Chopin Management System: improving Windows infrastructure monitoring and management

Marta Pacuszka[1,*], *Sebastian* Bukowiec[2,**], *Esteban* Puentes[2,***], and *Guillaume* Metral[2,****]

[1]Warsaw University of Technology
[2]CERN

Abstract. CERN Windows server infrastructure consists of about 650 servers. The management and maintenance is often a challenging task as the data to be monitored is disparate and has to be collected from various sources. Currently, alarms are collected from the Microsoft System Center Operation Manager (SCOM) and many administrative actions are triggered through e-mails sent by various systems or scripts.
The objective of the Chopin Management System project is to maximize automation and facilitate the management of the infrastructure. The current status of the infrastructure, including essential health checks, is centralized and presented through a dashboard. The system collects information necessary for managing the infrastructure in the real-time, such as hardware configuration or Windows updates, and reacts to any change or failure instantly. As part of the system design, big data streaming technologies are employed in order to assure the scalability and fault-tolerance of the service, should the number of servers drastically grow. Server events are aggregated and processed in real-time through the use of these technologies, ensuring quick response to possible failures. This paper presents details of the architecture and design decisions taken in order to achieve a modern, maintainable and extensible system for Windows Server Infrastructure management at CERN.

1 Introduction

At the time of writing this paper, the CERN Windows Server Infrastructure consists of around 650 servers. Following the strategy of the IT department, the number of servers based on virtual machines has increased and now equals that of those based on physical hardware. The virtual machines are based on the CERN Cloud Infrastructure [1]. While providing many benefits, there are still some use cases that require physical servers e.g. the Distributed File System (DFS) that is a storage system with external disk arrays.

The hardware monitoring for the Linux systems at CERN is done using open source collectd [2]. The metrics are gathered by collectd from various sources, e.g. the operating system, applications, logfiles and external devices, and stores this information or makes it

*e-mail: m.pacuszka@stud.elka.pw.edu.pl
**e-mail: sebastian.bukowiec@cern.ch
***e-mail: esteban.puentes@cern.ch
****e-mail: guillaume.metral@cern.ch

available over the network. Support for Microsoft Windows collectd is provided by SSC Serv, a native Windows application that can collect and dispatch performance data using the collectd network protocol. The Windows-based client is not open source, does not have a large user community and does not have the same capabilities as the Linux version. This makes it difficult to gather Windows servers hardware events using collectd.

This paper describes the Chopin Management System project that allows automation and facilitates infrastructure management in real-time. The working system is planned to replace the operation based on various event-gathering scripts and Microsoft System Center Operation Manager (SCOM).

2 System requirements

The main objective to create the Chopin Management System was to remove the remaining manual actions in operating Windows Server Infrastructure and consolidate all the scripts into one system. Additionally, it was desired to remove all the e-mail notifications and replace them with a modern dashboard. To address these requirements a distributed system was designed that could quickly scale and at the same time, it could be easily extended.

From the high-level perspective, logs are sent to the Message Broker which acts as a log buffer and a communication hub. For this purpose, RabbitMQ was selected as it proved to be reliable in other projects and supports many client libraries including .NET, Java, and node.js. The extensibility was also a requirement for the agent that is installed on all the monitored servers. For a database backend, it was decided to use one of the offerings of the IT database group (IT-DB) Database on Demand Service providing mySQL instances. As the processing units, a custom code written in .NET Core together with Apache Flink providing stream processing capabilities were chosen. React and Redux were selected as the technologies for the visualization layer as they are widely used in the CERN IT department.

Finally, the Chopin Management System was designed to be deployed on the OpenShift container platform.

3 System architecture

The Chopin management system is a distributed system application that consists of the following modules, presented in figure 1:

- Agent - Windows service deployed on every server of the Windows Server Infrastructure, with the purpose of collecting data continuously and sending it to the message broker,
- Message broker - enables the communication between the modules and serves as a buffer for the messages,
- Data stream computation module (STREAM PROCESSING MODULE) - responsible for processing streams of data received from servers. It filters the potentially interesting events, e.g. it detects changes in hardware configuration according to the defined set of rules. It allows anomaly detection in the scope of seconds to hours.
- Message processing and persistence module (DATA PERSISTENCE MODULE) - responsible for further processing of events, persisting data in the database and inferring from both incoming and historic data. This module is the core of anomaly detection in the scope of days to months.
- Dashboard - web application with user interface presenting the state of the infrastructure, including the latest failures and history of events and hardware configurations.

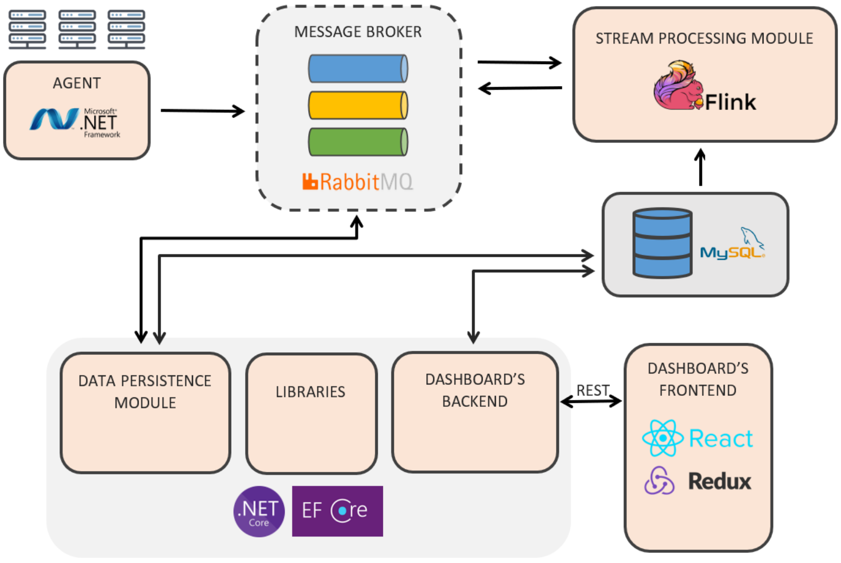


Figure 1. The Chopin management system architecture.

3.1 Indirect communication via message queues

The modular structure of the system required adapting an efficient and fault-tolerant communication mechanism that would be space and time-uncoupled. We decided to leverage a Message Oriented Middleware (MOM) technology, which supports the combination of publish-subscribe and message queues architectural patterns. In the MOM model, the basic unit of data is expressed with the concept of a message - an event with a predefined structure, and the core of it is a message broker - an intermediary software responsible for storing and routing or multicasting received messages to the appropriate consumers.

Having overviewed a range of open-source message brokers, we decided to use RabbitMQ by Pivotal [3], which is modeled on the AMQP standard. It provides a point-to-point communication model, using the concept of a queue storing messages sent by a producer (or producers) until they are received by the consumer and provides mechanisms such as Acknowledgements and Confirms to assure reliable delivery, even in case of a failure in any part of the system.

The message queue communication model is used not only for communication between data processing modules but also for delivering the unprocessed events from the agents installed on every server, the number of which is subject to changes. Hence, the model had to be independent of the number of producers. It has been achieved by creating a queue per category of a message, and a server identifier is included in a message. That is to say, a queue accepts messages from all the servers, and identifying the producers happens at the later stage. The communication model is presented in figure 2. Messages are sent to a direct exchange that routes them to the appropriate queue based on matching routing and binding keys.

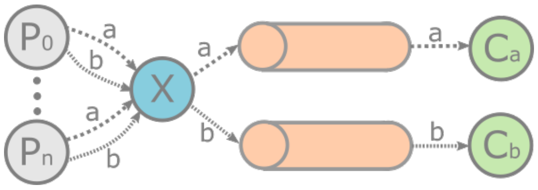


Figure 2. Communication model with RabbitMQ.

3.2 Collecting data with distributed agents

The Windows Agent is an application written in .NET Framework and runs in background as a Windows service on every monitored machine. The role of the agent is to gather necessary logs and send them to the monitoring system, to be more precise to the message broker (AQMP). The agent is designed in a modular way and it can be easily extended via plugins.

The core idea behind the agent is to send data in a raw form or only marginally processed to not cause a heavy load on the monitored machine itself. Performing the message filtering at a further stage of the pipeline ensures that all the required messages are sent and most of the system modifications and adjustments do not require the change or redeployment of the agent.

The agent is designed on the component-based architecture. The functional components communicate through interfaces. This facilitates development since integrating new functionalities requires only the implementation of an interface rather than integrating this logic in the agent itself.

The plugins are created as Dynamic Link Libraries (DLL), in which the agent is able to identify and load through the implemented interfaces. The two types of plugins can be differentiated, depending on their purpose, input and output. The former is responsible for collecting data from the server, while the latter sends this data to the either mentioned message broker or the console output. The agent allows sending information to multiple outputs of the same type e.g. the same information is sent to the quality assurance AQMP instance and to the production AQMP instance.

The data is collected from the Windows system event log in the real-time and from any data source specified in the plugins e.g. storcli.exe, a tool used to gather information about the RAID controller and disks. Each plugin runs based on a configured schedule e.g. information about the installed and available Windows updates is obtained every hour, a heartbeat with basic information about the system execution is generated every two minutes.

The agent has an auto-update feature that periodically (every hour) checks for the newer versions of the agent and all the plugins. The update is carried out in a separate process that is invoked by the agent itself. This process stops the application, replaces all the components with a newer version available and starts it again.

3.3 Data stream processing module

The main goal of the module is to process all the messages received from the servers as fast as possible (ideally, close to real-time) and detect the ones that (a) indicate critical events, failure or anomalies, or (b) may prove to be useful in detecting anomalies in the future, and therefore should be persisted.

Apache Flink is an open-source streaming processing engine, based on the operator-based computational model. Implementing a streaming job using this framework has been considered a plausible solution to the problem for several reasons.

1. Speed. One of the Flink's architecture assumptions is that data and computation are co-located, which results in computations being performed at in-memory speed. It is also very easily scalable, with no effect on consistency, due to the two types of processes that the Flink runtime consists of: the JobManagers, of which only one is active and coordinates tasks, checkpoints, recovery, etc, and TaskManagers (one or many) acting as workers executing the tasks. [4]

2. Stream processing. Flink's DataStream Api [5] enables stateful computations over the unbounded data stream, which matches the character of machine logs. The stream is processed continuously and events are handled just after they have been ingested, triggering actions such as state changes or producing other messages.

3. Fault tolerance. In case of failure, it is important to be able to restore the state from before the failure and make sure that no message is missed, nor processed more than once. Exactly-once state consistency is achieved by periodically writing checkpoints to remote persistent storage.

4. Rule-based anomaly detection. Many hardware changes or failures can be detected by finding well-defined sequences of specific types of logs. Flink features a library for Complex Event Processing (CEP) [6], that provides an API to specify patterns of events. The patterns are evaluated on data streams, leveraging all the advantages of Flink's DataStream API.

5. Easy development. The development of a job is an iterative process or implementation, validation, tests, and improvements, therefore one of the key requirements was to choose a tool that is modifications-friendly. Flink features savepoints that enable updating the job or adapting the scale of the application while preserving the application state from before a change. External connectivity is exposed via an HTTP/REST endpoint, used by, among others, a command-line tool, which makes it easy to submit, inspect and modify jobs.

The data stream processing module in the Chopin Management System provides the following functionality:

- filtering of the incoming log entries based on their severity and user-defined dynamic rules,
- filtering of the incoming data collected periodically from a range of monitoring software installed on machines, based on changes in the defined part of a message,
- detecting changes in hardware configuration, failures and automatic restores of devices based on static rules,
- detecting idle producers, using the windowing features of DataStream API.

3.4 Message final processing and persistence module

This module is an application responsible for processing already filtered messages received from the stream processing module altogether with data already stored in the database, persisting new data to the database and generating warnings and notifications if a failure or unexpected behavior has been detected.

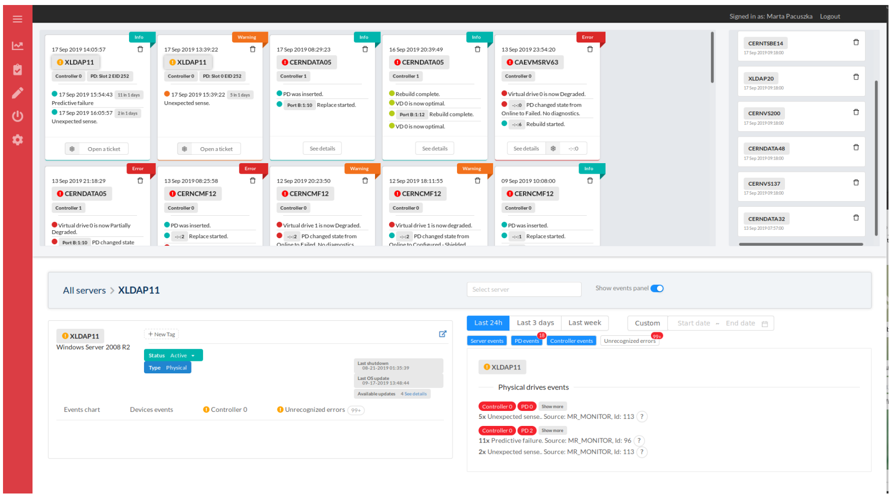
Having access to the historic data, the module is able to perform more informed decisions about the importance of incoming events, but it also means that this module is slower when compared with the stream processing module (see: 3.3) due to the database queries it has to perform. However, since most of the insignificant messages (which constitute most of the time a vast majority of all incoming data) have been discarded already by the stream processing module, at this stage more expensive operation can be performed without risk of clogging the system.

3.5 Dashboard

The indispensable part of a Chopin monitoring tool is a web application where the following information is presented:

- notifications containing information about the latest alerts, triggered rules and configuration changes. They get updated when a related event happens and can be managed by the user,
- list of servers that have stopped sending messages,
- list of historic events of possible significance, aggregated by a server, type of device and date. For some devices, the amount and frequency of events is visualised,
- list of all monitored servers, including information about the warranty of the operating system and available updates,
- history of a hardware configuration for every server,
- the status of every device, calculated based on notifications related to the device,
- the list of currently defined rules for alert triggering, with the possibility to modify it.

The exemplary screenshots from the dashboard are presented in the figures 3 and 4.


Signed in as: Marta Pacuszka Logout

Info — 17 Sep 2019 14:05:57 — XLDAP11 — Controller 0 — PD: Slot 2 EID 252
17 Sep 2019 15:54:43 — 11 in 1 days — Predictive failure
17 Sep 2019 16:05:57 — 2 in 1 days — Unexpected sense.
Open a ticket

Warning — 17 Sep 2019 13:39:22 — XLDAP11 — Controller 0 — PD: Slot 0 EID 252
17 Sep 2019 15:39:22 — 5 in 1 days — Unexpected sense.
Open a ticket

Info — 17 Sep 2019 08:29:23 — CERNDATA05 — Controller 1
PD was inserted.
Port B:1:10 Replace started.
See details

Info — 16 Sep 2019 20:39:49 — CERNDATA05 — Controller 1
Rebuild complete.
VD 0 is now optimal.
Port B:1:12 Rebuild complete.
VD 0 is now optimal.
See details

Error — 13 Sep 2019 23:54:20 — CAEVMSRV63 — Controller 0
Virtual drive 0 is now Degraded.
-:-:0 PD changed state from Online to Failed. No diagnostics.
-:-:6 Rebuild started.
See details -:-:0

Error — 13 Sep 2019 21:18:29 — CERNDATA05 — Controller 1
Virtual drive 0 is now Partially Degraded.
Port B:1:10 PD changed state

Error — 13 Sep 2019 08:25:58 — CERNCMF12 — Controller 0
PD was inserted.
-:-:2 Replace started.

Warning — 12 Sep 2019 20:23:50 — CERNCMF12 — Controller 0
Virtual drive 1 is now Degraded.
-:-:2 PD changed state from

Warning — 12 Sep 2019 18:11:55 — CERNCMF12 — Controller 0
Virtual drive 1 is now degraded.
-:-:2 PD changed state from

Info — 09 Sep 2019 10:08:00 — CERNCMF12 — Controller 0
PD was inserted.
-:-:1 Replace started.

CERNTSBE14 — 17 Sep 2019 09:18:00
XLDAP20 — 17 Sep 2019 09:18:00
CERNVS200 — 17 Sep 2019 09:18:00
CERNDATA48 — 17 Sep 2019 09:18:00
CERNVS137 — 17 Sep 2019 09:18:00
CERNDATA32 — 13 Sep 2019 07:57:00

All servers > XLDAP11 — Select server — Show events panel

XLDAP11 — Windows Server 2008 R2 — + New Tag
Status Active
Type Physical
Last shutdown 08-21-2019 01:35:39
Last OS update 09-17-2019 13:48:44
Available updates 4 See details
Events chart — Devices events — Controller 0 — Unrecognized errors 99+

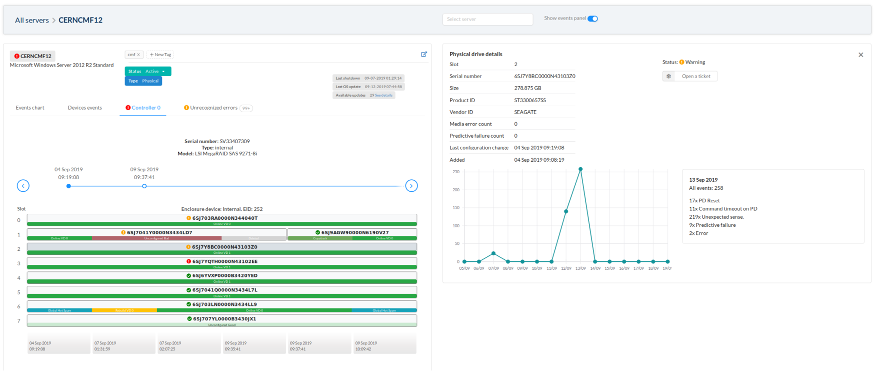
Last 24h — Last 3 days — Last week — Custom — Start date ~ End date
Server events — PD events 18 — Controller events — Unrecognized errors 99+

XLDAP11
Physical drives events
Controller 0 — PD 0 — Show more
5x Unexpected sense.. Source: MR_MONITOR, Id: 113 ?
Controller 0 — PD 2 — Show more
11x Predictive failure. Source: MR_MONITOR, Id: 96 ?
2x Unexpected sense.. Source: MR_MONITOR, Id: 113 ?

Figure 3. Server view in the main page of the dashboard.

4 System lifecycle

The modules are deployed in the Platform as a Service (PaaS) type of infrastructure. The solution used at CERN is the OpenShift container application platform, developed by Red Hat and built on top of Kubernetes. In order to provide quick building and automatic deployments of applications, directly from the Git repository, we have used the CI/CD pipelines provided


All servers > CERNCMF12

Select server   Show events panel

CERNCMF12
Microsoft Windows Server 2012 R2 Standard

cmf ×   + New Tag

Status Active
Type Physical

Last shutdown 09-07-2019 01:29:14
Last OS update 09-12-2019 07:44:58
Available updates 29 See details

Events chart   Devices events   Controller 0   Unrecognized errors 99+

Serial number: SV33407309
Type: internal
Model: LSI MegaRAID SAS 9271-8i

04 Sep 2019 09:19:08   09 Sep 2019 09:37:41

Enclosure device: Internal. EID: 252

Slot		
0	6SJ703RA0000N344040T (Online VD 0)	
1	6SJ7041Y0000N3434LD7 (Online VD 0, Unconfigured Bad, Deleted)	6SJ9AGW90000N6190V27 (Copyback, Online VD 0)
2	6SJ7Y8BC0000N43103Z0 (Online VD 1)	
3	6SJ7YQTH0000N43102EE (Online VD 1)	
4	6SJ6YVXP0000B3420YED (Online VD 1)	
5	6SJ7041Q0000N3434L7L (Online VD 1)	
6	6SJ703LN0000N3434LL9 (Global Hot Spare, Rebuild VD 0, Online VD 0, Global Hot Spare)	
7	6SJ707YL0000B3430JX1 (Unconfigured Good)	

04 Sep 2019 09:19:08 | 07 Sep 2019 01:31:59 | 07 Sep 2019 02:07:25 | 09 Sep 2019 09:35:41 | 09 Sep 2019 09:37:41 | 09 Sep 2019 10:09:42

Physical drive details

Slot	2
Serial number	6SJ7Y8BC0000N43103Z0
Size	278.875 GB
Product ID	ST3300657SS
Vendor ID	SEAGATE
Media error count	0
Predictive failure count	0
Last configuration change	04 Sep 2019 09:19:08
Added	04 Sep 2019 09:08:19

Status: Warning
Open a ticket

13 Sep 2019
All events: 258

17x PD Reset
11x Command timeout on PD
219x Unexpected sense.
9x Predictive failure
2x Error

Figure 4. Server configuration history and physical drive details.

by Gitlab. This allowed us to test and deploy software at the level of every commit. A sequence of jobs that we have developed for the persistence module, as well as backend and frontend of the dashboard including tests, building a Docker image and publishing it to the GitLab Docker registry. Then, the image is imported in the corresponding OpenShift project and the new version of the application is released.

In the case of the stream processing module, the Flink cluster is deployed on OpenShift container platform, and the client is run using the GitLab's CI/CD. The new jobs are submitted using the HTTP/REST endpoint. The process is presented in the figure 5.

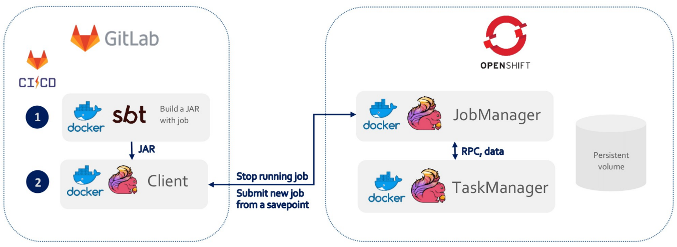


Figure 5. Deployment of new Flink jobs.

5 Conclusion

The management and maintenance of large infrastructures is often a challenging task as disparate data has to be monitored and collected from various sources. The Chopin Management System designed and deployed at CERN addresses these issues for the operation of the Windows Server Infrastructure.

Chopin receives and processes messages from the surveillance agent to present received data in a simplified and understandable way. The system does not aim to provide performance metrics but is intended to provide a tool that was missing in the administrators' toolbox.

Future plans include the extension of the system to provide general service availability to the CERN central monitoring system. This is currently done for Windows Servers using Microsoft System Center Operations Manager (SCOM).

References

[1] Bell T, Bompastor B, Bukowiec S, Castro Leon J, Denis M, van Eldik J, Lobo M Fermin, Fernandez Alvarez L, Fernandez Rodriguez D, Marino A, Moreira B, Noel B, Oulevey T, Takase W, Wiebalck A and Zilli S 2015 Scaling the CERN OpenStack cloud, J. Phys.: Conf. Ser. 664 (2015) 022003

[2] collectd, *The system statistics collection daemon*, https://collectd.org/, Accessed: 2020-03-10

[3] Pivotal, *RabbitMQ*, https://www.rabbitmq.com/, Accessed: 2020-02-07

[4] Apache Foundation, *Flink concepts: Distributed Runtime Environment*, https://ci.apache.org/projects/flink/flink-docs-stable/concepts/runtime.html, Accessed: 2020-02-07

[5] Apache Foundation, *Flink DataStream API Programming Guide*, https://ci.apache.org/projects/flink/flink-docs-stable/dev/datastream_api.html, Accessed: 2020-02-07

[6] Apache Foundation, *FlinkCEP - Complex event processing for Flink*, https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/libs/cep.html, Accessed: 2020-02-07