall end-to-end latency.

The objective of replacing the local file-based message queue with a local message broker while keeping the system flexible and performant has been achieved.

9 Comments

The main part of my work as an OpenLab student has been to get up to speed with the Nagios monitoring software and the basics of messaging with an ActiveMQ message broker. Afterwards, I also had to set up a little monitoring infrastructure with Cacti.

Real *coding* has only represented a very little part of my time : I spent the biggest part of it discussing the more abstract side of the work, learning about the tools, helping others getting in touch with them, to run the system and to test it.

Besides the many interesting tools and knowledge I learned about, it was the first time I participated in such a project and worked in an environment of this scale, and I am happy to see that I appreciated this too.

Those two months have been a very good experience and I really enjoyed my stay at CERN as an OpenLab student.

10 Links

Project-related

- SVN currently holding the new data : https://svnweb.cern.ch/cern/wsvn/nagActMQ
- Twiki page about the project : https://twiki.cern.ch/twiki/bin/view/DefaultWeb/WebHome?topic=LCG.UseLocalActiveMQForMessaging
- The messaging systems namespace : https://twiki.cern.ch/twiki/bin/view/EGEE/MsgNamespace
- JMX Query Tool : JMXQueryTool < LCG < TWiki
- Configuration Details : NagiosActiveMQConfiguration < LCG < TWiki

Bibliography

- Grid-monitoring SVN : http://www.sysadmin.hep.ac.uk/svn/grid-monitoring/
- Fuse : http://fusesource.com/products/enterprise-activemq/
- ActiveMQ Documentation : http://activemq.apache.org/
- Camel Documentation : http://camel.apache.org/
- Nagios Documentation : http://nagios.sourceforge.net/docs/3_0/
- Cacti - monitoring software : http://www.cacti.net/
- Jmx4Perl - get informations about JMX Mbeans on a server : http://search.cpan.org/~roland/jmx4perl/
- STOMP - simple messaging protocol : http://stomp.codehaus.org/
- Net-Stomp - Perl module implementing STOMP : http://search.cpan.org/~lbrocard/Net-Stomp-0.34/