EVOLUTION OF THE MONITORING IN THE LHCB ONLINE SYSTEM

C. Haen*, E. Bonaccorsi, N. Neufeld, CERN, Geneva, Switzerland

Abstract

The LHCb online system relies on a large and heterogeneous I.T. infrastructure: it comprises more than 2000 servers and embedded systems and more than 200 network devices. The low level monitoring of the equipment was originally done with Nagios. In 2011, we replaced the single Nagios instance with a distributed Icinga setup presented at ICALEPCS 2011. This paper will present with more hindsight the improvements we observed, as well as problems encountered. Finally, we will describe some of our prospects for the future after the Long Shutdown period.

INTRODUCTION

LHCb [1] is one of the four large experiments at the Large Hadron Collider at CERN. This experiment relies on a large computing infrastructure to control the Data Acquisition system and the detector, as well as to manage the data it produces. In addition to the supervision of PLCs and readout boards done by the SCADA system WinCC, a lower level monitoring is needed at the system level. Until 2011, a single instance of Nagios [2], an industry standard monitoring tool, was used to accomplish this task. It was replaced by a distributed Icinga setup [3], an open source fork of Nagios, primarily to increase the performance of our monitoring infrastructure. After two years of usage, we have a much clearer view of the pros and cons of such a solution, and the long shutdown period of the LHC is a good opportunity to look at potential alternatives.

CURRENT INFRASTRUCTURE

A full description of our current setup is visible in [4].

Nagios, Icinga and many other tools, monitor hosts (servers, switches, etc) and services (software, resources, etc) by periodically executing light weight programs called plugins. A plugin will return the status of the resource (like 'OK' or 'CRITICAL') and eventually extra information (e.g. error message, performance data). Typically, the amount of checks to be executed is in the order of tens of thousands in a few minute interval. This explains that the monitoring of large environments, like the LHCb one, cannot be achieved by a single instance of the monitoring software, without reaching huge latencies. The latency represents the difference of time between the scheduled execution time of the check, and the actual execution time.

For large environments the execution of the checks requires to be parallelized and distributed. The solution applied in LHCb is to have a central Icinga instance delegate the check execution to workers. This is achieved using the plugin functionality of Icinga: the mod_gearman plugin [5] intercepts the checks scheduled by Icinga and places them into queues. Remote servers, called workers, running a gearman client program fetch instructions from the server queues, execute them, and put the result back in a result queue. This result queue is treated by the Icinga instance as if it executed the checks itself. This setup requires the workers to have the plugins executed by Icinga available locally. At LHCb, the Icinga server distributes its plugins with the workers using an NFS share.

By default, Icinga stores the check results in flat files. They are presented to the user using a CGI based web interface which parses these files. The id2db plugin allows to dump the monitoring data into a database. An advanced and flexible web interface makes use of the database and even offers a REST API. We are using a MySQL database running on the same server as the Icinga instance.

Users are alerted about problems after they have been tested faulty a configurable amount of time. The notification is made using external plugins, as for the checks. The most common one is of course the 'mail' command. To avoid receiving too many emails, we use Nand[6], which aggregates emails for ten minutes before sending them all in only one email. Critical alerts bypass this buffering.

FEEDBACK

The setup previously described has been running since two years in the LHCb online environment. Although the general setup remain the same, it was several times slightly changed. In overall terms, we are happy with this configuration, but a two year experience allows us to spot weaknesses in it.

Positive Aspects

The very positive aspect of this setup is its performance. We are running about 40 000 checks in a 5 minute window with absolutely no latency. This performance is made possible thanks to mod_gearman, which allows us to balance the check executions over 60 nodes. The server running the Icinga instance (8 cores @ 2.50 GHz, 16 Gb memory) could not achieve this in standalone mode.

A nice aspect of mod_gearman is the ease of adding new servers to the pool of workers. It requires very few packages on the worker side, and no configuration at all on the server side. It provides a very flexible way to increase the performance or the redundancy of the check execution.

An intrinsic characteristic of Icinga we are heavily using in LHCb is the group and inheritance functionalities of the configuration format. It allows to factorize objects — like

* christophe.haen@cern.ch

ISBN 978-3-95450-139-7

hosts and services — definitions. Properly used, it simplifies greatly the configuration management. The schema put in place for LHCb is such that a set of simple scripts are enough to manipulate all hosts and services.

The PHP based web interface using the database is a great improvement compared to the CGI one. The user experience is better: faster researches, many filters, custom views, etc. From the administrator perspective, there are many improvements as well: advanced authentication, finer grain management of users and permissions, REST api, etc.

Nand is a good way to leverage the massive amount of emails that can be produced, and we certainly could not run without it.

Negative Aspects

The first major flaw is that the Icinga instance is a single point of failure in our monitoring system. While the number of workers is enough to handle several failures, a problem on the central server means a complete unavailability of our monitoring infrastructure. Redundancy or failover mechanisms were not included in the original design of Nagios/Icinga. Moreover, the plugin system does not allow to add such functionality. A hand-crafted solution is to have a second server setup the same way as the first one but in a "standby mode". The second server monitors the first one, and starts up in case of unavailability of the primary instance. This solution relies on Icinga's capabilities only. Another solution would be to use third party tool such as Pacemaker and Corosync to ensure the failover mechanism. None of these solutions is really satisfying because we would run into several problems, such as the synchronization of the database backend, the current status being lost, and the NFS share for the workers to be moved.

It is possible in the configuration to inform Icinga about the dependencies between hosts and services. Thanks to this, when several problems occur at the same time because of a common root cause, you get notified about the root cause only. System administrators greatly appreciate this functionality. However, the definition of dependencies is very tremendous, and several experts reported performance problems related to them.

A big problem which is encountered with Icinga is in case of a big failure in the environment. When a big portion of the infrastructure fails at the same time, Icinga needs a lot of time to detect all of it and the latency increase tremendously. There is also a knock-on effect with the latency. If it reaches a high threshold —around 300 s in our setup— it keeps increasing for ever.

One reproach that could be done to Icinga is that it is very static. Macros are available in the configuration to refer to various static information such as the host address or the email of the contact. But a check cannot inquire about runtime values such as its previous results or the output of other checks. Another illustration of the static behavior of Icinga is that the configuration cannot be changed at runtime – a restart or reload of the process is necessary. This might be a flaw in environments making heavy use of virtualization.

The critical aspect with having to restart Icinga to change the configuration is that the parsing of it can be very long. 60 000 services are parsed in about 25 seconds, while around 8 minutes are required for 200 000 services. The loading of the configuration in the database backend also takes several minutes to be completed, making the monitoring system blind for this time. Fortunately, the very latest version of Icinga (1.9) tackles the loading issue by changing the implementation, and brings the delay in the order of seconds.

ALTERNATIVES

Our investigations for the evolution of the LHCb Online monitoring setup focused on three options: Nagios4, Icinga2, Shinken.

Nagios4

Nagios4, presented as a major release but still in beta version at the time of writing, consists mainly in Core's performance improvements. Hot spots have been identified in the design and better algorithms are being implemented — e.g. for the event queue insertion or the macro resolution. The philosophy of the Nagios developers is that a heavily used plugin should be part of the Core. The mod_gearman worker being one of those plugins, Nagios now has worker processes, but still running on the same server. The configuration parsing performance has also been improved.

Thanks to these changes, the developers claim a decrease of 87% in disk IO operations, -42% of CPU utilization and -64% of memory consumption, which leads to six times more checks executed.

The configuration logic changed slightly. If it does not bring major variations, existing configurations might silently produce different output than with previous version. Users will have to be extremely careful to this.

Shinken

Shinken [7] is the pioneer of the next generation tools. It keeps the principles of the Nagios-like tools — services applied to hosts — but is a complete Python rewrite in which the original logic is extend and the technical design totally different (Fig. 1). The code is split in modules, each of them with a specific role, and each module is a different daemon. There can be as many of each daemon as the user wishes and spread on different servers. This naturally provides horizontal and vertical scalability and redundancy. Examples of daemon are the Arbiter (reads ands shares the configuration between Schedulers, manages the high availability), the Scheduler (schedules the execution of checks, makes correlation) or the Poller (actively executes the checks).

The developers praise the dynamism of the solution since it makes use of Python's flexibility. The user can easily define services whose status are based on other services' status, using boolean operators or mathematical functions.

ISBN 978-3-95450-139-7

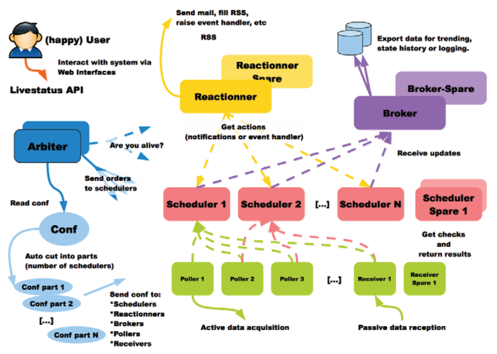
(happy) User
Interact with system via Web Interfaces
Livestatus API
Send mail, fill RSS, raise event handler, etc
RSS
Reactionner Spare
Reactionner
Export data for trending, state history or logging.
Broker-Spare
Broker
Arbiter
Are you alive?
Get actions (notifications or event handler)
Receive updates
Send orders to schedulers
Read conf
Scheduler 1
Scheduler 2
[...]
Scheduler N
Scheduler Spare 1
Conf
Get checks and return results
Auto cut into parts (number of schedulers)
Poller 1
Poller 2
Poller 3
[...]
Receiver 1
Receiver Spare 1
Conf part 1
Conf part 2
[...]
Conf part N
Send conf to: *Schedulers *Reactionners *Brokers *Pollers *Receivers
Active data acquisition
Passive data reception

Figure 1: Shinken architecture.

It also supports clusters with spare parts. Shinken has a native support for virtualization: it can make use of VMware tools or libvirt in order to automatically generate dependency rules between the virtual machines (VM) and the hosting servers. These rules are automatically updated if the VM is moved.

Shinken can directly use the configuration of Nagios. However, the logic was extended and new functionalities added for sake of factorization. For example, contrary to Nagios, Shinken can apply services to host templates, which avoids the definitions of groups. Another innovation is the possibility to apply services on host templates using complex boolean expressions. By considering templates as a flat set (e.g. 'linux', 'MySQL', 'Apache') instead of a hierarchical organization (e.g. 'linuxMySQL', 'linuxApache', 'linuxApacheMySQL', etc) , it reduces the amount of services to be defined when hosts can have several roles. Other noticeable facts for the configuration is a 'foreach' macro to duplicate objects and a flexible discovery system to generate configuration based on user's rules and plugins.

Another big difference between Shinken and its ancestors is that Shinken is firmly business oriented. The whole software is meant to provide different features (tools, views, notifications, etc) to the administrators — who are more interested in the source of problems — than to the managers — who focus on the impacts of problems.

Icinga2

At the time of writing, Icinga2 [8] is still in its very early stages — some features mentioned here are not yet implemented and are still subject to changes. The shortcomings of Icinga we described in the previous section were experienced by many users. Fixing these issues would involve changes which would break the compatibility with Nagios and the previous versions. For this reason, the developers decided to go for a parallel development branch, with a completely new code and approach. Regarding the actual implementation, they made the choice of C++ with heavy use of Boost libraries (but no C++11 features). Icinga will be compatible with Windows.

The first major change for users is that Icinga2 abandons the single core method. Very similarly to Shinken, Icinga2 will be implemented as components which can be loaded in various instances set up as a cluster (Fig 2).

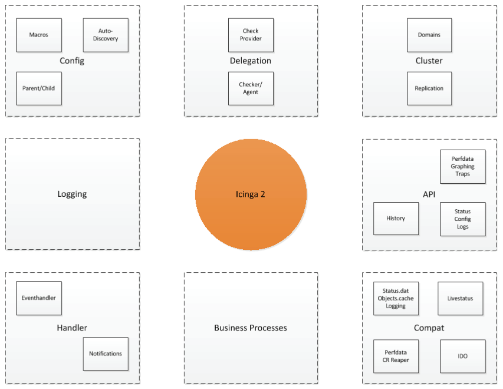

Figure 2: Icinga2 architecture.

The configuration is an entirely different approach. Not only has the grammar changed a lot — a conversion script is available — but the philosophy of it as well. For example, only services can be checked: a command is not anymore associated to a host to know its state, but the host takes the status of an associated service. Checks themselves will have a different interface, even though the Nagios style plugin will remain compatible.

Icinga2 will have an agent running on remote machines to replace plugins such as NRPE [9]. The reason is security, which is only an afterthought in NRPE and likes.

A standardize interface to interact with Icinga2 will be developed. It should provide an easy access to the internal state, as an input or an output. Plans include new database backend, configuration or creation of objects at runtime, auto-discovery of objects and real-time export of events (e.g. performance data).

Icinga2 will, as Shinken, be more business oriented and will propose correlation of status using business rules, that can be used for dependencies for example.

BENCHMARK

While the comparison is currently a bit unfair — Icinga and Shinken are production solutions, Nagios4 is a beta version and Icinga2 is at the beginning of its development — the purpose of this benchmark was to see the raw performances of these solutions. We are comparing our present solution (Icinga and Mod gearman with fine tuning, referred as icinga_gearman) with a tuned setup of Shinken, and with Nagios4 and Icinga2, both taken out of the box. The gearman workers were spread over 15 servers but the three other solutions were running on a single server. The benchmark was meant to see how were the solutions performing under normal circumstances and under heavy load. 60 000 dummy services were defined on 2000 hosts. The 2000 hosts are in OK state all along. At the startup, the 60 000 services are in OK state. After 1000 seconds, 90% of the services fail. They all recover at once after another 1000 seconds. The programs start with no historical data.

Figure 3 compares the average latency of each solution. The knock-on problem of Icinga mentioned earlier is clearly visible. The latency of Nagios4 is null all along, and so is the one of Icinga2 except at the startup. Shinken also seems to suffer from big environment failures.

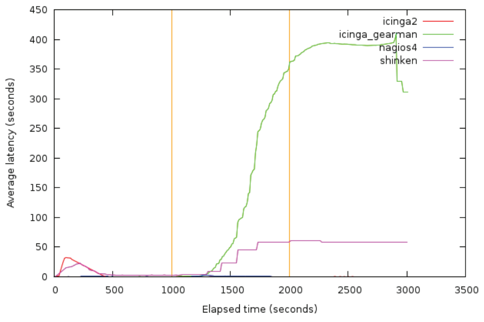

Figure 3: Latency.

Figure 4 compares the total amount of service checks executed since the startup of the program. Icinga2 clearly leads in this domain, and the high latency it has at the startup could be explained by the steepness of its curve during the first instants. After the big failure, the amount of checks executed by Shinken increases in a step manner. It is difficult to know whether the information is calculated and published less frequently or if the check execution really follows that curve.

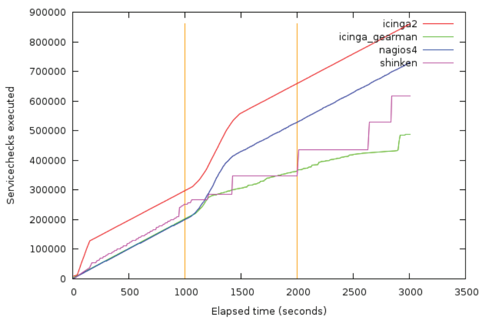

Figure 4: Servicechecks.

Figure 5 illustrates the speed at which each solution knows the real state of all services. The curves should reach 60 000 between 0 and 1000 seconds and between 2000 and 3000 seconds. They should drop at 6000 between 1000 and 2000 seconds. Icinga2 proves again to be the fastest. Nagios4's curve sticks to the Icinga2 one, except for the startup checks. Shinken seems to be the slowest, but the same incertitude as before remains.

This benchmark clearly shows the bottle neck of Icinga, even when finely tuned and used with mod_gearman. Even though they are not completely mature yet, Nagios4 and Icinga2 seem to overcome these limitations. On this benchmark, Shinken does not seem to perform so well. This might be explained by the usage of Python compared to the C++. Further tests should be done to verify if good performance can be achieved by spreading and multiplying the components over the infrastructure.

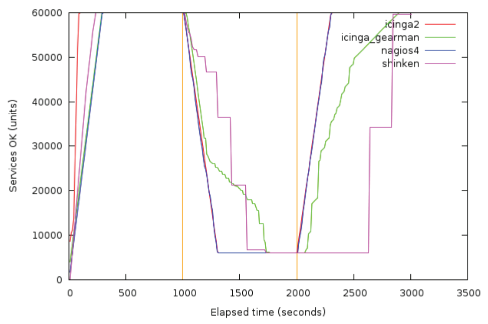

Figure 5: Reaction time.

The performance is only one aspect, and the amount of tools and features available for each tool should be compared before deciding for one or the other. Because Nagios4 remains a single core software with no possibility to distribute and balance the task on several servers, we do not consider it as a viable alternative. Icinga2 seems the most promising solution in terms of performance, but one has to wait for a production version to see what it will be capable of and what functionalities will be offered. This test does not exclude Shinken as a potential solution, it just shows that it cannot be used on a single server. The growing community, the flexibility, the numerous functionalities and the good user usability are as many arguments in its favor.

CONCLUSION

After two years of daily usage we could find some flaws in our monitoring system, both from the administrator's and the user's points of view. We have identified two potential solutions for future replacement: Icinga2 and Shinken. The first one, still in its early stages of development, seems very promising performance wise. It still has to be seen what will be the user experience with it. Shinken, already available in a stable version, seems not to perform as well but the usability and the amount of features are really good. We will reconsider these solutions when Icinga2 will be available for production, normally in the coming months.

REFERENCES

[1] A. Augusto Alves et al. The LHCb Detector at the LHC. JINST, 3:S08005, 2008.

[2] http://www.nagios.org/about

[3] https://www.icinga.org/about/

[4] C. Haen, E. Bonaccorsi, N. Neufeld "Distributed monitoring system based on Icinga" WEPMU035 ICALEPCS 2011

[5] http://labs.consol.de/lang/de/Nagios/mod-gearman/

[6] https://www.monitoringexchange.org/inventory/Utilities/AddOn-Projects/Notifications/NAN—Nagios-Notification-Daemon

[7] http://www.shinken-monitoring.org/

[8] https://www.icinga.org/about/icinga2/

[9] http://nagios.sourceforge.net/docs/nrpe/NRPE.pdf