Facultad de Ciencias

DISEÑO Y DESPLIEGUE DE UNA PLATAFORMA DE MONITORIZACIÓN PARA CENTROS DE PROCESAMIENTO DE DATOS

(Design and Deployment of a Monitoring Platform for Data Processing Centers)

Trabajo de Fin de Máster
para acceder al

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

Autor:
Jaime Iglesias Blanco

Directores:
Álvaro López García
Miguel Ángel Núñez Vega

Mayo – 2025

Resumen

Los centros de procesamiento de datos son infraestructuras críticas no solo para la provisión de servicios científicos y tecnológicos, sino también para el desarrollo de la sociedad actual, potenciado por la transformación digital a nivel global. A medida que estas infraestructuras crecen en tamaño y complejidad, impulsadas por una demanda cada vez mayor, resulta imprescindible contar con sistemas de monitorización que supervisen su rendimiento, eficiencia y sostenibilidad. En este contexto, el Trabajo de Fin de Máster tiene como objetivo el diseño, desarrollo y despliegue de una plataforma de monitorización escalable, modular y extensible para el centro de procesamiento de datos del Instituto de Física de Cantabria (IFCA-CSIC-UC), basada íntegramente en herramientas de código abierto.

La nueva plataforma sustituye al sistema previo incorporando una arquitectura moderna, eficiente y adaptable, diseñada para facilitar el mantenimiento, reducir los costes operativos y permitir la integración sencilla de nuevas fuentes de datos. Su diseño modular y escalable se adapta de manera óptima a entornos de computación heterogéneos, caracterizados por una gran variedad de equipamiento y tecnologías. La solución permite monitorizar en tiempo real el estado y utilización de máquinas físicas y virtuales, dispositivos de red, y software de gestión de la infraestructura, como los servicios cloud basados en OpenStack. Asimismo, permite realizar un seguimiento exhaustivo del consumo energético global de la instalación, así como a niveles más detallados, como racks, servidores, componentes específicos de los mismos o máquinas virtuales. A partir de estos datos, se calculan en tiempo real métricas e indicadores clave que permiten mejorar la eficiencia energética en las diferentes escalas medidas, además de ser utilizadas en este caso específico para proyectos europeos como *GreenDIGIT* y *AI4EOSC*, así como en otras iniciativas relacionadas con el impacto ambiental, como el plan de sostenibilidad del CSIC.

Tras su implementación y despliegue, la plataforma ha mejorado significativamente su eficiencia operativa, reemplazando el sistema anterior con una solución más escalable y flexible, que abarca más sistemas y equipamiento, ofreciendo una visión más detallada de la utilización, capacidad y disponibilidad de la infraestructura. A futuro, se planea expandir la plataforma con nuevos servicios e integrar capacidades para calcular el impacto ambiental en operación, consolidándola como una herramienta fundamental para apoyar la toma de decisiones y una gestión más sostenible del centro de procesamiento de datos.

Palabras clave: Centro de procesamiento de datos | Monitorización | Telemetría | Utilización de recursos | Consumo energético

This page intentionally left blank.

Abstract

Data processing centers are critical infrastructures not only for the provision of scientific and technological services, but also for the development of today's society, powered by the global digital transformation. As these infrastructures grow in size and complexity, driven by an ever-increasing demand, it is essential to have monitoring systems that oversee their performance, efficiency and sustainability. In this context, the objective of this work is the design, implementation and deployment of a scalable, modular and extensible monitoring platform for the data processing center at the Institute of Physics of Cantabria (IFCA-CSIC-UC), based entirely on open-source tools.

The new platform replaces the previous system with a modern, efficient and adaptable architecture designed to facilitate maintenance, reduce operating costs and allow easy integration of new data sources. Its modular and scalable design is optimally suited to heterogeneous computing environments, characterized by a wide variety of equipment and technologies. The solution enables real-time monitoring of the status and usage of physical and virtual machines, network devices, and key infrastructure management software, such as the cloud services based on OpenStack. It also allows exhaustive monitoring of the overall energy consumption of the installation, as well as at more detailed levels, such as racks, servers, specific components or virtual machines. From this data, key metrics and indicators are calculated in real time to improve energy efficiency at all scales that are measured and also used in this specific case for European projects such as *GreenDIGIT* and *AI4EOSC*, as well as in other initiatives related to environmental impact, such as the CSIC's sustainability plan.

Upon implementation and deployment, the platform significantly improved its operational efficiency, replacing the previous system with a more scalable and flexible solution that covers more systems and equipment, offering a more detailed view of infrastructure usage, capacity and availability. In the future, there are plans to expand the platform with new services and integrate capabilities to calculate the environmental impact in operation, consolidating it as a fundamental tool to support decision-making and a more sustainable management of the data processing center.

Keywords: Datacenter | Monitoring | Telemetry | Resource Usage | Energy Consumption

This page intentionally left blank.

Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a todos los miembros y compañeros del grupo de Computación Avanzada y e-Ciencia del Instituto de Física de Cantabria. Su apoyo, cercanía y colaboración han sido fundamentales para poder llevar a cabo este proyecto.

Muy especialmente, quiero agradecer a Álvaro López García, director de este trabajo, por su constante apoyo, su orientación y su disponibilidad en todo momento. Sus consejos y su experiencia han sido una guía clave a lo largo de todas las fases del proyecto. También quiero hacer una mención especial a Miguel Ángel Núñez Vega, codirector de este trabajo, por su implicación, ayuda y supervisión. Su cercanía, su implicación personal en cada detalle y su apoyo técnico y humano han sido determinantes para sacar adelante este trabajo.

Quiero extender también mi agradecimiento al resto de compañeros del servicio de computación del IFCA: Ibán Cabrillo, Aida Palacio y Pablo Izquierdo, por su ayuda constante y su predisposición para colaborar siempre que lo he necesitado. Su experiencia, su apoyo técnico y su implicación han sido clave para resolver los problemas que han ido surgiendo y para poder desarrollar el proyecto en un entorno estable y eficiente.

Del mismo modo, agradezco a todos los profesores del máster por todo lo aprendido en estos años. Su dedicación y el conocimiento que han compartido conmigo han sido fundamentales para afrontar con garantías los retos de este proyecto.

Por último, quiero dar las gracias de corazón a mi familia, mis amigos y mis compañeros del máster. Siempre han estado ahí, apoyándome, animándome y dándome fuerzas para seguir adelante, incluso en los momentos más exigentes. Su confianza en mí y su cariño han sido el impulso que he necesitado. Este logro también es suyo, y no puedo estar más agradecido por tenerlos a mi lado.

This page intentionally left blank.

Contents

List of Figures

List of Tables

Listings

Acronyms

AC Alternating Current 6, 38

API Application Programming Interface 7, 9, 11, 12, 20, 23, 40, 49

BMC Baseboard Management Controller 4, 5, 9, 63

CPU Central Processing Unit 4, 7, 9, 10, 15–18, 27, 29, 32, 34, 37, 38, 41, 47, 49, 61, 62, 65

CSIC Spanish National Research Council 1, 48

DC Direct Current 6, 38

DRAM Dynamic Random Access Memory 4, 10, 38, 40, 65

DSL Domain-specific language 20

EMI Electromagnetic Noise 37

GPFS IBM's General Parallel File System 8, 30, 49, 67

GPU Graphics Processing Unit 4, 8–10, 15, 16, 18, 23, 27, 32, 33, 41, 46–48, 62, 63, 65

HACS Home Assistant Community Store 35, 36

HPC High Performance Computing 1, 5, 8, 43, 49

IaaS *Infrastructure as a Service* 7

ICT Information and Communication Technology 1, 18

ICTS Unique Science and Technology Infrastructures 1

IFCA Institute of Physics of Cantabria 1–7, 14, 15, 21, 29–31, 40, 42, 43, 49, 57, 58, 61, 66

IPMI Intelligent Platform Management Interface 9, 27, 37, 38, 40, 49, 63

IT Information Technology 3, 4, 6, 9, 14, 21, 57

LDAP Lightweight Directory Access Protocol 14, 18, 42, 66

LQI Link Quality Indicator 36

MaaS *Monitoring as a Service* 2

MIB Management Information Base 9, 10, 34, 59

MSR Model-specific register 10

NaaS *Networking as a Service* 7

NIC Network Interface Card 4, 7

NVIDIA DCGM NVIDIA Data Center GPU Manager 8, 23, 27, 32, 46

OID Object Identifier 10, 34, 59

PCH Platform Controller Hub 10

PDU Power Distribution Unit 6

PSU Power Supply Unit 4, 6, 9, 37, 38

PUE Power Usage Effectiveness 2, 15, 17, 64

RAPL Running Average Power Limit 10, 14, 27, 37–40, 46, 49

RI Research Infrastructure 1

RSSI Received Signal Strength Indicators 36

SaaS *Software as a Service* 43

SDN Software Defined Network 7

SNMP Simple Network Management Protocol 9, 12, 26, 34

SoC System-on-chip 10, 38

SRS Software Requirements Specification 2, 16, 29, 33

SSO Single Sign-On 14, 18, 42, 66

TSDB Time Series Database 12, 13, 23, 26, 28, 31, 34, 41

UC University of Cantabria 1

UPS Uninterruptible Power Supply 6

VCS Version Control System 14

VLAN Virtual Local Area Network 5, 26, 29–31, 34, 42, 46, 61

VM Virtual Machine 4, 5, 7, 10, 15, 17, 21, 26–28, 30, 39–43, 46–49, 57, 58, 61, 64–67

WAL Write-ahead Log 13

This page intentionally left blank.

CHAPTER 1

Introduction

1.1 Motivation

The Institute of Physics of Cantabria (IFCA) is a joint center between the Spanish National Research Council (CSIC) and the University of Cantabria (UC), focused on basic science research. To this end, it has its in-house datacenter that offers various computing services, including HPC, Grid, Cloud and Artificial Intelligence to users and researchers both nationally and internationally [1]. This infrastructure is managed by the *Institute's Computing Service*.

Bearing this in mind, the use of a monitoring system is essential due to the infrastructure's criticality and complexity. Its heterogeneous nature, layered architecture and high user demand require continuous oversight to ensure performance, resource efficiency and fault detection. Real-time monitoring is crucial to detect bottlenecks and prevent failures that could impact research activities, making it indispensable for maintaining stability, reliability and efficiency.

Currently, there is a monitoring system in place, consisting of multiple agents running on the machines, InfluxDBv1 as the time-series database, which has been deprecated [2], and Grafana for visualizing system metrics. The database update introduced changes, including the migration of the primary query language from InfluxQL to Flux, a complex language that requires a complete overhaul of the queries in the dashboards to fully leverage its performance. Moreover, it appears that Flux will be deprecated in the next version, possibly due to its poor adoption [3]. Even after the update, the system still struggled with scalability, making it difficult to use and maintain. Therefore, a redesign of the system is proposed in this work, with a focus on reevaluating technologies and avoiding third-party solutions that tend to be overly complex or fail to meet all specific needs. Additionally, the emerging energy monitoring requirements outlined below are challenging to integrate into existing datacenter observability software.

On the one hand, a particularly relevant aspect of regulations and emerging initiatives within this sector is the focus on energy monitoring, aimed at assessing its impact and implementing countermeasures. For instance, the European Commission's *Data Centres in Europe – Reporting Scheme* [4], outlines the requirement for reporting on the energy performance and sustainability of datacenters within the European Union. Additionally, observing and reporting energy usage is a strategic interest for one of the lines of action described in the *Sustainability Plan from CSIC (2024-2026)* [5], which promotes the responsible and efficient usage of energy and water within the Information and Communication Technology sector, aiming to raise awareness about the environmental impact of computing RIs. Moreover, the document emphasizes that monitoring and measurement systems in this context have not yet been implemented, and that they are a critical component to support the actions that need to be taken to mitigate the environmental impact of Unique Science and Technology Infrastructures (ICTS).

On the other hand, there is a need for certain European projects in which the *Advanced Computing and e-Science Group* is involved, in incorporating data regarding electrical energy consumption. Starting with *GreenDIGIT (Greener Future Digital Research Infrastructures)*, the focus is on obtaining metrics from research infrastructures (RIs) to develop solutions that promote more sustainable and efficient computing. Secondly, *AI4EOSC (Artificial Intelligence for the European Open Science Cloud)* aims to measure the energy consumption and environmental impact of the platform being developed, while raising awareness among users about these factors.

1.2 Objectives

The leading purpose of this project is to build a scalable, extendable and modular observability platform for datacenters, focused on IFCA facilities, based on open-source existing projects, services and tools, to oversee the usage, power consumption and availability of its resources and key services, ensuring its adaptability to future needs and the integration of new data sources.

To achieve this, the project will design and develop a modular framework with multiple entry points for integrating new monitorable data producers, so that once a new source is connected through a defined entry point, acting as an agent for monitoring, the processes and tools for data processing, management, storage and visualization of these metrics are already settled.

Additionally, the project aims to replace and simplify the current monitoring platform, which is based on an outdated open-source software stack made up deprecated versions of the collectd and node-exporter agents, InfluxDBv1 and Grafana, to better adapt to the evolving needs of *IFCA Computing Service* technicians, for whom the platform should be intuitive, useful, clear and user-friendly. The new platform must minimize maintenance and operational costs, as well as simplifying the integration of new monitoreable assets, ensuring that the system operates autonomously without requiring additional infrastructure for its upkeep. Moreover, the platform seeks to optimize the usage of resources it consumes, including processing, networking and storage, while generating the least possible overhead on the datacenter.

In accordance with the goal of enhancing monitoring capabilities, the project drives the upgrading of the technologies used and the operational model by adopting a more flexible and resilient architecture. Delivering as a cloud-based *Monitoring as a Service* (MaaS) functionality but on-premises, with particular emphasis on data privacy and avoiding reliance on third-party solutions. Moreover, the platform will act as a technological enabler for obtaining key metrics, such as *Power Usage Effectiveness (PUE)* or *site carbon footprint* and will support other relevant projects and initiatives already in place.

1.3 Methodology

For the successful implementation of the project, it was necessary to select an execution and development methodology that would allow for flexible adaptation to emerging needs and ensure continuous delivery of functionalities. In this context, an incremental and iterative methodology was chosen. This methodology allows for the progressive integration of new features or improvements as needs are identified throughout the project, whether due to client demands [6] or own discoveries regarding previously unmonitored systems.

As new needs arise, a waterfall development model is applied, which involves sequential phases. The corresponding user stories are generated, which describe the required functionalities [7]. Based on these user stories, new requirements are drafted and incorporated into the Software Requirements Specification (SRS) document. Subsequently, all the necessary components are integrated into the architecture diagrams and the functionality is implemented and deployed. This mixed approach facilitates continuous delivery, as each iteration is completed with the release of new operational features, allowing the project to evolve in constant progression.

The incremental methodology is complemented with agile techniques, enabling a flexible and adaptive development process. Close collaboration with stakeholders and constant feedback are key elements to ensure that the final product meets the client's expectations and needs [8].

Project management is organized using a Kanban board, a tool that allows tasks to be structured and prioritized according to their progress, ensuring that tasks are completed successfully.

CHAPTER 2

Background and Technologies

This chapter presents the context in which the monitoring platform is implemented. It explains, from a monitoring perspective, both the fundamental concepts of a mainstream datacenter environment and the site-specific characteristics of the IFCA facilities, illustrated in Figure 2.1, which are essential for understanding the approach and requirements outlined in this report. Finally, the major technologies used for its development are described in detail.

2.1 Data Processing Centers Landscape: Infrastructure and Services

A data processing center, or datacenter, is a physical facility used to house high-performance computer systems that are interconnected through a high-speed network, alongside large storage systems and other computer infrastructure. These resources are utilized by organizations to collect, process, store and deliver large amounts of data [9]. Organizations often rely heavily on the applications, services and data contained within these centers, making them critical assets. This dependency means that their proper functioning is essential and therefore, it must be precisely monitored to ensure continuous operation and minimize potential disruptions.

Effective datacenter monitoring addresses this need by continuously tracking both operational metrics and environmental conditions using specialized tools and sensors. It also involves collecting data concerning the health of key components such as servers, storage devices, networking equipment and supporting infrastructure like power supply and cooling systems. Critical factors like uptime, performance, temperature, humidity and power usage are monitored, triggering alerts when any of these parameters deviate from predefined thresholds [10].

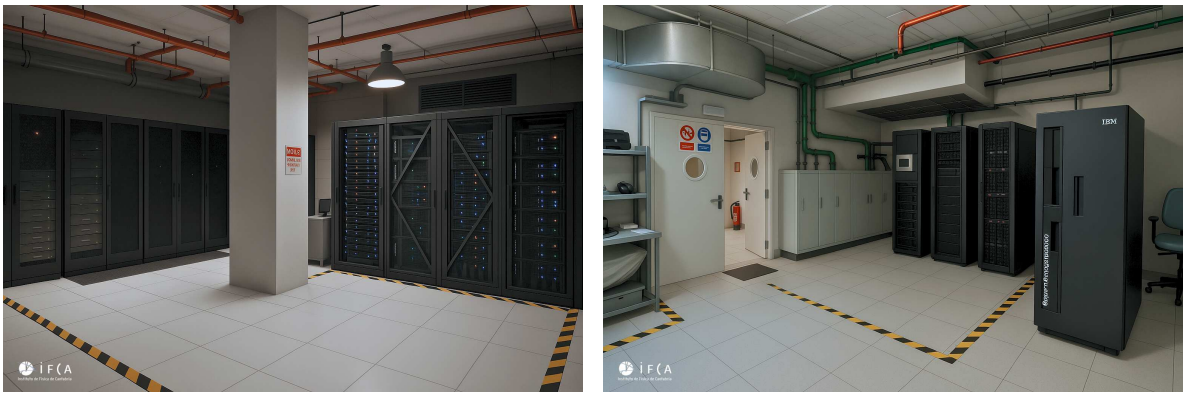


Figure 2.1: Data Processing Center at the Institute of Physics of Cantabria (IFCA) [1]
(Images created using generative artificial intelligence based on real photographs)

Additionally, in a research center like IFCA, having these monitoring systems in place within their own computing infrastructure allows for the capture of massive amounts of data on the operation of these systems. This data can then be analyzed to study their performance and develop customized solutions to improve their efficiency and reliability.

To gain a deeper understanding of how effective monitoring should be carried out, it is essential to first examine the core systems that make up a datacenter. At the hardware infrastructure level, these systems can be broadly categorized into two primary building blocks: Information Technology (IT) and Non-IT infrastructure, each contributing significantly to overall functionality.

2.1.1 IT Infrastructure

The computational or IT infrastructure consists of essential components that support workload execution, data processing, and communication, including servers, networking devices, and storage systems. These elements are standardized in form factor and usually stacked in rows within racks, optimizing space efficiency, cooling, and ease of maintenance [11].

2.1.1.1 Computing Equipment

A server is a high-performance computing system designed to provide services, process data and manage networked resources. Unlike consumer-grade computers, servers are optimized for reliability, scalability and continuous operation [9]. Their hardware composition typically includes multiple multi-core processors, each with its own dedicated main memory bank (DRAM), allowing for efficient parallel processing. Servers also feature multiple high-speed network interface cards (NICs) for data throughput and often specialized accelerators such as graphics processing units (GPUs) for intensive computing tasks. In terms of storage, servers are commonly equipped with multiple high-capacity and high-speed disks, often configured in redundant arrays (e.g., RAID), to support both performance and data reliability requirements.

To further enhance uptime and manageability, servers are also equipped with redundant fans and power supply units (PSUs) to ensure continued operation in case of hardware failure [9]. Furthermore, each server usually carries a management controller called the Baseboard Management Controller (BMC). The basic functions include monitoring of various hardware sensors, managing various hardware and software alerts, booting up and shutting down the server and providing remote management capabilities [11].

Servers come in various formats, such as rack-mounted, chassis-mounted, and blade servers, each with its unique design and characteristics. Rack-mounted servers are the most common in datacenters, designed to fit into standard 19-inch-wide racks, which can occupy anywhere from a single rack unit (1U) to multiple units, depending on its hardware configuration. Chassis-mounted servers are installed in a shared frame that occupies several rack units. Each node is often half the width of a standard server, so two can fit side by side in one rack unit. They also share parts like power supply units, which saves space and improve efficiency. Lastly, blade servers are even more compact, with multiple server blades housed in a single chassis, sharing PSUs, fans, backplane interconnect, and management infrastructure [11]. When monitoring server performance, health or energy consumption, it is important to account for the common elements shared by these server types to avoid reporting duplicated values and ensure accurate performance tracking across different server formats.

Taking this definition of a server into account, server monitoring is the process of continuously observing system resources, such as CPU usage, memory utilization, disk operations, network traffic or GPU usage, to ensure optimal operation and uptime. It specifically oversees the health and performance of both physical servers and hosted virtual machines (VMs), generating alerts and notifications for issues such as hardware failure or resource saturation [10]. It is also relevant to track power consumption at multiple levels to ensure optimal energy usage.

At IFCA, virtualization is a cornerstone of the computing infrastructure, with most of user-facing computing services running on virtualized environments powered by KVM-QEMU virtualization stack, where KVM acts as the hypervisor and QEMU provides hardware emulation. This approach enables efficient resource allocation, scalability and flexibility by deploying virtual machines on top of physical host servers. Consequently, monitoring must cover not only the physical hardware but also the virtual machines themselves, tracking their performance and resource usage to ensure reliability and service continuity.

From an administrative perspective, the computing infrastructure at IFCA is divided into two main types of machines: Datacenter hosts and user cloud machines. Datacenter hosts include physical servers and statically defined virtual machines, which are not dynamically provisioned and provide persistent management services. Cloud machines, on the other hand, are virtual instances created dynamically using a cloud orchestration platform, in this case OpenStack. This distinction influences how monitoring and management are approached. Host machines are typically organized using naming conventions based on hardware models, while cloud machines are grouped by OpenStack projects, each assigned a unique Virtual Local Area Network (VLAN). This network-based organization allows for easier implementation of access policies and filtering rules for the monitoring platform, without relying on machine names or metadata.

2.1.1.2 Network Equipment

The network infrastructure of a datacenter includes essential components such as multilayer switches, firewalls, routers and load balancers, which ensure device interconnection, security and performance. These devices typically have a management interface that provides features for monitoring their performance. The network is typically organized in a hierarchical topology, with top-of-rack switches handling internal rack connections and core switches managing inter-rack communication for scalability and redundancy [9]. At IFCA, Ethernet technology is the standard for general connectivity, while Infiniband is used for High Performance Computing (HPC) applications due to its low latency and high-speed data transfer capabilities [11].

Network monitoring is the process of continuously observing a datacenter network's performance, health and availability to help identify bottlenecks, performance degradation and potential security breaches within the network. It specifically oversees devices such as switches, routers, firewalls and load balancers, as well as overall network traffic and throughput [10]. As datacenter monitoring involves monitoring distributed systems, it is necessary to use the network itself for this purpose. A separate management network is used, which includes its own management switches [11]. This network connects to the management network interface of each server's BMC and to the management interface of each switch.

2.1.1.3 Storage Equipment

Storage in datacenters can be categorized into two main types: Private storage, local to individual computing tasks, and shared storage, which forms part of the global state in distributed workloads [9]. Private storage typically resides on local disks, is managed by a single process and lacks replication. In contrast, shared storage must be durable, resilient and accessible to many clients simultaneously, requiring a more complex and distributed storage infrastructure.

Physically, storage systems may be integrated with servers or provided through specialized units. High-performance storage is often housed in dedicated "storage towers" that abstract the underlying hardware complexity and provide seamless remote access. Alternatively, storage may be deployed in modular units commonly known as "storage bricks" or enclosures, either placed within rack slots or directly attached to servers [11]. In all scenarios, high-speed network connectivity is essential to ensure efficient data access.

Storage monitoring refers to the continuous oversight of these resources to maintain performance, manage capacity and ensure data integrity [10]. For distributed storage systems, monitoring is typically performed both at the machine level, tracking the physical hosts exporting the drives and at the service level, supervising the distributed software components that manage and expose storage to clients, as further detailed in Section 2.1.3.

2.1.2 Non-IT Infrastructure

On the other hand, the supporting infrastructure ensures that the computational systems operate within safe and stable environmental conditions. This includes various subsystems which are essential for maintaining the stability and efficiency of the datacenter.

2.1.2.1 Cooling Systems

Cooling is one of the primary subsystems that helps maintain the proper operating temperature for the IT equipment. At IFCA site, cooling is achieved through both free cooling and chillers. Free cooling utilizes external air during colder weather conditions to cool the servers without using additional energy, while chillers provide cooling during hotter periods, ensuring a stable temperature environment for the sensitive hardware.

2.1.2.2 Power Distribution Systems

Power supply is another crucial element of the datacenter infrastructure, which this work primarily focuses on. The power distribution system at IFCA facility follows an *alternating current (AC) architecture* [9], where three-phase AC high-voltage power enters the facility and is distributed through various circuits to different parts of the datacenter as shown in Figure 2.2.

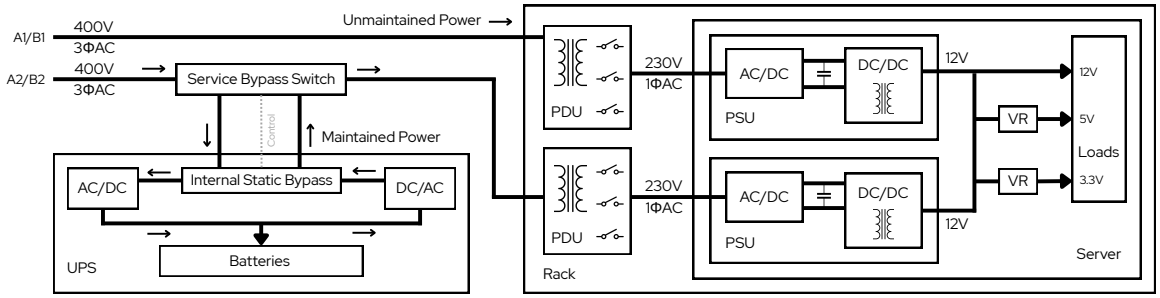


Figure 2.2: IFCA's Datacenter AC Power Distribution Architecture

There are four incoming 400V three-phase AC lines: Two (A1/B1) supply unmaintained power, feeding the load directly, while the other two (A2/B2) provide maintained power. The maintained lines are connected to uninterruptible power supply (UPS) units and a service bypass switch, which enables direct power supply during UPS maintenance [12]. The service bypass operates as an external selector between direct utility power and UPS-processed power. Within the UPS, an internal static bypass allows input AC to feed the output directly under normal conditions, improving efficiency without interrupting battery charging, which continues via a parallel AC/DC converter path. In case of a power outage, the batteries supply DC that is converted back to AC for the load. Regardless of the path, the output feeds the maintained power circuits. Both maintained and unmaintained power sources supply the multiple power distribution units (PDUs) installed in each rack. The PDUs distribute 230V single-phase AC power to the two redundant power supply units (PSUs) of servers, switches, and other equipment, where it is finally converted into low-voltage DC (12V, 5V and 3.3V) to power the internal electronic components.

Power monitoring is the tracking and analysis of electrical power usage, distribution and efficiency to help identify areas of high energy consumption and potential imbalances, allowing for optimized power usage and proactive measures against outages. It uses power meters and sensors to track power consumption of individual IT devices, specific power components, entire racks and cabinets, computer rooms and the overall datacenter facility meter [10].

Taking all of this into account, a comprehensive inventory of the IFCA's datacenter hardware equipment, including detailed specifications of each system, is provided in Appendix A.

2.1.3 Software Infrastructure

To gain a more comprehensive view of the datacenter's status, the services that manage it can be monitored, as they will provide crucial information about performance and operational health. In this context, the scope of the services to be monitored within this project is specifically focused on OpenStack, as it is the core service responsible for managing and orchestrating most of the resources within the datacenter environment. Given its critical role in resource allocation, provisioning and overall infrastructure management, it is essential to closely monitor its performance and operational health.

2.1.3.1 OpenStack

OpenStack is an open-source project offering a set of software components that provide common services for cloud computing infrastructures [13]. It controls large pools of compute, storage and networking resources, all managed through APIs, which employs standard authentication mechanisms. Client-side tools and libraries are available for interaction, along with a dashboard that lets administrators control services and users provision resources via a web interface.

OpenStack is broken up into services, allowing cloud administrators to plug or play components based on needs. In addition to standard *Infrastructure as a Service* (IaaS), additional modules offer orchestration, fault management, and service management to ensure high availability. As of this work, the following components are used in the IFCA's datacenter to support the institute's cloud computing service:

- **Keystone:** A service that handles client authentication, service discovery and multi-tenant authorization across the platform, via an API. It serves as the first point of interaction for users, granting access to other services after authentication. Keystone also manages identity resources such as projects, which represent ownership units, and domains, which act as containers for projects, users and groups, ensuring a unique namespace for each.
- **Nova:** This service provisions compute instances, including virtual machines, bare metal servers, and provide support for system containers. It operates through a collection of daemons (agents) running on Linux servers to deliver these services. It also supports the use of flavors, which define the resource allocation for instances.
- **Neutron:** It delivers *Networking as a Service* (NaaS) through an API that lets users configure network connectivity and addressing in the cloud. Neutron manages virtual networks, switches, subnets, ports, and routers in a software defined networks (SDNs) way, enabling communication between interface devices (e.g., vNICs) managed by other services (Nova).
- **Glance:** The Image service allows users to upload and access data assets intended for use with other services. Glance image services include discovering, registering and retrieving virtual machines (VMs) images. It provides a RESTful API for querying VM image metadata.
- **Cinder:** Is the OpenStack Block Storage service that provides block storage volumes to virtual machines. It virtualizes block storage management and offers a self-service API, to request and consume resources without needing to know the details of the device.
- **Placement:** Originally part of Nova, now functions as an independent component, to track resource inventories and usage through an HTTP API, supporting the management and allocation of resources across multiple OpenStack compute services. It tracks resource consumption through predefined classes, such as virtual CPUs, memory and disk space.
- **Horizon:** It is the official implementation of OpenStack's Dashboard, offering a web-based user interface for different services, simplifying administrative tasks and user interactions.

2.1.3.2 Other Services

In addition to OpenStack, other services of significant importance that are also relevant for monitoring include Slurm, which is widely used for job scheduling and resource management in High Performance Computing (HPC) environments. Furthermore, the distributed storage systems Ceph and GPFS are also crucial components within the datacenter. Monitoring these systems will contribute to maintaining optimal functionality and addressing any potential issues proactively. However, it is important to highlight that while these services are considered in the design of the platform, their implementation is not included within the scope of this work.

2.2 Monitoring Tools and Technologies

The different tools and technologies that compose the observability platform are described below. They are classified in several areas, depending on the stage of the monitoring process in which they are used.

2.2.1 Data Collection

This stage focuses on retrieving raw metrics and telemetry data directly from monitored systems, services, hardware components and sensors. The tools used for this purpose, typically agents, are deployed close to the data sources to ensure high accuracy in data acquisition.

2.2.1.1 Collectd

Collectd [14] is an open-source plug-in based Linux daemon for collecting periodically system and application performance metrics. Metrics are gathered from a variety of sources, including the operating system, applications, log files and external devices. The collected data is stored or made accessible via the network. These statistics can be used to forecast future system demand, identify performance bottlenecks and monitor systems. Collectd is being used on `v5.12.0`.

The main reasons for its choice are due to its metrics export model and its main features. First, it works following a push-based model, which allows each agent to send metrics autonomously to a monitoring server, without the need to configure anything on the monitoring server for each agent deployed. This is a differential factor to some alternatives such as node-exporter for Prometheus, which follows a pull-based model, so it is necessary to configure on the server each of the target agent URL to scrape them. This in a large deployment, with hundreds, even thousands of servers, as is the case in a datacenter, is not very scalable without the help of discovery services that automate this process. That is why this option was discarded.

In addition, its most relevant features are that it is written in `C` language for performance and portability. At the same time, it includes optimizations to handle hundreds of thousands of metrics. The daemon comes with over 100 built-in plugins, ranging from standard cases, such as system metrics, to highly specialized and advanced topics focused on specific technologies or devices. It provides powerful networking features and is extensible in numerous ways. Last but not least, it is actively developed and supported and well documented.

2.2.1.2 NVIDIA Data Center GPU Manager

NVIDIA DCGM [15] is an open-source suite of tools for managing and monitoring NVIDIA datacenter GPUs in cluster environments. It includes active health monitoring, comprehensive diagnostics, system alerts and governance policies including power and clock management. Version `4.0` is currently being used for this project.

The service is supported by *Linux* operating systems on *x86-64*, *ARM* and *POWER (ppc64le)* platforms. The installation packages include libraries, binaries and source code examples for interacting with the APIs.

A diagram of the typical deployment of the tools is shown in Figure 2.3. The service, named in the figure as "DC GPU Manager", communicates directly with the GPU driver. This last one can monitor the hardware at a low level and with a very fine grain of detail, at the compute kernel level. The service exposes monitored metrics and configuration commands through an API.

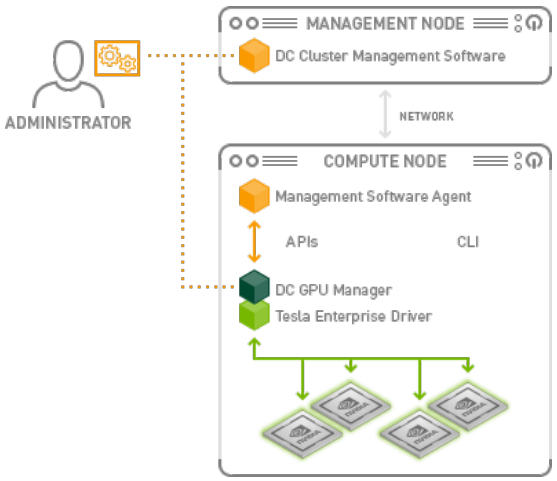


Figure 2.3: NVIDIA DCGM Operation Schema [15]

2.2.1.3 Intelligent Platform Management Interface (IPMI)

Intelligent Platform Management Interface (IPMI) is a set of specifications that provide a standardized interface for a hardware-level subsystem that offers remote platform management services [16]. In this context, it encompasses the monitoring of hardware systems in servers, such as system temperatures, voltages, fans and power supply units [17], among other subsystems, as shown in the diagram in Figure 2.4.

The specification is led by *Intel* [17] and was first published on September 16, 1998. It is supported by almost all the major IT infrastructure providers, including *Cisco*, *Dell*, *IBM*, *Hewlett Packard Enterprise*, *Lenovo*, *Fujitsu* and *Intel* itself, to name a few. Although IPMI is a standard specification, each manufacturer has its own platform implementation, meaning that the Baseboard Management Controller (BMC) may not always monitor the same hardware components, depending on the server manufacturer.

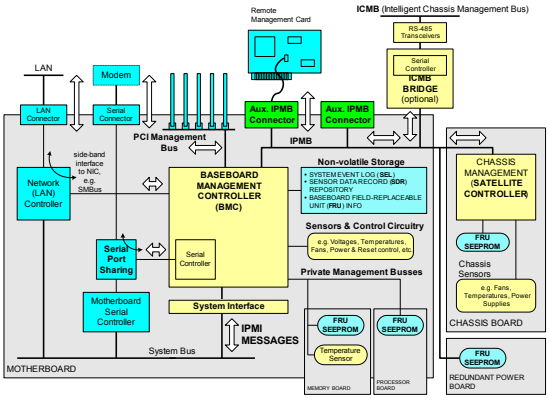


Figure 2.4: IPMI Block Diagram [17]

A key feature is the platform's independence from the CPU, boot firmware (BIOS or UEFI) and operating system. As a result, platform management functions remain available even when the server is off, as long as at least one power supply is connected and receiving power.

2.2.1.4 Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) is a TCP/IP application layer protocol for managing devices on IP networks [18]. It is based on a set of operations that offer the possibility to read and modify the configuration of the device, relying on SNMP configuration objects. This makes it one of the most commonly used technologies when it comes to network monitoring.

The information and configuration in each network device are stored in hierarchical databases called the Management Information Bases (MIBs), which consists of managed objects that provide insight into the device's performance, status and configuration. These objects can be either scalar (single instance) or tabular (multiple instances grouped) [19]. To identify each of

them, Object Identifiers (OIDs) are used, providing unique addresses across the entire hierarchy that allow the system to access specific objects on a device.

While devices may implement various MIBs, they all support a standard one: MIB-II [18], defined in *RFC1213* [20], which provides general TCP/IP management information. These MIBs are managed by an agent that handles incoming requests using the protocol's operations. This enables network monitoring tools to query the device, gather performance metrics and modify configurations when needed. This project uses the second version of the protocol.

2.2.1.5 Running Average Power Limit (RAPL)

Running Average Power Limit (RAPL) is a feature on Intel/AMD x86 CPUs, manufactured after 2012, that allows to set limits on power used by the CPU and other nearby components [21]. It also allows, as feedback, to report accumulated energy consumption of various system-on-chip (SoC) power domains, at very fine granularity and a high sampling rate [22]. RAPL energy data is exposed to the platform via host-software-accessible model-specific registers (MSRs).

RAPL supports multiple hierarchical power domains, each representing a physically meaningful region for power management, as shown in Figure 2.5. They are defined as:

- **Package (PKG):** Measures total socket energy consumption, including cores, integrated graphics and uncore components, such as last level caches.
- **Power Plane 0 (PP0):** Measures energy usage of overall the processor cores on the socket.
- **Power Plane 1 (PP1):** Tracks energy consumption of processor graphics (GPU) on desktop models.
- **DRAM:** Monitors energy usage of DRAM DIMMs connected to the socket integrated memory controller.
- **PSys:** Oversees thermal and power management of the entire SoC, including package domain, System Agent, Platform Controller Hub (PCH) and eDRAM.

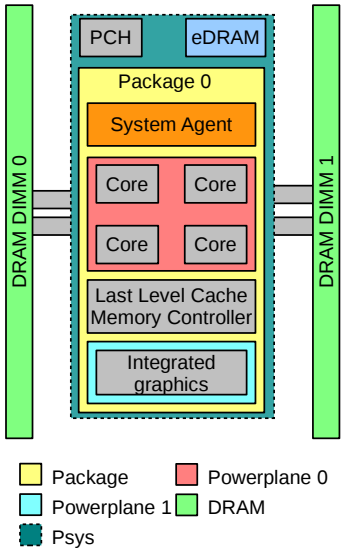


Figure 2.5: Socket Power Domains supported by RAPL [22]

For multi-socket server systems, each socket reports its own RAPL values independently [22].

2.2.1.6 Scaphandre

Scaphandre [21] is an open-source monitoring agent, dedicated to energy and power consumption metrics developed by Hubblo. Its purpose is to help measuring and thus understanding tech services energy consumption patterns. Scaphandre is being used for this project on its latest release: `v1.0.2`.

This agent uses RAPL registers to obtain energy measurements. The reason for using this agent instead of directly reading the registers is because measuring power consumption in tech service infrastructures is challenging, as it requires physical devices and custom-built data pipelines to collect useful metrics. Moreover, these methods only provide host-level power consumption data, lacking fine-grained insights into applications or processes. This challenge is even greater in virtualized or cloud computing environments. By using this agent, energy measurements for virtual machines are provided, as they are QEMU-KVM processes on host machines.

Another relevant feature of this agent is that it has been designed and implemented as optimally, light and clean as possible so that it does not produce any overhead to the system, leading to higher system utilization, resulting in higher energy consumption [21].

2.2.1.7 OpenStack Exporter for Prometheus

OpenStack Exporter [23] is an open-source monitoring agent designed to collect and expose data from OpenStack services in a format compatible with Prometheus. It interacts directly with the existing APIs of various components to gather relevant metrics[1]. In this project, OpenStack Exporter is utilized in its latest available version, `1.7.0`.

2.2.2 Data Processing

The following tools are responsible for gathering metrics from agents, devices, or sensors, processing them according to the platform's needs to ensure data consistency and readiness for further handling. Besides, during this process, the metrics are standardized to simplify integration, establishing agreements on type and unit conversion. Finally, when routed to the storage backend, all metrics are converted to a common format aligned with OpenTSDB notation.

2.2.2.1 Telegraf

Telegraf [24] is an open-source, plug-in-driven powerful server agent for collecting and reporting time series data. Written in Go and compiled as a standalone binary, it can be executed on any system with no external dependencies. Telegraf also contains in-memory metric buffers to maintain data collection if the downstream database is temporarily unavailable. Version `1.28.3` is currently being used for this project.

In addition to the features already mentioned, this technology was selected for its versatility as the main entry point for metrics, offering over 300 plug-ins that cover from IoT data collection to system telemetry. It supports flexible parsing and serialization of various input data formats and can serialize data into systems like InfluxDB or Prometheus.

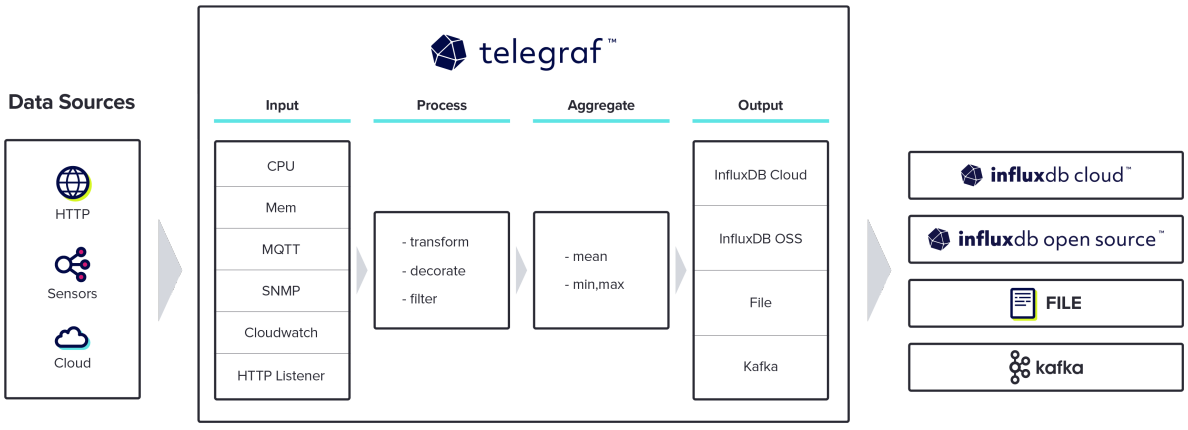


Figure 2.6: Telegraf Components [24]

As shown in Figure 2.6, Telegraf is composed of 4 types of modules, each with functionality and purpose within the metrics collection, routing and delivery process:

[1] OpenStack API Documentation: https://docs.openstack.org/api-quick-start/

- **Input plug-ins:** Collect metrics from communication protocols, services and third-party APIs. For this platform, the two types of input modules are a socket listener that receives metrics in collectd format and an SNMP collector that queries SNMP agents to gather network-related metrics.
- **Process plug-ins:** Transform, decorate and filter metrics before they are sent, sanitizing data as it arrives. In general, it classifies metrics by input labels and generates new ones, which serve as the primary mechanism to route metrics between plug-ins.
- **Aggregate plug-ins:** Create combined metrics, such as the average mean, minimum and maximum from the metrics that have collected and processed. The operations performed include scaling, unit conversion and derive additional metrics.
- **Output plug-ins:** Write data to a variety of datastores, services and message queues. For this proposal, a Prometheus server will be used, as explained in Section 2.2.3.1.

2.2.2.2 Home Assistant

Home Assistant [25] is an open-source home automation software designed for controlling devices, such as sensors, lights, switches, plugs, cameras and other smart home components. It emphasizes local control and data privacy, allowing users to manage their automation systems without relying on cloud services. With a highly customizable and extensible architecture, Home Assistant supports a wide range of integrations with third-party devices and platforms, enabling seamless automation and monitoring. Specifically, Home Assistant Container is used over Docker, a standalone container-based installation of Home Assistant Core, on version `2024.11.3`, as it allows for easy deployment and management of Home Assistant in an isolated environment.

2.2.3 Data Storage

The following stack of tools has the function of offering a high performance, widely available, large capacity, easily scalable and long-term storage system for the metrics collected through the different agents and aggregators available on the platform.

2.2.3.1 Prometheus

Prometheus [26] is an open-source, community-driven monitoring solution that collects metrics from monitored targets by scraping HTTP endpoints. These endpoints are often referred to as "exporters" when the component exposing the metrics runs in a separate process from the monitored target itself.

Prometheus stores its metrics as time series data, where each metric is a *float64* value, recorded with a millisecond-precision timestamp, along with optional key-value pairs known as labels. Every time series is uniquely identified by its metric name and optional labels [26]. This metric format is based on OpenTSDB notation. Version `2.48.0.rc.1` is currently being used.

Prometheus provides a functional query language called PromQL (Prometheus Query Language) that lets the user select and aggregate time series data in real time. It supports a wide range of operations, including mathematical expressions, statistical functions and temporal queries, enabling complex analysis and visualization of metrics across various dimensions.

By default, Prometheus stores the collected metrics for 15 days. Although the retention period can be extended by modifying configuration parameters, Prometheus is not designed to store and manage such a large amount of data on its own.

2.2.3.2 Grafana Mimir

Grafana Mimir [27] is an open-source, horizontally scalable, highly available, multi-tenant time series database (TSDB) for long-term storage for Prometheus, adding extended metric retention capabilities to the platform. Version `2.14.0` is currently being used for this project.

Mimir has microservices-based architecture. The system has multiple horizontally scalable microservices that can run separately and in parallel. These microservices are called components and are available through a single binary. The architecture is shown in Figure 2.7. It is divided into two backend paths: The write and the read path.

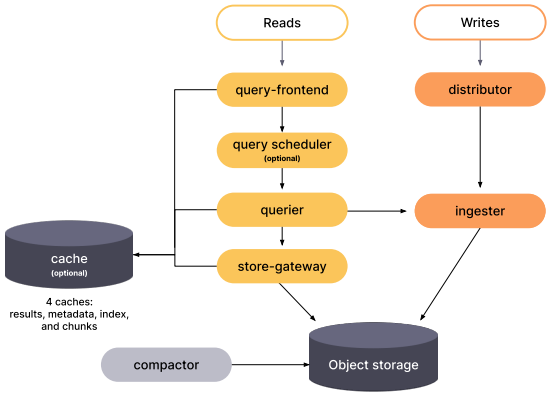


Figure 2.7: Grafana Mimir Main Components [27]

Its architecture ensures efficient data ingestion, storage and querying. Ingesters store incoming metrics, keeping them in memory and a write-ahead log (WAL) for recovery. Periodically, metrics are written to disk as TSDB blocks, uploaded to long-term storage and temporarily kept locally for quick access. To prevent data loss, the WAL should be on persistent disks. By default, each time series is replicated across three ingesters, with the Compactor merging blocks and removing duplicates to reduce storage. Queries enter through the query-frontend, which splits long-range queries and checks the cache. Cached results are returned instantly, while others are queued. Queriers pull queries, fetch data from store-gateways and ingesters, execute them and return results to the frontend for aggregation before sending them to the client. If used, the query-scheduler manages the queue instead of the frontend.

Some similar solution and commonly used is Thanos[2]. It has the same purpose but is oriented to be deployed following a microservices architecture application on a Kubernetes cluster. This alternative was not pursued because it increased the level of deployment complexity in excess for our platform, as it does not need to be deployed in a clustered environment for now. However, Grafana Mimir offers this possibility to obtain and guarantee the maximum system scalability possible, if required in the future.

2.2.4 Data Visualization

The last step of the pipeline of the monitoring platform is to show in an organized, concise, clear and visual way, the different metrics collected, with the objective of informing, controlling the systems and supporting decision-making with frequently updated real data. This is achieved by using the following software product.

2.2.4.1 Grafana

Grafana [28] is an open-source software that enables querying, visualizing, alerting and exploring metrics, logs and traces across different storage systems. Its data source plugins support connections to a wide range of platforms, including time series databases like Prometheus and CloudWatch, logging tools such as Loki and Elasticsearch, relational and NoSQL databases like PostgreSQL and InfluxDB, as well as CI/CD systems like GitHub and Jenkins.

[2]Thanos: Open source, highly available Prometheus setup with long term storage capabilities: https://thanos.io/

Grafana OSS provides powerful capabilities for building dynamic dashboards with detailed graphs, visualizations and alert rules. It also supports integration with multiple authentication systems via Single Sign-On (SSO) and Lightweight Directory Access Protocol (LDAP), making it suitable for secure, large-scale deployments with multiple users, centralized access control and collaborative dashboard management. In this platform, version `11.2.0` is used.

2.2.5 Other

Finally, we describe other technologies and tools that are not directly related to any stage of the monitoring process, although they have been used to build and manage the platform, i.e. deployment in the IFCA's datacenter.

2.2.5.1 Python

Although the platform is primarily built upon pre-existing software components, some programming has been necessary to address specific requirements. The custom codebase is written in either Python or Starlark[3], a Python dialect commonly used as a configuration language. Python is mainly employed to develop custom plug-ins for collectd, such as modules for reading the RAPL interface. On the other hand, Starlark is used within Telegraf to define custom rules for metrics processing [29]. The Python version used may vary across agents depending on the local environment, but in general, Python `3.8` is the most commonly deployed.

2.2.5.2 Puppet

Puppet [30] is an open-source configuration management tool that automates the process of managing infrastructure. It allows system administrators to define the desired state of systems and ensures configurations are consistently applied across servers. Using a declarative language, Puppet enables users to specify resources such as packages, services and files, along with their required states. The files that define these configurations are called manifests.

Operating in a client-server model, Puppet uses a central master to manage configurations and send them to agents running on managed nodes. These agents periodically check in with the master to receive and apply updates. Puppet is highly extensible, supporting a variety of operating systems and platforms and integrates with monitoring, logging and orchestration tools, helping IT teams automate tasks, enforce policies and maintain consistency across complex infrastructures. Puppet is utilized in its version `6.28.0`.

2.2.5.3 Bash Scripting

Bash scripting, which is a command-line shell scripting language commonly used in Unix-like systems, enables the creation of automation scripts for deploying platform agents onto specific areas of the infrastructure, where Puppet does not manage the automatic deployment process.

2.2.5.4 Git

The project has been developed using a version control system synchronized with a remote repository in gitlab.ifca.es. Different types of files of the project are being hosted there: The platform architecture and its modeling diagrams, the deployed services configuration files, the source code of the different custom plug-ins used, several Grafana's dashboards templates.

[3] Starlark Language: https://github.com/bazelbuild/starlark

CHAPTER 3

Requirements Specification

The requirements specification phase is a fundamental step in software development, as it lays the foundation for building a system that aligns with its intended purpose and ensures a smooth development process. This chapter outlines the platform and details the requirements identification and analysis process. Through user stories, key functionalities and constraints were identified and refined into a formal specification, which serves as a clear and consistent reference for the software design and implementation.

3.1 Platform Overview

The monitoring platform provides a comprehensive solution for tracking and analyzing key infrastructure metrics within the datacenter. It integrates data collection from diverse sources, enabling real-time insights and long-term analysis to support operational efficiency, resource optimization and sustainability goals.

Figure 3.1: Platform Logo

3.1.1 Key Features

The platform offers a range of capabilities to monitor, analyze and optimize datacenter operations. These features provide in-depth visibility across multiple areas:

- **Machines statistics:** Monitors the status and availability of machines, tracking CPU and GPU usage, network interfaces bandwidth, main and GPU memory usage, swap and disk usage to ensure optimal performance and detect potential failures.
- **Network performance:** Tracks network switches and interfaces utilization and availability, ensuring real-time visibility into link status, bandwidth and error rates.
- **OpenStack services state:** Provides insights into the status and performance of core OpenStack services like Keystone, Nova, Neutron, Glance and Placement.
- **Power consumption tracking:** Measures power usage at different levels of detail, from the whole datacenter power distribution grid to individual assets, such as chillers or machines, specific components like CPUs or GPUs and even virtual machines.
- **Energy efficiency metrics:** Computes indicators like *Power Usage Effectiveness (PUE)* across different scopes, providing actionable insights to enhance energy efficiency.

The platform's modular design ensures scalability and extensibility, allowing for seamless integration of new services and sensors as infrastructure evolves. Its data-driven approach empowers administrators to proactively manage resources, minimize downtime and improve the overall sustainability of the datacenter.

3.2 Requirements Identification and Analysis

To undertake a more detailed analysis of the software requirements, the different metrics to be incorporated and the challenges associated with the observability platform, several collaborative meetings were held with stakeholders. Among them were system administrators from *IFCA's*

Computing Service, who are responsible for managing the infrastructure and providing computing services to researchers. In addition, members of the *Advanced Computing and e-Science Group* who contributed to the development of the energy consumption and environmental impact reporting capabilities of the *AI4EOSC* project, as well as those overseeing the collection of infrastructure metrics for the *GreenDIGIT* project, also attended. The results of these discussions were captured in user stories, which provided a structured way of documenting the identified needs and functionalities.

User stories are a technique in agile development to define system requirements from the user's perspective, focusing on the value the software will deliver. They involve three essential aspects: A *Card* with a concise description of the requirement, a *Conversation* to refine acceptance criteria through collaboration between the development team and the product owner and a *Confirmation* where the product owner ensures the requirements are correctly understood. This process promotes clarity, collaboration and flexibility throughout development [31, 7]. The complete set of user stories cards is attached in Appendix B. Based on these, the Software Requirements Specification (SRS) is extracted as described below.

3.3 Software Requirements Specification

3.3.1 Functional Requirements

The functional requirements describe specific functions of the system or its behaviors in response to certain inputs, defining what the software must do to fulfill its purposes [32]. It specifies expected outcomes, interactions and system responses, ensuring that the software meets user needs and operational goals. Table 3.1 lists the complete set of functional requirements for the monitoring platform.

Table 3.1: Functional Requirements

ID:	Description:
FR-01	The platform must collect and store statistics for each machine operating system, including hostname for identification, host status, uptime, CPU usage, memory usage, swap usage, disk usage, network activity and logged-in users.
FR-02	The platform must continuously monitor the performance and link status of all network switches, displaying key metrics such as the device's uptime, CPU and memory usage and temperature. Each switch must be identifiable by its name or management IP.
FR-03	The platform must monitor and report on each network interface's properties, including identification data, configured link speed, interface type, link status, bandwidth usage, aggregate link usage over time and packet error rate.
FR-04	The platform must monitor and report the utilization, memory usage and power consumption of NVIDIA GPUs, identifying each device by its hostname and UUID. Additionally, the platform should track the GPU temperature, if possible.
FR-05	The platform must automatically collect and store historical data from the Schneider Electric sensors ecosystem, with immediate monitoring of any newly connected sensors as soon as they are linked to the system.

Table 3.1: Functional Requirements

ID:	Description:
FR-06	The platform must provide real-time power consumption in watts for each physical machine in the datacenter, using accurate sensors and methods to ensure measurements reflect actual energy usage.
FR-07	Data regarding power consumption, in watts, of each virtual machine used to deploy the *AI4EOSC* platform should be collected and stored.
FR-08	Energy efficiency metrics, such as PUE, must be calculated by the platform based on the consumption data collected across all scopes.
FR-09	The platform must track and display the live status of each OpenStack service.
FR-10	For Keystone, the platform must allow the visualization of multiple projects relevant information and their statuses, along with displaying a counter of registered users.
FR-11	For Nova, the platform must monitor the flavors and their characteristics, track the resource usage by each project, display the administrative status of the VMs and show the status of the hosts where the VMs are deployed.
FR-12	For Neutron, the platform must monitor the different networks and their characteristics, track the usage of public floating IPs and subnets and display the status of the network agents on the host machines.
FR-13	For Glance, the platform must monitor the size of each image and the overall size, as well as display their properties, including status, visibility and owner.
FR-14	For Placement, the platform must display the total usage of central processing units (CPUs), memory and storage, as well as usage per host.
FR-15	The status of all OpenStack service agents across the host machines should be collected and stored, displaying the current status and tracking the evolution of each agent over time.
FR-16	The platform must allow easy extension to monitor additional external critical services for the management and operation of the infrastructure.
FR-17	The platform should support monitoring of local services deployed on individual machines, such as proxies and databases.
FR-18	Metrics shall be presented in the most visually effective and comprehensible way, utilizing multiple chart types to enhance clarity and ease of analysis.
FR-19	The dashboards shall support data visibility customization, enabling administrators to filter metrics and adjust the time range according to their monitoring needs.
FR-20	Platform must ensure data visibility is restricted: VM metrics must be visible only to the respective owner and computing service operators, while data from the entire datacenter infrastructure should be accessible only to system administrators.

3.3.2 Non-Functional Requirements

The non-functional requirements are system qualities that guide the architecture of the product and often serve as design constraints. A widely used approach to define them is using the product quality model, as defined in *ISO/IEC 25010 standard* [33]. This groups the Information and Communication Technology (ICT) quality properties into eight characteristics: Functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability. Each of them is composed of various related subcharacteristics. The following table (3.2) shows the non-functional requirements, classified according to this quality model.

Table 3.2: Non-Functional Requirements

ID:	Type:	Description:
NFR-01	Security Confidentiality	Access to the monitoring platform must be through an existing Single Sign-On (SSO) authentication system using LDAP.
NFR-02	Maintainability Analyzability	All the systems used for monitoring must have a log file to supervise the status and detect errors in all the running services.
NFR-03	Portability Installability	The agents and services must be deployed and configured using a configuration and automation management tool: Puppet, if possible. Otherwise, a shell script must be provided.
NFR-04	Compatibility Interoperability	The platform must be able to monitor a heterogeneous environment, consisting of machines with both Intel and AMD CPUs, NVIDIA GPUs, as well as multiple switches brands.
NFR-05	Compatibility Interoperability	The platform must be able to run on Linux operating systems, specifically on distributions of the Debian and Red Hat families.
NFR-06	Compatibility Interoperability	The monitoring platform agents must support deployment in both virtualized and bare metal environments with minimal configuration changes.
NFR-07	Performance Time behavior	Data should be collected at intervals with a period of at least 30 seconds for infrastructure equipment and 5 minutes for services.
NFR-08	Reliability Availability	The data collected must persist and be accessible through the platform for a minimum period of 3 years.
NFR-09	Usability Operability	The monitoring dashboard must provide real-time visualizations of collected data, minimizing delays as much as possible.
NFR-10	Reliability Fault Tolerance	The system must continue operating in a degraded mode if one or more components fail, ensuring that critical monitoring functionalities remain available.
NFR-11	Security Confidentiality	All monitoring data must be stored entirely on site, ensuring no external data storage or cloud-based dependencies.
NFR-12	Maintainability Modifiability	All components of the monitoring platform must be based on open-source software with active community support to ensure long-term sustainability and adaptability.

CHAPTER 4

Design and Architecture

Design and architecture establish the foundation for building a scalable and maintainable system. A well-structured software architecture, along with clear modeling diagrams, enhances team communication, improves decision-making and helps identify potential risks throughout the software development process. This chapter describes the modeling technique used and the architecture designed for the monitoring platform for data processing centers.

4.1 Modeling Method: C4 Model

The C4 model is a streamlined graphical notation technique for modeling the architecture of software systems. Created by Simon Brown between 2006 and 2011, it aims to modernize software modeling techniques in alignment with modern software development trends and agile methodologies [34]. It follows a hierarchical decomposition based on four levels of abstraction:

- **Systems:** It is the highest level of abstraction, representing an entity that delivers value to its users, whether human or automated.
- **Containers:** Contexts or boundaries where code is executed, or data is stored. Containers are essential for the functioning of the overall software system.
- **Components:** Logical groupings of related functionality encapsulated behind well-defined interfaces. Unlike containers, components are not independently deployable.
- **Classes:** Highly detailed elements that can be easily transformed into code and data structures. Normally either UML classes, interfaces or entity-relationship models are used.

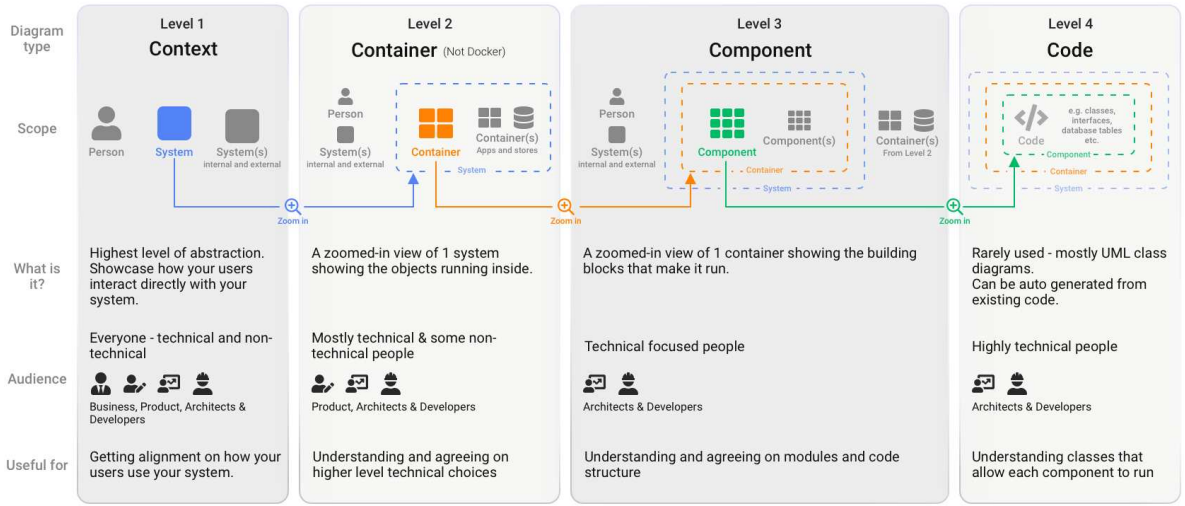


Figure 4.1: C4 model infographics: Types of diagrams, scope, description, audience and usage [35]

A key advantage of this graphical modeling technique is its adaptability. It allows adjusting the level of detail according to the audience, technical expertise and architectural needs. To achieve this, it employs four hierarchical diagram types, as shown in Figure 4.1:

- **Context:** Provides a high-level overview of how the software system interacts with external entities, such as users, external systems and services. It helps to define system boundaries and clarify its role within a larger ecosystem.
- **Containers:** Zooms into the system, displaying its high-level technical building blocks (e.g., applications, databases, APIs) and their interactions. It illustrates how different containers communicate and what technologies they use.
- **Components:** Details the internal structure of a container, showing its constituent components, their responsibilities and their interactions. This helps in understanding the modular organization of the system and the dependencies between different components.
- **Code:** Represents the internal implementation details of a component, typically using UML diagrams or similar notations. It provides a low-level view of classes, methods and relationships to help developers understand and navigate the codebase.

4.2 Design of the Monitoring Platform for Datacenters

To model the monitoring platform, the collaborative diagramming tool *IcePanel* [35] was used. This tool enables interactive, drag-and-drop diagram creation following the C4 model principles. The platform model is also available in the project repository[1] in two formats: As domain-specific language (DSL) code generated using *Structurizr*[2], which follows a *diagrams-as-code* approach and as the visual views generated by the tool itself.

4.2.1 Context Diagram

The datacenter environment consists of two major system groups and two user types, as illustrated in Figure 4.2. The relationships among these entities from the monitoring platform's perspective are described below:

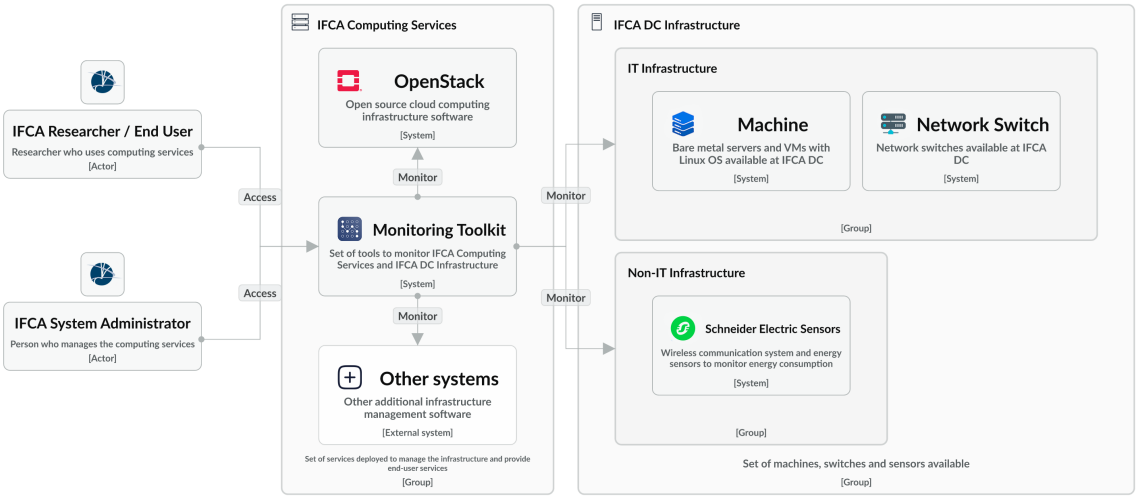


Figure 4.2: Monitoring Platform System Context Diagram

The services encompass both infrastructure management and user-facing computing services. For this first approach to the project, only OpenStack is considered, though additional services may be integrated in the future. The monitoring platform itself is part of this service stack.

[1] Platform architecture model: https://gitlab.ifca.es/iglesiasj/ifca-monitor-toolkit/-/tree/main/architecture
[2] Structurizr: https://structurizr.com/

The datacenter infrastructure, where these aforementioned services are deployed, includes IT equipment, ranging from physical servers to cloud-based virtual machines, as well as networking equipment such as switches. Additionally, non-IT infrastructure is monitored via a Schneider Electric sensor grid, which tracks energy consumption across various facility assets.

The platform has two main actors:

- System Administrators manage the computing services and infrastructure, using the platform to monitor system health, track performance metrics and resolve any issues that arise.
- IFCA Users and Researchers oversee the status of their deployed resources, such as cloud virtual machines, ensuring availability and optimal performance for computational tasks.

4.2.2 Container Diagrams

4.2.2.1 IFCA Computing Monitoring Toolkit Container Diagram

As shown in the platform's container diagram (Figure 4.3), the system is composed of eight containers, part of the main open-source software used to build the platform. Some of these containers have been described in Section 2.2, so their main functions are straightforward.

Starting with the less user-facing containers, these connect to the datacenter systems to ingest or request metrics gathered by agents deployed within the infrastructure. In addition to collecting this data, they also handle processing, filtering and labeling the metrics. Telegraf plays a dual role by receiving the reported metrics from the machines and requesting specific metrics from network switches. Home Assistant is responsible for obtaining data from the Schneider Electric sensors, acting as an intermediary to ensure the smooth integration of electrical system data into the monitoring framework. Prometheus scrapes metrics from various services monitored by compatible agents, known as Exporters, such as those used with OpenStack. It also aggregates metrics from Telegraf and Home Assistant, creating a centralized and unified metrics system.

This architecture uses a hybrid metrics intake model: A pull-type model via Prometheus for static services and a push-type model via Telegraf and Home Assistant for dynamic systems. Indeed, Telegraf and Home Assistant act as a static target for Prometheus. This hybrid architecture, considering the wide cardinality of the different monitored systems, provides scalability and compatibility in the data collection and delivery process using push-type agents and Telegraf and the optimized backend for metrics management and storage using Prometheus and Mimir.

Continuing with the last-mentioned container, Grafana Mimir stores the time-series data collected by Prometheus in a more efficient and compact way since Prometheus is not built for long-term storage. Mimir saves this data in a container, either using an object storage system (S3 compatible) or a local/remote file system. Since there's enough free disk space on the machine where the platform will be deployed, it was decided to use the local file system storage option. These metrics are then accessed and displayed through Grafana, which queries Mimir for the data and shows it on different dashboards with charts like timelines, stats, histograms and gauges. Additionally, there is an SQLite database, which holds static system data that does not change often and is not needed for long-term tracking. This database is manually updated. In practice, it is used for filtering variables values in the dashboards.

Lastly, web access to Grafana is routed through a NGINX reverse proxy. This helps prevent exposing specific service ports through the public IP, enhancing security by masking internal services behind a single-entry point. It also allows for SSL termination, offloading the encryption/decryption process from Grafana and ensuring secure communication over HTTPS.

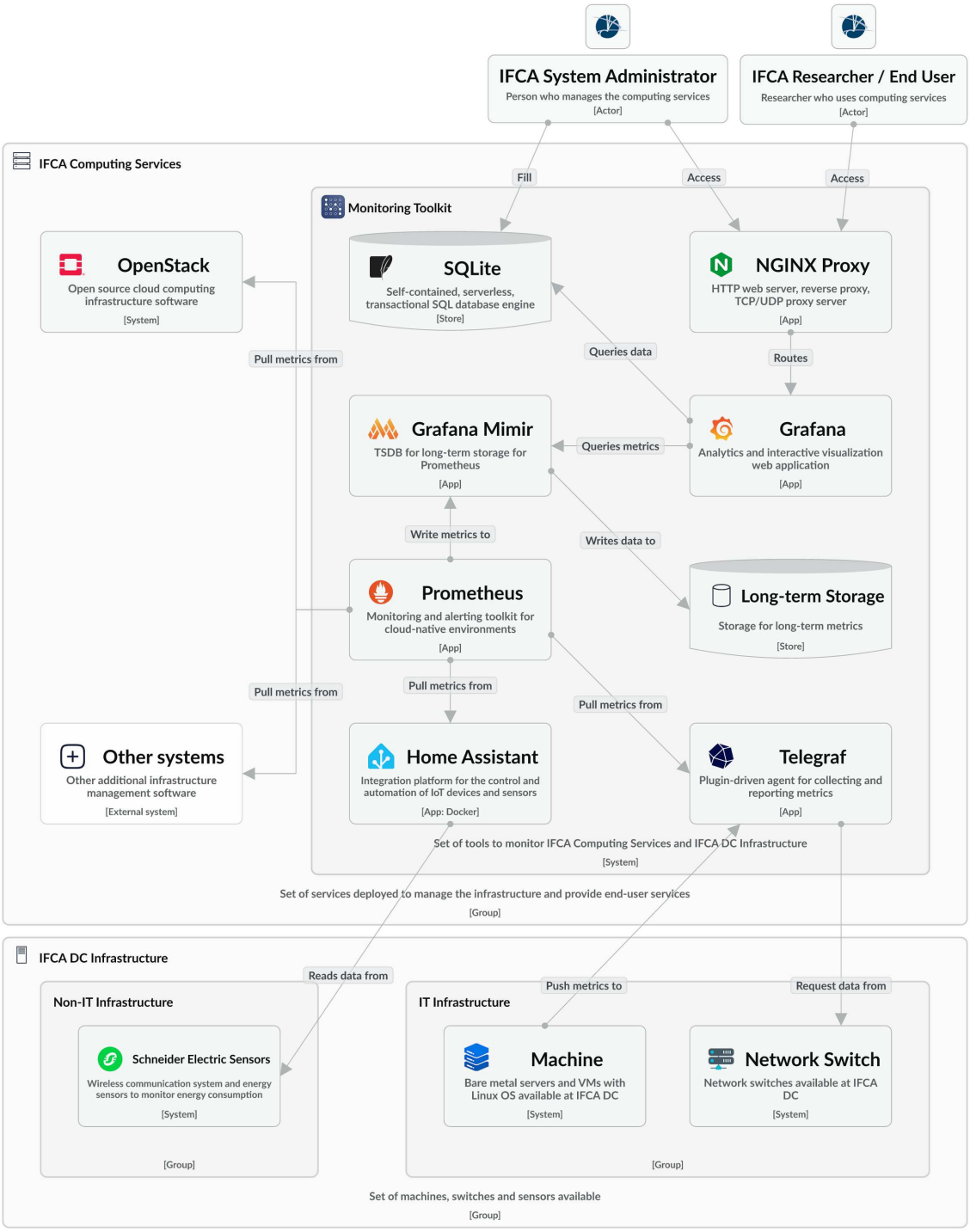


Figure 4.3: IFCA Computing Monitoring Toolkit Container Diagram

4.2.2.2 Machine Container Diagram

All the machines available in the datacenter have been instrumented with a series of agents for monitoring, as shown in Figure 4.4. This setup is flexible, with the number of containers varying based on the machine's specific characteristics. Each machine runs a collectd daemon, which collects performance metrics from the operating system, hardware components and services. These metrics are then forwarded to Telegraf for processing. The most flexible containers, which may or may not be present depending on the machine's configuration, are as follows:

If a machine has a GPU installed, it also includes NVIDIA's monitoring service, NVIDIA Data Center GPU Manager (NVIDIA DCGM), which communicates with the GPU driver via a proprietary protocol. Collectd retrieves the metrics from this service as well.

Unlike virtualized environments, physical machines can directly monitor energy usage because they have access to the underlying hardware. To measure this, a Scaphandre agent is installed on the physical machines, which allows monitoring not only the overall energy consumption of the machine but also the consumption associated with specific processes running on the host. These metrics are also requested by the collectd agent from the Scaphandre agent.

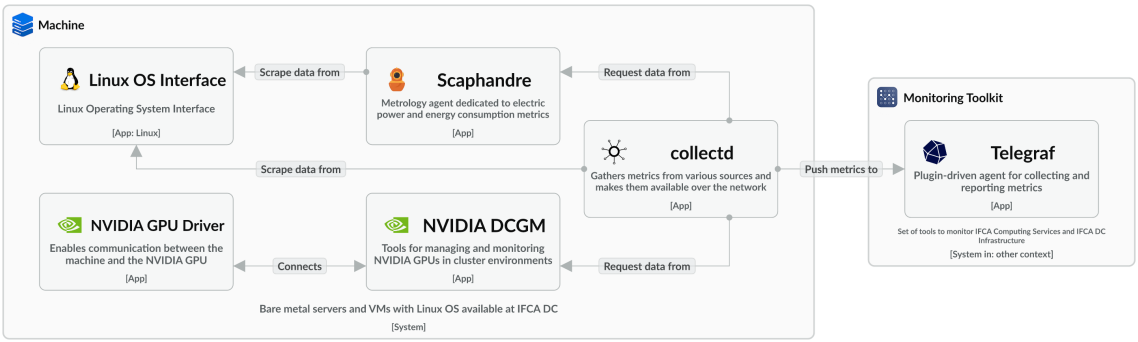


Figure 4.4: Machine Container Diagram

4.2.2.3 OpenStack Container Diagram

The monitoring of all OpenStack components is done through an OpenStack Exporter, as shown in Figure 4.5. This exporter acts as an intermediary agent that periodically makes requests to the APIs of various OpenStack components (such as Nova, Neutron, and Cinder) to collect real-time metrics about their performance and health. The gathered data is then transformed into the OpenTSDB format, which is optimized for time-series storage. Once the data is processed, it is exposed via HTTP endpoints, allowing a Prometheus server to pull the metrics.

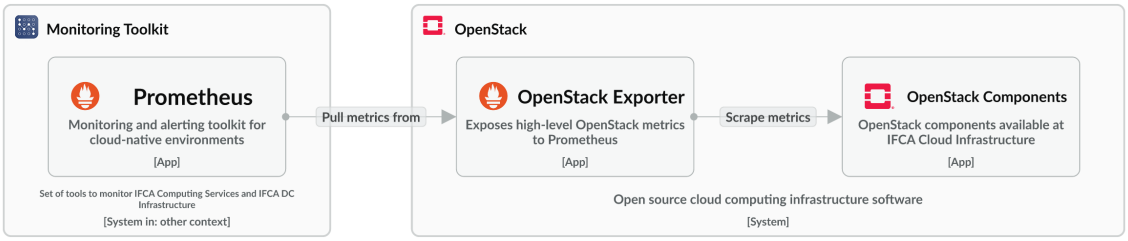


Figure 4.5: OpenStack Container Diagram

4.2.2.4 Schneider Electric Sensors Container Diagram

The Schneider Electric sensor network follows a hub-and-spoke topology, consisting of multiple sensors models, referred to as PowerTags, and a central communications hub, also known as the gateway. As illustrated in Figure 4.6, the gateway interconnects these sensors and transmits measurement data over a LAN network. Communication within the sensor network is handled via the Modbus protocol, while the connection between the gateway and the LAN is established using Modbus/TCP, which encapsulates Modbus messages within TCP/IP packets. These packets can be interpreted by the Home Assistant instance.

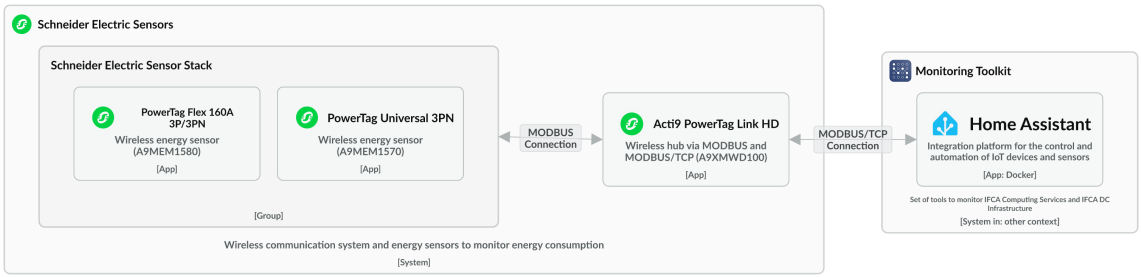


Figure 4.6: Schneider Electric Sensors Container Diagram

4.2.3 Components Diagrams

4.2.3.1 Grafana Components Diagram

The Grafana component diagram is straightforward, as it primarily illustrates the relationships between the application and its data sources. As shown in Figure 4.7, Grafana uses two connectors to interact with the main data sources of the platform. The first connector allows Grafana to execute queries using PromQL, but it is important to note that, for performance reasons, the queries are executed against Grafana Mimir rather than directly on Prometheus. The second one enables Grafana to execute SQL queries to the SQLite database, which is used for storing non-time-series data. These connectors are shown as components in the diagram because they are plug-ins, offering flexibility and modularity to Grafana and enabling seamless integration with future data sources.

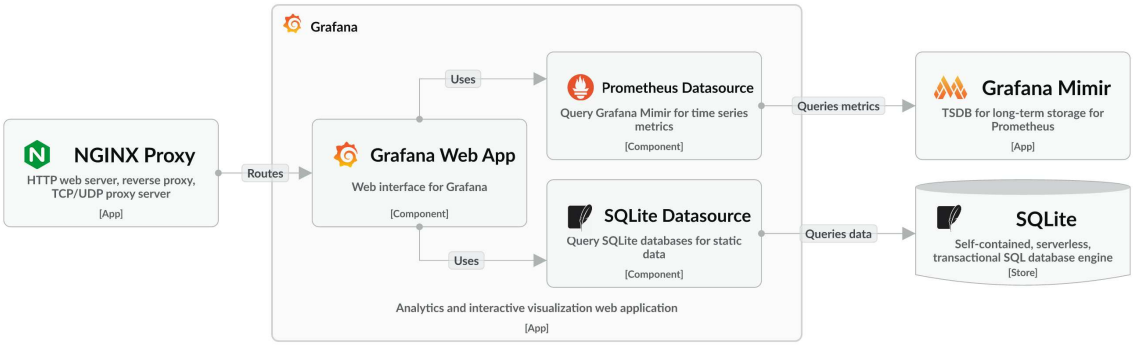


Figure 4.7: Grafana Components Diagram

4.2.3.2 Grafana Mimir Components Diagram

Grafana Mimir, as previously explained in section 2.2.3.2, consists of six distinct components: The Query-Frontend, the Querier, and the Store-Gateway, which together form the read path and enable Grafana to retrieve metrics efficiently. The Distributor and Ingester components are part of the write path; their primary function is to receive metrics samples from Prometheus and persist them to disk. Finally, there is the Compactor, which optimizes the storage of time-series data on disk. The relationships between these individual components are illustrated in Figure 4.8.

Although the components and communication between them are already established and implemented in the software itself, it is necessary to consider their function and relationship within the system in order to configure and deploy each of them, according to the needs of

the platform. This is, for example, to configure how many instances of each component exist to have a high availability, how much is necessary to dimension the system to support massive data ingestion, how to configure the data retention and the connection with the storage system and so on. This is why these components are defined in the diagram, even though it is not necessary to define them and their built-in relationships.

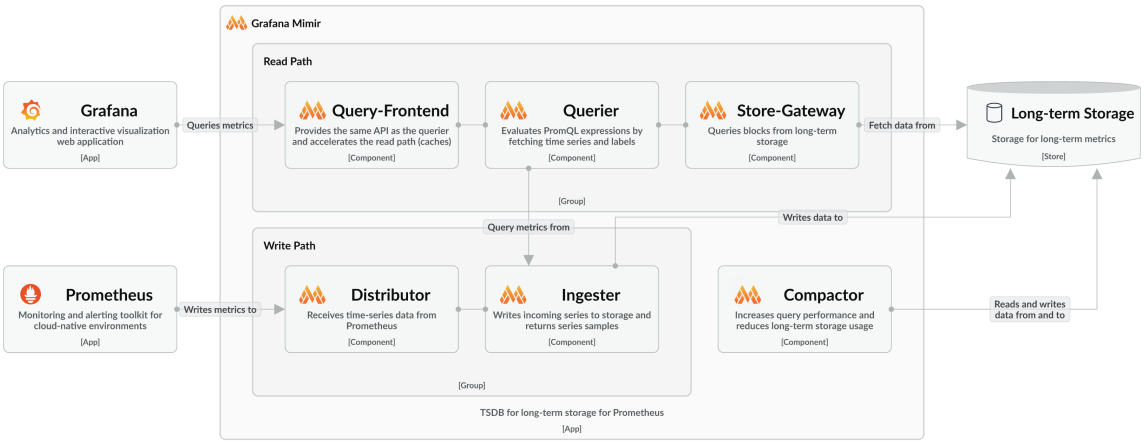


Figure 4.8: Grafana Mimir Components Diagram

4.2.3.3 Prometheus Components Diagram

Figure 4.9 shows the Prometheus components. These components are not functional blocks of this software but represent the different configuration blocks necessary for it to work. Specifically, these are called jobs and with a certain configuration, they are in charge of scraping the different targets of the agents that expose the different metrics in OpenTSDB format over the network.

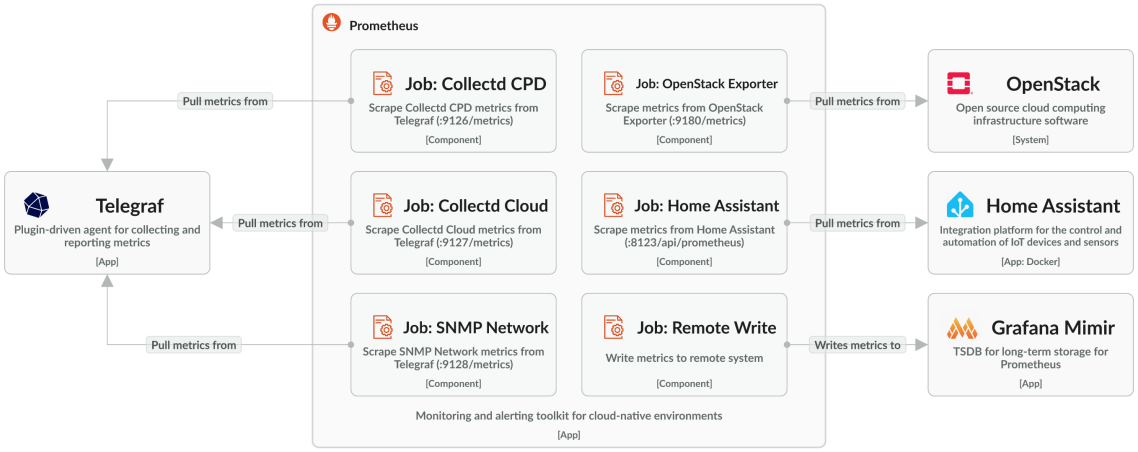


Figure 4.9: Prometheus Components Diagram

There are six jobs in total: The three leftmost ones are responsible for obtaining the metrics of each of the three Telegraf output streams, as shown in Figure 4.10. Then, there are two jobs that scrape metrics from OpenStack and Home Assistant endpoints. Finally, there is a component with a distinct function that handles sending the metrics to Grafana Mimir, a process known as remote write in Prometheus terminology.

4.2.3.4 Telegraf Components Diagram

Telegraf's metrics processing pipeline consists of four stages: Input, processing, aggregation and output. It processes data coming from both machines and network switches, with different data flows depending on the source. On the one hand, it acts as a gateway for the metrics sent by the collectd agents, while on the other hand, it uses an SNMP client to make requests for various network switches. In addition, because there are several types of machines which require different processing, there are two data flows whose source is the collectd agent. In all, the three flows are represented in Figure 4.10.

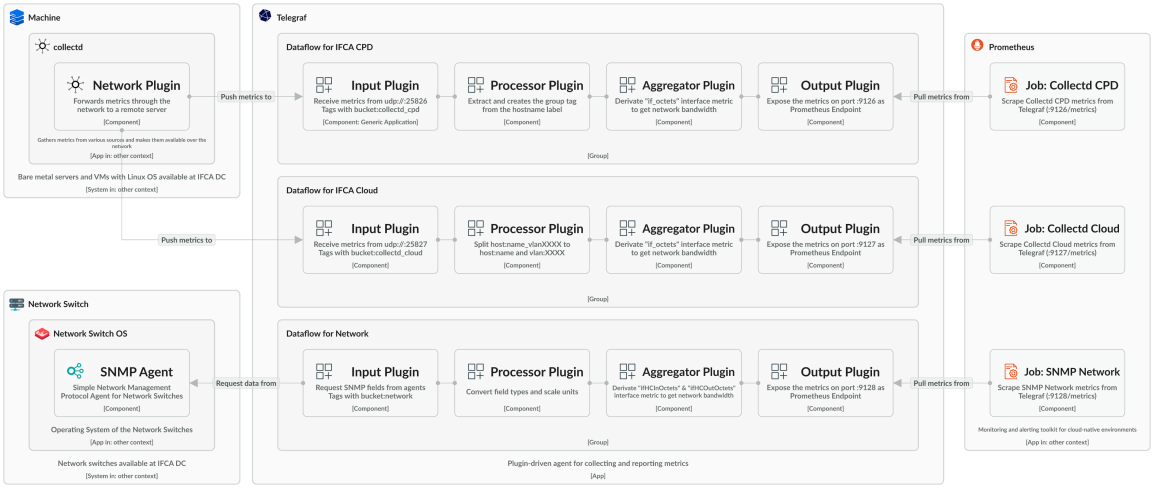


Figure 4.10: Telegraf Components Diagram

Telegraf receives the metrics from collectd using a socket type input plug-in, with a network open port listening. To distinguish between the two main groups of machines in the datacenter, those managed by the computing service and those in cloud used directly by users, two different ports are used. This plug-in tags the metrics based on their flow before passing them to the next stage. At this point, the data flow diverges: For the datacenter host machines (named as *IFCA CPD* in diagram), the name of the group is extracted from the hostname. In the case of user cloud machines (named as *IFCA Cloud* in diagram), the VLAN number is taken from the hostname label. This feature enables group machines by its VLANs number, in order to control access to VM data to users. Subsequently, for both machine types, using the aggregation plug-in, derived metrics are generated from the aggregation of other more basic metrics, for example the bandwidth of the network interfaces, from the number of bytes (*octets*) that have been sent/received by the network interface. Finally, these metrics are formatted following the OpenTSDB nomenclature in the output plug-in, so that Prometheus can scrape them.

Finally, the operating system of the network switches includes an SNMP agent, which can be queried using Telegraf's own prompt and the corresponding input plug-in. The obtained metrics are processed through various transformations, such as type conversion and unit scaling, as switches from different manufacturers may use different data type definitions. As well as for the machine's data flows, bandwidth metrics are derived from the interface's octet counters, and the processed metrics are then exposed as a Prometheus endpoint.

4.2.3.5 Collectd Components Diagram

The mode of operation of collectd is via plug-ins. Figure 4.11 illustrates the attached ones. By default, the monitoring daemon includes a bunch of plug-ins for monitoring various operating

system components. For simplicity, OS metrics are grouped into a single component in the diagram, as all the targeted metrics are OS-related. However, they are implemented as multiple built-in plug-ins. These metrics include CPU activity, memory, swap and disk usage, processes statuses, network interfaces statistics, port usage and status, system uptime and logged-in users.

Another OS-related plug-in that is modeled as a separate component is the IPMI Plug-in, as it is only relevant for physical machines. This plug-in retrieves information via IPMI from various hardware sensors, such as power consumption. Communication is carried out through an operating system library (*OpenIPMI*). Following the energy monitoring topic, there are two additional plug-ins: The first one allows reading directly the records provided by the RAPL interface to obtain other power consumption measurements, focusing only on the CPU and memory. The second one enables reading the output generated by Scaphandre agent to obtain the energy consumption of virtual machines hosted on the given host. More details about the latter are provided in section 4.2.3.6.

Similarly to the last ones, by means of an external plug-in developed in Python, it is possible to read the metrics gathered from the installed GPUs in the system, by the NVIDIA DCGM agent.

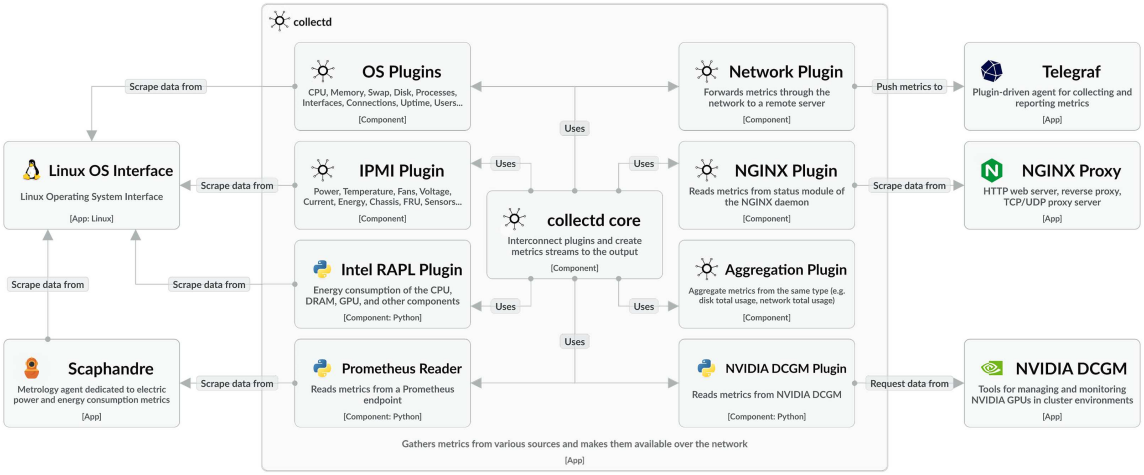


Figure 4.11: Collectd Components Diagram

In addition, an example of a service that is monitored locally from the serving machine, such as a NGINX proxy, is included. From a plug-in within collectd, the statistics for this service can be obtained and included as an additional set of metrics for the machine concerned. This can be extended to cover further services, such as local databases, if required in the future.

Collectd daemon has an aggregation plug-in that allows metrics to be aggregated locally. This is useful to aggregate statistics from all the network interfaces or from all the disks attached. Finally, all metrics are sent via network to Telegraf, using the integrated network plug-in.

4.2.3.6 Scaphandre Components Diagram

Scaphandre, the energy consumption metrology agent, is made up of three high-level types of components from a functional point of view: Sensors, the core and exporters, as shown in Figure 4.12. First, the sensors are the component that reads from the RAPL interface exposed through the Linux kernel. This kernel feature is called *powercap/intel_rapl*. This component provides the accumulated energy consumption of each of the domains or hardware components defined in the standard. These measurements are transferred to the central component, which is

in charge of transforming the accumulated energy consumption (*joules*) into power consumption (*watts*). Furthermore, this agent is deployed for the primary function of calculating the power consumption of each KVM/QEMU virtual machine hosted on the server. Finally, all these metrics can be reported in several formats. In our case, in the same format as practically as the rest of the platform: OpenTSDB, using a Prometheus exporter, scraped by a collectd custom plug-in.

The reason why this exporter is scraped by collectd agent instead of the main platform Prometheus instance, is because in this way, these metrics are tagged as other set of collectd host metrics, so it can be also classified and processed likewise in Telegraf. So, it is easier to introduce a new metric into the existing metric flow rather than create an additional parallel metric flow between Scaphandre and Prometheus with a separate or replicated configuration for classification.

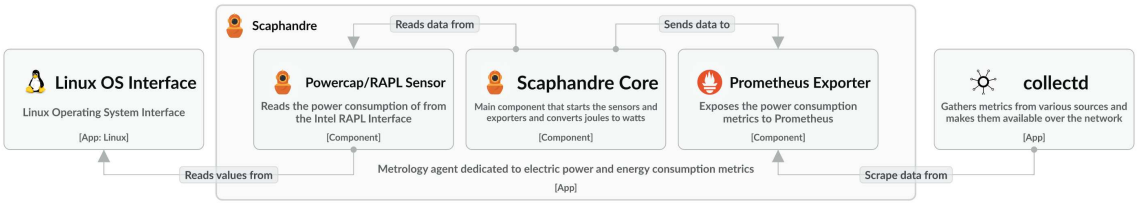


Figure 4.12: Scaphandre Components Diagram

4.2.3.7 Home Assistant Components Diagram

The Home Assistant instance has 2 additional components on top of the factory's preset ones available by default in this software, as shown in Figure 4.13. The first one, EcoStruxure PowerTag Link Gateway, is a custom integration that allows communication with the Schneider Electric sensor hub. The second integration, available in the official Home Assistant's catalog, is Prometheus Integration, which allows to expose over the network, in an OpenTSDB format, all the metrics available in this system to be scraped by a Prometheus server.

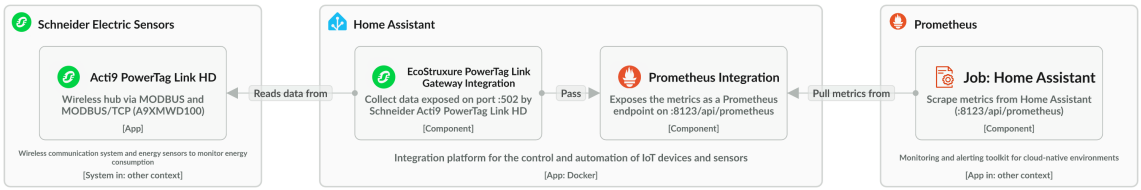


Figure 4.13: Home Assistant Components Diagram

The diagram in Figure 4.13 illustrates that one integration communicates with the other for simplicity, ignoring the rest of the Home Assistant components. The metrics collected in the first integration are passed through to be exposed by the second, as the internal operation of Home Assistant is not relevant from the platform's perspective, requiring no additional configuration and treating it as a black box.

4.2.4 Code Diagrams

This is an optional level of detail and is usually available on demand in tools such as IDEs [34]. It should be interesting for more complex components, with the need to define their design at a high-level of detail. In addition, as in this case the different systems used are already developed, the low-level architecture is the own of each one.

CHAPTER 5
Implementation

This chapter covers the configuration of each service in the datacenter's monitoring platform at IFCA, detailing both backend services and distributed agents. It also highlights implementation details and references the guides and documentation followed throughout the process. Additionally, the chapter provides a comprehensive overview of the available metrics, along with descriptions of the dashboards developed and the components that comprise them.

5.1 Computing Equipment Monitoring

5.1.1 Machines

5.1.1.1 Collectd Agent Installation and Basic Configuration

The installation of the collectd agent is straightforward for both major operating system families present in the datacenter. By installing the package named "collectd", the agent is promptly prepared for use, with most built-in plugins included by default, although some may require additional packages, typically following the naming convention "collectd-[pluginname]".

The basic configuration involves setting the machine `Hostname` to uniquely identify the metrics. For cloud instances, the VLAN number is also added to the `Hostname` label, as there is no other label that could be created for this purpose. Later, the Telegraf Starlark Processing Plug-in will split this label into two chunks and create the appropriate labels for further processing. Another essential parameter that must be provided is the `Interval`, which defines the period between samples. On our platform, based on the Software Requirements Specification, it is set to 30 seconds to ensure high-resolution metric collection suitable for detailed performance analysis.

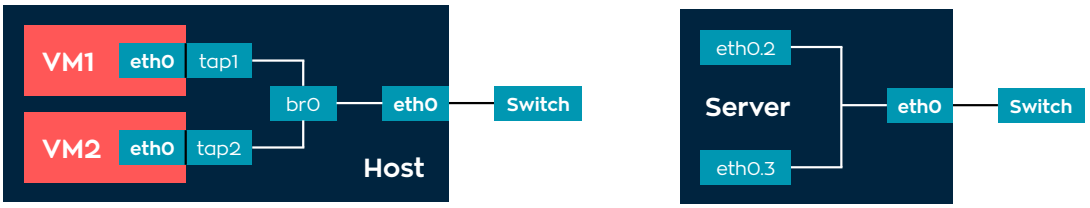
Finally, logging functions are enabled to capture and store relevant system events and data, ensuring that the process runs smoothly, and any issues can be easily tracked and diagnosed.

5.1.1.2 Collectd Plug-ins Configuration

The next step in configuring the collectd agent is to enable the desired plug-ins, which are already specified in the architecture diagram in Figure 4.11. Each selected plug-in is configured to fulfill particular requirements, and their individual settings are outlined below.

- **cpu:** This plug-in reports CPU usage metrics. It is configured to report by state (`idle`, `nice`, `interrupt`, `softirq`, `steal`, `system`, `user` and `wait`) [36]. To prevent excessively detailed data, it aggregates metrics at the processor level, or per host, given that most machines have multiple sockets, instead of reporting per core. Additionally, it is configured to report values as percentages rather than the default jiffies.
- **load:** This plug-in gathers the load metrics. These numbers give a rough overview over the utilization of a machine. The system load is defined as the number of runnable tasks in the run-queue and is provided by many operating systems as one-, five- and fifteen-minute averages, representing short-, mid- and long-term load [14]. No configuration is needed.
- **processes:** Collects information about local system processes. No additional configuration is needed, as it reports the number of processes in each state (`blocked`, `paging`, `running`, `sleeping`, `stopped` and `zombies`) [36].

- **memory:** This plug-in report memory usage statistics. No further configuration is needed, as it reports values in absolute units (i.e., bytes) by default and percentage values can be computed later if required.
- **swap:** This plug-in collects information about used and available swap space. No configuration needed. By default, the summary over all swap devices is reported only, i.e. the globally used and available space over all devices measured in bytes.
- **df:** This plug-in is configured to report filesystem usage statistics, focusing exclusively on local filesystems mounted directly on the host's disks. This is achieved by applying exclusion filters for network filesystems (e.g., GPFS, Ceph, rclone) and virtual or temporary ones (e.g., Docker/Nomad/Kubernetes volumes, Snap images). It also reports inode usage.
- **aggregation:** This plug-in, as its name suggests, combines other statistics into compound ones. It must be explicitly configured to define the desired aggregations. In this case, it is set up to sum up `df` plug-in statistics grouped by host, providing overall values instead of per-filesystem breakdowns.
- **interface:** Reports network traffic statistics, including the number of octets, packets and errors for each interface or network device. This plugin requires extensive filtering because multiple virtual network interfaces exist within a physical device. The goal is to include only outward-facing interfaces to avoid counting traffic multiple times on both virtual and physical interfaces. On IFCA's hosts two network interfaces topologies are used for virtual networking, as shown in Figure 5.1.



(a) Bridge Topology (b) VLAN Topology

Figure 5.1: Linux Interfaces Schemes for Virtual Networking [37]

The first provides connectivity from host to virtual machines using a Linux bridge, which behaves like a network switch. It forwards packets between interfaces that are connected to it [37]. This creates a bridge device named `br0` and sets two TAP devices (`tap1`, `tap2`) and a physical device (`eth0`) as its slaves, as shown in Figure 5.1a. In this case, only `eth0`, from both host and VM, should be monitored in order to track the real traffic on the host interface, as well as the portion of the traffic forwarded to/from each VM.

The second topology, as shown in Figure 5.1b, involves the use of Virtual Local Area Networks (VLANs) within a physical interface. This allows the broadcast domains to be separated by adding labels to the network packets [37]. The main issue is that counting the network statistics for each available device may result in counting the traffic on the physical device (`eth0`) as well as part of the traffic forwarded to the virtual interfaces (`eth0.2`, `eth0.3`), leading to an overestimation. Only `eth0` device should be tracked.

- **tcpconns:** This plug-in gathers network statistics regarding the number of TCP connections to specific local and remote ports and their TCP state, as defined on *RFC793* [38].
- **nginx:** Collects the number of connections and requests handled by a NGINX daemon. It queries the `ngx_http_stub_status_module` module, which is not compiled by default.

- **uptime:** Reports system uptime statistics, showing the time elapsed since last system boot.
- **users:** Reports the number of the current logged in users on host. No configuration needed.
- **network:** This plugin sends data to a remote server, Telegraf's Socket Listener Plug-in in our case. The server address and port must be specified. For datacenter hosts, port :25826 is used, while for user cloud machines, port :25827 is used.

5.1.1.3 Telegraf Processing Pipeline

Once the metrics are received in Telegraf, they are tagged according to the input plug-in based on the network port where they were received, allowing the separation of the metric streams from the two main groups of machines, as each requires slightly different processing steps.

On one hand, for metrics received through the input designated for datacenter hosts, processing involves extracting the group name which they belong to, from the hostname label. At IFCA, the naming convention assigns characteristic names to groups of machines based on their model, with digits appended to identify individual machines. Telegraf removes these digits and creates a new label containing the group name. On the other hand, for metrics from user machines in the cloud, since hostnames are user-defined without a specific naming convention, there is no inherent way to group these machines. To address this, the VLAN number is included in the hostname field by the collectd agent. Telegraf then splits this field into two separate labels, hostname and VLAN, which are originally combined and separated by a slash character.

Next, the metrics go through the aggregation plug-ins. At this point both data streams converge and do not differ in machine type. This plug-in is used to obtain the bandwidth per network interface, from the number of octets or bytes in transit through the interface in that interval, in this case 30 seconds. Finally, it is also converted from bytes to bits, to obtain bits/s. Mathematically, the exact bandwidth is represented as the derivative of the amount of data with respect to time, as defined in Equation 5.1:

$$B_w = \frac{dOctecs}{dt} \times 8 = \left(\lim_{h \to 0} \frac{Octecs(t+h) - Octecs(t)}{h} \right) \times 8 \approx \frac{Octecs(t_1) - Octecs(t_0)}{t_1 - t_0} \times 8 \tag{5.1}$$

However, when continuous data is not available, as is the case in network monitoring systems where we obtain samples at discrete time intervals, we can approximate the bandwidth by using the difference in data sent between two points in time, divided by the time interval between them. This approximation becomes more accurate as the time points get closer together.

It is important to note that for Telegraf and the calculation of derived metrics, all hosts must have synchronized time with the monitoring server. This was already the case in the datacenter due to the NTP server. If any host's time is not synchronized, Telegraf will not be able to calculate the derived metrics, as those metrics will fall outside the aggregation window in which it operates.

5.1.1.4 Prometheus Scraping Jobs

Finally, the set of metrics processed in OpenTSDB format are exposed on port :8126 for the datacenter machines data flow and on port :8127 for the user cloud machines data flow, allowing Prometheus to collect them. To achieve this, two separate jobs are configured on the Prometheus server, each polling its respective port at 30-second intervals. The metric samples remain available until they are either updated with new data or expire after a maximum of 120 seconds, ensuring sufficient time for collection.

5.1.1.5 Metrics Validation

To validate the data collected by the collectd agents, the transformation and aggregation of metrics by Telegraf plug-ins and to ensure the platform proper functioning, stress tests were performed on the machines, focusing on CPU and GPU, intensive memory and disk operations, synthetic network traffic generated by iPerf and InfiniBand performance tools, as well as machine reboots. These tests were designed to simulate operational workloads and ensure that any changes in the host states are accurately reflected in the collected metrics.

5.1.1.6 Dashboards

Since a large amount of data is collected from monitoring various aspects of numerous machines, it was decided that organizing this information in an accessible manner, without losing detail, would be best achieved by establishing a three-tier dashboard hierarchy. Each level has its own structure, purpose and focus, allowing users to navigate between general and detailed views seamlessly. Links between the dashboards enable users to move from general overviews to specific details with ease, creating a fluid navigation experience across all levels.

The first level (Figure C.1) offers a high-level view of CPU usage across all monitored machines, displaying percentages, color scales and aggregated metrics such as the total number of monitored machines, average CPU usage and average uptime, among other statistics, providing a comprehensive overview.

The second level (Figure C.2) presents more metrics, such as network usage, memory and disk usage, with each row representing a different machine, as shown in Figure 5.2. This level offers focused views on specific machines, with filters to narrow results by service or group, as well as time series and bar graphs that illustrate the evolution of key metrics like network bandwidth of the target hosts selected.

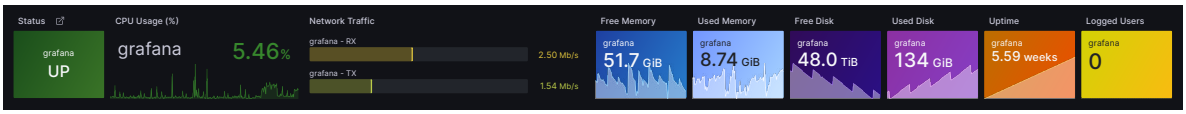


Figure 5.2: Level 2 Dashboard Row Showcasing Key Metrics of Monitored Host

The third level (Figure C.3) dives into granular details for the selected machine, offering additional metrics like CPU usage by state, system load, processes status, network interface statistics (e.g., sent packets, errors and drops) and network ports usage, TCP connections status and usage of inodes from different mounted file systems.

5.1.2 NVIDIA GPUs

To obtain real-time analytics about GPUs, NVIDIA Data Center GPU Manager is used. This process is being guided using a technical blog post published on the NVIDIA Developer website [39]. For troubleshooting, the official documentation [15] was consulted. It is an important requirement that the NVIDIA DCGM relies on the host engine service (*nv-hostengine*), which is included as part of the GPU driver installation and must be running in order to collect GPU telemetry data.

5.1.2.1 NVIDIA DCGM Service Connection with collectd Agent

The NVIDIA DCGM package includes a sample collectd plug-in implemented using the NVIDIA DCGM Python binding. The plug-in needs to be installed and configured to use it with collectd. The main configuration that has to be done manually is to configure dependency libraries paths,

as they differ across operating system distribution families. Then, it is necessary to instantiate the Python plug-in in the collectd agent. The plug-in is simply a Python program that imports a module and executes it upon loading. For this, the module's path and its name must be provided.

5.1.2.2 Selected Metrics

At the date of this work, with the latest version available (v4.0), there is a set of 1216 variables that can be measured by this service [15]. As described in the Software Requirements Specification, in chapter 3, the following entities have been selected:

- `DCGM_FI_DEV_MEMORY_TEMP` 140 : Memory temperature for the device, in degrees Celsius.
- `DCGM_FI_DEV_GPU_TEMP` 150 : Current temperature of the device, in degrees Celsius.
- `DCGM_FI_DEV_MEM_MAX_OP_TEMP` 151 : Maximum operating temperature for GPU memory.
- `DCGM_FI_DEV_GPU_MAX_OP_TEMP` 152 : Maximum operating temperature for this GPU.
- `DCGM_FI_DEV_POWER_USAGE` 155 : Power usage for the device, in Watts.
- `DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION` 156 : Total energy consumption for the GPU in *mJ* since the driver was last reloaded. Later it is converted to watts per hour in Grafana.
- `DCGM_FI_DEV_SLOWDOWN_TEMP` 158 : Slowdown temperature for the device, in ºC.
- `DCGM_FI_DEV_SHUTDOWN_TEMP` 159 : Shutdown temperature for the device, in ºC.
- `DCGM_FI_DEV_POWER_MGMT_LIMIT` 160 : Power limit for the device, in Watts.
- `DCGM_FI_DEV_GPU_UTIL` 203 : GPU utilization. Range: 0.0-1.0
- `DCGM_FI_DEV_MEM_COPY_UTIL` 204 : GPU memory utilization. Range: 0.0-1.0
- `DCGM_FI_DEV_FB_TOTAL` 250 : Total Frame Buffer (main GPU memory) of the GPU in MB.
- `DCGM_FI_DEV_FB_FREE` 251 : Free Frame Buffer (main GPU memory) in MB.
- `DCGM_FI_DEV_FB_USED` 252 : Used Frame Buffer (main GPU memory) in MB.
- `DCGM_FI_DEV_FB_RESERVED` 253 : Reserved Frame (main GPU memory) Buffer in MB.

5.1.2.3 Dashboards

To present the metrics collected from the NVIDIA GPUs in the clearest and most organized way, two levels of dashboards are provided. Links are available to allow interaction and navigation between them. Both dashboards feature a set of filters that allow users to select GPUs by service, group, host, GPU number and UUID. These make it easier to navigate through the data and focus on specific GPU. The filters enhance the user experience by enabling a more targeted analysis.

The first dashboard, shown in Figure C.4, displays the usage and energy consumption of each monitored GPU through time series, providing a quick overview of these measurements for all GPUs. Additionally, at the top, a series of statistics are shown, summarizing key variables such as the number of GPUs currently monitored, the number of GPUs being used, average utilization across all GPUs, memory used, free memory, average temperature, average power consumption and total energy consumed.

The second dashboard displays, as shown in Figure C.5, a detailed view of a single GPU. It presents identifying information at the top, such as the machine where is located on and the GPU UUID, among others. Below, it shows statistics for the selected GPU, including both current values in gauge format and their evolution over time in time series charts.

With these two levels of dashboards, users can have a comprehensive view of the overall GPU usage and power consumption, as well as drill down into the performance and energy metrics of individual GPUs, providing detailed and actionable insights for system monitoring and optimization.

5.2 Network Equipment Monitoring

5.2.1 Network Switches

The monitoring of the datacenter network switches is carried out through the Telegraf SNMP input plug-in. Each network switch is queried by a dedicated instance of the plug-in. Some initial connection configuration is required for each instance, including the target SNMP agent URL, protocol version and community as the authentication mechanism. Afterwards, the fields or tables to be monitored must be defined for each target agent, specifying the name of each metric and the OIDs of the device MIBs from which the value or set of values will be retrieved.

5.2.1.1 Management Information Base Object Identifiers

In order to select the necessary objects from the MIBs, it must be taken into account that certain objects are common amongst switches, such as the device name or the number of interfaces, since they are defined as part of the MIB-II hierarchy, which follows a certain standard as specified in *RFC1213* [20], but other objects structure that are dependent on some intrinsic characteristic of the switch model, such as the number of CPUs, are found in a MIB that is proprietary to each manufacturer or model.

To identify the different OIDs of each switch model and manufacturer, *Observium MIB Database* [40] has been used, as well as the documentation and manuals of the devices. The selected enterprise OIDs are detailed for each model in Appendix A.2, as well as in the configuration files themselves, available at the associated folder in the project repository[1].

5.2.1.2 Telegraf Processing Pipeline

All input metrics are labeled with the device name, as well as the interface identifier if applicable. Also, all string values, such as management IP address, device description, interface alias or interface MAC address, must be embedded into other metrics labels, as OpenTSDB format does not allow any kind of value type without direct conversion to float type.

Once the metrics are gathered from the input plug-in every 30 seconds, some processing must be done in order to standardize field names, scale units, data formats and types of the uncommon MIB objects, as each manufacturer has their own naming agreement. Subsequently, the aggregation plug-in calculates the bandwidth in bits per second from the cumulative value of outgoing and incoming octets for each interface, as was already done for the network interfaces on machines. Finally, the output plug-in exposes all the metrics on port `:9128` via the Prometheus client, with a metrics expiration window of 120 seconds. At the other end of the communication, a Prometheus job is configured to scrape the metrics every 30 seconds at that endpoint.

5.2.1.3 Selected Metrics

Once all metrics have been processed and derived, they can be categorized into two main groups: Device metrics and interface metrics:

- **Device metrics:** Include the device name, model description, number of interfaces, which encloses physical and virtual interfaces (VLANs), uptime, management IP address, CPU usage, memory usage and device temperature.

[1] https://gitlab.ifca.es/iglesiasj/ifca-monitor-toolkit/-/tree/main/server/etc/telegraf/telegraf.d/snmp_network

- **Interface metrics** cover several aspects:
  - **General information:** Interface name, interface alias, configured link speed, interface type (as defined by IANA registry[2]), MTU and MAC address.
  - **Status:** Administrative and operational status and interface connector presence.
  - **Usage:** Bandwidth, including current value, historical time series, and histogram distribution, aggregated utilization (hourly, daily, weekly, and monthly), and packet statistics classified by type (unicast, multicast and broadcast), along with packet discard and error statistics.

5.2.1.4 Dashboards

A single dashboard is provided, offering two filters: Device selection and interface selection within the device. Relatively static data, such as name, description and interface general information, such as index or alias, are presented using a stat chart, showing only the latest value. More dynamic data, such as utilization, temperature, statuses and bandwidth, are visualized through time-series charts, bar charts and histograms. A dashboard screenshot can be seen in Figure C.6 in Appendix C.

5.3 Energy and Power Consumption Monitoring

5.3.1 Technical Room

The connection to the electrical sensor system installed in the facility is made through an already available open-source Home Assistant custom add-on[3]. This enables real-time data collection from the sensors system gateway, allowing values to be retrieved and stored externally, thus integrating these measurements into the datacenter's monitoring platform.

5.3.1.1 Home Assistant Installation and Configuration

The installation of Home Assistant as a Docker container is carried out using Docker Compose with the official Home Assistant Docker image. Additionally, it is possible to deploy a custom image using a Dockerfile, which has the integration embedded directly into the image, simplifying deployment but increasing the maintenance efforts of the integration. For this, an alternative installation method is used: Home Assistant Community Store (HACS), a third-party, open-source add-on that allows the installation of custom components and plug-ins into a Home Assistant instance. This method enables managing updates of the custom integration directly from its official repository releases, avoiding the need to manually update the integration code attached to the container. The installation was carried out following the official documentation[4].

5.3.1.2 Schneider Acti9 PowerTag Link HD Configuration

It is assumed that the Gateway and all associated PowerTags have been properly installed and configured according to the manufacturer's guidelines. For the integration with Home Assistant to function correctly, the Modbus/TCP service must be enabled on the Gateway, which is typically enabled by default. To confirm this setting, users should access the device's administration and configuration webpage. By default, Modbus/TCP operates on port `:502`; however, if a different port has been specified during setup, this must be updated accordingly within the Home Assistant

[2] IANAifType-MIB: https://www.iana.org/assignments/ianaiftype-mib/ianaiftype-mib
[3] Home Assistant integration for EcoStruxure Gateways: https://github.com/Breina/PowerTagGateway
[4] HACS User documentation: https://hacs.xyz/docs/use/

integration settings [41]. Furthermore, it is recommended to enable the discovery service on the Gateway, as this simplifies the integration process by allowing Home Assistant to automatically detect the device on the network.

5.3.1.3 EcoStruxure PowerTag Link Gateway Installation and Configuration

The installation and integration of the EcoStruxure PowerTag Link Gateway begins by downloading the custom integration repository through HACS and restarting Home Assistant. The integration is then added. If the discovery service is enabled, the Gateway is automatically detected; otherwise, the host address and Modbus/TCP port must be entered manually. Once the Gateway is successfully added, devices and entities are created, one device per PowerTag and one entity for each measurement obtained through the Gateway.

5.3.1.4 Prometheus Connection to Home Assistant

The official Prometheus endpoint plug-in[5] is used to export the metrics to a Prometheus instance. The code provided on the official website is added to the Home Assistant configuration file (`configuration.yaml`). Afterwards, custom filters are applied to remove unnecessary metrics, such as Home Assistant zones, users, and other items unrelated to Schneider sensor data.

On the main server side, a new job is created in the Prometheus configuration file. This job is configured to pull metrics from the "`/api/prometheus`" path every 30 seconds from the Home Assistant target.

5.3.1.5 Available Metrics

We obtain from each electric sensor the following metrics:

- **Current (A):** Measured per phase and neutral. Electrical current is the flow of electric charge through a conductor.
- **Voltage (V):** Measured per phase. Voltage represents the electrical potential difference between two points.
- **Power (W):** Includes active, reactive and apparent power per phase and overall. Also includes total active power demand. Their relationship is represented in Figure 5.3.
  - **Active power (W):** Power used by the equipment to perform useful work.
  - **Reactive power (VAR):** Power that flows back and forth without doing useful work.
  - **Apparent power (VA):** Total power, combining active and reactive.

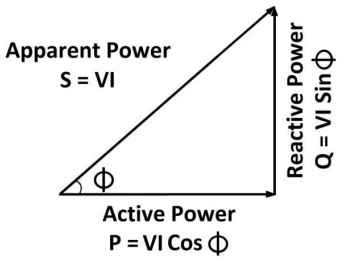


Figure 5.3: Power Relationships in AC Circuits [42]

- **Energy (Wh):** Total active, reactive and apparent energy accumulated since the last reset. Energy represents the total power consumed over time.
- **Power Factor (%):** The ratio between active power and apparent power. This measure indicates how much of the energy consumed is actually used to do useful work and how much is wasted. In other words, it reflects how efficiently electricity is being used [42].
- **Diagnostics:** The Schneider integration also provides additional metrics related to sensor status and wireless connectivity, including gateway status, link quality indicator (LQI), received signal strength indicators (RSSI), packet loss ratio, connection status and more.

[5] Home Assistant Prometheus integration: https://www.home-assistant.io/integrations/prometheus/

5.3.1.6 Dashboards

Based on the three-tier dashboards for machine monitoring, a similar two-tier structure is used for Schneider Electric Sensors, offering linked navigation between dashboards and filters to select one or multiple sensors for visualization.

The first dashboard (Figure C.10) provides an overview of total power related metrics per sensor, displaying active, reactive, apparent power and power factor through time series charts. Additionally, by aggregating data from the sensors on the main incoming electrical panels, a comprehensive view of the entire datacenter's power consumption is obtained.

The second dashboard (Figure C.11) dives into more detailed measurements for each sensor, presenting both real-time values and historical trends. It covers power, voltage, current and power factor, broken down per phase, enabling thorough monitoring of each variable's behavior over time.

Finally, a dashboard was created to visualize the various diagnostic metrics of the sensors (Figure C.12). Its purpose is to verify and measure the connectivity performance of the sensors, as several issues were identified during the project's deployment phase. These connectivity problems arise because the sensors are installed inside electrically shielded panels for safety reasons. Additionally, a datacenter contains a high concentration of electronic equipment, such as servers, power supply units (PSUs), cooling systems, all of which can generate electromagnetic noise (EMI) that interferes with Wi-Fi signals. Furthermore, metal racks and other equipment can reflect or absorb Wi-Fi signals, creating dead zones or areas with weak coverage. The physical layout of the hardware and the presence of barriers can also contribute to signal degradation [43].

5.3.2 Physical Servers

In this section, the energy and power monitoring for physical servers in a datacenter is addressed, as they are among the most energy-consuming hardware components. The IPMI plug-in, integrated into the collectd library, allows for collecting and filtering power-related data from various sensors. Additionally, we explore the use of Intel's RAPL interface for more precise power monitoring, specifically for CPU power consumption, further enhancing the accuracy of energy tracking on these servers.

5.3.2.1 IPMI Plug-in Integration

The IPMI plug-in is already integrated into the collectd library and can be used by simply instantiating it in the configuration. However, since this interface provides access to numerous sensors and system parameters, it is necessary to filter those that report power data. To achieve this, the plug-in supports filtering based on a regular expression, which matches if the sensor name contains the word "power" or its multiple existing abbreviations. It is important to note, however, that not all sensors with the word "power" in their name actually measure power. For example, sensors related to power supply unit (PSU) parameters may not measure power directly; they could measure current, voltage, status, or other variables. To address this, additional processing and filtering are performed in Telegraf by checking the unit of measurement of the filtered readings, discarding those that are not power measurements in watts.

Finally, it is essential to consider that, since we are measuring hardware elements within the server enclosure, some of these components may be shared across multiple nodes, as in the case with chassis or blade servers. In these server format, multiple nodes share the power supply units, and each node may report the power consumption of the PSU as if it were exclusively its

own. To address this discrepancy, we normalize the power consumption for each node by dividing the total power usage of the shared power supply unit by the number of nodes utilizing it.

5.3.2.2 Intel RAPL Integration

After reviewing public open-source code repositories, an Intel RAPL Reader Python plug-in for collectd was found[6]. However, some modifications were necessary, as the original plug-in does not support multi-socket hosts. To adapt the code, a unique numeric identifier was added to each RAPL domain name to distinguish each socket. The source code of the modified plug-in is in the project repository[7].

Additionally, some processing must be performed in Telegraf, as this plug-in report energy rather than power. Energy can be converted to instant power by deriving energy over time. By differentiating the energy every 30 seconds (or the monitoring period), we can calculate the average instantaneous power for that particular interval. The exact instantaneous power is derived from energy with respect to time, as defined in Equation 5.2. As mentioned earlier, since the data is sampled at discrete intervals, a *Difference Quotient Approximation* is used.

$$P_{inst} = \frac{dE}{dt} = \lim_{h \to 0} \frac{E(t+h) - E(t)}{h} \approx \frac{E(t_1) - E(t_0)}{t_1 - t_0} \tag{5.2}$$

5.3.2.3 Metrics

The metrics collected by the IPMI plug-in vary depending on the server manufacturer and model, as not all systems provide the same set of sensors. In general, it is possible to retrieve the whole server instant power consumption, AC input, and DC output power for all installed power supply units, as well as the power usage of the fans, DRAM, and CPU.

From the RAPL interface, we have measured the average instantaneous power consumption per sampling interval, broken down by the various available RAPL domains for each processor model. Although the total system-on-chip (SoC) power consumption is not reported directly, it can be calculated later in Grafana, by aggregating the non-inclusive domains (`PKG` and `DRAM`).

5.3.2.4 Dashboards

As previously, a two-tier hierarchical approach has been used to organize data into two dashboards. Both offer three filters to refine the view by service, group and host, as well as navigation links that enable seamless switching between the two dashboards.

The first dashboard displays the current power consumption per machine, measured by IPMI, RAPL, or both, as shown in Figure C.7. It also shows summary statistics, such as the number of monitored machines, measurements by type and overall consumption, calculated as the maximum between methods for each machine.

The second dashboard, shown in Figure C.8, details the temporal evolution of measurements, breaking them down by method and domain. It also includes ratios comparing CPU and memory consumption across both measurement methods. From these ratios, it can be concluded that the measurements align closely, showing minimal discrepancies that could be due to measurement precision or unit conversion.

[6] https://github.com/jsastriawan/intel_rapl/blob/master/intel_rapl.py

[7] https://gitlab.ifca.es/iglesiasj/ifca-monitor-toolkit/-/blob/main/clients/usr/lib/collectd/python-plugins/intel_rapl.py

5.3.3 Virtual Machines

Finally, at this level of detail, we focus on the energy consumption of the main asset used by users and the platforms deployed on them in a cloud computing environment: virtual machine. This level of technicality is more complex than the one described above since it is a virtual entity, which runs on a physical resource, shared among multiple instances through virtualization. To achieve this, the Scaphandre agent is added to the stack of agents running on the physical host machines, as described below.

5.3.3.1 Scaphandre Installation and Configuration

Installing the Scaphandre agent is straightforward: simply download and install the package from the official repository, then run the main program with the `--qemu` option to map QEMU-KVM processes to virtual machines, thereby adding VM metadata to process metrics. Additionally, specify the exporter to use, Prometheus in this case, along with the required port [21]. A systemd service is created to manage the execution of the agent.

Additionally, it is necessary to ensure that RAPL is enabled in the system's kernel, which can be done by running `modprobe intel_rapl_common`, and that QEMU-KVM is installed. To verify that data is being collected, at least one hosted virtual machine should be running.

5.3.3.2 Scaphandre Connection with collectd Agent

To read the metrics provided by Scaphandre through the Prometheus exporter from the host's collectd agent, a plug-in capable of reading and parsing those metrics is required to convert them into collectd-compatible metrics. Since this functionality is quite specific, the simplest approach is to implement it through a Python script, leveraging two available libraries: The Prometheus client for Python, which facilitates both requesting and parsing the metrics, and the collectd library, which handles composing the metrics in collectd's format and connecting them with the agent. Following a review of public open-source code repositories, it was discovered that this plug-in had already been developed by the community and is accessible in this repository[8].

However, when using that plug-in, some useful native Scaphandre metric labels, such as `exe`, which indicates the executable name of the process metric, `cmdline`, which provides additional information about the process and `vmname` in case the process is a virtual machine, were lost during metric parsing. These are custom labels added by the Scaphandre Prometheus exporter and were therefore not expected or considered by the generic plug-in. Since these labels are essential to univocally identify the time series of each virtual machine, we modified the code to parse these new labels and add them as collectd labels. One restriction is that collectd metrics have finite and predefined labels, unlike the Prometheus metrics format, so it was necessary to encapsulate several of these Scaphandre metrics labels into a single one. Later, the Telegraf Starlark processor plug-in separates this tag into multiple labels. The source code of the modified plug-in is in the project repository[9].

5.3.3.3 Metrics

The Scaphandre agent adds two new measurements to the platform regarding the power consumption of the infrastructure. It provides the consumption of each host in microwatts,

[8] https://github.com/ryarnyah/collectd-prometheus/blob/master/collectd_prometheus.py

[9] https://gitlab.ifca.es/iglesiasj/ifca-monitor-toolkit/-/blob/main/clients/usr/lib/collectd/python-plugins/collectd_prometheus.py

calculated as an aggregation of several measurements to give a try on the power usage of the whole host. It might be the same as RAPL-PSYS domain or a combination of RAPL-PKG and RAPL-DRAM domains [21]. Neither does it represent the real consumption that can be measured with IPMI. The most interesting measure from this data source is the power consumption in microwatts per existing process on the host, focusing only on the QEMU-KVM processes that are virtual machines. Each of the VMs can be identified by its instance name within the host's KVM domain, as well as the VM UUID assigned by the VMs manager, OpenStack.

5.3.3.4 Dashboards

To display these measurements in Grafana, the same dashboard structure is used as for the machine's consumption monitoring. For each physical machine, the total consumption provided by Scaphandre is shown, followed by the consumption breakdown per virtual machine hosted on the host. Ratios are also calculated to compare how much the measurements calculated by Scaphandre represent out of the total machine consumption measured with IPMI, if such a measurement is available. Finally, two charts display the aggregated consumption of all monitored servers and all VMs fitted with this data source. All consumption charts are plotted as time series to visualize historical evolution, while stat charts display real-time values and counters, such as the number of monitored machines.

Since this dashboard handles a large amount of data that identifies different hosts and virtual machines, five filters are available: By host service, by host group, hostname, KVM domain and VM UUID. A screenshot of this dashboard is shown in Figure C.9.

Additionally, to verify and validate the measurements previously taken from the RAPL counters by the collect agent plug-in, a third column of charts was added to the second dashboard described in 5.3.2.4, showing the detailed energy consumption breakdown per machine, as shown in Figure C.8. This allows comparison between the value computed by Scaphandre and the sum of those retrieved directly from the RAPL interface.

5.4 Service Monitoring

5.4.1 OpenStack Exporter for Prometheus

As previously mentioned, an OpenStack Exporter for Prometheus is a tool designed to expose OpenStack metrics in a format that Prometheus can scrape and monitor. This is especially useful for keeping track of cloud infrastructure health, performance and resource utilization.

There are currently several implementations of such exporters available, developed either by the community or third-party enterprises. Among the available options, a well-maintained, community-driven, open-source implementation has been selected[10], due to its broad support for all relevant services used at IFCA's cloud computing service.

5.4.1.1 Installation and Configuration

The OpenStack Exporter can be installed in multiple formats [23], including a straightforward deployment using the Snap package manager, which simplifies the process by bundling all dependencies into a single, self-contained package. After installing, configuration primarily involves setting the necessary environment variables to allow the exporter to authenticate and query metrics from the OpenStack APIs.

[10] OpenStack Exporter for Prometheus: https://github.com/openstack-exporter/openstack-exporter

5.4.1.2 Metrics

The first step was to check, classify and sort the various metrics provided by the exporter, as the official documentation [23] failed to define some metrics that were observed during execution. After this review, the following metrics are available in summary:

- **Identity (Keystone):** Metrics on domains, projects, groups and users registered in the identity service, including detailed information and service status.
- **Compute (Nova):** Metrics on instances, flavors, availability zones and resource limits (CPU, memory, disk). Also includes the status of compute agents and the service itself. The number of GPUs are not directly reported in flavors data, but they can be extracted and added based on the flavor name using Prometheus processing rules after scraping.
- **Network (Neutron):** A wide range of metrics on networks, subnets, ports, routers, floating IPs, agents and security groups. All has attached some metadata like status or ownership. Also includes IP address availability and service status.
- **Image (Glance):** Metrics related to the number of images, their size in bytes, some metadata such as status, visibility and creation dates. Also includes service availability status.
- **Storage (Cinder):** Metrics related to service state, volumes and snapshots; including storage limits and usage (in GB) for volumes and backups.
- **Allocation (Placement):** Metrics on resource allocation, usage, reservation and total availability in the compute infrastructure in terms of vCPUs, memory and storage, as well as the status of the placement service.

5.4.1.3 Dashboards

One dashboard is provided per service. At the top of each, the service status (`UP`/`DOWN`/`ERROR`) is shown both in real-time and as a time series. Relevant metrics are displayed in three ways: Representative values, like Keystone's project count, are shown as current values with time series charts to track changes. Tabular data from OpenStack Exporter is presented in tables, sometimes merged for clearer insights (e.g., VMs and their flavors), showing only current values. When historical trends are relevant, time series graphs are also included.

For agent status, a two-tier dashboard structure is used. The first tier gives an overview of all agents per host, showing their current state, while the second tier adds temporal evolution. Filters allow easy exploration by host and agent status, while navigation links provide seamless movement between dashboards.

Screenshots of the OpenStack dashboards developed have been excluded from this report due to the presence of sensitive information related to the cloud service and its entities, including users, projects, and networks, among others, to protect user privacy and confidentiality.

5.5 Data Management Backend

5.5.1 Grafana Mimir

After Prometheus has collected all the relevant metrics, as previously outlined, it is configured to write these metrics to an external TSDB service, specifically Grafana Mimir. The basic configuration of Mimir follows the official installation and configuration guidelines detailed in its documentation [44]. This ensures that the setup is properly aligned with the best practices and optimized for reliability and performance in a production environment.

Another important aspect of this service is the configuration of the storage backend. This service uses the local filesystem for storage, which is simple and fast. Since the storage is on the machine itself, there is no need to connect to a remote storage system. In this configuration, a retention policy is applied, ensuring that metrics are retained for a period of three years.

In addition, the ingestion rate of metrics is carefully adjusted to meet the specific requirements of the system. These adjustments ensure that the system can handle the expected volume of incoming data without performance degradation. The configuration parameters for ingestion rates are determined based on key Prometheus statistics, such as the number of generated time series, which provide insight into the overall load on the system. By analyzing these metrics, the ingestion process is optimized, ensuring both scalability and performance.

5.5.2 Grafana

Grafana dashboards are organized into two main folders: *Internal* and *External*. The *Internal* folder is restricted to service administrators, housing sensitive data related to the overall infrastructure, with all metrics from the monitored systems displayed. These permissions are configured in Grafana's administration menu via web interface. The *External* folder contains dashboards that display metrics gathered from cloud machines, designed to make this content generally available to registered users at IFCA, via SSO through LDAP. However, an additional layer of control is applied to restrict the information about which virtual machines users can view.

Additionally, each dashboard can have customized permissions, allowing precise control over who can view or modify it, based on user roles. This feature is particularly useful for granting access to specific project stakeholders. For example, it allows *GreenDIGIT* developers to view data regarding power consumption. The structure ensures proper data access management, granting users visibility according to their permissions while safeguarding sensitive information.

5.5.2.1 Grafana Filters

To enrich Grafana filters, an SQLite database is used to establish relationships between metric labels. Two aspects are addressed: Linking VLANs numbers to usernames to enable filtering of user-specific VLANs associated with user cloud virtual machines, and grouping hosts into services for functional categorization of the datacenter machines, as shown in Table A.1 in Appendix A. YAML files define these relationships, one mapping VLANs to username lists and another linking services to machine groups. Python scripts parse these files and convert them into SQLite databases, simplifying integration.

This approach adds flexibility, enabling more meaningful data visualization and improved organization. In the case of user VLANs, this implementation enforces the privacy and access control policy, ensuring that each user can only query data for the VLANs assigned to them in the database. This restriction is integral to the system, guaranteeing that access to sensitive data is strictly limited according to the user's specific VLAN assignments.

5.6 Data Backup and Recovery System

An essential component of the monitoring platform is the data it collects and stores, as it provides valuable insights into the system's performance over time. Given that this data must be retained for at least three years, implementing reliable backup systems is crucial to ensure its long-term availability and protection against data loss. This can be achieved by simply backing up the relevant directories where data is stored, leveraging existing tools in the datacenter environment, such as Bacula [45], to automate and manage these backups efficiently.

CHAPTER 6

Deployment

This chapter outlines the process of deploying and automating the platform configuration for IFCA's premises. The main objective of this process is to ensure that the configurations applied to the monitoring services are both persistent and replicable, while also enabling the automatic and unattended deployment of agents across the datacenter systems. This ensures that when a new server, network switch, or sensor is installed, it is automatically configured, integrated into the platform and begins being monitored with just a few steps.

Figure 6.1 shows the production deployment diagram of the platform's agents and services across the IFCA's infrastructure. The diagram maps all containers from the architecture diagrams in Chapter 4 to the respective datacenter resources and systems. The server-side components are deployed on two physical machines in the datacenter.

6.1 Server Configuration Management

The platform configuration is managed using Puppet to ensure a consistent and automated deployment of all necessary services. For the server side, two Puppet modules have been defined: The first module is structured into five classes which configure Grafana, Prometheus, Telegraf, SQLite and Mimir. These classes install the required software using system packages, attach predefined configuration files, as described in the previous chapter, enable the services, as all of them are being executed as systemd services and verify that they are running correctly. Additionally, other files, such as dashboard templates and YAML files for filters, are also attached. The second module handles the deployment of the Home Assistant instance as a pre-built Docker container, ensuring it is launched with the correct configuration attached and remains in a running state. This structured approach guarantees a modular, scalable and reliable deployment process, minimizing manual intervention while maintaining system consistency.

6.2 Agent Installation and Setup

Given the diversity in machine provisioning, two distinct deployment methods for agents and the required software are necessary. The first applies to centrally managed physical and virtual machines, including virtual instances (i.e., worker nodes) in the HPC and Grid Slurm queues, as well as VMs hosting *Software as a Service* (SaaS) applications. For these, system administrators handle deployment and configuration using Puppet. The second method concerns user-created cloud virtual machines, where end users are responsible for deploying and configuring the agents on their own machines, with a bash script provided to automate the setup.

6.2.1 Servers and Datacenter Services

6.2.1.1 Puppet Facts

In the Puppet context, the variables that define and categorize a host are called '*facts*'. These provide crucial details about the system's configuration and environment. Some facts are predefined, such as the operating system version, whether the host is virtual or physical and other system attributes [46]. These facts are essential in the deployment of the platform across the infrastructure, as they help tailor the monitoring setup to each specific host.

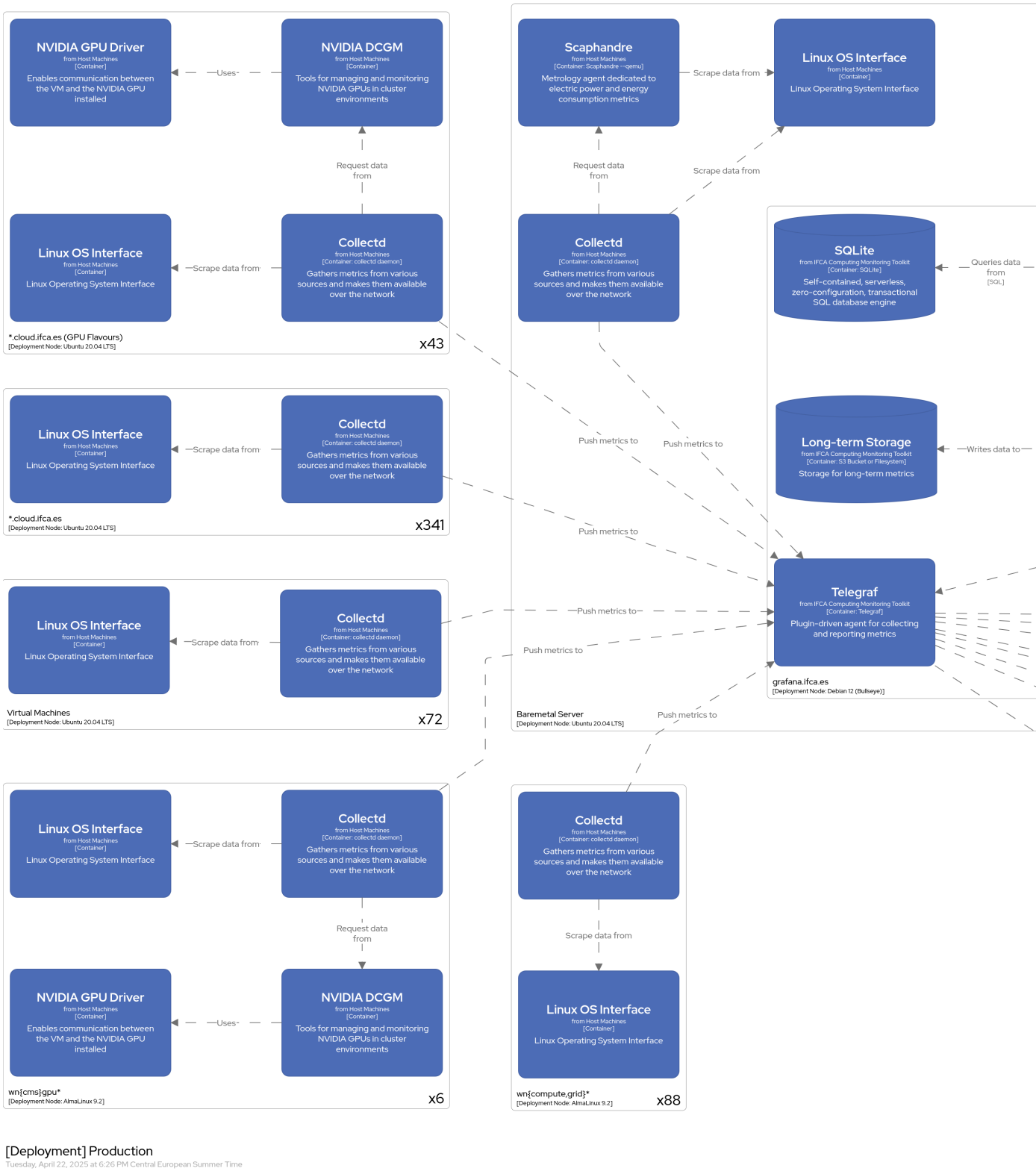
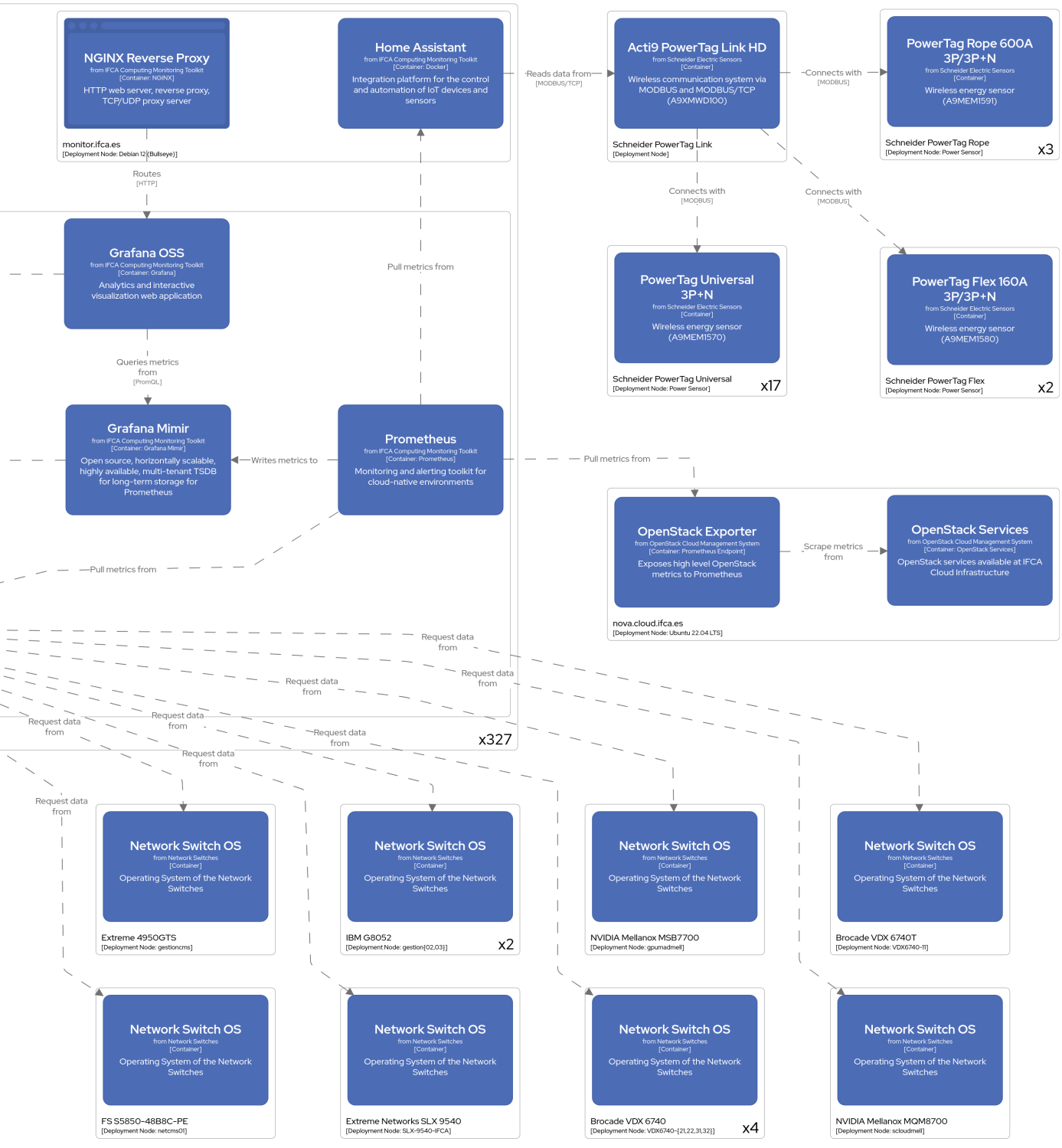


Figure 6.1: Monitoring Platform Production Diagram on IFCA's Datacenter


NGINX Reverse Proxy
from IFCA Computing Monitoring Toolkit
[Container: NGINX]
HTTP web server, reverse proxy, TCP/UDP proxy server
Home Assistant
from IFCA Computing Monitoring Toolkit
[Container: Docker]
Integration platform for the control and automation of IoT devices and sensors
monitor.ifca.es
[Deployment Node: Debian 12 (Bullseye)]
Reads data from
[MODBUS/TCP]
Acti9 PowerTag Link HD
from Schneider Electric Sensors
[Container]
Wireless communication system via MODBUS and MODBUS/TCP (A9XMWD100)
Schneider PowerTag Link
[Deployment Node]
Connects with
[MODBUS]
PowerTag Rope 600A 3P/3P+N
from Schneider Electric Sensors
[Container]
Wireless energy sensor (A9MEM1591)
Schneider PowerTag Rope
[Deployment Node: Power Sensor]
x3
Routes
[HTTP]
Grafana OSS
from IFCA Computing Monitoring Toolkit
[Container: Grafana]
Analytics and interactive visualization web application
Pull metrics from
Connects with
[MODBUS]
Connects with
[MODBUS]
PowerTag Universal 3P+N
from Schneider Electric Sensors
[Container]
Wireless energy sensor (A9MEM1570)
Schneider PowerTag Universal
[Deployment Node: Power Sensor]
x17
PowerTag Flex 160A 3P/3P+N
from Schneider Electric Sensors
[Container]
Wireless energy sensor (A9MEM1580)
Schneider PowerTag Flex
[Deployment Node: Power Sensor]
x2
Queries metrics from
[PromQL]
Grafana Mimir
from IFCA Computing Monitoring Toolkit
[Container: Grafana Mimir]
Open source, horizontally scalable, highly available, multi-tenant TSDB for long-term storage for Prometheus
Writes metrics to
Prometheus
from IFCA Computing Monitoring Toolkit
[Container: Prometheus]
Monitoring and alerting toolkit for cloud-native environments
Pull metrics from
Pull metrics from
OpenStack Exporter
from OpenStack Cloud Management System
[Container: Prometheus Endpoint]
Exposes high level OpenStack metrics to Prometheus
Scrape metrics from
OpenStack Services
from OpenStack Cloud Management System
[Container: OpenStack Services]
OpenStack services available at IFCA Cloud Infrastructure
nova.cloud.ifca.es
[Deployment Node: Ubuntu 22.04 LTS]
Request data from
Request data from
Request data from
Request data from
Request data from
Request data from
Request data from
x327
Request data from
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
Extreme 4950GTS
[Deployment Node: gestioncms]
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
IBM G8052
[Deployment Node: gestion[02,03]]
x2
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
NVIDIA Mellanox MSB7700
[Deployment Node: gpumadmell]
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
Brocade VDX 6740T
[Deployment Node: VDX6740-11]
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
FS S5850-48B8C-PE
[Deployment Node: netcms01]
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
Extreme Networks SLX 9540
[Deployment Node: SLX-9540-IFCA]
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
Brocade VDX 6740
[Deployment Node: VDX6740-[21,22,31,32]]
x4
Network Switch OS
from Network Switches
[Container]
Operating System of the Network Switches
NVIDIA Mellanox MQM8700
[Deployment Node: scloudmell]

Three new boolean facts have been added to the Puppet server utilities to dynamically manage the loading of different agents and their configurations, considering the machine characteristics:

- `nvidia_gpu`: This fact has `True` value if the machine has an NVIDIA GPU driver installed. In this case, NVIDIA DCGM and the necessary collectd configuration will be installed on the system for integration. It has value `False` in any other case and therefore, does not perform any configuration.
- `intel_rapl`: This fact has value `True` if the machine has access to the path where the RAPL energy meter registers are read. In this case, the plug-in to read these registers will be enabled in collectd. As in the previous case, no action is taken in case of `False` value.
- `qemu_installed`: This fact has value `True` if the machine has qemu installed or `False` otherwise. If the previous fact is also `True`, Scaphandre agent will be installed in the server and its corresponding connector with collectd, in order to obtain the energy consumption of the hosted VMs.

6.2.1.2 Puppet Modules

As shown in Figure 4.4, there are three main agents deployed on the machines. The rollout of each is defined in its own dedicated Puppet module. The deployment and configuration of these modules are determined by the values of the facts outlined earlier.

- **Collectd Module:** Responsible for installing and configuring the collectd agent, following the configuration explained on section 5.1.1. Furthermore, if a machine is not virtual, it enables power-monitoring plug-ins and also provides custom plug-ins code.
- **NVIDIA DCGM Module:** Installs the NVIDIA DCGM service on the host machine, copies the Python bindings required for integration with the `collectd` agent, and enables the plug-in accordingly. It also verifies that the service is running.
- **Scaphandre Module:** Responsible for installing the Scaphandre agent, creating a systemd service to run it, adding the reader plug-in to the collectd agent configuration, and verifying that the service is active.

Lastly, the OpenStack Exporter snap package is deployed within the existing Nova module.

6.2.2 IFCA Cloud Virtual Machines

The deployment of the multiple agents, which involves several steps and configuration files, is automated through a bash script to ensure a seamless procedure for users. This script installs and configures the software on both Red Hat-based and Debian-based systems. While running, it retrieves the network configuration parameters to determine the VLAN, installs collectd daemon and enables the selected set of plug-ins. The script activates NVIDIA GPU monitoring if applicable. It also ensures required services are active and verifies the collectd service is running. The script can be downloaded and executed directly using the command shown in listing 6.1.

Listing 6.1: Automatic agents deployment on IFCA Cloud VMs

```
root@IFCAcloudVM:~# curl -s -S -L https://gitlab.ifca.es/monitor/main/install.sh | bash
```

The access to the platform must be requested through a support ticket. The administrator attached as a response the link and the requester has to provide as response the VLAN number code issued by the script. Then a system administrator could provide access to Grafana to the user and their machines, adding this username-VLAN mapping into relational SQLite database.

CHAPTER 7

Discussion and Conclusion

Finally, this last chapter presents the achievements obtained after carrying out this project, the lessons learned during the execution of this work and the points of improvement and growth for the future of the developed platform.

7.1 Achievements

This project has successfully developed a scalable, modular and extensible observability platform for data processing centers, leveraging open-source tools to monitor resource usage, power consumption and service availability. It features easy integration of new data sources through a flexible framework, ensuring efficient data processing and metrics visualization. The outdated monitoring stack was replaced, making the platform more intuitive, user-friendly, performance and cost-effective. Moreover, efforts have focused on deploying the platform with a minimal number of agents, ensuring easy interconnections and maximizing data collection per agent. Network usage was optimized by reducing the number of connections, as telemetry generates continuous network traffic.

Moreover, it should be noted that the implementation of this platform has enabled the integration of monitoring capabilities for systems that were previously unsupported:

- **Physical machines:** In addition to usage metrics, the platform now tracks the energy consumption of each individual node and their components, which was not possible before.
- **Cloud virtual machines:** The system now supports monitoring of both resource usage and power consumption of virtual machines deployed in cloud environments.
- **Schneider Electric sensors:** Energy monitoring has been enhanced by fully integrating Schneider Electric sensors, which were previously limited to reporting only current values.
- **NVIDIA GPUs:** Previously unavailable, GPUs observability has now been added, including detailed metrics such as status, utilization, temperature, and power consumption.
- **Network infrastructure:** The platform monitors CPU usage, memory, and temperature of the network switches, as well as the status and bandwidth of each of their interfaces.
- **OpenStack:** The state of OpenStack agents and services is now integrated into the observability stack. While this information was already accessible via OpenStack's native interface, the unified platform offers a more intuitive view for system administrators.

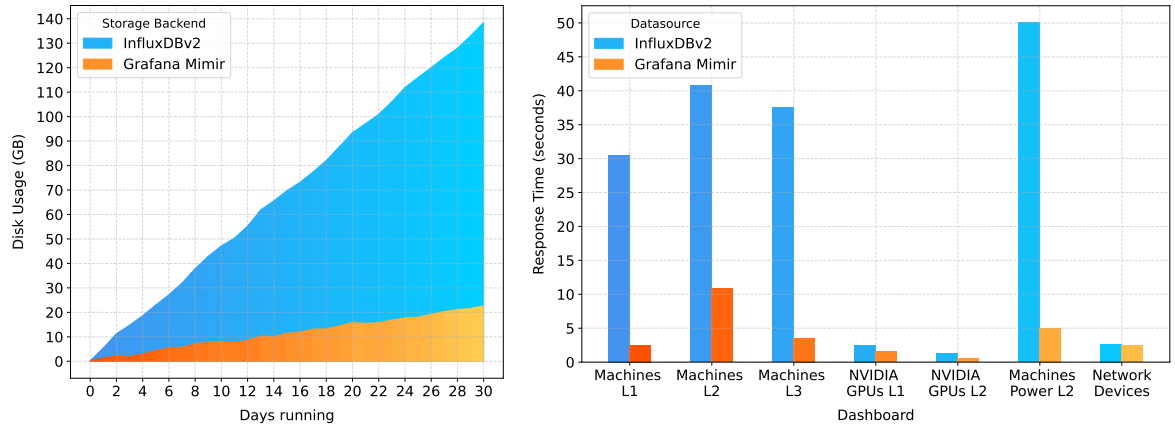
The development of this work has led to internal enhancements of the existing monitoring system, as well as external improvements to other systems through accurate monitoring and the use of this information to support decision-making.

7.1.1 Performance Enhancements

As shown in Figures 7.1a and 7.1b, the new platform's performance has improved considerably, comparing it with the InfluxDB update alternative, in terms of disk usage and response time of the dashboards. Both tests have been developed under equal conditions and in the same period of time, so the amount of metrics and their properties are the same for both systems.

Regarding Figure 7.1a, it shows the disk space occupied by the data stored in it, by means of both backends, over the course of a month. The difference is overwhelming and there is not much to comment on. This ensures that the platform is far more scalable in terms of disk capacity availability to ensure that all the metrics collected can be stored for as long as required.

Subsequently, Figure 7.1b shows the response time, including data request and plots rendering, of a set of dashboards that were developed for both backends, with the aim of carrying out this benchmark. To perform the measurements, the Firefox Profiler is used, a tool available within the browser's web developer tools, with the cache option disabled. For dashboards with a small data set and few charts, response times are not divergent in comparison. However, the difference becomes noticeable in dashboards that display a large volume of metrics, such as those showing multiple metrics for all machines in the datacenter. This is where Grafana Mimir has the advantage. These results evidence that the platform has become significantly more usable than before, as the time required for user interaction to use the platform has decreased.



(a) Storage Backend Disk Usage During 30 Days

(b) Grafana Dashboards Response Time with Both Data Sources

Figure 7.1: Backend Performance Comparison (InfluxDBv2 vs. Grafana Mimir)

7.1.2 Resource Placement Improvement

In addition to tracking the health of systems and services in real time, the platform provides data that enables the analysis of usage patterns to optimize resource placement. By offering both historical and real-time data, inefficiencies in resource allocation can be identified. A clear example is the optimization of the distribution of virtual machines in the cloud infrastructure among servers with graphics processing units. By analyzing their usage, something that was not possible before, many virtual machines use a compute flavor that provides more graphics processing unit than actually needed, based on their historical usage, which resulted in some graphics processing unit allocated but being left underutilized.

This insight led to a refinement in resource provisioning to better align graphics processing unit allocation with real demand. As a result, resource utilization improved, idle capacity was reduced, and available hardware could be more effectively leveraged. These adjustments contributed to lower energy consumption and enhanced scalability of the cloud infrastructure.

7.1.3 Monitoring Dataset

As a result of collecting high-resolution data across various relevant areas in the datacenter, a dataset has been generated along with its corresponding data descriptor. This dataset is currently being prepared for publication in DIGITAL.CSIC [47], the institutional repository of the Spanish National Research Council.

The dataset includes, on one hand, data collected over a period of 4 months from various usage parameters of 130 physical machines, including CPU, memory, disk, and network usage, as well as power usage measured at multiple levels. This energy consumption data includes the readings reported by IPMI and the measurements obtained through the RAPL interface and the Scaphandre agent, all with the highest possible granularity. A detailed hardware inventory by model is also provided for the set of physical machines.

On the other hand, the dataset includes energy consumption data obtained via the Scaphandre agent from 612 virtual machines running on the 130 servers. For each VM, additional information is provided to establish associations, such as the hypervisor, flavor, user, project, and domain within OpenStack. All identifying data have been anonymized to ensure data privacy.

The primary goal of this dataset is to provide data for an in-depth study of the relationship between the underlying hardware configuration, resource usage, or flavor characteristics in the case of VMs, and their resulting energy consumption, for both physical and virtual machines. This will contribute to the analysis of energy efficiency in computing systems, helping optimize energy consumption in datacenters and improve resource management in cloud infrastructures.

7.2 Future Work

7.2.1 Scale Up the Number of Monitored Services

Initially, the project focused on monitoring OpenStack, as it manages most of the trackable computing resources. However, as expected, there are other key services at the IFCA's datacenter, including distributed storage systems like Ceph and IBM Spectrum Scale (formerly GPFS), as well as the Slurm task queue manager for HPC and Grid jobs, that also need to be monitored leading to ease their maintenance. Integrating these services into the platform is not too far-reaching, involving the deployment of compatible Prometheus exporters with an API that each service provides and connecting them to the Prometheus instance. Finally, relevant dashboards need to be rolled out in Grafana to display the metrics in an accessible manner.

This process is already underway and demonstrates that the system is both scalable and expandable with new modules, thus supporting once again that the objectives of the project have been met.

7.2.2 Include New Related Data Sources for Completeness

An important future task involves incorporating additional data sources to enable real-time estimation of the datacenter's environmental impact, with granularity ranging from the whole facility down to the virtual machine level. Building on existing energy consumption data, the system could quantify the impact in terms of gCO_2eq, standing for the emissions associated with the generation of the electricity needed to operate the infrastructure. This functionality would reinforce ongoing efforts aimed at improving energy efficiency and minimizing the environmental footprint, while also enabling consistent reporting and dissemination of the environmental impact of IFCA's computing infrastructure.

7.2.3 Closing the Loop with Meta-Monitoring

Finally, it would be interesting to actively monitor the services deployed on the platform, especially those that receive and manage the huge amount of metrics acquired. For example, it would be convenient to monitor the data ingestion performed by Telegraf, the amount of data series created in Prometheus or the disk usage by Grafana Mimir writes, from the platform itself.

This page intentionally left blank.

Bibliography

[1] Advanced Computing and e-Science Group. *IFCA Confluence*. URL: https://confluence.ifca.es/spaces/IC/overview (visited on 29/03/2025).

[2] InfluxData Inc. *InfluxDB OSS and Enterprise Roadmap Update from InfluxDays EMEA*. May 2021. URL: https://www.influxdata.com/blog/influxdb-oss-and-enterprise-roadmap-update-from-influxdays-emea/ (visited on 10/03/2025).

[3] InfluxData Inc. *The future of Flux | Flux Documentation*. URL: https://docs.influxdata.com/flux/v0/future-of-flux/ (visited on 10/03/2025).

[4] European Commission. *Data centres in Europe: energy performance and sustainability reporting scheme*. en. Text. January 2024. URL: https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/13818-Data-centres-in-Europe-reporting-scheme_en (visited on 16/02/2025).

[5] Consejo Superior de Investigaciones Científicas. *Plan de sostenibilidad del CSIC (2024-2026)*. Español. February 2025. URL: https://www.csic.es/sites/default/files/2025-02/plan_sostenibilidad_CSIC_24-26_0.pdf.

[6] Kent Beck et al. *Manifesto for Agile Software Development*. URL: https://agilemanifesto.org/ (visited on 13/03/2025).

[7] Mike Cohn. *User stories applied: for agile software development*. Addison-Wesley signature series. Boston: Addison-Wesley, 2004. ISBN: 978-0-321-20568-1.

[8] James A. Highsmith. *Agile project management: creating innovative products*. Agile software development series. Boston: Addison-Wesley, 2004. ISBN: 978-0-321-21977-0.

[9] Luiz André Barroso, Urs Hölzle and Parthasarathy Ranganathan. *The Datacenter as a Computer: Designing Warehouse-Scale Machines*. en. Synthesis Lectures on Computer Architecture. Cham: Springer International Publishing, 2019. ISBN: 978-3-031-00633-3 978-3-031-01761-2. DOI: 10.1007/978-3-031-01761-2. URL: https://link.springer.com/10.1007/978-3-031-01761-2 (visited on 29/01/2025).

[10] Mary Zhang. *Data Center Monitoring: A Comprehensive Guide*. en-US. March 2024. URL: https://dgtlinfra.com/data-center-monitoring/ (visited on 05/03/2025).

[11] Krishna Kant. 'Data center evolution: A tutorial on state of the art, issues, and challenges'. In: *Computer Networks*. Virtualized Data Centers 53.17 (December 2009), pp. 2939–2965. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2009.10.004. URL: https://www.sciencedirect.com/science/article/pii/S1389128609003090 (visited on 29/01/2025).

[12] Schneider Electric, Inc. *How is a single phase service bypass panel wired?* en-GB. April 2015. URL: https://www.se.com/uk/en/faqs/FA240007/ (visited on 26/04/2025).

[13] OpenInfra Foundation. *OpenStack | Open Source Cloud Computing Infrastructure*. en. URL: https://www.openstack.org/software/ (visited on 05/11/2024).

[14] Florian octo Forster, Sebastian tokkee Harl. *collectd*. en-US. URL: https://www.collectd.org/ (visited on 17/07/2024).

[15] NVIDIA Corporation. *NVIDIA DCGM Documentation*. en. Documentation. May 2024. URL: https://docs.nvidia.com/datacenter/dcgm/latest/index.html (visited on 03/07/2024).

[16] Thomas Krenn. *IPMI Basics - Thomas-Krenn-Wiki*. URL: https://www.thomas-krenn.com/en/wiki/IPMI_Basics#cite_note-ipmispec-1 (visited on 29/08/2024).

[17] Intel Corporation. *Intelligent Platform Management Interface Specification v2.0 rev. 1.1*. en. October 2013. URL: https://www.intel.com/content/www/es/es/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html (visited on 29/08/2024).

[18] Douglas Mauro and Kevin J. Schmidt. *Essential SNMP*. eng. 2nd ed. Sebastopol: O'Reilly Media, Inc, 2009. ISBN: 978-0-596-00840-6.

[19] Fortinet. *¿Qué es el Protocolo simple de administración de red (SNMP)? ¿Es seguro?* es. URL: https://www.fortinet.com/lat/resources/cyberglossary/simple-network-management-protocol.html (visited on 20/03/2025).

[20] Marshall T. Rose and Keith McCloghrie. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*. Request for Comments RFC 1213. Num Pages: 70. Internet Engineering Task Force, March 1991. DOI: 10.17487/RFC1213. URL: https://datatracker.ietf.org/doc/rfc1213 (visited on 08/04/2025).

[21] Hubblo. *Scaphandre Docs*. Documentation. URL: https://hubblo-org.github.io/scaphandre/book/ (visited on 09/09/2024).

[22] Kashif Nizam Khan et al. 'RAPL in Action: Experiences in Using RAPL for Power Measurements'. en. In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 3.2 (June 2018), pp. 1–26. ISSN: 2376-3639, 2376-3647. DOI: 10.1145/3177754. URL: https://dl.acm.org/doi/10.1145/3177754 (visited on 30/03/2025).

[23] Jorge Niedbalski. *openstack-exporter/openstack-exporter*. URL: https://github.com/openstack-exporter/openstack-exporter (visited on 30/03/2025).

[24] InfluxData Inc. *Telegraf - Open Source Server Agent | InfluxDB*. January 2022. URL: https://www.influxdata.com/time-series-platform/telegraf/ (visited on 03/07/2024).

[25] Open Home Foundation. *Home Assistant*. en. URL: https://www.home-assistant.io/ (visited on 29/01/2025).

[26] Prometheus. *Prometheus - Monitoring system & time series database*. en. URL: https://prometheus.io/ (visited on 31/10/2024).

[27] Grafana Labs. *Grafana Mimir OSS | Prometheus long-term storage*. en. URL: https://grafana.com/oss/mimir/ (visited on 31/10/2024).

[28] Grafana Labs. *Grafana OSS | Leading observability tool for visualizations & dashboards*. en. URL: https://grafana.com/oss/grafana/ (visited on 17/07/2024).

[29] Al Sargent. *How to Use Starlark with Telegraf*. August 2020. URL: https://www.influxdata.com/blog/how-use-starlark-telegraf/ (visited on 17/03/2025).

[30] Perforce Software, Inc. *Puppet Infrastructure & IT Automation at Scale*. en. URL: https://www.puppet.com/ (visited on 30/03/2025).

[31] Ronald E Jeffries. *Essential XP: Card, Conversation, Confirmation*. URL: https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/ (visited on 25/03/2025).

[32] Ian Sommerville. *Software engineering*. eng. 10. ed. Boston, Munich: Pearson, 2016. ISBN: 978-0-13-394303-0.

[33] *ISO/IEC 25010:2011(en), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en (visited on 22/11/2024).

[34] Simon Brown. *The C4 model for visualising software architecture*. en-US. URL: https://c4model.com/ (visited on 03/11/2024).

[35] IcePanel Technologies Inc. *IcePanel | Explain complex software systems*. en. URL: https://icepanel.io/ (visited on 04/11/2024).

[36] Terrehon Bowden et al. *The /proc filesystem*. Documentation. The Linux Fundation, October 1999. URL: https://www.kernel.org/doc/Documentation/filesystems/proc.txt (visited on 08/04/2025).

[37] Hangbin Liu. *Introduction to Linux interfaces for virtual networking*. en. October 2018. URL: https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking (visited on 08/11/2024).

[38] USC Information Sciences Institute. *Transmission Control Protocol*. Request for Comments RFC 793. Num Pages: 91. Internet Engineering Task Force, September 1981. DOI: 10.17487/RFC0793. URL: https://datatracker.ietf.org/doc/rfc793 (visited on 08/04/2025).

[39] Scott McMillan. *Setting Up GPU Telemetry with NVIDIA Data Center GPU Manager*. en-US. January 2019. URL: https://developer.nvidia.com/blog/gpu-telemetry-nvidia-dcgm/ (visited on 31/10/2024).

[40] Observium Limited. *Observium MIB Database*. Database. URL: https://mibs.observium.org/ (visited on 09/09/2024).

[41] Breina. *Breina/PowerTagGateway*. original-date: 2022-07-24T20:29:01Z. January 2025. URL: https://github.com/Breina/PowerTagGateway (visited on 01/04/2025).

[42] Schneider Electric, Inc. *Difference Between Active Power, Reactive Power and Apparent Power*. en. January 2024. URL: https://eshop.se.com/in/blog/post/difference-between-active-power-reactive-power-and-apparent-power.html (visited on 04/11/2024).

[43] 7SIGNAL, Inc. *Interference: What Is Wi-Fi Interference?* en. URL: https://www.7signal.com/wi-fi-interference (visited on 04/03/2025).

[44] Grafana Labs. *Grafana Mimir documentation | Grafana Mimir documentation*. en. URL: https://grafana.com/docs/mimir/latest/ (visited on 31/03/2025).

[45] Bacula.org. *The best open source backup software for Linux*. en-US. URL: https://www.bacula.org/ (visited on 08/04/2025).

[46] Puppet, Inc. *Puppet Core - Facter: Core Facts*. en. Documentation. URL: https://puppet.com/docs/puppet/7/core_facts.html (visited on 02/04/2025).

[47] Jaime Iglesias Blanco and Álvaro López García. *IFCA Cloud energy consumption metrics*. eng. Accepted: 2025-05-12T12:37:01Z. 2025. DOI: 10.20350/digitalCSIC/17271. URL: https://digital.csic.es/handle/10261/388993 (visited on 13/05/2025).

This page intentionally left blank.

Appendices

This page intentionally left blank.

APPENDIX A

Datacenter Inventory

This appendix details the different types of equipment and assets involved in the monitoring of the Institute of Physics of Cantabria's datacenter. This includes both the physical IT infrastructure of the facility, such as servers, network switches and electrical energy meters. Furthermore, since the datacenter is mostly virtualized and part of the platform is focused on them, virtual machines are also detailed as computing resources.

Additionally, this inventory serves as a checklist to organize, prove and verify whether the monitored features match the system's characteristics and to determine what monitoring agents are required for each group of machines.

A.1 Compute Machines

This section provides a detailed overview of the machines available in the datacenter at the Institute of Physics of Cantabria [1]. The infrastructure is organized and classified into distinct services and groups. It encompasses both physical servers and virtual machines (VMs).

Table A.1: Machines available at IFCA's datacenter under monitoring

Group	#	VM	Server Model / NVIDIA GPU	#CPU	#GPU	Memory	Disk	Infiniband
Service: HPC Compute (*Altamira v2*)								
node	158	☐	*IBM System iDataPlex dx360 M4*	2	0	64GB	500GB	☑
Service: HPC Management								
login	2	☐	*IBM System x3550 M4*	1	0	185GB	220GB	☑
ojancano	2	☑	-	-	0	4GB	90GB	☐
Service: HPC Worker Nodes								
wncompute	55	☑	-	-	0	460GB	300GB	☑
wngpu	5	☑	-	-	2	122GB	300GB	☑
Service: Grid Compute								
cmsfj	18	☐	*Fujitsu PRIMERGY BX924 S4*	2	0	90GB	400GB	☐
cmsgiga	8	☐	*Lenovo ThinkSystem SR650*	2	0	188GB	1.2TB	☑
cmsgpu	1	☐	*Supermircro SYS-420GP-TNR / A30*	2	8	250GB	1TB	☑
cmsln	20	☐	*Lenovo ThinkSystem SD530*	2	0	192GB	450GB	☐
Service: Grid Management								
pool	10	☑	-	-	0	5GB	26GB	☐
...	11	☑	-	-	0	-	-	☐
Service: Grid Worker Nodes								
wngrid	33	☑	-	-	0	700GB	600GB	☑
wncmsgpu	1	☑	-	-	8	230GB	300GB	☐

Table A.1: Machines available at IFCA's datacenter under monitoring

Group	#	VM	Server Model / NVIDIA GPU	#CPU	#GPU	Memory	Disk	Infiniband
Service: Cloud Compute								
aitken	2	☐	*IBM System x3850 X5*	4	0	1TB	2.5TB	☐
fcloud	5	☐	*Lenovo ThinkSystem SR630*	2	0	775GB	250GB	☐
gpumad	20	☐	*Lenovo ThinkSystem SR650 / V100*	2	2	188G	2TB	☑
gpumax	10	☐	*Supermircro SYS-4028GP-TNR / T4*	2	8	360GB	500GB	☑
mcloud	8	☐	*Dell Inc. PowerEdge C6420*	2	0	192GB	4TB	☐
metcloud	40	☐	*Gigabyte H262-Z61-00*	2	0	500GB	2TB	☑
scloud	65	☐	*Lenovo ThinkSystem SR630 V2*	2	0	370GB	1TB	☑
Service: Cloud Management								
openstack	15	☑	-	-	0	-	-	☐
net	1	☐	*Lenovo System x3550 M5*	1	0	24GB	400GB	☐
Service: Management								
bacula	1	☐	*Lenovo System x3550 M5*	1	0	61GB	450GB	☐
db	3	☐	*Fujitsu PRIMERGY RX2530 M4*	1	0	14GB	200GB	☐
geoffrey	1	☐	*Lenovo ThinkSystem SR570*	2	0	30GB	4TB	☐
grafana	1	☐	*Dell Inc. PowerEdge R730*	1	0	62GB	50TB	☐
hutch	1	☐	*Lenovo ThinkSystem SR630*	2	0	188GB	210GB	☐
monitor	1	☐	*IBM System x3650 M2*	2	0	47GB	1.6TB	☐
repo	1	☐	*Lenovo ThinkServer TD350*	1	0	16GB	77TB	☐
starsky	1	☐	*IBM System x3550 M4*	1	0	16GB	490GB	☐
wngw	1	☐	*Dell Inc. PowerEdge R320*	1	0	16GB	460GB	☐
virt	5	☐	*Fujitsu PRIMERGY RX200 S7*	1	0	4GB	20GB	☐
zeus	7	☐	*Fujitsu PRIMERGY RX2530 M4*	2	0	188GB	70GB	☐
...	12	☑	-	-	0	-	-	☐
Service: Ceph Storage								
cephiscsi	3	☑	-	-	0	2GB	16GB	☐
cephmon	3	☑	-	-	0	4GB	12GB	☐
cephosd	6	☐	*Supermicro SSG-6029-E1CR24L*	2	0	60GB	400GB	☐
cephrgw	2	☑	-	-	0	4GB	16GB	☐
Service: GPFS Storage								
dss	2	☐	*Lenovo ThinkSystem SR650 V2*	2	0	500GB	500GB	☑
quorum-dss	1	☑	-	-	0	7GB	100GB	☑
Service: IFCA Local Services								
...	13	☑	-	-	0	-	-	☐

As of the date of completion of this work, 384 user virtual machines in the IFCA cloud remain to be detailed, which could also be monitored through the platform if their owner wishes.

A.2 Network Switches

This section details the Object Identifiers (OIDs) used for monitoring various parameters within the datacenter's network switches. It lists the OIDs for each monitored field, along with their corresponding types, grouped by switch model. As some of them are common across switches, they are shown in Table A.2.

Table A.2: Common MIBs: Summary of selected OIDs

Field Name:	OID:	Type:
Device Name	`RFC1213-MIB::sysName.0`	Field
Device Model	`RFC1213-MIB::sysDescr.0`	Field
Uptime	`DISMAN-EXPRESSION-MIB::sysUpTimeInstance`	Field
Num Interfaces	`IF-MIB::ifNumber.0`	Field
Interfaces	`IF-MIB::ifTable` and `IF-MIB::ifXTable`	Table

Table A.3: IBM RackSwitch G8052 MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	`SNMPv2-SMI::enterprises.26543.2.7.7.3.1.1.2.1.2.1`	Field
CPU Usage	`HOST-RESOURCES-MIB::hrProcessorLoad.1`	Field
Memory Usage	`SNMPv2-SMI::enterprises.1588.2.1.1.1.26.6.0`	Field
Temperature [1-8]	`SNMPv2-SMI::enterprises.26543.100.100.14.[11-13,32-36].0`	Field

Table A.4: FS S5850-48B8C-PE MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	`SNMPv2-SMI::enterprises.52642.1.43.1.1.1.2.0`	Field
CPU Usage	`SNMPv2-SMI::enterprises.52642.1.1.9.1.0`	Field

Table A.5: Extreme Networks ERS 4950GTS MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	`SNMPv2-SMI::mib-2.14.8.1.1.(IP Address).0.0`	Field

Table A.6: ExtremeSwitching SLX 9540 MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	`SNMPv2-SMI::enterprises.1588.2.1.1.1.1.25.0`	Field
CPU Usage [1-8]	`HOST-RESOURCES-MIB::hrProcessorLoad.1966[08-15]`	Field
Memory Usage	`SNMPv2-SMI::enterprises.1588.2.1.1.1.26.6.0`	Field
Temperature [1-8]	`SNMPv2-SMI::enterprises.1588.2.1.1.1.1.22.1.4.[1-8]`	Field

Table A.7: Mellanox MQM8700-HS2F MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	`IP-MIB::ipAdEntAddr.(IP Address)`	Field
CPU Usage [1-4]	`HOST-RESOURCES-MIB::hrProcessorLoad.1966[08-11]`	Field
Memory Usage	`SNMPv2-SMI::enterprises.1588.2.1.1.1.26.6.0`	Field
Temperature [1-9]	`SNMPv2-SMI::mib-2.99.1.1.1.4.(200020011|200100011|200110011|200110012|200400011|200410011|200450011|601240011|602240011)`	Field

Table A.8: Mellanox MSB7700-ES2F MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	IP-MIB::ipAdEntAddr.(IP Address)	Field
CPU Usage [1-2]	HOST-RESOURCES-MIB::hrProcessorLoad.1966[08-09]	Field
Memory Usage	SNMPv2-SMI::enterprises.1588.2.1.1.1.26.6.0	Field
Temperature [1-8]	SNMPv2-SMI::mib-2.99.1.1.1.4.(200020011|200100011|200110011|200110012|200400011|200410011|200450011|601240011)	Field

Table A.9: Brocade BR-VDX6740 MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	SNMPv2-SMI::enterprises.1588.2.1.1.1.1.25.0	Field
CPU Usage	SNMPv2-SMI::enterprises.1588.2.1.1.1.26.1.0	Field
Memory Usage	SNMPv2-SMI::enterprises.1588.2.1.1.1.26.6.0	Field
Temperature [1-4]	SNMPv2-SMI::enterprises.1588.2.1.1.1.1.22.1.4.[1-4]	Field

Table A.10: Brocade BR-VDX6740T-56-1G-F MIB: Summary of selected OIDs

Field Name:	OID:	Type:
Management IP	SNMPv2-SMI::enterprises.1588.2.1.1.1.1.25.0	Field
CPU Usage	SNMPv2-SMI::enterprises.1588.2.1.1.1.26.1.0	Field
Memory Usage	SNMPv2-SMI::enterprises.1588.2.1.1.1.26.6.0	Field
Temperature [1-3]	SNMPv2-SMI::enterprises.1588.2.1.1.1.1.22.1.4.[1-3]	Field

A.3 Electric Sensors

There are three types of sensors: PowerTag Rope 600A 3P/3P+N (A9MEM1591)[1], in Figure A.1a, PowerTag Flex 160A 3P/3P+N (A9MEM1580)[2], in Figure A.1b and PowerTag Universal 3P+N (A9MEM1570)[3], in Figure A.1c. All of them are connected to a Schneider Acti9 PowerTag Link HD wireless communication hub (A9XMWD100)[4], shown in Figure A.1d, which offers an administration web interface, to manage and configure all the sensors.

(a) PowerTag Rope 600A 3P/3P+N (b) PowerTag Flex 160A 3P/3P+N (c) PowerTag Universal 3P+N (d) Acti9 PowerTag Link

Figure A.1: Wireless Sensors and Communication Hub Device from Schneider Electric

Two out of the three sensors of the first type are used to measure the two unmaintained facility's main energy input lines. The third sensor, together with one sensor of the second type, is used to monitor the two maintained facility's main energy input lines. The remaining sensor of the second type, along with the seventeen sensors of the third type, is distributed across various assets, including cooling systems, fans, chillers, and high-power racks.

[1] PowerTag Rope 600A 3P/3P+N: https://www.se.com/es/es/product/A9MEM1591
[2] PowerTag Flex 160A 3P/3P+N: https://www.se.com/es/es/product/A9MEM1580/
[3] PowerTag Universal 3P+N: https://www.se.com/es/es/product/A9MEM1570/
[4] Acti9 PowerTag Link: https://www.se.com/es/es/product/A9XMWD100/

APPENDIX B

User Stories

This appendix presents the user stories made in the requirements analysis process. They describe in a detailed and understandable way all the user needs and the system capabilities or features to be developed. In addition, the different acceptance criteria for the subsequent evaluation of the project are specified, ensuring clear expectations and project alignment. For each user story, the stakeholder requesting the specific feature is also clearly identified.

ID:	`US01`
Name:	Monitor operating system statistics of each machine.
Description:	As a system administrator, I wish to track system indicators across all machines, to ensure proper functioning and to quickly detect any failures or anomalies.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. Each machine must be uniquely identified by its hostname. 2. Monitor host status (`UP`/`DOWN`) and uptime to ensure availability and reliability. 3. Track CPU usage, system load and process status for performance monitoring. 4. Record memory usage in bytes and swap usage to ensure sufficient memory resources on the host. 5. Monitor network activity, including interface usage (bytes sent/received), bandwidth (in bits/sec) and port status to detect any network issues. Also, track packet statistics (sent/received) and errors by interface. Only real network interfaces must be considered. 6. Track disk usage and partitions in bytes and monitor inode usage to ensure sufficient storage is available on each machine. Also, only real disk partitions must be considered. 7. Number of logged-in users on each machine. 8. Data must be collected at regular intervals with a period of at least 30 seconds and must be available for a minimum period of 3 years. 9. A filtering system must exist to categorize machines, allowing filtering by group and service for IFCA computing machines and by VLAN or project name for user virtual machines (VMs).
Comments:	• It must be applicable to all types of machines, including physical, VMs and user-owned cloud instances, ensuring broad coverage. • Machines may run a variety of operating systems, specifically distributions from the Debian family (*Debian 11/12, Ubuntu 20/22*) and Red Hat (*AlmaLinux 8/9*).

ID:	US02
Name:	Monitor the network switches performance and link status.
Description:	As a network administrator, I need to continuously monitor the status of all network switches to ensure reliability and quickly detect failures.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. Each device must be identified by its name or management IP. 2. For each network device, it should display the number of available interfaces, the device's uptime, CPU and memory usage and its temperature, as some of them are equipped with optical transceivers. 3. For each network interface on the device, some identification data must be provided: Interface name or alias and MAC Address. 4. For each network interface of the device, it might be useful to report some link properties: Configured link speed and Interface type. 5. For each network interface of the device, the link status (UP/DOWN), bandwidth (bits/s), aggregate link usage by hour, day, week and month (bytes) and packet error rate should be reported. 6. Data must be collected at regular intervals with a period of at least 30 seconds and must be available for a minimum period of 3 years.
Comments:	• Network switches are from various manufacturers (See annex A.2)

ID:	US03
Name:	Monitor the utilization and power consumption of NVIDIA GPUs.
Description:	As a system administrator, I wish to know how the GPUs installed in the datacenter are actually being used, as well as their power consumption.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. Some identification data must be collected to identify the device clearly. Hostname of the host where is mounted and device UUID. 2. GPU utilization, device memory utilization and instant power usage should be reported. 3. Data must be collected at regular intervals with a period of at least 30 seconds and must be available for a minimum period of 3 years. 4. In addition, it could be relevant to track the GPU temperature, to detect possible heat dissipation problems that lead to potential failures of the device.
Comments:	• NVIDIA GPUs models: See annex A.1. More models will be added.

ID:	US04
Name:	Store historical data about the datacenter power distribution grid.
Description:	As an infrastructure administrator, I want to be able to have a historical evolution of the different electrical parameters obtained by the Schneider Electric sensors.
Stakeholders:	IFCA Computing Service & GreenDIGIT Project.
Acceptance criteria:	1. The data must be collected from a Schneider Acti9 PowerTag Link HD wireless communication hub (A9XMWD100). 2. The integration and incorporation of a new sensor into the platform should be automatic once it is linked to the existing sensor system. 3. Data must be collected at regular intervals with a period of at least 30 seconds and must be available for a minimum period of 3 years.
Comments:	• The installation and configuration of the sensors and hub is already done and operational, so the metrics are now available in real time.

ID:	US05
Name:	Monitor the datacenter physical machines power consumption.
Description:	As an infrastructure administrator, I would like to get and provide the instant power usage in watts of each physical machine placed.
Stakeholders:	IFCA Computing Service & GreenDIGIT Project.
Acceptance criteria:	1. Machine must be identified by its hostname. 2. The platform must provide instant power usage in watts for each physical machine in the datacenter. 3. Measurements should be as close to real consumption as possible, using accurate methods and available sensors. 4. Additional metrics, such as isolated GPU consumption, should also be reported to provide more detailed insight into the machine's energy usage. 5. Data must be collected at regular intervals with a period of at least 30 seconds and must be available for a minimum period of 3 years.
Comments:	• Most servers have a Baseboard Management Controller (BMC) that allows IPMI access for hardware monitoring, but some do not.

ID:	US06
Name:	Monitor the AI4EOSC platform power consumption.
Description:	As a platform developer, I am interested in getting the instant power consumption (W) of the virtual machines where the platform is placed.
Stakeholders:	IFCA Computing Service, GreenDIGIT Project & AI4EOSC Project.
Acceptance criteria:	1. VM must be uniquely identified using its instance name or by its UUID given by the cloud resource manager: OpenStack. 2. Data must be collected at regular intervals with a period of at least 30 seconds and must be available for a minimum period of 3 years.
Comments:	• The AI4EOSC platform is deployed in a cloud computing environment.

ID:	US07
Name:	Calculation of energy efficiency metrics.
Description:	As an infrastructure administrator, I want to calculate energy efficiency metrics, such as PUE and related indicators, at different scopes based on energy consumption data gathered overall infrastructure.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. Energy efficiency metrics must be computed automatically using real-time and historical data from Schneider Electric sensors and other power monitoring sources. 2. The system must support calculations at different scopes, including overall datacenter efficiency and specific subsystems or equipment.
Comments:	• The methodology for computing PUE and other metrics should align with industry best practices to ensure accuracy and comparability.

ID:	US08
Name:	Visualize service metrics on dashboards.
Description:	As a system administrator, I want to visualize metrics on interactive dashboards so that I can easily monitor and analyze the performance and health of services using various visual representations.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. Dashboards should display service metrics through various visual elements, such as line graphs, bar charts, histograms and counters. 2. Dashboards should update in real-time to reflect the latest metrics collected from the monitored systems. 3. The data displayed must be customizable (e.g., time range, metric filters) to meet specific monitoring needs on demand.

ID:	US09
Name:	Monitor the OpenStack computing resources and main services.
Description:	As a cloud administrator, I am interested in getting all the information available from the cloud computing resources and the status of the OpenStack services that manages them.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. To be able to see the current status of each of the OpenStack services. 2. Specifically for Keystone, identity service, I want to be able to visualize information about the projects created and their status. Additionally, it is interesting to show a counter of registered users. 3. For Nova, provisioner of compute instances (i.e. virtual machines), there are different elements to monitor: The flavors and their characteristics (CPUs, DRAMs, Disk and GPUs). The use of these resources by each project. On the other hand, to show the status, from the administrative point of view of the different VMs and finally the status of the different hosts where they are deployed. 4. For Neutron, network service, monitor the different networks and their characteristics, availability of public floating IPs, subnets and the status of the different network agents on the host machines. 5. For Glance, the image service, I want to be able to monitor the size of each image, as well as the overall size. In addition, it might be useful to visualize their properties such as status, visibility, owner. 6. For Placement, cloud inventory and resource usage service, show total usage of CPUs, Memory and Storage, as well as per host. 7. In addition, combined data from the different services can be displayed, e.g. GPUs or floating IPs used per project. 8. Data must be collected at regular intervals with a period of at least 5 minutes and must be available for a minimum period of 3 years.
Comments:	• More services are expected to be monitored in the future.

ID:	US10
Name:	Monitor the OpenStack agent's status.
Description:	As a cloud administrator, I would like to check the status of the multiple OpenStack service agents distributed across the host machines.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. It is possible to see the current status of all agents (Nova and Neutron) and the evolution of each of them over time. 2. Data must be collected at regular intervals with a period of at least 5 minutes and must be available for a minimum period of 3 years.
Comments:	• More services agents are expected to be monitored in the future.

ID:	US11
Name:	Platform access, user authentication and data visibility permissions.
Description:	Access to the monitoring platform to visualize the data should be open to all users with an account in the IFCA Computing Services.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. Access to the monitoring platform must be through an existing Single Sign-On (SSO) authentication system using LDAP. 2. Data extracted from user virtual machines must be visible only to the own user and the administrators of the computing service (External). 3. Data extracted from all the datacenter infrastructure must be visible only to the system administrators of the computing service (Internal).
Comments:	• There may be occasions when certain users need internal data. There must be the possibility to give read-only permissions for certain data.

ID:	US12
Name:	Monitoring platform requirements and design constrains.
Description:	The monitoring platform must meet specific software and deployment requirements to ensure reliability, maintainability and full control over the collected data. All software used for data collection, processing, storage and visualization must adhere to open-source licensing. Additionally, the platform must be entirely self-hosted within the IFCA's infrastructure, avoiding dependencies on external services.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. All components of the monitoring system, including data collection agents, storage backends and visualization tools, must be based on open-source software with active community support and a track record of reliability. 2. The entire platform must be deployed on premise within the IFCA's infrastructure, ensuring that no external services are required for data storage, processing, or authentication. 3. 100% of the monitoring data must be stored within IFCA's infrastructure, with no external data storage or cloud dependencies. 4. Every deployed software component must generate logs to allow verification of correct operation, debugging and security auditing. 5. The system must be designed to ensure maintainability, allowing updates and patches without significant downtime or disruption to the monitoring process.
Comments:	• If a software component does not meet long-term reliability or maintainability expectations, it should be possible to replace it with an alternative open-source solution.

ID:	US13
Name:	Automated deployment of the monitoring platform.
Description:	As an infrastructure administrator, I want the deployment and configuration of the monitoring platform to be automated to ensure consistency, repeatability and ease of maintenance.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. The deployment of all infrastructure-related components of the monitoring platform must be automated using Puppet to the greatest extent possible. 2. Puppet must handle the installation, configuration and management of services, ensuring that all nodes maintain a consistent state over time. 3. For areas where Puppet cannot be applied, such as user-managed virtual machines (VMs), alternative automation mechanisms (e.g., shell scripts) must be available. 4. The automation process must be documented and include clear instructions for execution, troubleshooting and updates. 5. Any configuration changes must be version-controlled, allowing rollback and auditability of modifications.
Comments:	• The automation system should support scalability, allowing new machines to be integrated with minimal manual intervention. • Regular validation should be carried out to ensure that automated deployments remain functional and up to date.

ID:	US14
Name:	Extensible monitoring for additional services.
Description:	As a system administrator, I need the platform to be extensible so that we can monitor other general services and local machine services in the future to ensure comprehensive system oversight.
Stakeholders:	IFCA Computing Service.
Acceptance criteria:	1. The platform must be designed to allow the addition of new monitoring capabilities for external already deployed services, such as Slurm task scheduler, or storage systems such as GPFS or Ceph. 2. The platform should support monitoring of local machine services, such as proxies (NGINX, Apache) or databases (MySQL, PostgreSQL). 3. It must be possible to add and configure monitoring of additional services without requiring major code changes, ensuring easy scalability. 4. Monitoring data for all added services must be integrated with the existing system monitoring and presented in a unified interface.

This page intentionally left blank.

APPENDIX C
Dashboards

This appendix shows the dashboards developed in Grafana to display the data collected by the monitoring platform. The different dashboards have been designed to show in a clear and fast way the different metrics collected, in multiple levels of detail and volume of information.

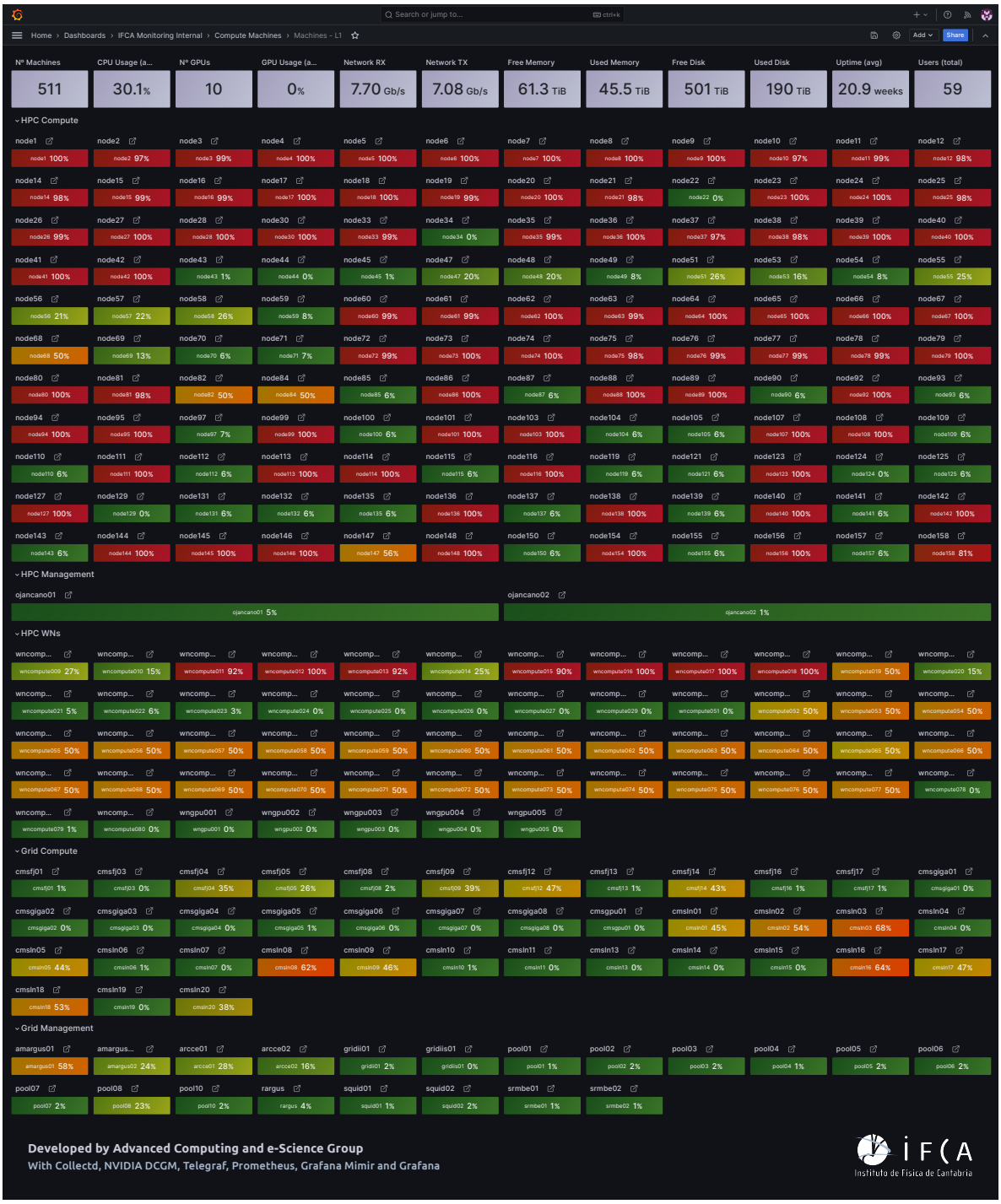


Figure C.1: Dashboards > Compute > Machines L1

Figure C.2: Dashboards > Compute > Machines L2

Figure C.3: Dashboards > Compute > Machines L3

Figure C.4: Dashboards > Compute > NVIDIA GPUs L1

Figure C.5: Dashboards > Compute > NVIDIA GPUs L2

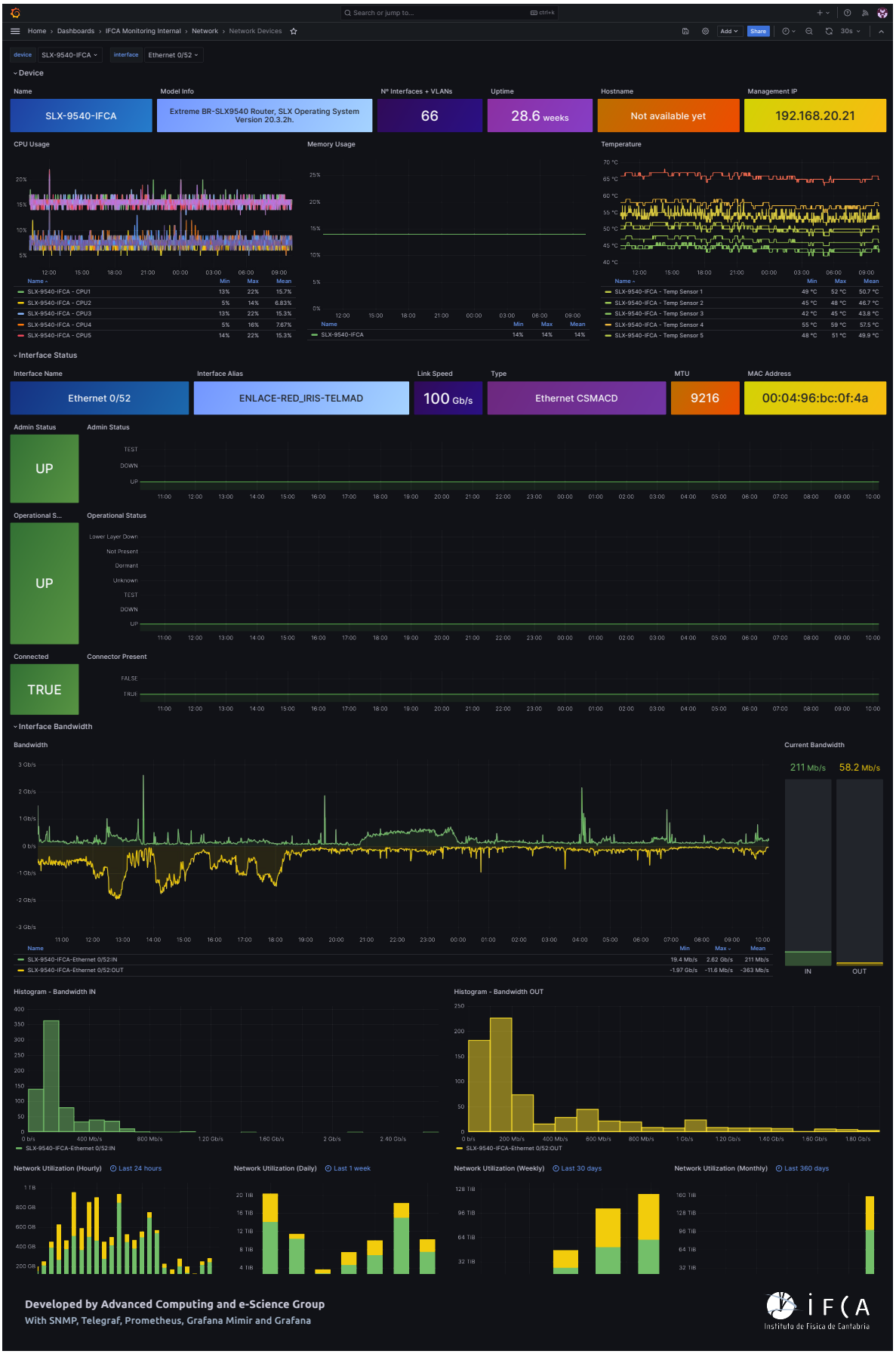


Figure C.6: Dashboards > Network > Network Devices

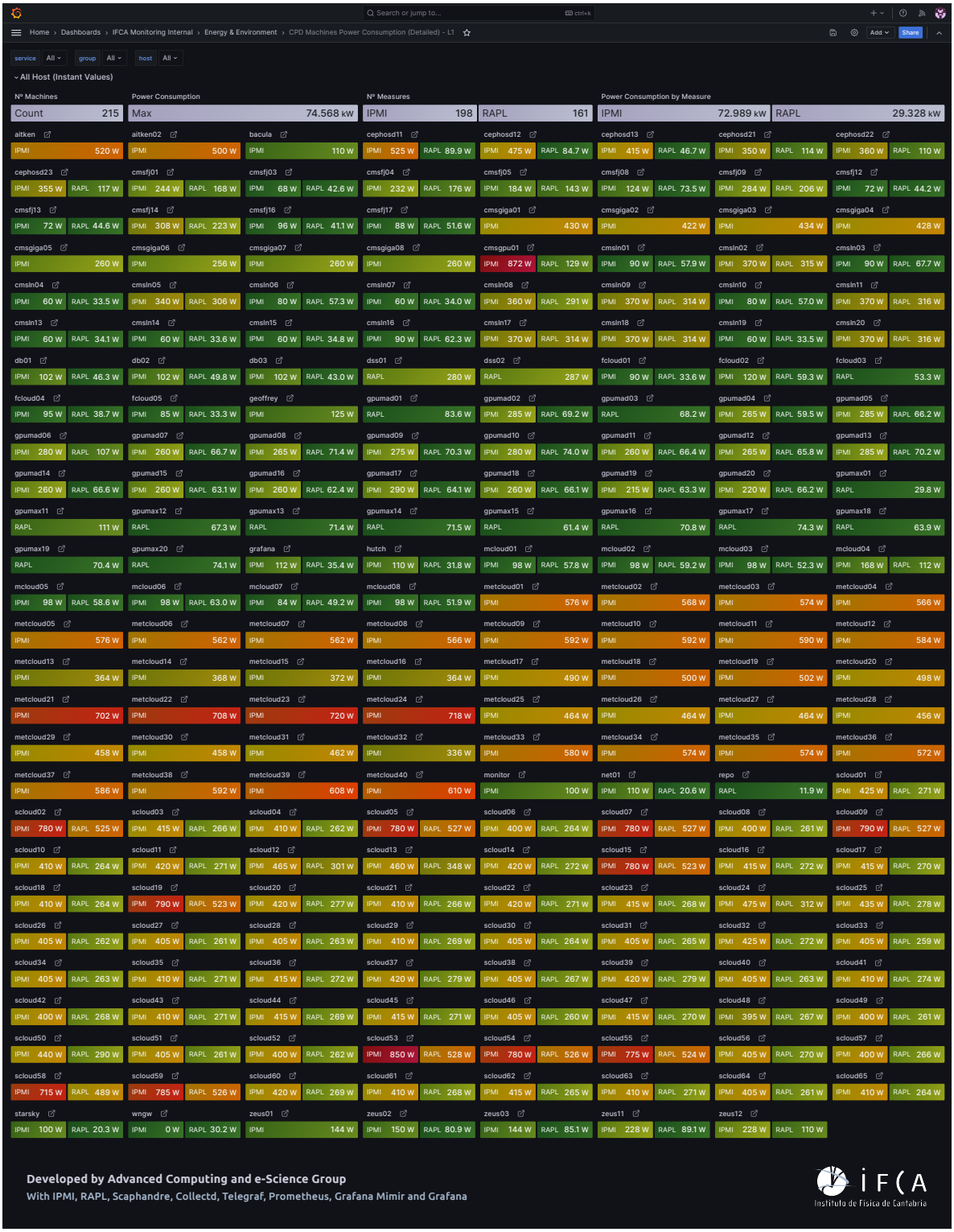


Figure C.7: Dashboards > Energy > Machines Power Consumption L1

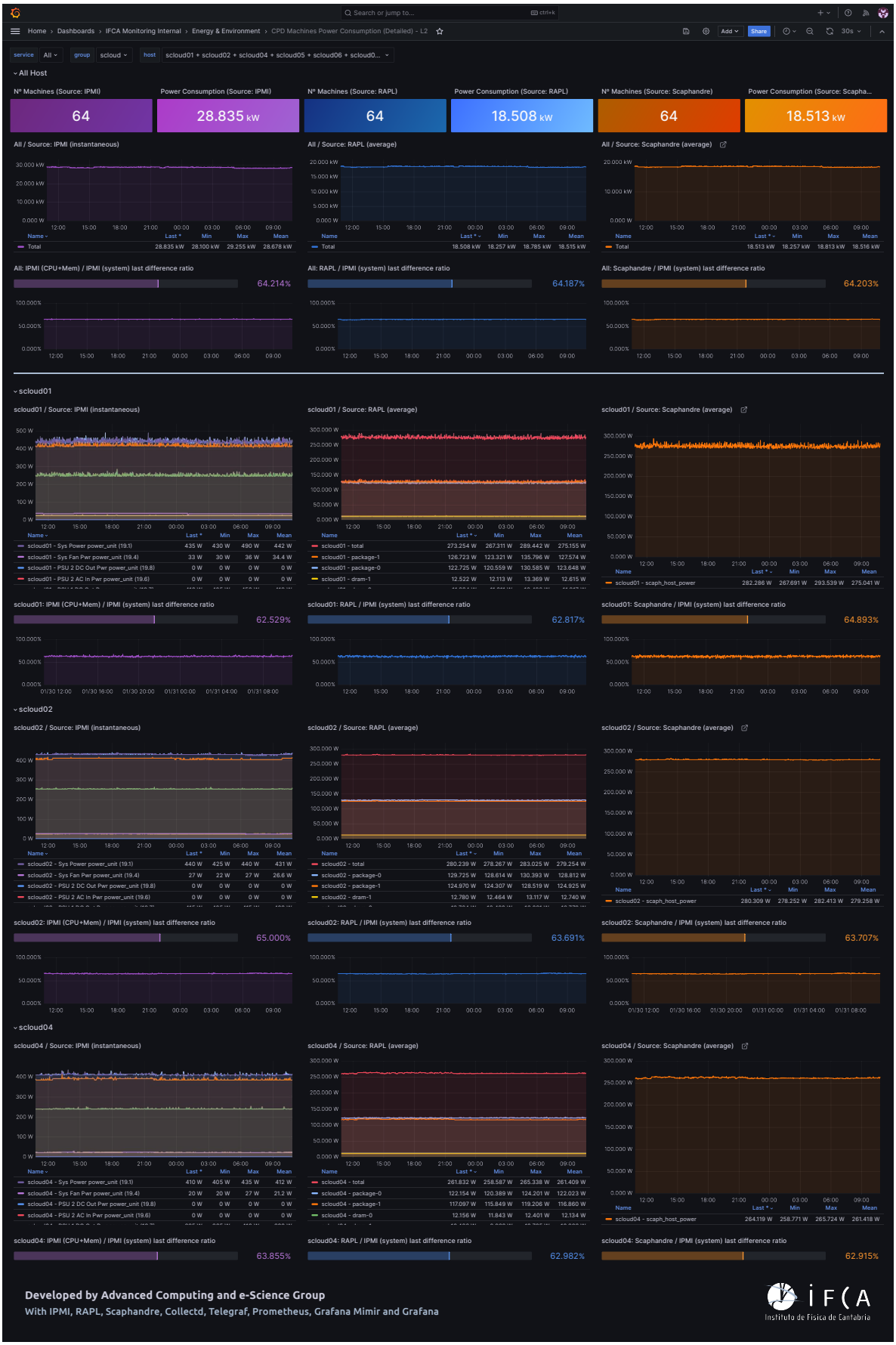


Figure C.8: Dashboards > Energy > Machines Power Consumption L2

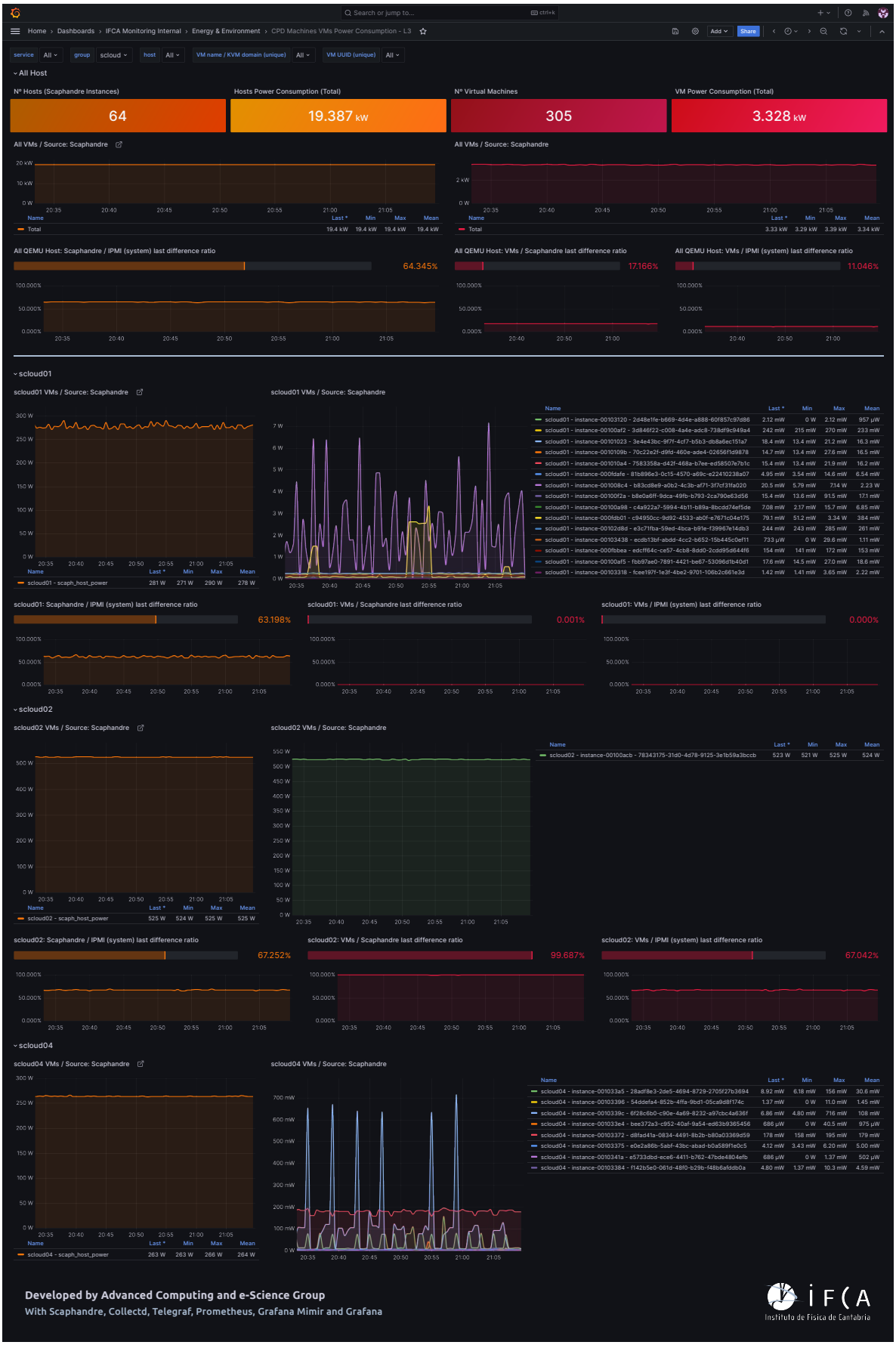


Figure C.9: Dashboards > Energy > Virtual Machines Power Consumption L3

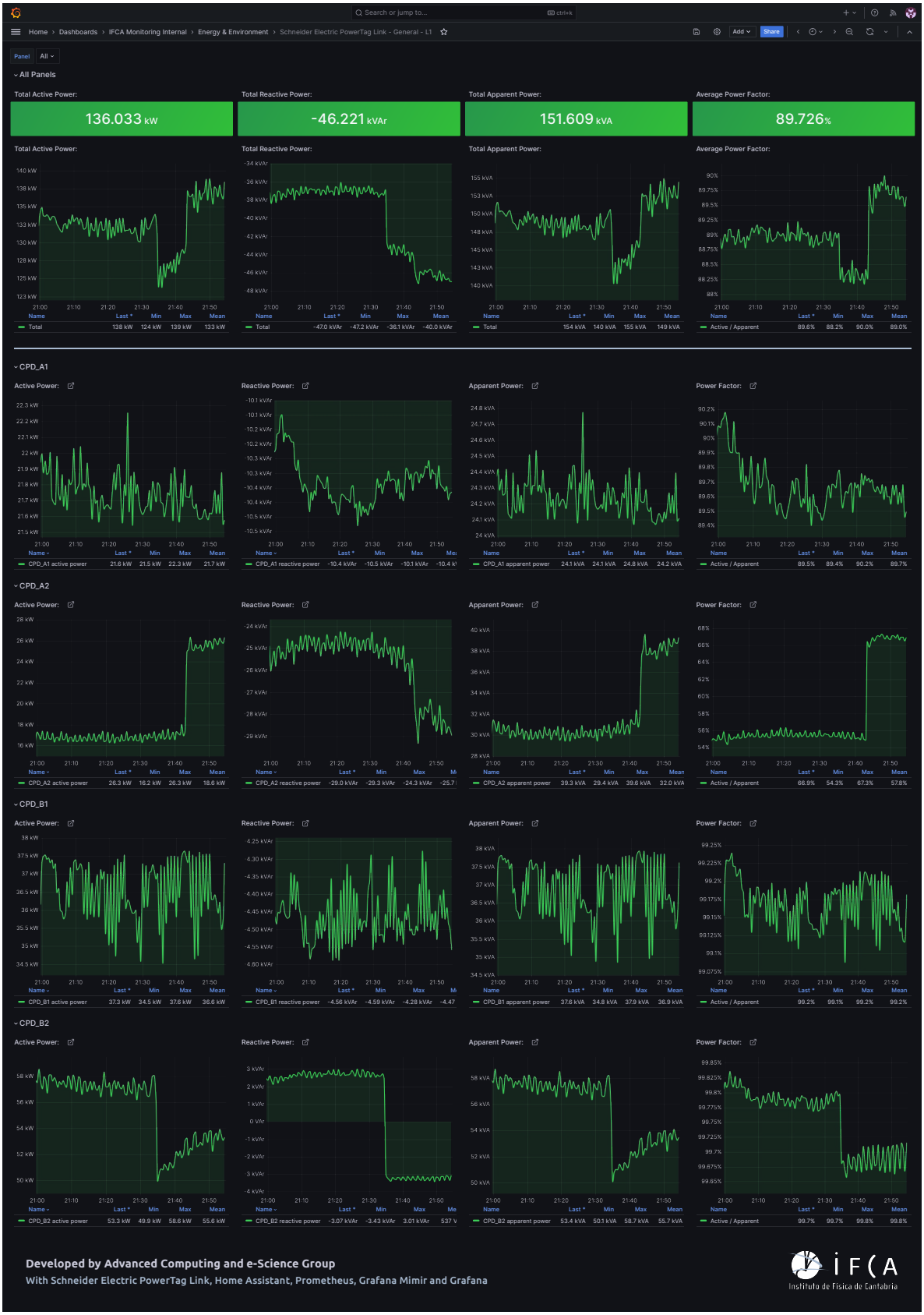


Figure C.10: Dashboards > Energy > Schneider Electric - L1

Figure C.11: Dashboards > Energy > Schneider Electric - L2

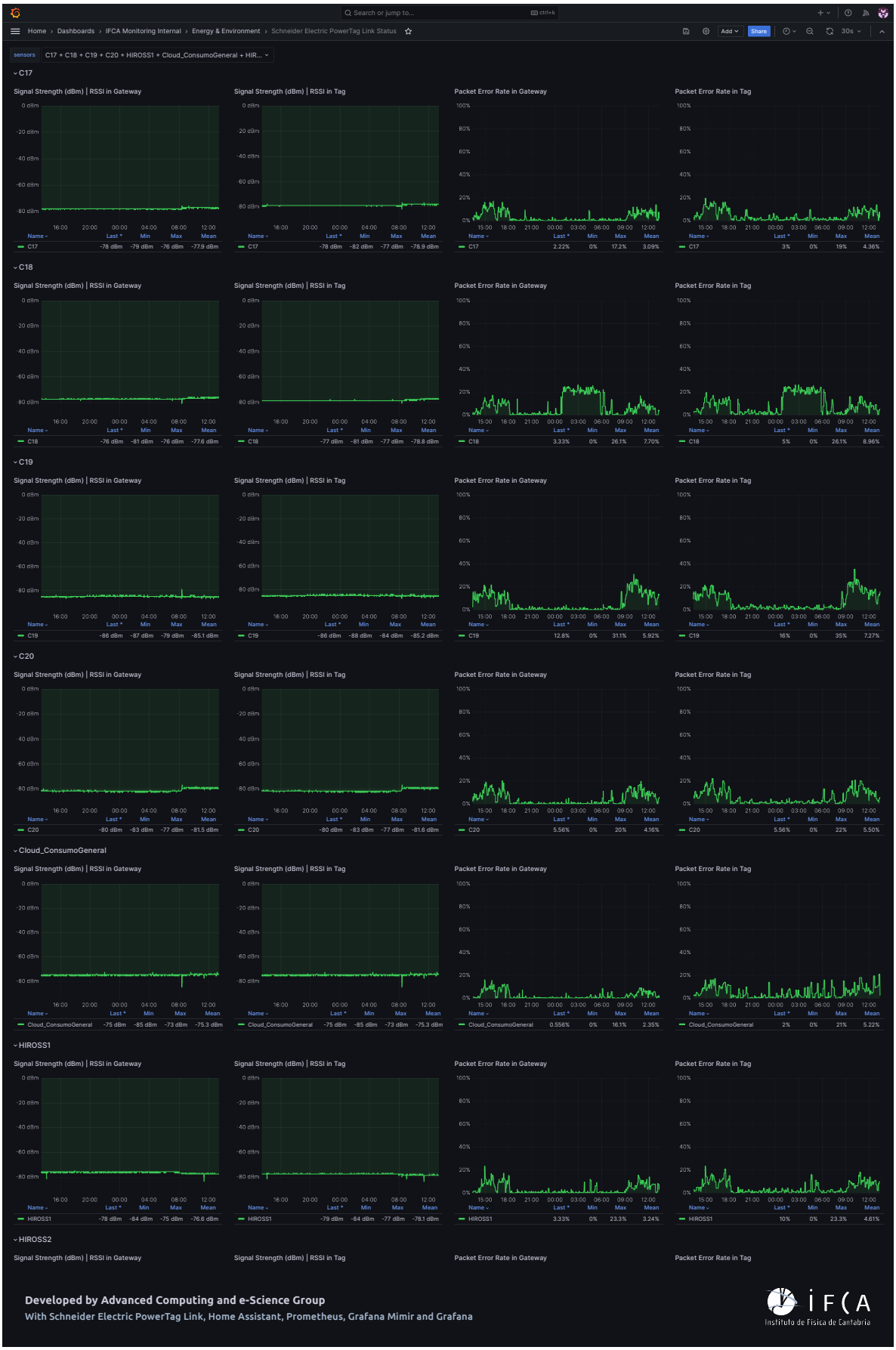


Figure C.12: Dashboards > Energy > Schneider Electric PowerTag Link Status