

Health and performance monitoring of the online computer cluster of CMS

G Bauer⁶, U Behrens¹, O Bouffet², M Bowen², J Branson⁴, S Bukowiec²,
M Ciganek², S Cittolin⁴, J A Coarasa Perez², C Deldicque², M Dobson²,
A Dupont², S Erhan³, A Flossdorf⁴, D Gigi², F Glege², R Gomez-Reino², C Hartl²,
J Hegeman^{2a}, A Holzner⁴, Y L Hwang², L Masetti², F Meijers², E Meschi²,
R K Mommsen⁵, V O'Dell⁵, L Orsini², C Paus⁶, A Petrucci², M Pieri⁴, G Polese²,
A Racz², O Raginel⁶, H Sakulin², M Sani⁴, C Schwick², D Shpakov⁵, M Simon²,
A C Spataru² and K Sumorok⁶

¹ DESY, Hamburg, Germany

² CERN, Geneva, Switzerland

³ University of California, Los Angeles, Los Angeles, California, USA

⁴ University of California, San Diego, San Diego, California, USA

⁵ FNAL, Chicago, Illinois, USA

⁶ Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

^a Now at Princeton University

E-mail: Olivier.Raginel@cern.ch

Abstract. The CMS experiment at the LHC features over 2'500 devices that need constant monitoring in order to ensure proper data taking. The monitoring solution has been migrated from Nagios to Icinga, with several useful plugins. The motivations behind the migration and the selection of the plugins are discussed.

1. Introduction

The CMS experiment's [1][2] online cluster at the LHC [3] consists of 2'300 computers and around 200 switches or routers operating on a 24-hour basis. This large infrastructure must be monitored in a way that the administrators are pro-actively warned of any failures or degradation in the system, in order to avoid or minimize downtime of the system which can lead to loss of data taking. Each minute of colliding beam in the machine is precious, and therefore fast and reliable monitoring and alerting is crucial. Different groups working on diverse parts of the detector use most of these computers, and the administration and monitoring of all of them is the responsibility of the DAQ (Data AcQuisition) group.

The original monitoring solution, based on Nagios, was found to be unsatisfactory, and was replaced by Icinga, with several useful plugins. The motivations behind the migration and the selection of the plugins are discussed.

2. Issues and solutions

2.1. *Upgrading from a Nagios installation*

The overall experience with Nagios in CMS was satisfactory, but there were problems with scaling in the CMS growing cluster. The simplicity of its plugins and their extensibility are its key assets, however the difficult-to-maintain process of splitting the configuration in order to scale the monitoring with the growing infrastructure and the old standards used for the web interface forced a reconsideration of what to use to re-architecture the monitoring system.

While evaluating the different alternatives possible, mostly three products were considered. The latest version of Nagios [4] (version 3.2.3, because 3.3.1 had serious bugs), the latest version of Icinga [5] (1.3 at the time, now 1.7.1), and Shinken [6].

All these products are able to read the previous configuration with very little modifications, and they can reuse the existing plugins, making any migration easier.

The latest version of Nagios did not have the additional features or enhancements required, i.e. it still used a CGI web interface written in C, which needed a recompilation for each modification.

The first evaluated version of Icinga was not available in RPM format. An effort was put into packaging it for version 1.3.1, and giving it back to the community, with the benefit that nowadays there are available RPMs in the DAG [7] repository. The latest version of Icinga has an improved web interface following modern standards, which can be easily extended through widgets (called "cronks"), which are XML files. Icinga also offers JSON output for most of the pages (since version 1.4.1), which allows very easy integration with other tools. Icinga had also a very active community behind it.

Shinken looked promising, but seemed still in its beta stage at the time. Icinga was selected considering the points aforementioned.

2.2. *Scalability with respect to a growing cluster*

The configuration is stored in plain text files. There are modules allowing to store the configuration in a database, but those were not considered, because generating flat files or filling up a database is equivalent, so the database only increases the complexity.

It was difficult to maintain the process of changing the configuration during cluster growth. The load on each monitoring server is proportional to the number of checks and the number of hosts monitored, and once the maximum load on a single server was reached, the configuration had to be split across several servers. In order to show a unified view of the whole cluster, the monitoring information had to be aggregated onto a central dashboard, which added another layer of complexity. The splitting was a manual task and required both guessing as to how the cluster would grow and cautiousness in writing the configuration so that it could be re-used across servers.

Gearman provides a generic application framework for farming out tasks to other machines or processes that are better suited to do the work. It allows work to be executed in parallel, to balance the

processing load on several machines. It can be used in a variety of applications, from high-availability web sites to the transport of database replication events. It is the nervous system through which distributed processing communicates [8].

Icinga can leverage the power of Gearman by using a module, which is installed as a broker directly inside the main executable and farms out all checks needed to be run to the Gearman server and checks for results to be processed. This allows easy scaling; the central server is only a scheduler, there can be as many worker nodes as needed, and they each have the same configuration.

Icinga makes use of the multiple queues allowed by Gearman (see Fig. 1). Two queues hold the hosts checks and services checks respectively, consumed by the workers, injected by the scheduler. The next queue is consumed by the scheduler, injected by the workers and check_multi and holds the results of the checks. Finally, the scheduler fills one last queue, which holds the performance data to be processed, consumed by PNP4nagios. Each worker also has its own queue so one can also monitor that it is alive, using a special plugin called check_gearman.

Gearman quickly became the central piece for the scaling of our monitoring infrastructure, because of its inherent efficiency. It is very stable and reliable, and has proved to be a good choice.

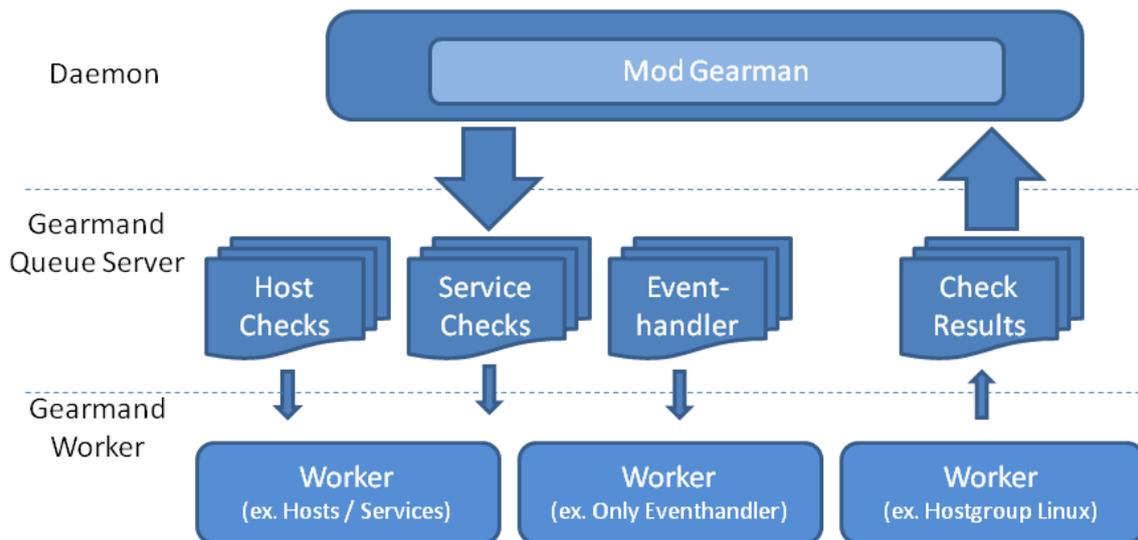


Figure 1 Scalability with respect to the number of checks

A standard check can either be run locally on the monitoring server or remotely through some mechanisms. The most common ones are SSH [9], NRPE [10] and SNMP [11]. SSH has more overhead and is potentially more secure. SNMP is light and fast but difficult to extend and complicated to setup to have good security (SNMPv3). NRPE (Nagios Remote Plugin Execution) is the best of both worlds: easy to setup, secure enough, fast and easy to extend. It has a drawback that required patching: the limit of the output on a 32-bit operating system was limited to 4096 characters.

Regardless of the mechanism to execute remote checks, for each check run from the monitoring server a connection to the remote server needs to be open. Therefore, the load on the monitoring server scales linearly with the number of checks done on a single machine, which was already problematic when having over 20 checks on all hosts.

To reduce this overhead, a plugin called check_multi [12] is used. It can run on the monitoring server one single remote check that can cascade on the remote monitored client and check several plugins in a row. Currently the CMS monitoring server runs two remote check_multi checks per host instead of 27 remote checks. The problem posed by this solution is that all results appear as one single service in Icinga. This causes issues because the alerting is done on a service, which contains multiple checks, some of which requiring alerting and some not. Furthermore a failure on one check causes the whole service to be critical, and will therefore hide any additional check failures in this service. To

remedy this, the tests are then split as passive ones, which allow fine-tuned notifications, and proper alerting.

In summary, for the standard 27 tests run on all the monitored hosts, two active tests are defined in Icinga, which make use of `check_multi`. These `check_multi` checks output XML data, which is piped into `send_multi` (part of `mod_gearman`), which then takes care of parsing and splitting the 27 results and sending them back through the Gearman queue. The master Icinga scheduler processes the 27 passive tests. The configuration is a little bit more complex than in plain mode, but `check_multi` provides tools for generating the passive checks from the local/remote `check_multi` configuration, reducing the effort required in configuration and maintenance.

As most of the checks take as long to run as the overhead of the NRPE connection, tunneling the 27 passive checks through `check_multi` reduces the load on the monitoring server roughly by a factor of 15 and cuts the time needed to run them all in half.

2.3. Configuration management

The CMS cluster is managed by a configuration management tool called Quattor [13]. To avoid the duplication of information, a perl script that extracts information out of Quattor and generates a configuration suitable for Icinga has been developed. The script also generates the network topology, in order to hide all failures located behind a failed switch or router.

Alerting is based on host groups, generated from a wiki page, and alerts can be sent either by email or by SMS. The wiki page only contains which server or quattor profile belong to which group, and who should be notified would anything happen to any machine within this group.

To extend services for each sub-detector, services just need to be put into groups, and the groups need be filled from the wiki page. This results in a flexible and clear, yet simple configuration management, and additions/suppressions can be very easily implemented.

2.4. Performances

The use of these performance enhancing modules and the work on grouping the checks has yielded impressive performance improvements over the previous Nagios infrastructure allowing for the monitoring of 65'000 checks run every 2 minutes on average, compared to 20'000 every 5 minutes on the previous system. It has also allowed the scaling down of the hardware requirements from four 8-core machines to one 4-core machine generating all the RRDs, and two 8-core machines, one running the Icinga server and the other running all the checks. Furthermore the design allows the easy growth of the infrastructure without the need to rethink the monitoring system as a whole.

2.5. PNP4nagios

Most of the executed checks generate performance data, which are simple metrics used to monitor the health of a machine and its variation. For example the checks for the disk occupancy, the memory usage, the ping response time, and other such variables generate performance data. They are very useful to graph, in order to see trends and correlations in the data. PNP4nagios [14] graphs any plugin's performance data that follows the Nagios plugin development guidelines [15]. This also allows developing custom plugins and having the results graphed.

PNP4nagios can also merge plots, in order to show the metrics from several hosts on a single graph. One example for this is the monitoring of the temperature of the hosts (see Fig. 2). Local checks are in place to ensure hosts don't overheat, and turn themselves off in case they reach some thresholds. It is useful to be able to correlate the temperature within a building, a room or a rack, to pinpoint a cooling issue for example, or simply to observe the heat impact of data taking.

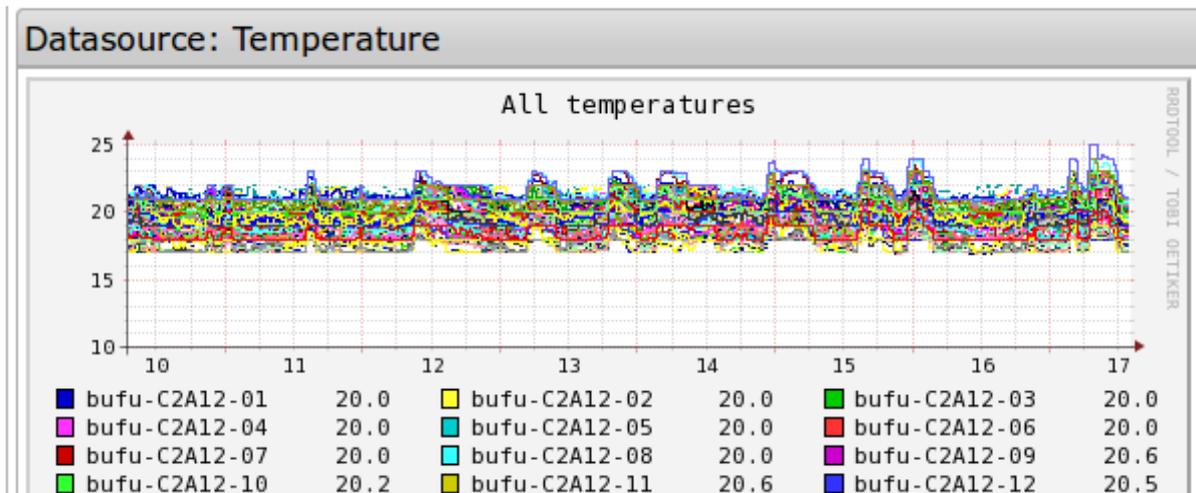


Figure 2 Overall temperature display of 1028 computers from the High Level Trigger farm over a week

PNP4nagios also allows the aggregation of plots. As all the data are stored in RRDs [16], it is easy to collect metrics from several hosts/checks, and graph them together, either merged like in Fig. 1, or stacked like in Fig. 3. For example, in CMS the Storage Manager machines, which are used to store the data selected by the High Level Trigger [17] on disk before shipping it to CERN's Tier-0 [18] for storage and processing, have four network interfaces that can all receive data during taking data. In order to graph the total bandwidth usage, all four network cards for all 16 storage managers need to be stacked, which is very easy to do with PNP4nagios (see Fig. 3).

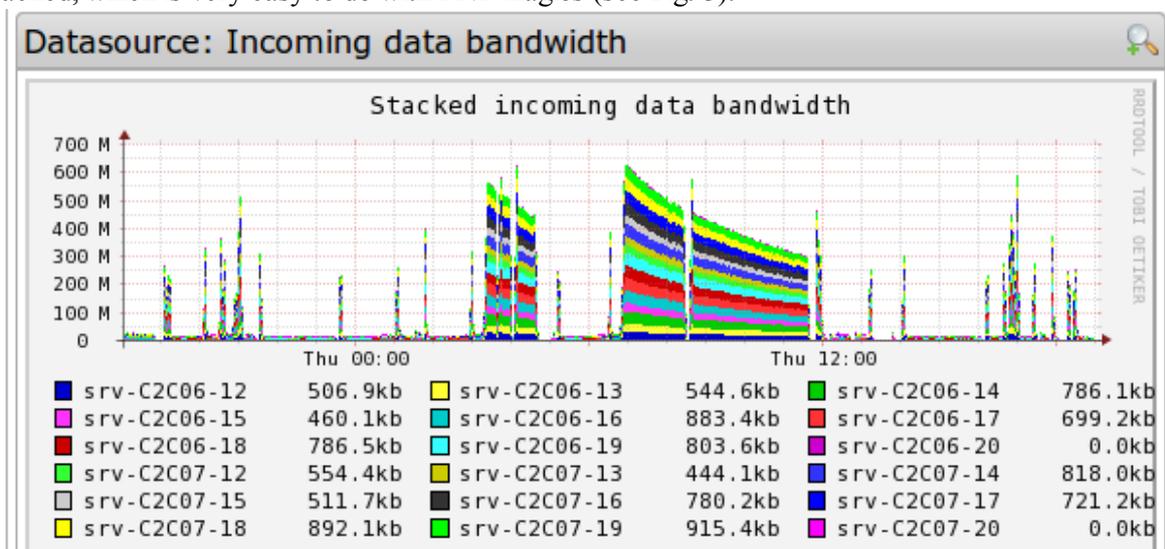


Figure 1 Stacked incoming data bandwidth on all 16 Storage Managers over 24h

Thanks to caching (see next section), the load on the disks isn't too high. The only issue that needed tuning was that PNP4nagios writes a summary file in XML for each service check, that describes the metrics available in the service. Because of the number of checks with metrics for each host (~10) and the number of hosts (~2300), the default refresh time of 5 minutes was too short to handle the graphing of all the metrics of the entire cluster. Indeed, that would result in writing over 70 files per second. Therefore the frequency of writing these XML files to disk has been decreased to once per day, which means one file written every 4 seconds, a compromise against an acceptable delay when new checks are added.

2.6. RRDcached

PNP4nagios generates many RRD files, which needs a lot of I/O to disk. In order to leverage the load, RRDcached is used. It simply caches all updates to a RRD file. Changes are synchronized to disk only every 5 minutes by default. The RRDcached daemon has a feature that the client can use to tell it to force the file to be synchronized to disk right away. This feature can be used in order to serve the latest data. As trends are more used than live statistics, this feature is not used. RRDcached is part of RRDtool since version 1.4.

2.7. Integration with external tools

Some sub-detectors need to integrate Icinga's results within their own monitoring architecture, in order to show a coherent overall status. This is done using JSON [19] standard outputs (available from version 1.4.1 in Icinga). Most of the monitoring pages in our infrastructure can output JSON data, which can then easily be parsed and displayed inside another application.

3. Conclusion

The new Icinga based monitoring system is able to monitor over 2'300 computers and 200 switches, with a granularity of two minutes. Most of the configuration for the hosts and switches are generated from Quattor, getting the hosts and groups from Quattor profiles. On all those hosts, 27 standard checks are run through check_multi, with additional checks added on a host-by-host basis depending on its functionality.

Although the decision to migrate from Nagios to Icinga was mostly motivated by the philosophy behind the Icinga project, and its newer interface and wider feature list, it has exposed many very useful plugins used in the new system. This experience has been presented to other groups at CERN. Icinga, along with some of the plugins presented here, have been tested and are being used by other CERN experiments [20].

The use of these performance enhancing modules and the work on grouping the checks has yielded impressive performance improvements. This has allowed all sub-detectors to reliably monitor their production critical machines, and some have even implemented their own checks.

The system is fully commissioned and its design allows the easy growth of the infrastructure, providing enough scalability to survive well beyond the upcoming long shutdown.

4. Acknowledgement

This work was supported in part by the DOE and NSF (USA) and the Marie Curie Program.

References

- [1] The CMS Collaboration, *The Compact Muon Solenoid Technical Proposal*, CERN LHCC 94-38, 1994.
- [2] The CMS Collaboration (Adolphi R et al.) *The CMS Experiment at CERN LHC*, JINST 3 S08004 361, 2008.
- [3] Lyndon Evans and Philip Bryant (eds) (2008). *LHC Machine*. Journal of Instrumentation JInst 3 S8001E
- [4] Nagios Is The Industry Standard In Infrastructure Monitoring, Nagios.org, <http://nagios.org>
- [5] Icinga takes open source monitoring to the next level, icinga, <http://icinga.org>
- [6] Shinken, The next Industry Standard in IT Monitoring, <http://shinken-monitoring.org>
- [7] DAG, RPM packages for Red Hat, RHEL, CentOS and Fedora, <http://dag.wieers.com/rpm>
- [8] Gearman provides a generic application framework to farm out work to other machines or processes that are better suited to do the work, <http://gearman.org/#introduction>
- [9] The Secure Shell (SSH) Authentication Protocol, <https://tools.ietf.org/html/rfc4252>
- [10] NRPE, Nagios Remote Plugin Executor, <http://docs.icinga.org/1.6/en/nrpe.html>

- [11] A Simple Network Management Protocol (SNMP), <https://tools.ietf.org/html/rfc1157>
- [12] check_multi is a multipurpose wrapper plugin which takes benefit of the Nagios 3.x capability to display multiple lines of plugin output, Matthias Flacke, http://my-plugin.de/wiki/projects/check_multi/discussion
- [13] Quattor is a system administration toolkit, <https://trac.lal.in2p3.fr/Quattor/wiki/Web>
- [14] PNP is an addon to Nagios which analyzes performance data provided by plugins and stores them automatically into RRD-databases, <http://docs.pnp4nagios.org/pnp-0.6/start>
- [15] Nagios Plug-in Development Guidelines, Nagios Plugins Team, <http://nagiosplug.sourceforge.net/developer-guidelines.htm#AEN201>
- [16] RRDtool is the OpenSource industry standard, high performance data logging and graphing system for time series data, Tobias Oetiker, OETIKER+PARTNER AG, <http://oss.oetiker.ch/rrdtool>
- [17] The CMS High Level Trigger System: Experience and Future Development, Sparatu & all, *International Conference on Computing in High Energy and Nuclear Physics (CHEP), NY, May 2012, Proceedings CHEP2012212*
- [18] CMS Tier-0: Preparing for the future, Dirk Hufnagel, *International Conference on Computing in High Energy and Nuclear Physics (CHEP), NY, May 2012, Proceedings CHEP2012309*
- [19] JSON (JavaScript Object Notation) is a lightweight data-interchange format., <http://json.org>
- [20] Tools and strategies to monitor the ATLAS online computing farm, Scannicchio & all, *International Conference on Computing in High Energy and Nuclear Physics (CHEP), NY, May 2012, Proceedings CHEP2012348*