ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA085

CERN CONTROLS OPEN SOURCE MONITORING SYSTEM

F. Locci[*], F. Ehm, L. Gallerani, J. Lauener, J. Palluel, R. Voirin
CERN, Geneva, Switzerland

Abstract

The CERN accelerator controls infrastructure spans several thousands of computers and devices used for Accelerator control and data acquisition. In 2009, a fully in-house, CERN-specific solution was developed (DIAMON) to monitor and diagnose the complete controls infrastructure. The adoption of the solution by a large community of users, followed by its rapid expansion, led to a final product that became increasingly difficult to operate and maintain. This was predominantly due to the multiplicity and redundancy of services, centralized management of data acquisition and visualization software, its complex configuration and its intrinsic scalability limits. At the end of 2017, a completely new monitoring system for the beam controls infrastructure was launched. The new "COSMOS" system was developed with two main objectives in mind: firstly, detecting instabilities and preventing breakdowns of the control system infrastructure. Secondly, providing users with a more coherent and efficient solution for development of their specific data monitoring agents and related dashboards. This paper describes the overall architecture of COSMOS, focusing on the conceptual and technological choices for the system.

INTRODUCTION

The CERN Accelerator Control System [1] relies on many components and a substantial infrastructure, which must be available 24 hours a day, 7 days a week. This hardware and software infrastructure needs to be monitored in order to anticipate or detect failures and fix them as quickly as possible. The Controls Open-Source Monitoring System (COSMOS) project was launched in 2017 to renovate the existing in-house solution [2] [3], which was suffering from its hyper-centralized model, the multiplicity of the solution, service overlap and scalability issues.

THE CONTEXT

In monitoring, the term ‘host’ refers to a device with an IP address (responsive to ping) while ‘service’ refers to any application, resource or hardware component (network device, module, sensor, etc.) providing a particular function on the host.

The accelerator control system has just under 7000 hosts (Fig. 1), mainly Linux CentOS CERN 7 computers (the use of Windows is declining in the domain of accelerator controls) and specific Ethernet devices (BMCs[1], PLCs[2], etc.). The number of Linux computers is constantly increasing, by 5 to 8% per year, while disk space has increased by a factor of 500 in a decade.

[*] frank.locci@cern.ch
[1] Baseboard Management Controller
[2] Programmable Logic Controller

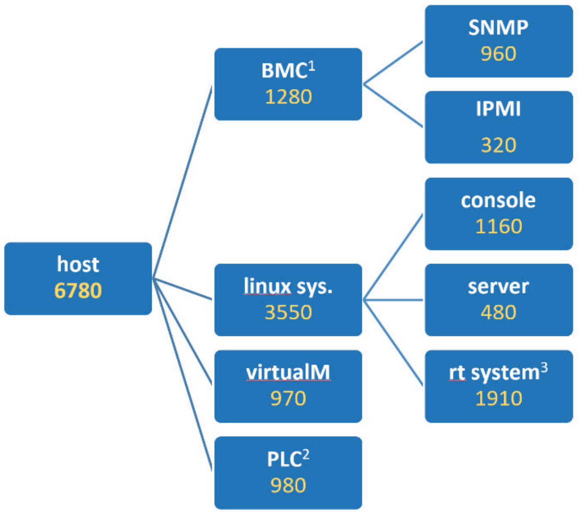


Figure 1: Main types of control system hosts.

OBJECTIVES AND SCOPE

Reviewing the existing system and evaluating major products in the monitoring field (collectd, Icinga2, Zabbix, Prometheus) helped us to define the main objectives of the COSMOS project and laid the foundations for the future solution.

Preliminary Study Recommendation

Recommendations emerging from the preliminary study were the following:

- Align the new monitoring system with CERN IT services (e.g. the central “DB on Demand” service) and industry standards in order to allow us to focus on our core business.
- Use de-facto standard technologies and open-source software as far as possible.
- Propose a new paradigm where specific aspects of the monitoring are delegated to experts who become responsible for collecting their metrics, define alerts and setup their own dashboards.

Scope of the COSMOS Monitoring System

When designing a monitoring system, it is important to consider the origin and the nature of data we want to monitor. We can distinguish at least two types of information intended for users with different objectives:

- Functional monitoring to detect infrastructure related failures, to alert the system administration team or equipment experts and to assist in taking technical decisions.
- Business monitoring focused on operational data and providing support for controlling the accelerator.

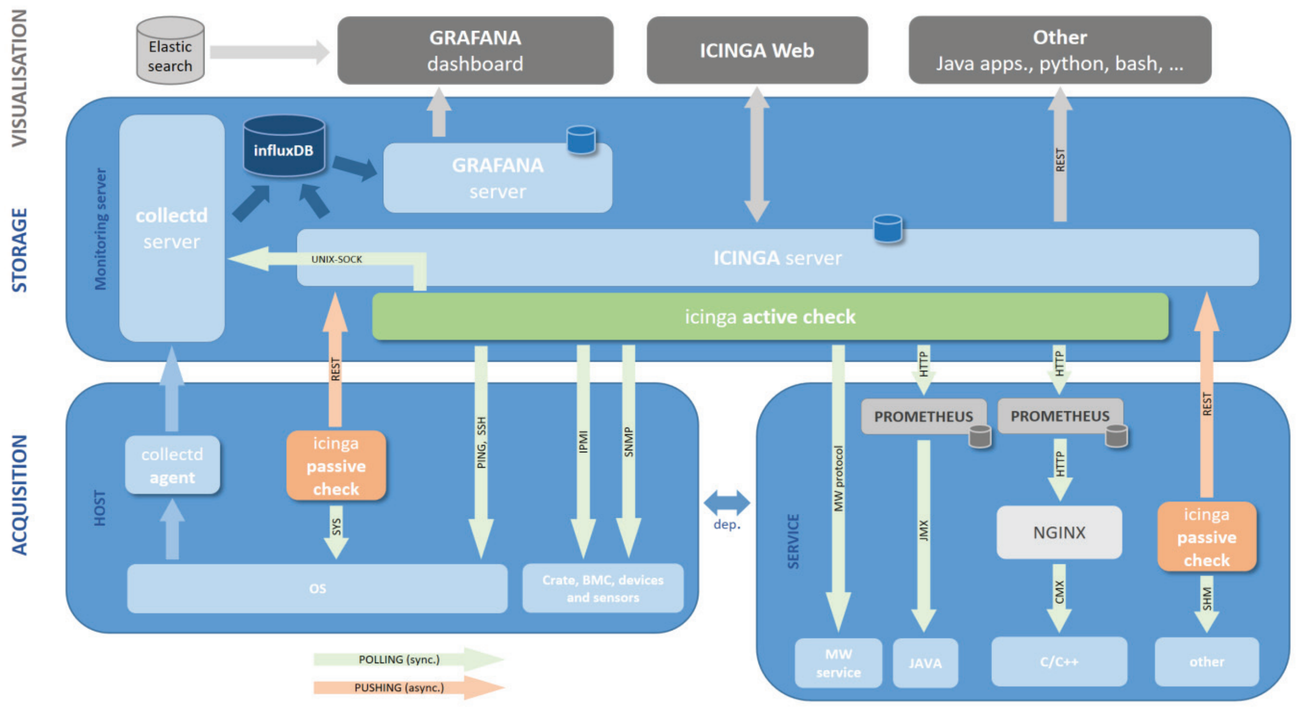


Figure 2: Integration and interconnection of the software component.collectd: acquisition of OS Metrics.

Table 1: Selected Open-Source Software

Product	Motivation
Collectd (5.8.1)	MIT License, easy to install, file-based configuration, modular layout, large plugin collection, CERN support
Prometheus (1.7.1)	Apache License 2.0, easy to install/run, rich data model, functional query language, powerful rule processing, graphing and alerting, HTTP API
Icinga2 (2.10.4)	GPLv2, file-based configuration, ~75% of needs covered, extensive features, collectd support, scalability, availability, multi-tenancy, large user community, support, complete documentation
InfluxDB (1.6.3)	MIT License, write and query performance, on-disk compression, scalability
Grafana (5.4.2)	Apache License 2.0, easy to install, file-based configuration, dashboard flexibility, data-sources support (Influx, Elastic Search), large user community

Experience shows that it is very difficult to combine both aspects and to provide simple and efficient tools that take the different needs into account. Therefore, COSMOS is exclusively dedicated to the functional monitoring of the controls infrastructure.

We wanted to avoid the pitfall of a Unified Monitoring Infrastructure (UMI) solution, which does not fit the size and diversity of our infrastructure. As discussed later, it has been possible to propose a perfectly customized and compact solution, based on a limited number of targeted open-source software (OSS) components, see Table 1 for details.

Finally, with a more modular approach, we wanted to share development between teams and thus clarify the responsibilities of each stakeholder (system administrators, application experts, operation, etc.).

THE SOLUTION

Overall Architecture

Figure 2 shows the COSMOS architecture. At its heart, one finds an open-source product called **Icinga2** [4]. Icinga covers most of our needs out of the box and perfectly fits the collaborative and distributed model that we need to monitor our heterogeneous infrastructure (from the hardware, software and human point of view).

Icinga2, which started as a fork of Nagios, introduces the ‘plugin’ concept, a standalone extension to the Icinga2 core. Each plugin instance (commonly called ‘check’) executes a specific logic and produces a health report of the related component (see Table 2 for details). The result of the check is made of a functional status report of the component and optional additional metrics (“performance data”) that are sent to the Icinga2 server. The server then generates a notification (by email or SMS) according to the user configuration, and sends status and performance data to the IDO[1] (MySQL) or time series (InfluxDB) database as appropriate.

In parallel, COSMOS uses **collectd** [5] agents to gather system metrics from hosts, related devices (disks, memory, etc.) and the network. collectd makes this information available over the network to the central server, where data is stored into the InfluxDB database as well.

[1] Icinga Data Output

Table 2: Common Host and Services Checks

Check	Mode	Source
basic connectivity (host alive: ping, ssh)	active	system
boot diagnostic (reboot count., Eth. speed, etc.)	passive	system, REST
Crate and BMC[1] metrics (fan speed, power sup., temp., batt., bus, led, etc.)	active	IPMI
Timing network, specific devices and sensors	active	SNMP
Real-time fieldbus and sub-network agents	passive	REST (proxy)
Disk partition usage, CPU load, std. and EDAC[2] memory, etc.	active	collectd-unixsock
PLC[3]	active	JMX [6] (proxy)
Process status (up/down) and diagnostics (CPU, memory, etc.)	passive	systemd, system, REST
Application status and functional metrics	active	JMX, CMX, NGINX [7]

Whenever it is not possible to compute the status of a service from a single Icinga2 check, or in order to detect trends, COSMOS uses Prometheus as an intermediate agent. **Prometheus** [8] then generates the service status using its functional expression language (see 'Prometheus' chapter below for details).

The central IDO database is used by expert tools such as IcingaWeb, to establish, in real time, a complete diagnostic of each component, to manage notifications and downtime, and to provide detailed statistics and event histories.

Finally, data can be visualized and analyzed by expert users thanks to a dedicated **Grafana** [9] instance.

An Open-Source Software Based Solution

In accelerator control, as in other fields, the network size, the number and complexity of deployed equipment and the rate of change are constantly increasing. It makes monitoring a vital part of system administration activities. At the same time fortunately, some extremely powerful, open source monitoring tools, have appeared on the market. By selecting and integrating several of these tools, we have built a lightweight and efficient solution that best met our objectives (Fig. 3).

collectd is the piece of software that we use to gather operating system metrics on every host covered by COSMOS. This tool, dating back to 2005, still receives regular commits and is released under the MIT License: this is a very stable product that was successfully adopted beforehand by CERN's IT department. We use both sides of collectd's client-server model. The client is a daemon running on every host within the scope of accelerator controls, whether diskless (front-end computers) or disk-based (servers) and technical consoles). Every five minutes a predefined set of metrics is collected through the daemon. The typical amount of system-related metrics per host is between 60 and 90. Covered areas are – non-exhaustively – CPU usage and statistics, RAM usage, disk status (with S.M.A.R.T. attributes when applicable) and partition usage, as well as network measurements. Metrics are tailored to the hardware that collectd is running on: for instance, we automatically detect if extra partitions are present on the system, or if SSDs of a particular brand are physically present, in order to get relevant information about their depreciation. Once collected locally on a host, metrics are sent to an instance of collectd acting as a server and running on the COSMOS server. This instance plays three roles:

- Transferring metrics to Icinga2 that will determine whether measurements are within an acceptable range.
- Pushing metrics to the Influx database, which is the main source of data for Grafana, COSMOS's graphical visualization layer.
- Writing metrics into RRD files (Round-Robin database), one for each metric. RRD files provide a low-level data visualization alternative to Grafana, mostly used as a fallback method by system administrators.

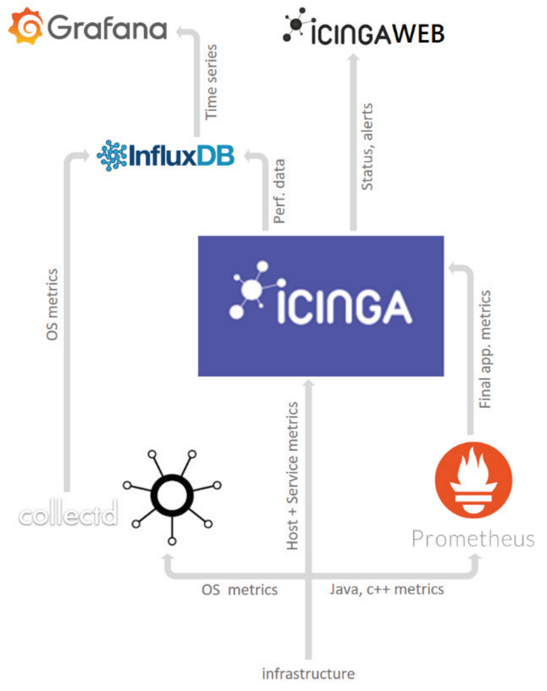


Figure 3: An OSS solution fitting each kind of metrics.

Icinga2: Hosts and Services Checks

An Icinga2 check is a piece of code running on monitored nodes or directly on the Icinga2 central server. In the

[1] Baseboard Management Controller
[2] Error Detection and Correction
[3] Programmable Logic Controller

latter case, it uses custom or dedicated protocols such IPMI or SNMP to gather raw data from which it computes the final status of the service, based on pre-configured thresholds. A check can be as simple as a few lines of Bash, C or Python. It can be active or passive. An active check is triggered by the central server in polling mode and takes metrics from the nodes on a periodic basis. A passive check is a standalone agent (usually running on the monitored node or proxy), pushing data to the central server asynchronously. With COSMOS, it is used in particular to reach hosts deployed on Ethernet subnets like White-Rabbit, CERN's Control timing system. The Icinga2 REST API is then used to send the result to the Icinga2 server. In the case of systems for which it is critical to fork processes and allocate resources at any moment, passive checks are also recommended. For the same reason, it is suggested to use high-performance languages such as C or C++ for real-time systems. For instance, acquisition of metrics for low-level processes is performed with C++ Management Extension (CMX [10]) agents, as described later in this document.

In addition to custom plugins developed by users, there are over 3,000 third-party Nagios plugins [11] we can use to easily enhance the monitoring system.

Prometheus: Application Metrics Acquisition

In many cases, application monitoring is based on observation over time, instead of a single indicator such as a counter. Basic service checks turn into complex, stateful agents. Instead of designing, implementing and maintaining such agents, the idea is to use Prometheus and let it act as an agent for Icinga2, using its embedded data storage and powerful query language (Fig. 4).

The Prometheus language allows the definition of any kind of rule, from a simple comparison of a single metric, to a complex check against an aggregation of metrics over time. This provides a lot of flexibility, whilst hiding the implementation details from the upper layer. An Icinga2 active check (prom-check) queries alerts and metrics from Prometheus in order to expose them as performance data. To ease aggregation and classification inside the higher layer, a set of standard labels has been defined: hostname, application and service. Besides the rules, which are manually defined, the Prometheus configuration is automatically derived from the Icinga2 configuration.

It is trivial to integrate Prometheus with Java applications through the standard JMX protocol [6]. However, for native applications that are monitored through the C++ CMX interface [10], custom development was needed. Such native applications run on soft real-time, diskless computers called Front-Ends (FECs). As Prometheus performs HTTP GET calls to scrap the raw metrics, it is necessary to run a HTTP server directly on the FEC. Several solutions were evaluated: Boost Beast, Apache, Lighttpd and NGINX [7]. Due to its stable memory usage, small footprint and low CPU usage, NGINX became the webserver of choice. A native module was developed to read CMX metrics from the shared memory and publish them through HTTP via a heap buffer [12]. The module is real-time compliant: everything is done in-memory, without disk access.

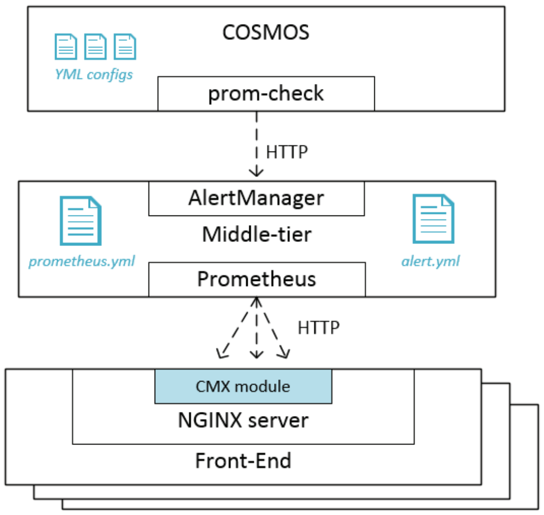


Figure 4: Overview of the CMX monitoring stack.

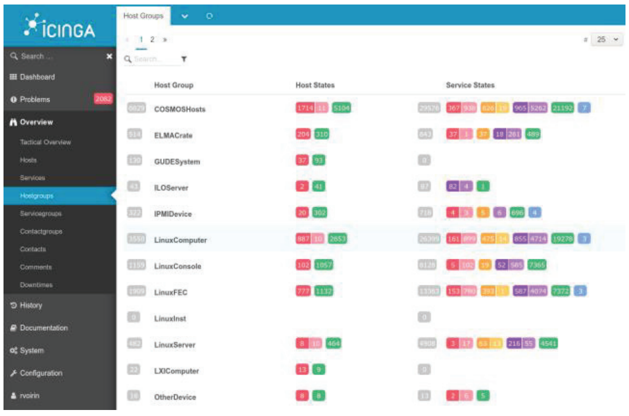
Graphical User Interfaces

COSMOS presents monitoring data in various fashions. Firstly, operators can still use the previous DIAMON console, refactored to use COSMOS components as data providers (or backends), without modifying the overall appearance and behaviour of the GUI. From custom tree views, the user can quickly visualize failing systems and restart them if necessary.

Secondly, Grafana (Fig. 5) is the main entry point for visualizing infrastructure metrics. As collectd ingests data and sends it to the Influx database, Grafana is used to query and display results in graphical manner. Teams can create their own dashboards, focusing on particular aspects of the infrastructure. For instance, displaying a chart of fan speeds for specific kinds of hardware or retrieving the amount of RAM used by a process over time, in order to spot potential memory leaks.

Figure 5: Visualization of server metrics in Grafana.

The final user interface is IcingaWeb (Fig. 6), the front-end to Icinga2, which is very efficient at providing an overview of problems occurring in the infrastructure, as well as a detailed history of events. In addition to its powerful search tool and its integrated and customizable views, IcingaWeb offers a complete, multi-user interface to interact with the monitoring process (events control, downtime period, checks scheduler, etc.).



	Host Group	Host States	Service States
[illegible]	COSMOSHosts	1714 11 5104	[illegible] 367 938 820 19 905 5262 21192 7
514	ELMACrate	204 310	843 37 1 37 18 261 489
130	GUDESystem	37 93	0
43	ILOServer	2 41	87 82 4 1
322	IPMIDevice	20 302	718 4 3 5 6 696 4
3550	LinuxComputer	887 10 2653	26399 161 899 475 14 855 4714 19278 3
1159	LinuxConsole	102 1057	8128 5 102 19 52 585 7365
1909	LinuxFEC	777 1132	[illegible] 153 790 393 1 587 4074 7372 3
0	LinuxInst		0
482	LinuxServer	8 10 464	4908 3 17 63 13 216 55 4541
22	LXIComputer	13 9	0
16	OtherDevice	8 8	13 2 6 5

Figure 6: IcingaWeb showing host groups.

Checks Deployment and Configuration

COSMOS includes an automatic configuration tool, developed in Python, to generate Icinga2 configuration files on a daily basis, from three different data sources:

- The controls configuration database (Oracle) describing each host, service and their dependencies;
- The list of hosts managed by Ansible and their predefined variables;
- The user-specific data (alarms setup, check arguments, etc.).

If we choose to delegate the development and maintenance of checks to the users, it is also essential to provide adequate means to store and deploy these pieces of software. COSMOS uses GitLab for this purpose and in particular its CI/CD module to build (when necessary) and deploy checks whenever the user makes changes in his code.

CONCLUSION AND FUTURE PLANS

Less than one year (1.25 man-years) has been required to build the core part of COSMOS (Icinga2, collectd), covering 80% of the basic infrastructure monitoring. The second phase of the project, which is ongoing, is dedicated to the monitoring of processes, application metrics (Prometheus) and to the integration of specific diagnostics and critical accelerator equipment (cryogenics, magnet protection, etc.). Today, a single Linux computer (2xCPU 2.2GHz/10 Cores, 128Gb RAM) is hosting the COSMOS services (Icinga2, collectd and Grafana) and easily supports the load.

It performs on average 30,000 checks every 5 minutes and generates a large number of metrics, as shown in Table 3.

System administrators and control experts quickly adopted the COSMOS tools for diagnosis and daily maintenance work. We must now provide the system with a framework dedicated to a wider audience (equipment groups, operation) and allow the configuration of custom checks (setup, notification, etc.) in a more autonomous way. The use of IcingaWeb may also be extended to all users. To do this we must guarantee that the system can support a large number of simultaneous connections, that it is secure, provides proper backup services and the required level of availability. We are considering implementing some of the high-availability features of Icinga2 to improve overall reliability (clustering, failover mechanism, etc.)

Table 3: Metric Statistics

Item	Source	~Number
OS metric types	collectd	250
OS metric cardinality	collectd	236,000
Check types	Icinga2	80
Check cardinality	Icinga2	29,600
Perf. data types	Icinga2	2,500
Perf. data cardinality	Icinga2	1,378,000

The experience gained so far shows that the new collaborative model is being well-received by users and works perfectly. The project study also demonstrated that no single monitoring tool on the market could cover 100% of our needs. However, it is relatively easy to build a complete system by integrating a small number of open-source software tools, providing that the tools are properly selected according to their functionality, complementarity and interconnectivity.

REFERENCES

[1] *Introduction to the BE-CO Control System*, CERN, GVA, Switzerland, 2019 edition, chapter 23.

[2] P. Charrue, M. Buttner, F. Ehm, and P. Jurcso, “Improving Software Services Through Diagnostic and Monitoring Capabilities”, in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 1070-1072. doi:10.18429/JACoW-ICALEPCS2015-WEPGF155

[3] M. Buttner, P. Charrue, J. Lauener, and M. Sobczak, “Diagnostic and Monitoring CERN Accelerator Controls Infrastructure: The DIAMON Project - First Deployment in Operation”, in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper TUP019, pp. 128-130.

[4] Icinga2, https://icinga.com/products

[5] collectd, https://collectd.org

[6] JMX, https://en.wikipedia.org/wiki/Java_Management_Extensions

[7] NGINX (doc), https://www.nginx.com

[8] Prometheus, https://prometheus.io

[9] Grafana, https://grafana.com

[10] F. Ehm, Y. Fischer, G. M. Gorgogianni, S. Jensen, and P. Jurcso, “CMX - A Generic Solution to Expose Monitoring Metrics in C and C++ Applications”, in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper THPPC014, pp. 1118-1121.

[11] Nagios, https://www.nagios.org/projects/nagios-plugins

[12] NGINX (git), https://gitlab.cern.ch/smith/nginx-cmx-module