*Energy data visualization tools from VM*

*Abstract*- **Today, the energy management of enterprise IT systems requires the management of heterogeneous and distributed devices. In the past, the IT infrastructure was limited to computers connected by ethernet, and a few local servers. From now on, the computer park has diversified with mobile devices such as laptops and smartphones, internet terminals such as WiFi and routers, local and remote servers such as thecloud, as well as various connected objects.** **The objective of this article is to highlight two tools for measuring and visualize the energy consumption of a virtual machine in order to easily manage it energy.**

***Keywords: Scaphandre API, Prometheus, energy consumption***

Introduction

The general theme of this article is green IT. It is a set of techniques aimed at limiting the environmental consequences of information and communication technologies. Green IT appeared when the IT infrastructure became one of the causes of the increase in the greenhouse effect due to its relatively high consumption of electrical energy. This article highlights the combination of two fairly effective tools for collecting and then graphically displaying the energy consumption metrics of a virtual machine. The tool used to collect energy consumption data is an API, called Scaphandre, and the data visualization tool in the form of graphs for setting up a final dashboard summarizing, by real time, the data collected for a data virtual machine, is called Grafana. Grafana is part of an ecosystem called Prometheus. The originality of this article is the detailed explanation, the implementation and the display of the result in the form of a dashboard of the combination of the two tools the Scaphandre API and the Prometheus ecosystem.

To do this, we will first understand how the Scaphandre API and the Prometheus ecosystem work. Next, we will look at the implementation of these two technologies. Finally, we will see the final result that we obtained thanks to these tools.

Related work

How does the Scaphandre API work?

Scaphandre is an API aimed at measuring the energy consumption metrics of technological services. It provides the energy consumed by a single process on a server or virtual machine. In order to easily understand the calculation of the amount of energy consumed by a single process executed on a machine (server or virtual machine), we can imagine a rectangle containing lines each representing the calculation time allocated to each process. For machines working on different processes at the same time (i.e. working on one process for a short interval of time and then another time interval and so on), we call these intervals the jiffies. Each process keeps a running total of the total number of jiffies allocated to it. Thus, to know the quantity of resources of a machine used by a given process, it suffices to know the total number of jiffies used. In order to know the power used by a given process, we first count the jiffies used by this process when it is running. Then, for each jiffy, we check the amount of power consumed at those specific times. By aggregating all the power readings for all the jiffies over a specific time frame, we can arrive at a usable figure for the amount of power used in terms of watt hours.

Without deploying Scaphandre, or an API of the same type, it is of course possible to know the most important part of the resources of a machine in terms of use by a given process. On the other hand, if we want to find the amount of energy used by process, we must necessarily know how much energy is used by the machine. Energy consumption information is extracted using RAPL technology. It is a technology integrated in Intel processors and AMD processors, having the x86 architecture, produced after the year 2012. The powercap module, located between Scaphandre and this energy consumption data, writes the energy consumption in files, which will then be read by Scaphandre. After being read, Scaphandre API stores this data in buffers, and therefore allows further processing through the various exporters.

The Scaphandre API is divided into two main parts; a sensor and an exporter.

The 1st part, the sensor, is therefore intended to obtain the energy consumption metrics of the host, and make them available to the exporter. For example, PowercapRAPL obtains and transforms metrics from the powercap Linux kernel, which serves as an interface to obtain RAPL function data from x86 processors. The PowercapRAPL sensor first collects energy consumption measurements and then converts them into energy consumption measurements. Each time the exporter, like the prometheus exporter, requests a measurement, for example each time a request arrives, which we call request ”i”, PowercapRAPL reads the values ​​of the energy meters from powercap. Then this sensor stores them and performs the same operation for CPU usage statistics and for each process running on the machine in real time. Now, between 2 measurement requests, request “i” and request ” i+1”, we have the possibility to obtain the energy consumption subset related to the PID of a process. So to find out what a service actually consumes, simply join the consumption of all the associated PIDs. It is important to note that this functionality is not available directly for virtual machines. In this case, with the QEMU exporter, we will first have to run Scaphandre on the hypervisor, and then make the VM metrics available.

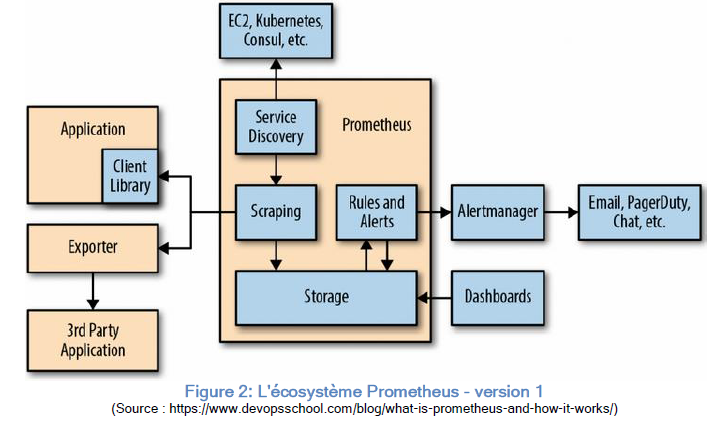
Regarding the 2nd part, the exporter is intended to request the sensors to obtain new measurements and store them for possible later use. The exporter therefore allows you to export the current metrics. For example, the prometheus exporter exposes metrics on an HTTP endpoint, to be retrieved by a prometheus instance. Then the stdout exporter just exposes the metrics to standard output. Regarding the qemu exporter, it is intended to collect metrics related to the execution of virtual machines on a Qemu/KVM hypervisor. These metrics will be made available to each virtual machine by running the PowercapRAPL sensor with the –vm option. The Qemu exporter puts virtual machine metrics into files the same way the powercap kernel module does. He imitates this behavior so that the sensor can act the same as it would on a non-virtual machine.

How does the Prometheus ecosystem work?

Prometheus is a free software application created in 2012 by SoudCloud. Since 2016, this application has been part of the Cloud Native Computing Foundation. It is used to monitor any purely digital time series by recording and processing them. It gathers, organizes, and stores metrics from infrastructure platforms, applications, and services, including HTTP endpoint metrics. Prometheus can also collect his own health metrics and monitor them. This ecosystem is therefore a solution that encompasses both the management of a multidimensional data model and the collection of scalable metrics while maintaining a certain operational simplicity.

It is developed with the Go programming language and has its own query language, promQL, which will facilitate the manipulation and analysis of the collected data. Prometheus is an ecosystem with several components working together for the purpose of generating reports on the performance of a system. Thus, Prometheus collects and stores metrics from applications that expose metrics in a plain text format through HTTP endpoints.

The diagrams below are complementary. They allow us to better understand the architecture of the prometheus ecosystem that we used to build our dashboard.



The first component of this ecosystem is the Prometheus server. He takes care of both storage of metrics, and planning of monitoring tasks by querying the data sources at a predefined polling frequency. Monitoring tasks are managed through the YAML configuration file, and configured using the directive “scrape config”.

In the diagram above we have an element that is in the server Prometheus who is called SD. Indeed, in order to know the targets from which the collection data will be made, Prometheus relies on several SD mechanisms. For instance, we can have file based SD than custom implementations SD can use. This is therefore possible by directly managing a YAML type file containing a whole list of targets. In addition, Prometheus also provides other SD implementations like Kubernetes or Amazon Elastic Compute Cloud.

The second component of this ecosystem is the Prometheus exporter. Exporters allow collecting metrics from a specific third-party system and rendering them by suite available for Prometheus servers to retrieve. The client libraries, used by applications, enable an HTTP endpoint where internal metrics are exposed and then collected by Prometheus servers.

The third component is the client libraries. In fact, applications cannot provide metrics only after adding instrumentation to their codes using the Prometheus client libraries directly.

The fourth component is Alertmanager. The main purpose of this element is to manage alerts sent by the Prometheus server by sending notifications through several means.

The fifth component is the pushgateway. This component allows you to retrieve the short-lived external services metrics. Finally, we have the Web UI component. It is a web application intended to view the result of a query in graph or table form, in real time promQL.

Proposed method

Evaluation

Conclusion and future work

References