Universidad de Cantabria

Facultad de Ciencias

Gestión energética de infraestructuras cloud

(Energy management of cloud infrastructures)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Jaime Iglesias Blanco

Director: Enrique Vallejo Gutiérrez

Co-Director: Álvaro López García

Julio - 2023

Esta página se ha dejado en blanco intencionadamente.

Resumen

Debido a la tendencia alcista de la demanda de los servicios de computación en la nube que ofrecen los centros de procesamiento de datos a nivel mundial, se ha generado un aumento considerable de la energía eléctrica consumida por estas instalaciones, para su correcto funcionamiento de forma ininterrumpida.

Por ello, se propone la creación de un sistema de gestión energética autónomo para este modelo de infraestructuras, tratando de mejorar así su eficiencia energética, sin disminuir el rendimiento. Para lograrlo se ha realizado un estudio del entorno, así como sobre las tecnologías empleadas en la gestión del mismo.

Para establecer un marco teórico que sustente las técnicas empleadas para alcanzar dicha mejora, se han analizado varias fuentes de ineficiencias energéticas presentes en estas infraestructuras. Como resultado de la selección de estas fuentes, se ha diseñado e implementado un software enfocado, en primer lugar, en la optimización del uso de los recursos de cómputo disponibles en el *datacenter*. De esta forma, se ofrece la máxima cantidad de instancias virtuales de cómputo, en el mínimo número de recursos físicos disponibles, mediante lo que se conoce como consolidación. En segundo lugar, el sistema controla y mide el aprovisionamiento de recursos físicos disponibles de una manera óptima, en función de la demanda actual, de tal forma, que se mantiene encendida únicamente la parte de la infraestructura en uso. Para lograr esto, el sistema es capaz de encender y apagar los servidores de forma dinámica.

Además, puesto que se trata de un entorno de computación heterogéneo, el sistema se ha caracterizado por tener un diseño modular, el cual permite una alta flexibilidad y adaptación a la infraestructura existente, por medio del uso de diferentes tipos de extensiones. En el caso particular de este proyecto, el sistema se ha diseñado para ser desplegado en el centro de procesamiento de datos del Instituto de Física de Cantabria (IFCA-CSIC-UC), el cual ofrece un entorno de computación en la nube gestionado con *OpenStack*, con una amplia variedad de servidores, que ofrecen distintos servicios basados en la computación en la nube.

El despliegue de este sistema, en este tipo infraestructuras que manejan un gran volumen de datos, permitirá que las organizaciones obtengan beneficios económicos y ambientales, promoviendo la sostenibilidad en los centros de procesamiento de datos.

Palabras clave: Computación en la nube | Eficiencia energética | Sistema de gestión modular | Consolidación de servidores | Apagado/Encendido dinámico de servidores

Esta página se ha dejado en blanco intencionadamente.

Abstract

Due to the upward trend in the demand for cloud computing services offered by datacenters worldwide, there has been a considerable increase in the amount of electricity consumed by these facilities for their uninterrupted operation.

For this reason, the creation of an autonomous energy management system is proposed for this model of infrastructures, in an attempt to improve their energy efficiency without reducing the performance. To achieve this, a study of the environment has been carried out, as well as a study of the technologies used in its management.

In order to establish a theoretical framework to support the techniques used to achieve this improvement, several sources of energy inefficiencies present in these infrastructures have been analysed. As a result of the selection of these sources, software has been designed and implemented focused, in the first place, on the optimization of the use of the computing resources available in the datacenter. In this way, it offers the maximum number of virtual computing instances, in the minimum number of available physical resources, by means of what is known as consolidation. Secondly, the system controls and measures the provisioning of available physical resources in an optimal way, based on the current demand, so that only the part of the infrastructure in use is kept powered on. To achieve this, the system is able to dynamically switch servers on and off.

Furthermore, since it is a heterogeneous computing environment, the system has been characterised by a modular design, which allows a high flexibility and adaptability to the present infrastructure through the use of different types of extensions. In the particular case of this project, the system has been designed to be deployed in the data processing centre of the Institute of Physics of Cantabria (IFCA-CSIC-UC), which offers a cloud computing environment managed with OpenStack, with a wide variety of servers, offering different cloud computing based services.

The deployment of this system, in this type of data-intensive infrastructure, will allow organisations to obtain economic and environmental benefits, promoting sustainability in datacenters.

Keywords: Cloud Computing | Energy Efficiency | Modular Management System | Server consolidation | Dynamic power on/off servers

Esta página se ha dejado en blanco intencionadamente.

Agradecimientos

En primer lugar, me gustaría agradecer a los miembros del grupo de Computación Avanzada y e-Ciencia del Instituto de Física de Cantabria. En especial al codirector de este trabajo, Álvaro López García, y a Miguel Ángel Núñez Vega, por su guía, asesoramiento y dedicación a lo largo de todo el proceso.

En segundo lugar, a Enrique Vallejo Gutiérrez, director de este trabajo, por su orientación y ayuda. Sus aportes ayudaron a mejorar significativamente la calidad y la relevancia de este trabajo.

Agradecer también a todos los profesores del grado, por su formación y todos los conocimientos que he adquirido durante los cuatro años, que sin ellos, no hubiera sido posible la realización de este proyecto.

Por último, agradecer a mi familia, amigos y compañeros por todo el apoyo incondicional y su motivación a lo largo de mi carrera universitaria.

Esta página se ha dejado en blanco intencionadamente.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de códigos

CAPÍTULO 1

Introducción

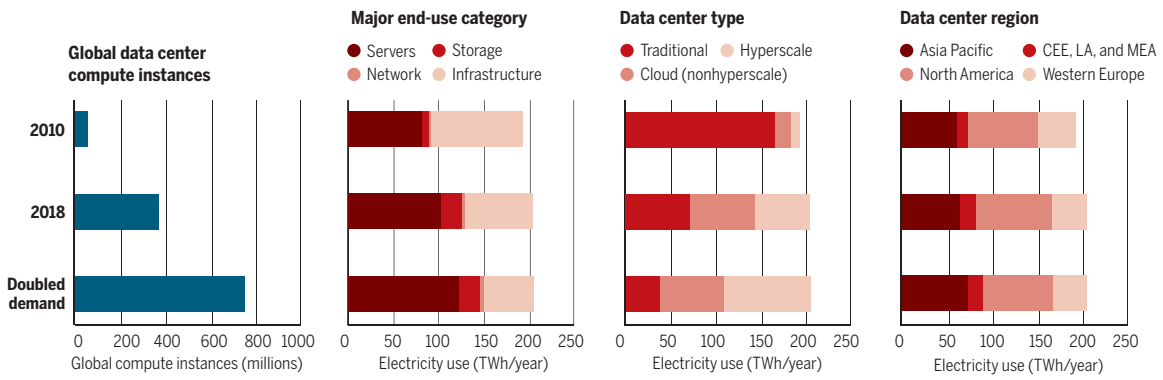
1.1. Motivación

En los últimos años, los centros de procesamiento de datos (CPD) o *datacenters* se han convertido en un pilar fundamental de la sociedad actual debido a la digitalización global, provocando un crecimiento de la demanda de servicios que requieren de una gran infraestructura de computación para ofrecer aplicaciones que manejan una gran cantidad de datos, tales como la inteligencia artificial, vehículos autónomos, plataformas de *streaming* o incluso la banca [1]. Esto supone un gran aumento de la cantidad de energía que requieren estas instalaciones para poder realizar su actividad y ofrecer un servicio de forma ininterrumpida.

Los primeros estudios que se realizaron al respecto estimaban que el consumo de energía por los *datacenters* había crecido desde 153 teravatios-hora (TWh) en 2005 y entre los 203 y 273 TWh en 2010, suponiendo entre un 1.1 y un 1.5 % del consumo de energía global [2].

Posteriormente se estimó que la infraestructura de los *datacenters* se triplicaría, incluso podría llegar a cuadriplicarse. Como consecuencia estos consumirían entre un 3 % y un 13 % de todo el gasto energético generado en el mundo en 2030 [3]. Estos datos son extrapolaciones de las estimaciones de consumo de entonces, sobre la demanda actual, sin embargo, no se tienen en cuenta que en el mismo tiempo que aumenta la capacidad de los centros de procesamiento de datos, también se realizan mejoras en cuanto al diseño y optimización de estos entornos. Por ejemplo, desde 2010, el consumo de electricidad por servidor se ha reducido en una cuarta parte, debido a mejoras de eficiencia presentes en los procesadores, así como reducciones de potencia en periodos de inactividad. Adicionalmente, el número de instancias de cómputo también se ha reducido, según lo previsto, debido a la virtualización de los recursos del *datacenter*.

Teniendo en cuenta dichas optimizaciones, la estimación para 2010 se redujo a 194 TWh. Para 2018, varios analistas estiman que se alcanzó los 205 TWh, lo que en términos de consumo a nivel mundial equivale a un 1 %. Esto representa un crecimiento de un 6 % en ocho años. Sin embargo, la capacidad de cómputo de los centros de procesamiento de datos aumentó en un 550 % durante el mismo periodo de tiempo [2], por lo que demuestra la importancia de las medidas de gestión energéticas en este ámbito.



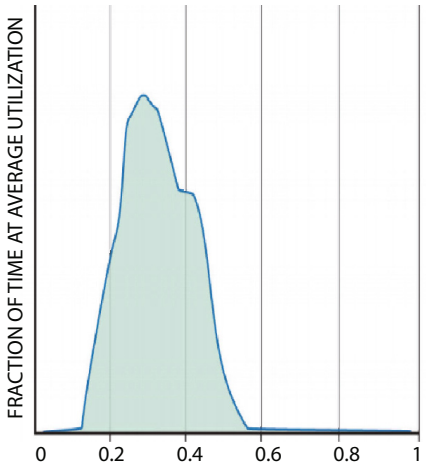
CEE, LA, and MEA, Central and Eastern Europe, Latin America, and Middle East and Africa; TWh, terrawatt-hour.

Figura 1.1: Uso histórico de la energía y uso previsto de la energía con una demanda de infraestructura duplicada [2]

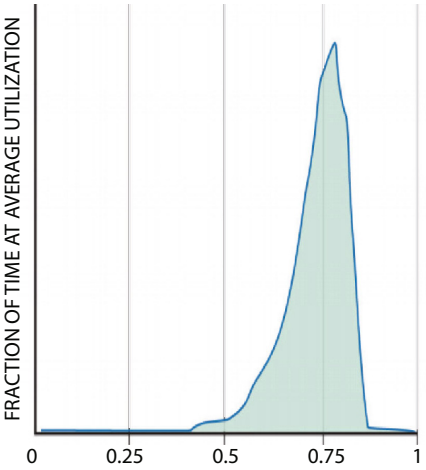
Como se puede ver en la figura 1.1, la duplicación de la demanda refleja la continuidad de las tendencias actuales de eficiencia junto con el crecimiento previsto de las infraestructuras de computación [2]. Duplicar la cantidad de servidores no supone un incremento del mismo orden de magnitud del gasto energético debido a las técnicas empleadas para minimizar el consumo.

1.2. Objetivos

El principal objetivo del trabajo es estudiar y realizar el diseño e implementación de un sistema software capaz de aplicar unas mejoras sobre la infraestructura de computación *cloud* para reducir el consumo energético. Dichas mejoras consisten en optimizar el uso de los CPD, ya que como se muestra en la figura 1.2a, la utilización media es inferior al 50 % de la capacidad total de cómputo del *datacenter*. Estas optimizaciones consisten en por medio de una estrategia de consolidación de servidores, basados en varias políticas, algoritmos y métricas configurables aumentar el número de servidores en estado de inactividad, para posteriormente reducir el consumo energético lo máximo posible de los servidores no utilizados, realizando apagados y encendidos dinámicos para adaptarse a la demanda de los clientes del servicio que ofrece la infraestructura *cloud* del *datacenter*.



(a) Comportamiento de un *datacenter* típico



(b) Comportamiento del *datacenter* más utilizado

Figura 1.2: Gráfica de distribución de la actividad media de dos *datacenters* de Google con 20.000 servidores [4]

Aplicando dichas optimizaciones, sin aumentar la carga de trabajo, se podría pasar del escenario de la figura 1.2a, con una utilización media del *datacenter* del 30 % al escenario de la figura 1.2b, con una utilización media del 75 %, por medio de reducir el número de máquinas disponibles en el mismo, es decir, la capacidad total del *datacenter*.

El sistema a desarrollar deberá estar diseñado en base a una arquitectura modular para que sea sencillo la incorporación de nuevos componentes que implementen nuevas funcionalidades o políticas de gestión de los recursos. De esta forma, por ejemplo, se podrían incluir nuevos algoritmos de optimización para ir ajustando la eficiencia todo lo que sea necesario. Además, la flexibilidad del sistema es de vital importancia, ya que el entorno de computación que controla es variante en el tiempo, debido a que se incorporan nuevos sistemas para la gestión del mismo, con los cuales el sistema de gestión energética podría interactuar en un futuro o mismamente nuevos servidores que requieran de otro mecanismo para poder controlarlos.

1.3. Planificación y metodología de desarrollo: Scrum

Scrum es una metodología de trabajo ágil o *framework* utilizada para la gestión de proyectos, especialmente en el desarrollo de software. Fue propuesta por primera vez en 1986 por

Hirotaka Takeuchi e Ikujiro Nonaka en un artículo de investigación titulado *"The New Product Development Game"*. Posteriormente, Jeff Sutherland y Ken Schwaber formalizaron y popularizaron, convirtiéndolo en lo que es hoy [5]. Está metodología ofrece una solución, asumiendo de forma intrínseca, que es imposible definir todo el proyecto desde el inicio, priorizando la capacidad de adaptación del equipo a los cambios inesperados en los requisitos con el objetivo de mejorar la velocidad y aumentar el flujo de trabajo durante la fase de desarrollo.

Se ha decidido usar esta metodología precisamente para solventar este problema. Al inicio del proyecto no estaba muy claro el alcance global del proyecto, debido a que, como se ha explicado en la sección 1.2, el enfoque del sistema está orientado a permitir la modularidad y extensión, por lo tanto, era inviable definir completamente el proyecto durante los primeros días del trabajo.

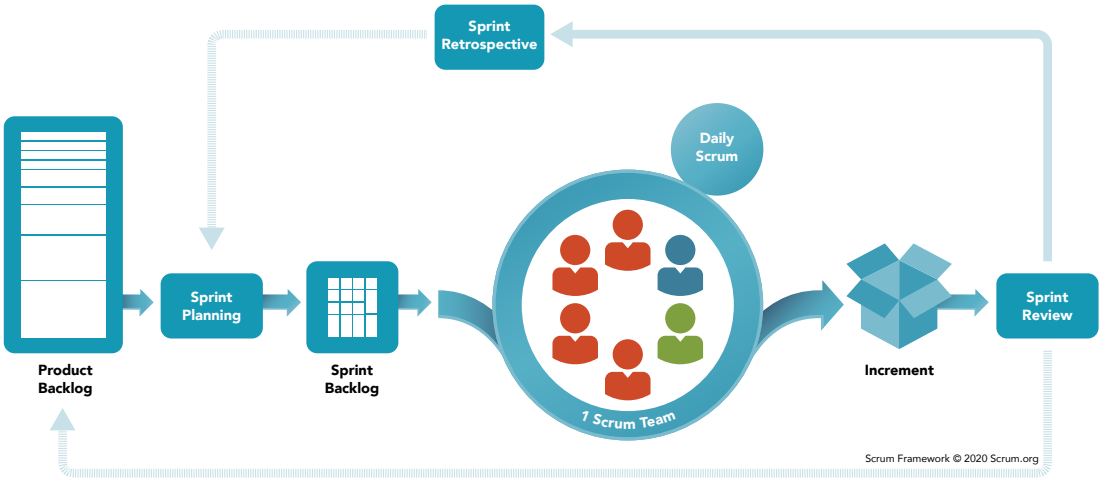


Figura 1.3: Diagrama de implementación del *framework* Scrum

Esta flexibilidad se consigue debido a que el *framework* se basa en un enfoque iterativo, incremental y colaborativo dividido en pequeños grupos de trabajo para la entrega de proyectos de manera constante en pequeñas fases, como se observa en la figura 1.3. De esta manera, este *framework* ha facilitado en gran medida la división de este proyecto en varias partes, sin la necesidad de definir un alcance global para el proyecto.

Scrum se compone de tres aspectos esenciales como son los artefactos o elementos de trabajo, los eventos o reuniones, y los roles del equipo de trabajo [6].

1.3.1. Artefactos o elementos de trabajo

En primer lugar, es necesario desgranar el producto software a realizar en pequeñas partes, dando lugar al primer artefacto Scrum, conocido como ***Product Backlog*** del proyecto. Conforma una lista dinámica priorizada de las características, funcionalidades, historias de usuario y requisitos del producto, las cuales se deben completar para que el proyecto tenga éxito. El trabajo definido en el *Product Backlog* se tiene que ir desarrollando en forma de iteraciones o *sprints*.

Un ***Sprint*** es un período de tiempo en el que el equipo desarrolla un incremento de software "potencialmente entregable", es decir, funcional y que permita ver al cliente el avance en el proyecto [5]. La duración de estos ha sido de entre una y dos semanas por *sprint*, dependiendo de la complejidad de la nueva funcionalidad a implementar.

Al inicio de cada iteración, se seleccionará los ítems del *Product Backlog* que han de completarse en dicha iteración, formado así el siguiente artefacto, denominado ***Sprint Backlog***.

El último artefacto es el incremento (***Increment***), y se genera al final de la iteración. Conforma la suma de todos los elementos desarrollados del *Product Backlog* completados durante la iteración actual y el valor de los incrementos de todos los *sprints* anteriores.

1.3.2. Roles del equipo Scrum

Para utilizar Scrum de forma correcta se requieren tres perfiles o roles en el equipo de trabajo:

- **Propietario del producto *(Product Owner)*:** Es la representación del cliente dentro del equipo de trabajo. Es responsable de gestionar el *backlog* del producto. Este rol lo ha llevado a cabo parte del grupo de Computación Avanzada y e-Ciencia del IFCA.
- **Scrum Master:** Es el responsable de que la metodología de trabajo es entendida y se realiza ajustándose a la teoría, prácticas y reglas de Scrum. Este rol lo ha llevado a cabo el autor de este Trabajo Fin de Grado.
- **Equipo de desarrollo:** Son los responsables de crear el producto y entregar los resultados del proyecto. Generan los incrementos de cada *sprint*. En este caso el equipo de desarrolladores se compone únicamente del autor de este Trabajo Fin de Grado.

1.3.3. Eventos o reuniones

La forma de colaborar y gestionar los artefactos de manera organizada es por medio de reuniones periódicas. Existen cuatro tipos de reuniones que implican a diferentes partes del equipo de trabajo, en función del nivel de avance la iteración actual:

- **Planificación del Sprint (Sprint Planning):** Se definen los objetivos para el siguiente *sprint*, seleccionando tareas del *backlog* de producto y los pasos necesarios para completarlas. Se ha realizado conjuntamente entre el autor y el co-director de este Trabajo Fin de Grado.
- **Scrum diario (*Daily Scrum*):** Scrum define que durante el desarrollo se han de realizar una serie de reuniones diarias, de breve duración, en las que participa todo el equipo de desarrollo y sirven como mecanismo de sincronización durante una iteración. En este caso, como el equipo de desarrollo lo conforma únicamente el autor del TFG, es innecesario realizar estas reuniones. Sin embargo, se han utilizado tableros *KanBan* para representar gráficamente el progreso dentro de cada tarea del *sprint* y mejorar la planificación.
- **Revisión del Sprint (*Sprint Review*):** Se realiza al completar la iteración y el equipo analiza las tareas completadas, cómo se han llevado a cabo, así como las no completadas debido a la falta de tiempo o por problemas que se han generado durante el desarrollo. También sirve como reunión de demostración a los clientes del producto realizado hasta la fecha.

  Como se ha indicado anteriormente, el *sprint* dura entre una o dos semanas, por lo que estas reuniones tienen lugar con la misma frecuencia. Además, han servido en este caso para para mantener a la dirección de este Trabajo Fin de Grado informada del estado y grado de avance del proyecto.
- **Retrospectiva del Sprint (*Sprint Retrospective*):** Al igual que la reunión anterior se realiza al completar un *sprint* para evaluar el resultado obtenido, pero en este caso se analiza la forma de trabajar, la calidad con la que se ha desarrollado la metodología y se buscan soluciones a los problemas surgidos para aplicarlas en las siguientes iteraciones.

CAPÍTULO 2

Conceptos y fundamentos previos

En este capítulo se presentan los conceptos fundamentales necesarios para permitir al lector comprender de manera clara y concisa los términos utilizados a lo largo de este Trabajo Fin de Grado. Para ello se explicarán los conceptos relacionados con el tema de estudio y se establecerán las bases teóricas necesarias para comprender el enfoque y los objetivos del trabajo. Por último, se citan las tecnologías empleadas en el desarrollo del mismo.

2.1. Computación en la nube (*Cloud Computing*)

El término *Cloud Computing* (que se traduce como "Computación en la nube", donde "nube" es una metáfora de Internet) se refiere a un paradigma informático que surge con el objetivo de proporcionar servicios informáticos a los usuarios, a través de la red, por lo tanto accesibles desde cualquier dispositivo con acceso a Internet [7]. Todo esto es posible gracias a la virtualización de los recursos físicos. Los proveedores pueden prestar un gran número de servicios de forma puntual y eficiente, permitiendo a los usuarios acceder a ellos de forma cómoda y segura.

Según la definición del *National Institute of Standards and Technology* (NIST), la computación en *cloud* se puede interpretar como "un modelo de computación distribuida que permite el acceso ubicuo, conveniente, bajo demanda en red a un conjunto de recursos informáticos configurables (redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente provistos y liberados con un mínimo esfuerzo de gestión o interacción con el proveedor" [8].

2.1.1. Características principales

El *cloud computing* es una tecnología que ha revolucionado la forma en que las empresas y organizaciones gestionan sus recursos informáticos. A continuación, se detallan algunas de sus principales características [8, 9]:

- **Servicio automático bajo demanda:** Sin necesidad de comunicarse directamente con cada proveedor de servicios, un cliente puede proporcionar unilateralmente recursos informáticos según sus necesidades de forma automática.
- **Amplio acceso a través de la red:** Los recursos están disponibles a través de la red y se puede acceder a ellos desde cualquier tipo de dispositivo conectado a Internet. Esto supone mínimos requerimientos por parte del cliente.
- **Agrupación de recursos:** Los recursos informáticos del proveedor se comparten entre múltiples usuarios, asignados y reasignados dinámicamente en función de la demanda de los consumidores. El cliente no suele tener control ni conocimiento de la ubicación, pero puede ser capaz de especificar la ubicación a un nivel abstracción como, por ejemplo especificando el país o el estado.
- **Servicio medido:** Los recursos utilizados se contabilizan de forma independiente y precisa, para establecer un modelo de negocio basado en el pago por uso (*pay-per-use*). El uso de los recursos puede ser monitorizado y controlado, proporcionando transparencia entre el proveedor del servicio y el usuario sobre el gasto de estos.

2.1.2. Tipos de despliegue

Existen cuatro tipos de modelos de despliegue de un entorno de computación *cloud* dependiendo del grupo de usuarios que tienen acceso a la infraestructura [8, 9]:

- **Privada:** El uso de la infraestructura de cómputo está restringido a la institución dueña de los recursos físicos.
- **Pública:** Ofrece un método de acceso para usuarios de todo el mundo a los servicios *cloud* y que suele emplear un modelo de negocio basado en políticas de pago por uso.
- **Colectiva:** Se identifica de esta manera cuando la infraestructura está compartida entre varias organizaciones o instituciones.
- **Híbrida:** Surge debido a la combinación de los modelos de desligue anteriores.

2.1.3. Modelos de servicio

Cada modelo de servicio de computación en la nube cubre diferentes necesidades de los usuarios y las empresas, proporcionándoles un nivel diferente de control, seguridad y escalabilidad [7, 8], como se muestra en la figura 2.1. Los modelos de servicio que presenta este paradigma de computación distribuida son los siguientes:

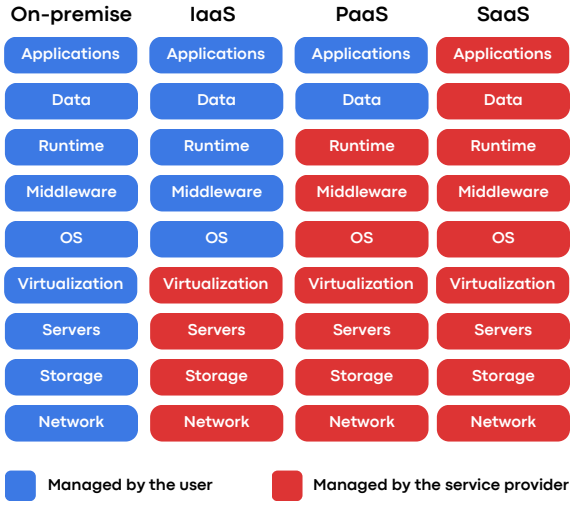


Figura 2.1: Reparto de gestión entre usuario final y proveedor del servicio *cloud*

- ***Software as a Service* (SaaS)**: Contiene las aplicaciones que se ejecutan en la nube y las cuales están ofrecidas bajo demanda como un servicio a los usuarios. Son aplicaciones escalables y gestionadas por el proveedor de *cloud*, ya que el usuario accede a ellas por medio de la red.
- ***Platform as a Service* (PaaS)**: Se encarga de proporcionar la infraestructura y un entorno de ejecución a las aplicaciones de la capa superior. Permite el desarrollo de aplicaciones *cloud* sin tener que preocuparse de la gestión y administración de los sistemas informáticos de la infraestructura. Para garantizar la escalabilidad, a menudo, se presenta virtualizada.
- ***Infrastructure as a Service* (IaaS)**: Se ubican los recursos físicos del sistema (servidores, redes y almacenamiento), los cuales se ofrecen como un servicio a los usuarios. También se suelen emplear técnicas de virtualización para poder albergar un número elevado de máquinas virtuales y reducir costes, gracias a la utilización más eficiente de los recursos.

2.2. Centro de procesamiento de datos

Un centro de procesamiento de datos (CPD), también conocido como *datacenter* es una instalación compuesta por computadores conectados por medio de una red de altas prestaciones, con la disposición de grandes sistemas de almacenamiento e infraestructura informática que las organizaciones utilizan para reunir, procesar, almacenar y difundir grandes cantidades de datos [4]. Una organización suele depender en gran medida de las aplicaciones, los servicios y

los datos contenidos en un centro de datos [10], lo que lo convierte en un activo fundamental para las operaciones cotidianas de la misma.

Además, en el ámbito científico, un *datacenter* presta a la organización en cuestión, una infraestructura robusta, con una gran potencia de cómputo y capacidad de almacenamiento, gestionada por un grupo de profesionales, para que los investigadores puedan utilizarla para la elaboración de sus trabajos, incrementando la calidad de los mismos.

2.2.1. Servicios de computación del IFCA

El Instituto de Física de Cantabria (IFCA) dispone de un centro de procesamiento de datos (CPD) *on-premise*, gestionado por el grupo de Computación Avanzada y e-Ciencia. Está compuesto por cuatro tipos de servicios, basados en una arquitectura *Intel® x86*, interconectados en una red con una velocidad de transmisión de 10 Gb/s, que ofrece múltiples opciones y arquitecturas de procesamiento [11]. Los servicios que ofrece son los siguientes:

- **AI/Machine Learning:** Dispone de 31 nodos de cómputo, con un total de 2.572 núcleos de CPU, 7792 GB de memoria principal y 40 GPUs *NVIDIA® GV100GL 32Gb* (con *Infiniband EDR* 100Gb/s), 10 *NVIDIA® 1080Ti* y 80 gráficas *NVIDIA® T4*.
- **Cloud:** Cuenta con 179 nodos de cómputo y ofrece un total de 2.368 núcleos de CPU con 33.219 GB de RAM. Este conjunto de nodos opera un modelo de computación en la nube, como el descrito en el apartado 2.1, para el cual se diseñará el sistema de gestión energética. Ofrece un modelo de servicio de *Infrastructure as a Service* (IaaS). Este entorno se describirá con mayor detalle en la sección 2.3.
- **Supercomputación (HPC) - Altamira:** Consiste en un sistema formado por 158 nodos, con un total de 5.056 núcleos de CPU, 10.112 GB de memoria principal e interconectados a través de una única red *Infiniband FDR10* a 40 Gb/s. Forma parte de la Red Española de Supercomputación (RES) y el Servicio Santander de Supercomputación (SSC).
- **Grid (HTC):** Dispone de 62 nodos de cómputo, reuniendo un total de 2.664 núcleos de CPU y 7.200 GB de memoria RAM.

Los cuatro tipos de servicios comparten un sistema de almacenamiento persistente de 1.6 PB a través de *IBM's General Parallel File System* (GPFS) y 400 TB de espacio de almacenamiento a través de *Ceph File System* (CEPH). La infraestructura se completa con un sistema de almacenamiento de cintas *IBM® TS3500* con una capacidad máxima de almacenamiento de unos 6 PB.

2.3. Entorno de computación *cloud*

En esta sección se describe la parte del sistema informático con el que deberá interactuar el sistema de gestión energética, tanto la infraestructura hardware del centro de procesamiento de datos (CPD) del IFCA, como el software empleado en la gestión de este, ya que será necesario tenerlo en cuenta a la hora de incluir un nuevo componente de gestión en el entorno.

2.3.1. Infraestructura Hardware

La infraestructura hardware la conforman los servidores, dedicados parte de ellos a ofrecer servicios generales y otros como nodos de cómputo, equipos de red, sistemas de almacenamiento en disco y robots para las copias de seguridad en cintas. Para este trabajo solo se tendrán en cuenta los servidores o máquinas físicas del *datacenter* del Instituto de Física de Cantabria.

2.3.1.1. Máquinas físicas

Como se ha comentado en la sección 2.2.1, se dispone de cuatro tipos de servicios de computación en el CPD del IFCA, aunque los servicios de computación *AI/Machine Learning* y de *Grid (HTC)* también se gestionan a través de la misma infraestructura software que gestiona el servicio de computación *cloud*. Por ello, el sistema de gestión energética también deberá tener en cuenta las máquinas físicas disponibles para estos servicios, aunque con alguna particularidad. Por ejemplo, las migraciones de máquinas virtuales (VMs) del servicio de *AI/Machine Learning* se realizan entre máquinas del mismo tipo, al necesitar de hardware especifico como puede ser una GPU u otro tipo de acelerador hardware.

El listado completo de máquinas físicas del centro de procesamiento de datos del Instituto de Física de Cantabria [12] gestionadas a través del sistema de computación cloud se encuentra en el anexo A.

2.3.1.2. *Intelligent Platform Management Interface* (IPMI)

Intelligent Platform Management Interface (IPMI) es un conjunto de especificaciones que proporcionan una interfaz estandarizada para un subsistema a nivel hardware que provee los llamados servicios de "gestión de plataforma" de forma remota [13]. En este contexto engloba la monitorización del equipamiento hardware de la infraestructura, como los servidores, sistemas de refrigeración, fuentes de alimentación, como su control a través del cual podemos encender y apagar los servidores. También permite gestionar el inventario de hardware presente en la infraestructura, así como hacer un *logging* y establecer alertas de eventos a nivel hardware.

Una característica clave es la independencia de la plataforma respecto de la CPU, el *firmware* (BIOS o UEFI) y el sistema operativo. Las funciones de gestión de la plataforma también permanecen disponibles cuando el servidor esté apagado, siempre que al menos una fuente de alimentación del servidor tenga corriente.

El componente principal de la arquitectura necesaria para utilizar IPMI es un microcontrolador integrado en la placa base de cada uno de los servidores, conocido como *Baseboard Management Controller* (BMC) [15]. Este gestiona la interfaz entre el software de gestión del sistema, por ejemplo, las *IPMItools* en *Linux*, y el hardware de la plataforma, como se puede ver en el esquema de la figura 2.2. Tiene su propio *firmware* y memoria RAM.

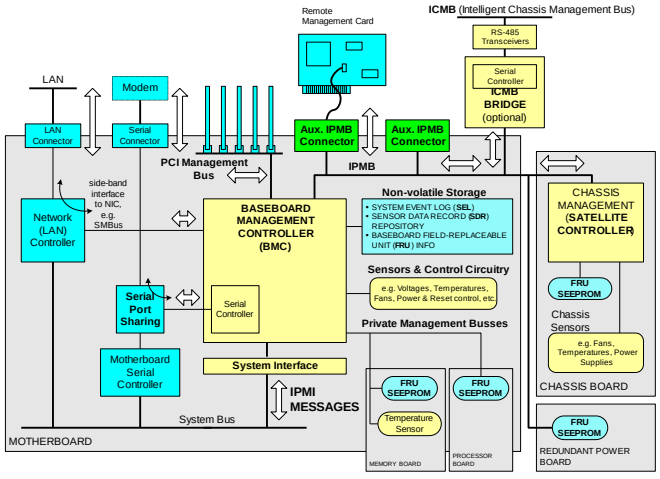


Figura 2.2: Interfaces del BMC a la placa base del servidor [14]

La especificación está liderada por *Intel* [14] y se publicó por primera vez el 16 de septiembre de 1998. Está respaldada por más de 200 proveedores de infraestructura IT, como *Cisco*, *Dell*, *Hewlett Packard Enterprise* y la propia *Intel* [15]. Aunque IPMI es una especificación estándar, cada fabricante tiene su propia implementación y para todo parámetro definido en la especificación no siempre medirá los mismos componentes del hardware, ya que esto es dependiente del fabricante del servidor.

2.3.2. Infraestructura Software

La infraestructura software dedicada al control de los servicios *cloud* del *datacenter*, con los cuales el sistema de gestión energética interactuará con ellos, son: *OpenStack* como plataforma para la gestión de computación en *cloud* y *Grafana* para la visualización y recolección de datos con el objetivo de monitorizar el entorno.

2.3.2.1. *OpenStack*

OpenStack [16] es una plataforma *open-source* para la gestión de infraestructuras de computación en la nube que proporcionan un modelo de servicio de *Infrastructure as a Service* (IaaS). Está escrita en lenguaje *Python* y distribuida bajo una licencia *Apache, versión 2.0*[1].

Permite la gestión de grandes conjuntos de nodos de cómputo, sistemas de almacenamiento y de red, repartidos en un CPD, a través de unas APIs que emplean mecanismos de autenticación estándar. Además, se ofrece un panel de control que permite a los administradores gestionar el sistema y a los usuarios aprovisionarse de los recursos a través de una interfaz web.

La plataforma está construida en una arquitectura modular, mediante componentes, mostrados en la figura 2.3, que proporcionan distintos servicios propios de un gestor de computación cloud para un modelo de servicio IaaS. Además, incluye componentes adicionales para la orquestación, gestión de fallos y gestión de los propios servicios que garantizan la alta disponibilidad de las aplicaciones de los usuarios.

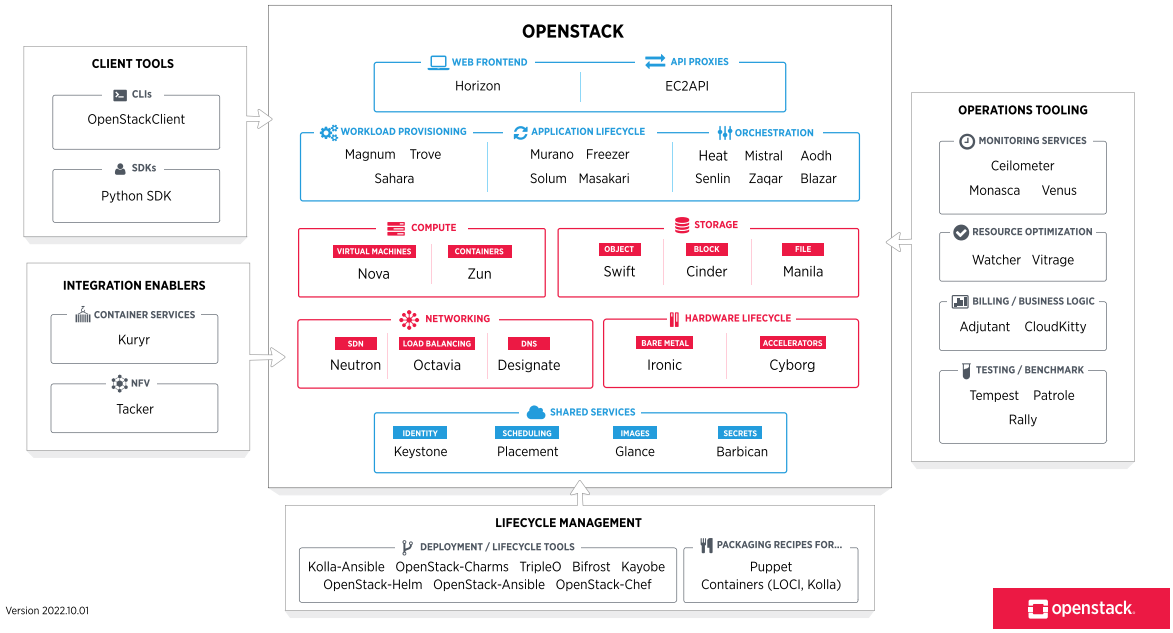


Figura 2.3: Componentes de *OpenStack* [16]

A fecha de realización de este trabajo, en el CPD del IFCA se utilizan los siguientes componentes:

- **Nova:** Proporciona una forma de aprovisionar instancias de computación (servidores virtuales). Soporta la creación de máquinas virtuales (VMs). Se ejecuta como un conjunto de demonios sobre servidores *Linux* existentes.

[1]Licencia *Apache, versión 2.0*: https://www.apache.org/licenses/LICENSE-2.0

- **Neutron:** Proporciona "conectividad de red como servicio" entre dispositivos gestionados por otros servicios *OpenStack*. Implementa la API de red.
- **Keystone:** Proporciona autenticación de usuarios, descubrimiento de servicios y autorización multiusuario distribuida mediante la API de identificación.
- **Horizon:** Proporciona una interfaz de usuario basada en un cliente web para gestionar los distintos servicios del *cloud*.
- **Cinder:** Es un servicio de almacenamiento de bloques. Virtualiza la gestión de dispositivos de almacenamiento en bloque y proporciona a los usuarios finales una API de autoservicio para solicitar y consumir esos recursos, sin la necesidad de saber dónde está desplegado realmente su almacenamiento o en qué tipo de dispositivo.
- **Glance:** Este servicio incluye el descubrimiento, el registro y la recuperación de imágenes de máquinas virtuales. Cuenta con una API RESTful que permite consultar los metadatos de las imágenes de máquinas virtuales y recuperar la imagen real.

OpenStack se desarrolla y actualiza en ciclos de 6 meses con una nueva versión, pero no es factible actualizar el CPD cada vez que hay una actualización de este software. En la actualidad, en el Instituto de Física de Cantabria, se emplea la versión *Ussuri* [2].

2.3.2.2. *Grafana*

Grafana es una plataforma de visualización y monitoreo de datos de código abierto, también distribuida bajo la licencia de Apache 2.0. Está escrita en lenguaje *Go* y *Typescript*. Permite la visualización y el formato de datos métricos de forma interactiva y en tiempo real, a través de la creación y configuración de paneles de mando con gráficos a partir de múltiples fuentes, como bases de datos, sistemas de monitoreo y servicios en la nube [17].

Entre los servicios que se conectan como fuentes de datos se incluyen: bases de datos relacionales (como *MySQL*, *PostgreSQL*), bases de datos NoSQL (como *MongoDB*, *InfluxDB*), sistemas de monitoreo (como *Prometheus*, *Graphite*), servicios en la nube (como *AWS CloudWatch*, *Google Cloud Monitoring*) y otros [17].

La plataforma es ampliamente utilizada en entornos de operaciones y desarrollo de software. El equipo de sistemas del grupo de Computación Avanzada y e-Ciencia del IFCA lo utiliza para la monitorización y configuración de alertas de eventos en el entorno, permitiendo realizar estudios de la utilización de sus infraestructuras. También lo utilizan para analizar *logs*, generados por otros sistemas software del entorno, de una manera más eficaz, a través de un sistema de gestión de registros, llamado *Loki* desarrollado por *Grafana Labs*. Esta herramienta se caracteriza por utilizar una arquitectura de almacenamiento distribuido, conocido como "*log* de índice invertido", en lugar de almacenar los registros de forma tradicional, como archivos de texto o bases de datos. Así los registros se almacenen de forma eficiente y se puedan buscar rápidamente.

2.4. Eficiencia energética en *datacenters*

La eficiencia energética ha sido un aspecto clave durante mucho tiempo en las áreas de computación móvil y embebidos, enfocada en extender la duración de la batería de los mismos y para reducir, por restricciones térmicas, la densidad de potencia máxima en chips en un área extremadamente pequeña. Sin embargo, la gestión energética es también un aspecto

[2] *OpenStack Releases - Ussuri:* https://releases.openstack.org/ussuri/index.html

de gran interés en los centros de procesado de datos, con el objetivo de reducir los costes relacionados con el consumo de energía, incluyendo gastos económicos, operacionales, así como medioambientales [4].

2.4.1. Principales fuentes de ineficiencia energética

Un primer paso a la hora de empezar a diseñar un sistema para la gestión energética de un entorno computacional es buscar los principales puntos en los que la infraestructura es más ineficiente. Este Trabajo Fin de Grado se ha centrado en los problemas de eficiencia energética generados por la parte IT de la infraestructura, en concreto el uso de la CPU, que como observamos en la figura 2.4, es el componente con el consumo energético más elevado. Obviamente, para hacer la instalación lo más respetuosa con el medioambiente habría que considerar las ineficiencias generadas también por el equipamiento no IT, como son los equipos de suministro eléctrico (PDUs y UPS) y los equipos de refrigeración o *cooling*. A continuación, se explican dos casos estudiados con una solución posible. Existen más fuentes, como el uso de un hardware obsoleto, pero que, para mitigar dicha fuente de perdidas energéticas, se requiere un desembolso de capital para la adquisición de nuevos componentes más eficientes energéticamente, lo cual no es el propósito de este trabajo.

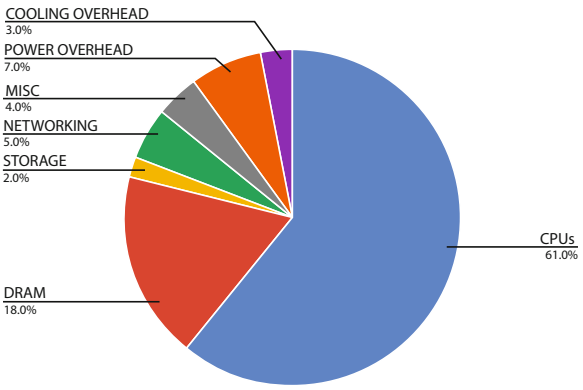


Figura 2.4: Distribución aproximada del uso de potencia máxima en un servidor de 2017 con 2 cores x86 y 12 DIMMs, con una utilización media del 80 % [4]

2.4.1.1. Servidores con un perfil de consumo de energía no proporcional

El principal problema existente ahora mismo con el consumo de energía en el ámbito de los servidores es que no tienen un perfil de consumo de energía proporcional a su uso. Idealmente, los sistemas deberían consumir energía de forma proporcional a su uso o carga de trabajo. Actualmente, ha mejorado considerablemente y los servidores tienen un perfil bastante proporcional. Sin embargo, este efecto se amplifica teniendo en cuenta que los servidores son utilizados entre un 10 % y 50 % de su carga máxima y en muchos casos con periodos de inactividad [1], lo que genera un consumo estático por parte de infraestructura no utilizada.

2.4.1.2. Sobreaprovisionamiento de servidores e infraestructura eléctrica

Hoy en día los *datacenters* están sobredimensionados, en el sentido de que están diseñados para manejar una carga pico, que en pocas ocasiones se da, como se explicó en la sección 1.1. Esto se plasma en servidores y hardware no utilizado, lo cual genera un consumo energético excesivo para el trabajo útil que se está realizando. Sin embargo, el sobreaprovisionamiento no es un problema si la energía que consumen los servidores es directamente proporcional al uso del mismo.

2.4.2. Métricas de eficiencia energética

La métrica *Power Usage Effectiveness* (PUE) refleja la calidad, en términos de eficiencia energética, del complejo donde está alojado la propia infraestructura, relacionando el consumo

total de todo el *datacenter* con el consumo generado del equipamiento IT, formado por los nodos de cómputo, equipos de red y almacenamiento. Se puede expresar y calcular a través de la siguiente formula:

$$PUE = Total\ Facilty\ Power/IT\ Equipment\ Power \tag{2.1}$$

Esta métrica cada vez está más en desuso, entre otras cosas porque no es comparable directamente, ya que en todas las medidas que se hagan sobre distintos *datacenters*, no se van a contabilizar las mismas pérdidas energéticas [4], ni bajo las mismas condiciones. Se suele usar en documentos enfocados al *marketing*, donde se enseña siempre el mejor caso, cuyos valores no son reales ya que son obtenidos de forma esporádica en condiciones ideales.

Además, al ser una métrica que representa cuánta potencia es perdida en equipamiento no IT (distribución y conversión eléctrica, así como refrigeración), no se centra en medir la mejora de la eficiencia con la que está enfocada este trabajo. El objetivo es aumentar la eficiencia energética de la infraestructura IT centrado en los nodos de cómputo o servidores. Si aplicamos la estrategia de mejora de eficiencia, que se explicará posteriormente en la sección 2.5, se reducirá en igual medida tanto el consumo total del CPD, como el consumo del equipamiento IT, por lo que el PUE permanece invariable.

Una forma más precisa de medir la eficiencia energética de un CPD es a través de la métrica *Data Center Energy Efficiency* $(DCEE)$, que se centra en estimar la cantidad de energía usada por equipamiento IT del *datacenter* que está realizando un trabajo útil. Se puede calcular a partir de la siguiente formula:

$$DCEE = ITU \times ITE/PUE \tag{2.2}$$

donde la utilización del equipamiento IT (ITU) denota la relación media del equipamiento usado sobre la totalidad de capacidad del *datacenter* y la eficiencia del equipamiento IT (ITE) representa la cantidad de trabajo útil realizado por cada julio de energía consumido, que exclusivamente depende de cada tipo de servidor y su nivel de utilización.

2.5. Sistemas de gestión energética

En el siguiente apartado se explican y justifican la elección de dos estrategias de optimización, que solventan las ineficiencias explicadas en el apartado 2.4.1, y las cuales se implementarán en el software de gestión energética.

En términos generales, la mejora de la eficiencia energética de la infraestructura se basa en la optimización de la utilización de las máquinas físicas. Conseguir una mayor eficiencia energética del sistema es una consecuencia de aprovechar correctamente los recursos, optimizar la utilización del sistema y únicamente equipar al CPD de los recursos hardware, en este caso servidores, necesarios para su correcta operabilidad con la carga de trabajo o uso actual.

2.5.1. Mecanismos para optimizar el consumo energético

Como primera aproximación para minimizar las fuentes de ineficiencias, descritas en el apartado 2.4.1, se ha decidido aplicar una estrategia de optimización basada en el apagado y encendido dinámico de máquinas físicas, para así suplir los problemas de sobreabastecimiento y en parte el problema del consumo de energía no proporcional a su uso en los servidores. Respecto a este último, la solución ideal se debe tratar en el diseño de los propios componentes informáticos, pero sí podemos eliminar el consumo de potencia estática de los servidores cuando no están siendo utilizados.

Para lograr esta mejora, el software de gestión energética que se va a desarrollar deberá controlar el nivel de utilización del entorno de computación *cloud*, y aplicar las acciones necesarias para que el consumo sea mínimo. En la práctica, el sistema de gestión energética deberá apagar los servidores cuya utilización sea nula, y así disminuir el consumo estático de los mismos.

Empíricamente, utilizando la fórmula 2.2 referente a la métrica $DCEE$, aplicando está medida estaríamos aumentando el ITU, ya que al apagar máquinas reducimos el número de servidores disponibles en el *datacenter*, respecto a su capacidad total, por lo tanto, la utilización del equipamiento IT se incrementa. Como la métrica es directamente proporcional a este valor y el PUE es constante para el tipo de optimizaciones que estamos realizando, se produce una mejora en la eficiencia energética. De igual modo, también podemos aumentar el ITE, ya que si en la política de apagado dinámico, establecemos algún parámetro de caracterización del servidor, en función a la relación rendimiento/eficiencia, podemos apagar los servidores con peor relación y por tanto peor eficiencia.

Adicionalmente, se obtiene otra optimización de forma indirecta: a menor número de servidores en funcionamiento, menor consumo y menor perdida de energía en forma de calor, por lo tanto, los sistemas de refrigeración consumirán menos, disminuyendo a su vez el PUE.

2.5.2. Mecanismos para optimizar el uso de recursos

La segunda aproximación se centra en la gestión y optimización de los recursos, a través de la virtualización y consolidación de servidores. La virtualización es el proceso más usado en la informática para solventar grandes problemas en forma de abstracciones [10]. En el ámbito de un *datacenter* y sus servidores, consiste en replicar una máquina física como dos o más máquinas virtuales (VMs), pudiendo ejecutar en cada una de ellas una tarea o servicio determinado [1]. De este modo, cada tarea se estará ejecutando en el servidor teniendo la percepción de que tiene el control total sobre la máquina física, cuando en verdad, varias máquinas virtuales se estarán ejecutando en el mismo servidor. Esto es posible porque realmente cada tarea o máquina virtual no necesita utilizar todos los recursos computacionales que ofrece cada servidor. Con esto, en primer lugar, aumentamos la capacidad de utilización de la infraestructura y, en segundo lugar, proporciona una nueva forma de mejorar la eficiencia energética de los centros de procesamiento de datos: la consolidación.

La consolidación de máquinas consiste en agrupar todas las máquinas virtuales (VMs) en ejecución, en el menor número de máquinas físicas posibles. El movimiento de las VMs se puede realizar en tiempo de ejecución de las mismas, a través de lo que se conoce como migración. La ubicación de las máquinas virtuales debe realizarse estratégicamente, considerando diferentes factores de los recursos disponibles para una explotación óptima de los mismos.

El software de gestión energética deberá monitorizar la distribución de VMs sobre la infraestructura física, y aplicar de forma automática la consolidación de máquinas. Como resultado, algunos servidores quedarán vacíos, por lo tanto, en estado de inactividad, y podrán ser apagados para ahorrar energía. Como consecuencia, aumenta la utilización de la infraestructura disponible (ITU), aumentando la eficiencia del *datacenter*. De igual modo, si la consolidación de máquinas tiene en cuenta la relación rendimiento/eficiencia, se podrían migrar las máquinas a los servidores con la mayor eficiencia posible, aumentando el ITE.

La consolidación de máquinas no es una tarea sencilla, de hecho, está categorizado como un problema NP-completo, comúnmente conocido como el “problema de la mochila”.

2.6. Tecnologías

En esta sección se describen las tecnologías y herramientas utilizadas para el desarrollo del software de gestión energética, desde el lenguaje de programación empleado, los *frameworks* y librerías, las bases de datos, herramientas para la realización de pruebas, así como para el control de calidad y el control de versiones.

Python v3.10

El lenguaje de programación seleccionado para el desarrollo de este proyecto es *Python* [18]. Se trata de un lenguaje de programación interpretado, de alto nivel, orientado a objetos y de propósito general. Sus características más relevantes son su sintaxis limpia, el tipado dinámico de variables y la utilización de sangría (indentación) para marcar bloques de código. Contiene una amplia biblioteca estándar la cual proporciona módulos y funciones para realizar una amplia gama de tareas. En concreto, se usa la versión de *Python* 3.10.6.

Además, para el desarrollo y el manejo de dependencias del proyecto se emplean los entornos virtuales, a través de los cuales se permite aislar y gestionar de forma independiente los paquetes y dependencias, así como sus versiones, en diferentes proyectos dentro de un mismo sistema.

Como se comentará en el capítulo 4, el sistema tiene una arquitectura cliente-servidor. Cada una de las partes se implementa como un paquete *Python* independiente. Los paquetes son una forma de estructurar el espacio de nombres de los módulos de *Python* (fichero con extensión `.py`) relacionados entre sí, a través del directorio que los contiene, para facilitar la importación y reutilización de los mismos.

El principal motivo de la elección de este lenguaje, además de por las ventajas ya expuestas, es por la existencia de una gran variedad de librerías y paquetes externos para la comunicación con otros sistemas o para disponer de funcionalidades ya implementadas, como por ejemplo, la existencia librerías para la comunicación a través de la API de *OpenStack* o para ejecutar los comandos de IPMI, así como el resto de librerías citadas a continuación.

FastAPI

FastAPI [19] es un *framework* de desarrollo web de alto rendimiento para crear APIs RESTFul en *Python*. Se basa en el estándar de tipado de *Python*, conocido como "*Python type hints*", lo que permite una mejor verificación estática de tipos. Es rápida y escalable ya que está construida sobre *Starlette*, que es un *framework* de desarrollo web que explota las ventajas de la programación asíncrona.

Una de las características destacadas de este *framework* y el principal motivo de su elección, es la generación automática de la documentación de la misma. Además, es interactiva, permitiendo probar el desarrollo en tiempo real. Utiliza para ello la especificación de *OpenAPI* (*Swagger*).

Stevedore

Stevedore [20] es una librería para *Python* que proporciona un conjunto de clases para la gestión e implementación de patrones comunes para el uso de extensiones cargadas dinámicamente. Utiliza los puntos de entrada en las *setuptools* del paquete, para definir y cargar *plug-ins*. Un punto de entrada (*entry-point*) es la forma estándar de referirse a un objeto con nombre definido dentro de un módulo o paquete de *Python*. Los puntos de entrada se registran utilizando un nombre en un espacio de nombres (*namespace*).

Existen diversos patrones de carga de *plug-ins*: Por un lado, existen los "*drivers*" (figura 2.5a), que siguen el patrón conocido como "*Single Name, Single Entry Point*", que señala que solo es necesario que esté cargado un *plug-in* por cada punto de entrada y se utilizan principalmente para comunicarse con un recurso externo. Por otro lado, están las "extensiones" (figura 2.5b), con un patrón "*Many Names, Many Entry Points*", en las cuales puede haber varios *plug-ins* cargados por cada *entry-point*. Estos últimos se utilizan para ampliar la funcionalidad del sistema.

(a) Patrón de carga de *drivers*

(b) Patrón de carga de extensiones

Figura 2.5: Patrones de carga de *plug-ins* usando *Stevedore* [20]

La arquitectura de esta librería se basa, como muchos sistemas de *plug-ins*, en la herencia de clases, donde existe una clase abstracta base para cada espacio de nombres, y un conjunto de clases que heredan de esta e implementan la funcionalidad de cada *plug-in* identificado con su punto de entrada.

Trio

Trio [21] es una librería de programación asíncrona para *Python* que se enfoca en la simplicidad y la seguridad. Proporciona una forma intuitiva de escribir código concurrente y paralelo, permitiendo que múltiples tareas se ejecuten de forma cooperativa y eficiente.

El principal motivo de su elección es que, a diferencia de otros *frameworks* para la programación asincrónica como *asyncio*, que se basan en un bucle de eventos y devoluciones de llamada, *Trio* utiliza un modelo de programación basado en corrutinas llamado *"async/await"*. Esto hace que el código sea más legible y fácil de entender, ya que se puede escribir de manera secuencial. En cuanto a la seguridad, esta librería está diseñada con el objetivo de evitar los errores más comunes de la programación concurrente como son las carreras críticas o los *deadlocks*.

La programación asíncrona con corrutinas en *Python* es un enfoque de programación que permite la ejecución concurrente y eficiente de tareas sin bloquear el flujo principal del programa. Se basa en el uso de corrutinas, que son funciones especiales que pueden suspenderse y reanudarse en puntos específicos de ejecución. Es muy beneficiosa en aplicaciones que predomina la entrada/salida, como es el caso del software a desarrollar, donde es necesario esperar a que ocurran eventos o esperar respuesta de otro sistema. *Trio* orquesta todas las corrutinas para ir siguiendo el flujo de ejecución del programa de forma concurrente, incrementando el rendimiento y la eficiencia del software.

SQLAlchemy

SQLAlchemy [22] es una librería de *Python* que proporciona una abstracción de alto nivel, a través de lo que se conoce como *Object Relational Mapper* (ORM), para interactuar con distintos motores de bases de datos relacionales que utilizan el lenguaje SQL.

La ventaja que proporciona esta librería es que, por medio de su ORM, permite mapear objetos a tablas de la base de datos y viceversa, por lo que el manejo de los datos de una aplicación se puede realizar a través de operaciones estándar de *Python*, en lugar de escribir consultas a la base de datos en lenguaje SQL. Además, permite de forma automática manejar las transacciones

de la base de datos y proporciona mecanismos para el control de concurrencia, como bloqueos y aislamiento de transacciones.

Es compatible con bases de datos muy utilizadas como por ejemplo *MySQL*, *PostgreSQL*, *SQLite* y *Oracle*. En concreto, para este software de gestión energética se una base de datos *MySQL*.

Typer

Typer [23] es una librería para el desarrollo de aplicaciones con una interfaz de usuario por linea de comandos (*Command-line interface* (CLI)). Es el compañero de *FastAPI* para este ámbito, ya que también se basa en el estándar de tipado de *Python*, disponible desde la versión 3.6. A raíz de esto, utiliza una sintaxis declarativa mediante el uso de decoradores y anotaciones de tipo para definir comandos, opciones y argumentos, lo que hace que el código sea más legible y mantenible. Se ha seleccionado porque está construida sobre otra conocida librería para el mismo propositivo, conocida como *Click*, por tanto, hereda casi todas las ventajas de *Click*, pero simplificando aún más su uso.

Permite la creación de comandos, subcomandos, paso de argumentos y opciones, lo que la hace muy flexible para desarrollar una CLI completa. Además, genera automáticamente la documentación con la descripción de cada uno de los comandos y sus opciones. Admite la generación automática de completado de línea de comandos para intérpretes de comandos como *Bash*, *PowerShell* y *Fish*. Esto facilita la interacción con la CLI y permite a los usuarios ver y autocompletar los comandos, opciones y argumentos disponibles.

Pytest

Pytest [24] es un *framework* para la creación y ejecución de pruebas en *Python*. Su gran ventaja respecto a otras librerías es que no requiere de la creación de clases de prueba o que los métodos de prueba tengan nombres específicos para que sean encontrados y ejecutados por la herramienta.

Ofrece también la capacidad de ejecutar pruebas en paralelo, la generación automática de informes, y la posibilidad de instalar complementos para añadir funcionalidades, como, por ejemplo, evaluar en el mismo momento la cobertura de los test de prueba. Por último, una capacidad interesante es la parametrización de las pruebas con distintos conjuntos de parámetros, lo que reduce y simplifica bastante la sintaxis, obteniendo un gran número de casos testeados para un solo método.

SonarQube

SonarQube [25] es una plataforma de código abierto para la inspección continua de calidad del código. Realiza una serie de análisis estáticos del código fuente para detectar problemas y malas prácticas en el código. Utiliza un conjunto de reglas predefinidas y personalizables para identificar posibles vulnerabilidades de seguridad, errores de codificación, complejidad excesiva en métodos, duplicación de bloques de código, entre otros problemas. Además, puede analizar la cobertura de las pruebas realizadas en el código, lo que permite identificar áreas del código que no están adecuadamente cubiertas por pruebas unitarias. Esto ayuda a garantizar una cobertura de pruebas adecuada y mejorar la calidad del código.

Git

Por último, se usa la herramienta *Git* para el control de versiones distribuido del proyecto.

CAPÍTULO 3

Análisis y especificación de requisitos

En este capítulo se aborda el análisis y definición de requisitos del sistema, un proceso fundamental para el éxito de cualquier proyecto de desarrollo de software. Tiene como objetivo identificar y definir las necesidades del cliente y requerimientos del producto que se va a desarrollar, para poder establecer una base sólida sobre la cual se construirá.

3.1. Descripción general del software

Se ha de diseñar un software para la gestión energética de máquinas de computación *cloud*, operadas a través de *OpenStack*, utilizando las estrategias mencionadas en la sección 2.5, de modo que se puedan apagar y encender máquinas dinámicamente cuando sea necesario. El sistema para ello debe monitorizar el estado de utilización de cada una de las máquinas. Este debe mantener un conjunto (*baseline*) de máquinas vacías, para satisfacer la posible demanda en el corto plazo, pero que cuando esta caiga, se apaguen las que correspondan. Además, debe ser extensible mediante *plug-ins* para, en un futuro, permitir la consolidación de máquinas y alguna otra funcionalidad.

Figura 3.1: Diseño de marca para el producto software

Con esta idea surge desarrollar **CEMS2**, un sistema software compuesto de módulos para la gestión de la eficiencia energética para las infraestructuras de *cloud computing*. Como se muestra en la figura 3.1, el nombre surge del acrónimo en inglés de la descripción de este: ***C**loud **C**omputing **E**nergy **E**fficiency **M**anagement **S**oftware **S**ystem*, omitiendo la letra repetida por cada pareja de palabras, lo que da motivo al digito **2** al final del nombre.

3.2. Análisis de los requerimientos

Para realizar un análisis más detallado del tipo de software y los requerimientos que debía superar, se realizaron varias reuniones de colaboración con los *stakeholders*, en este caso parte del grupo de Computación Avanzada y e-Ciencia del IFCA. Estos requerimientos se redactan mediante historias de usuario (*user stories*).

Las historias de usuario es una técnica de obtención de requisitos y funcionalidades, que se utilizan en el contexto del desarrollo de requisitos ágiles, como una herramienta de comunicación que combina las ventajas de los medios de comunicación escrito y oral [26]. Describe la funcionalidad que será valiosa tanto para el usuario como para el comprador de un sistema o software, desde su punto de vista, con el lenguaje que utilizará él mismo. La atención se centra en qué necesidades o problemas se resolverán con el sistema software que se construirá [27]. Forman parte de la fórmula de captura de funcionalidades definida en 2001 por Ron Jeffries [28], compuesta de tres aspectos fundamentales:

- ***Card*:** Una descripción escrita de la historia de usuario de forma sintetizada en una pequeña tarjeta. Sirve como recordatorio y promesa de una conversación posterior.
- ***Conversation*:** El equipo de desarrollo y el propietario del producto añaden los criterios de aceptación de cada historia antes de comenzar con su implementación. Debido a que esta técnica de obtención de requisitos se complementa con las metodologías de desarrollo ágiles, pueden realizarse cambios en estos criterios.
- ***Confirmation*:** El propietario del producto o usuario de negocio confirma que el equipo de desarrollo ha entendido y recogido correctamente sus requisitos revisando los criterios de aceptación.

A continuación, se muestra un ejemplo de una de las historias desarrolladas para este proyecto:

ID:	US-01
Nombre:	Registrar máquinas físicas en el sistema.
Descripción:	Como administrador quiero poder añadir, modificar, ver y eliminar cada una de las máquinas físicas o conjuntos de estas.
Criterios de aceptación:	1. El sistema deberá utilizar un fichero que establecerá los datos de las máquinas físicas del CPD disponibles. 2. Los datos para cada máquina son: *hostname*, grupo, modelo, dirección IP de gestión, usuario y contraseña de gestión, e interfaz de conexión a cada máquina. 3. Se deberá permitir definir grupos de máquinas físicas indicando los *hostnames* del grupo, al igual que las direcciones IP, en forma de rangos. 4. El sistema deberá validar los datos, por ejemplo, el *hostname* y la dirección IP de gestión deben ser únicos en todo el sistema. 5. Se deberá notificar al administrador los errores en el caso de que los datos no sean válidos, o incoherencias entre el fichero y las máquinas ya registradas en el sistema de forma persistente. 6. El sistema proporcionará el resto de los datos necesarios, como el estado energético actual o la fecha de creación y modificación. 7. Por defecto, toda máquina física añadida al sistema deberá registrarse con la opción de monitorización deshabilitada, hasta que el administrador indique lo contrario. 8. El sistema informa al administrador si la nueva máquina física se ha añadido correctamente.
Comentarios:	■ En el fichero se deberán especificar todos los campos, ya sea de manera individual o por grupos y rango de valores. ■ Cada máquina deberá tener 2 *flags* para indicar si la máquina física tiene que ser monitorizada y si está disponible en el CPD. ■ El estado energético actual de cada máquina se obtendrá a través de *Intelligent Platform Management Interface* (IPMI).

La totalidad de las historias de usuario redactadas se encuentran en el anexo B. A partir de estas, se extraen la especificación de requisitos de software, disponible en el siguiente apartado.

3.3. Especificación de requisitos de software (ERS)

La especificación de requisitos de software (ERS) define un documento detallado que describe las necesidades, características y funcionalidades de un sistema de software [29]. Es una etapa fundamental en el proceso de desarrollo de software, ya que establece la base para el diseño, implementación y pruebas del sistema.

Una especificación de requisitos de software bien elaborada proporciona una base sólida para el desarrollo del software, ayudando a los desarrolladores a comprender claramente qué se espera del sistema y a los clientes a tener una visión clara de lo que recibirán al final del proceso de desarrollo.

3.3.1. Requisitos funcionales

Los requisitos funcionales (RF) describen las funciones que debe ejecutar el software, cuando se cumplen ciertas condiciones. A veces se denominan capacidades o características. Un requisito funcional también puede describirse un conjunto finito de pasos de prueba para validar su comportamiento [29]. A continuación, se presentan los requisitos funcionales del sistema de gestión energética, recogidos en la tabla 3.1:

Tabla 3.1: Requisitos funcionales

ID:	Descripción:
RF-01	El administrador podrá iniciar sesión en el sistema utilizando un nombre de usuario y contraseña. También podrá cerrar sesión sin ninguna comprobación previa.
RF-02	El administrador podrá registrar los grupos de máquinas físicas en el sistema, con sus respectivas características y parámetros, a través de un fichero de tipo llave y valor (`yaml`), como el mostrado en el anexo D.1.
RF-03	El administrador podrá leer todos los datos de las máquinas físicas en la base de datos del sistema.
RF-04	El administrador podrá habilitar o deshabilitar las máquinas físicas o grupos de máquinas, para que el sistema no opere sobre el conjunto deshabilitado.
RF-05	El sistema monitorizará el nivel de utilización de cada uno de los servidores.
RF-06	El administrador podrá ver el estado energético actual de cada uno de los servidores (encendidos o apagados), así como su nivel de utilización.
RF-07	El administrador podrá consultar un histórico de utilizaciones de las máquinas.
RF-08	El sistema apagará máquinas dinámicamente, por medio de un algoritmo definido, cuando el número de máquinas físicas con utilización nula es mayor al *baseline*.
RF-09	El sistema arrancará máquinas dinámicamente cuando detecte que el número de máquinas físicas vacías es inferior al *baseline*.

Tabla 3.1: Requisitos funcionales

ID:	Descripción:
RF-10	Se deberán arrancar todas las máquinas físicas registradas, y detener todas las funciones del software, si el administrador decide desactivar el sistema.
RF-11	El sistema permitirá al administrador configurar ciertos parámetros del sistema, como el *baseline* de máquinas, el periodo de monitorización y el tiempo de acción.
RF-12	El sistema debe ser extensible mediante *plug-ins* para añadir nuevas funcionalidades y añadir más políticas de optimización.
RF-13	El administrador podrá consultar directamente los *plug-ins* instalados y habilitar su carga y uso a través del fichero de configuración.
RF-14	El sistema consolidará las máquinas de forma dinámica, mediante un algoritmo predefinido y realizará las migraciones necesarias para lograr dicha optimización.
RF-15	El sistema permitirá mostrar al administrador las optimizaciones conseguidas de los algoritmos disponibles, sin aplicarlos sobre la infraestructura.

3.3.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) son los que actúan para limitar la solución. Estos a veces se conocen como restricciones o requisitos de calidad. Pueden ser clasificados más a fondo según si son requisitos de adecuación funcional, rendimiento, compatibilidad, usabilidad, fiabilidad, seguridad, mantenibilidad o portabilidad [29, 30]. A continuación, se presentan los requisitos no funcionales del sistema de gestión energética, recogidos en la tabla 3.2:

Tabla 3.2: Requisitos no funcionales

ID:	Tipo:	Descripción:
RNF-01	Seguridad	El acceso al sistema está limitado a los usuarios logueados con los permisos necesarios (administradores).
RNF-02	Mantenibilidad	El sistema debe tener un *log* que registre las acciones aplicadas para tener un inventario de la evolución del sistema.
RNF-03	Rendimiento	Las modificaciones realizadas sobre la infraestructura deberán reflejarse en la interfaz del sistema de gestión en tiempo real.
RNF-04	Usabilidad	El registro de máquinas a través del fichero permitirá utilizar rangos para definir conjuntos de máquinas y no tener que ser definidas de manera individual.
RNF-05	Usabilidad	El sistema debe ser configurado y administrado a través de una interfaz de línea de comandos (*Command-line interface* (CLI)).
RNF-06	Compatibilidad	El sistema ha de poder ejecutarse íntegramente en un computador con sistema operativo *Linux*.
RNF-07	Compatibilidad	La base de datos del sistema debe ser relacional SQL, pero compatible con varios motores de bases de datos.

CAPÍTULO 4

Diseño y arquitectura

El diseño y la arquitectura del software son aspectos clave en el desarrollo de cualquier proyecto, ya que establecen las bases para la creación de un sistema robusto, escalable y mantenible. Unos buenos diagramas de arquitectura de software facilitan la comunicación dentro y fuera de los equipos de desarrollo de software, la incorporación eficaz de nuevos desarrolladores, las revisiones/evaluaciones de arquitectura, la identificación de riesgos, la modelización de amenazas, etc. En este capítulo se describe la técnica de modelado adoptada, así como el modelado realizado para el sistema de gestión energética.

4.1. Método de modelado: *C4 Model*

El modelo C4 es una técnica de notación gráfica que se utiliza para modelar la arquitectura de sistemas de software, creado por el arquitecto de software Simon Brown, entre 2006 y 2011, con el objetivo de estandarizar y simplificar las técnicas de modelado de software [31].

A raíz del crecimiento del uso de metodologías ágiles para el desarrollo de software, se redujo en gran medida el uso de técnicas convencionales para el modelado, ya que no era viable mantener actualizados los diagramas de arquitectura, mientras se iba desarrollando el software, debido su gran complejidad y notación estricta. Está técnica de modelado propone el uso de un conjunto de abstracciones, para simplificar la notación, y de una jerarquía de diagramas, la cual proporciona diferentes niveles de detalle, cada uno de los cuales es relevante para una audiencia diferente.

4.1.1. Abstracciones

Para diseñar los diagramas, o como lo define su creador "mapas de código", se han de definir una serie de abstracciones para crear un lenguaje que se pueda usar para describir la estructura del sistema software. Se usan las siguientes abstracciones:

- **Actor:** Representa a los usuarios que interactúan con el sistema informático.
- **Sistema:** Es el mayor nivel de abstracción, y representa cualquier cosa que ofrece una funcionalidad a sus usuarios, sean personas o no. Esto incluye el sistema de software que se está modelando y cualquier otro sistema software del que dependa (o viceversa).
- **Contenedor:** Representa una aplicación o una base de datos. Un contenedor[1] es algo que necesita ejecutarse para que funcione de forma general el sistema software.
- **Componente:** En el mundo del desarrollo de software, el término "componente" se utiliza mucho, pero en este contexto, se refiere a una colección de funcionalidades relacionadas que están contenidas detrás de una interfaz bien definida.

  Es fundamental tener en cuenta que todos los componentes de un contenedor suelen ejecutarse en la misma área de proceso. Los componentes no son unidades desplegables independientes en el concepto del modelo C4.

[1] No son contenedores *Docker* en este contexto

Con estas abstracciones podemos definir un sistema informático que está formado por uno o varios contenedores, cada uno de los cuales contiene uno o varios componentes que se implementan mediante uno o varios elementos de código (clases, interfaces, objetos, funciones, etc). Las personas o actores son capaces de utilizar los sistemas de software que se crean.

4.1.2. Diagramas

El modelo C4 se compone de 4 diagramas jerárquicos, cada uno de ellos con un nivel de detalle concreto, enfocado a una audiencia especifica como se muestra en la figura 4.1.

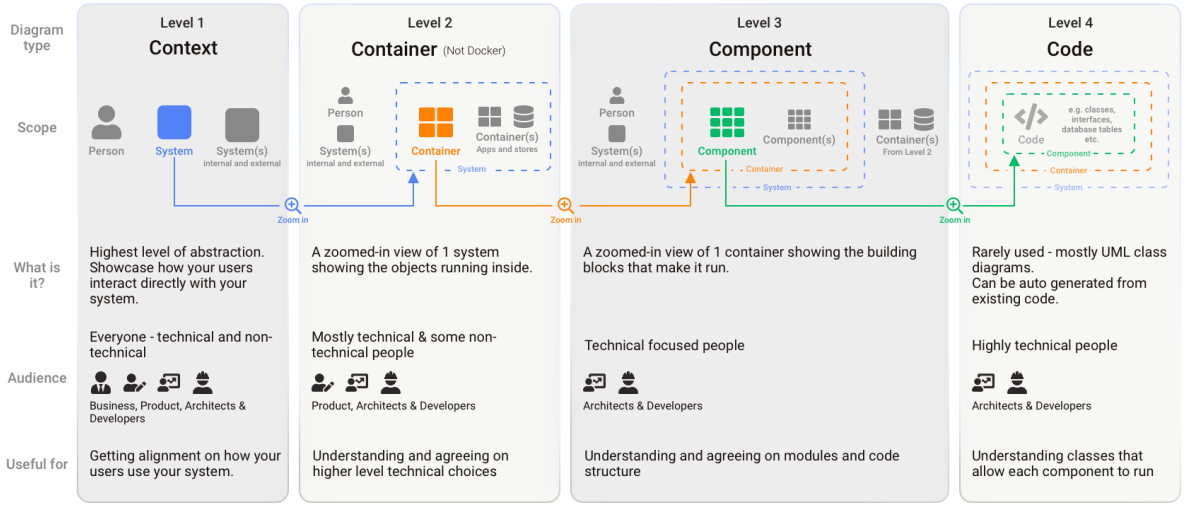


Figura 4.1: Tipos de diagramas en el modelo C4 [32]

- **Diagrama de contexto:** Proporciona un punto de partida, mostrando una idea general que describe cómo el sistema software encaja en el mundo que le rodea. Se enfoca en una audiencia general, ya que pueden ayudar a comprender el alcance del proyecto.
- **Diagrama de contenedores:** Se acerca al sistema de software en cuestión, mostrando los bloques de construcción técnicos de alto nivel (contenedores) y cómo interactúan. Estos pueden ser aplicaciones, bases de datos, microservicios, etc. Resulta útil tanto para los desarrolladores de software como para el personal de soporte y operaciones.
- **Diagrama de componentes:** Hace zoom en un contenedor individual, mostrando los componentes dentro de él y como se relacionan. Va dirigido a arquitectos y desarrolladores de software.
- **Diagrama de código:** Un diagrama de código (por ejemplo, de clases UML) puede utilizarse para hacer zoom en un componente individual, mostrando cómo se implementa ese elemento. Se trata de un nivel de detalle opcional, y que no tiene por qué implementarse para todos los componentes del sistema.

4.2. Modelado del sistema de gestión energética

Para modelar el sistema se ha utilizado la herramienta *IcePanel*[2]. Es un editor web que permite realizar este tipo de diagramas de un forma gráfica e interactiva, permitiendo el movimiento

[2] IcePanel | Explain complex software systems: https://icepanel.io/

entre los distintos tipos de diagramas y la creación de flujos de llamadas entre los distintos objetos, para representar la comunicación entre ellos.

4.2.1. Diagrama de contexto

El contexto en el que se enmarca el sistema de gestión energética, llamado **CEMS2**, está formado, tal y como se muestra en la figura 4.2 por un actor, el propio sistema y tres sistemas externos, ya descritos en las secciones anteriores de este Trabajo Fin de Grado (TFG).

El único actor que podrá acceder e interactuar con el sistema será el **administrador** de la infraestructura hardware y software del *datacenter*, concretamente en el área de computación *cloud*, ya que tendrá los permisos necesarios para poder aplicar las optimizaciones a través del sistema de gestión energética y configurar el acceso a los sistemas externos con los que se comunica.

El sistema **CEMS2** operará e interactuará de manera conjunta con otros sistemas externos, presentes como parte del software empleado para la gestión en el centro de procesamiento de datos del IFCA, como son la plataforma de gestión *OpenStack*, accedida a través de su **API** para la monitorización de la infraestructura y realizar la migración de máquinas virtuales, una fuente de datos, para posteriormente utilizando el servicio **Grafana**, obtener una visualización y realizar el almacenamiento de las métricas obtenidas, y las **máquinas físicas** del CPD para realizar los encendidos y apagados de forma dinámica, y así optimizar la eficiencia energética de la infraestructura.

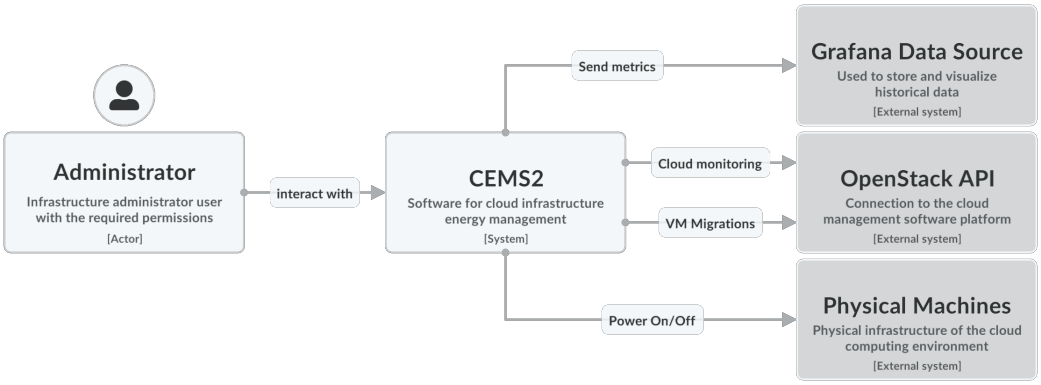


Figura 4.2: Diagrama de contexto del sistema de gestión energética

4.2.2. Diagrama de contenedores

Enfocándonos en el sistema **CEMS2**, si ampliamos la vista en un segundo nivel de detalle como es el diagrama de contenedores, observamos que el sistema se ha organizado en seis contenedores, como se muestra en la figura 4.3. Estos a su vez están divididos en dos grupos claramente diferenciados:

- Por un lado, el contenedor **CLI Application** implementa el *frontend* del sistema, a través de una interfaz de linea de comandos (CLI), que, por medio de esta, el administrador de la infraestructura podrá configurar, supervisar y operar con el sistema de gestión energética.
- Por otro lado, los otros 5 contenedores restantes conforman el *backend* del sistema de gestión energética que implementa toda la lógica del software.

En primer lugar, el contenedor **REST API Application** proporciona al sistema una interfaz de comunicación común entre el *frontend* y el resto de los contenedores presentes en el *backend*.

En segundo lugar, se encuentran los dos contenedores que implementan la lógica del sistema: **Cloud Analytics** se encarga de la monitorización de la infraestructura cloud, a partir de la obtención de métricas mediante la petición de solicitudes a la **API de OpenStack**, así como la gestión y envió de las métricas recogidas a servicios de visualización externos como **Grafana**, por medio de un *data-source*. El contenedor **Machines Control** implementa la lógica necesaria para el control de las máquinas, físicas y virtuales de la infraestructura del servicio de computación en la nube. Se comunicará, por medio de peticiones, con la **API de OpenStack** para realizar las migraciones de las máquinas virtuales (VMs) de forma automática y con las máquinas físicas de la infraestructura para realizar los encendidos y apagados de las mismas, de forma dinámica.

La comunicación entre ambos contenedores se realiza exclusivamente por medio de la **REST API Application**, la cual coordinará todo el proceso de comunicación entre los distintos contenedores. La comunicación principal entre estos dos bloques lógicos se basa en la petición y obtención de métricas, por parte del contenedor de control de la infraestructura al contenedor de monitorización, para poder generar optimizaciones a partir de estas, para posteriormente, una vez aplicadas, monitorizar de nuevo y comprobar el resultado de la optimización.

En tercer lugar, el sistema dispone de una base de datos, como mecanismo de almacenamiento de datos persistente, la cual es accesible a través de la **REST API Application**. Por último, el backend, utilizará un quinto contenedor, nombrado como **System Log**, que realizará la labor de registrar cronológicamente en un fichero, los acontecimientos que han ido afectando al sistema, a excepción de la base de datos, que utilizará su propio gestor de *logs* del motor de la base de datos utilizada.

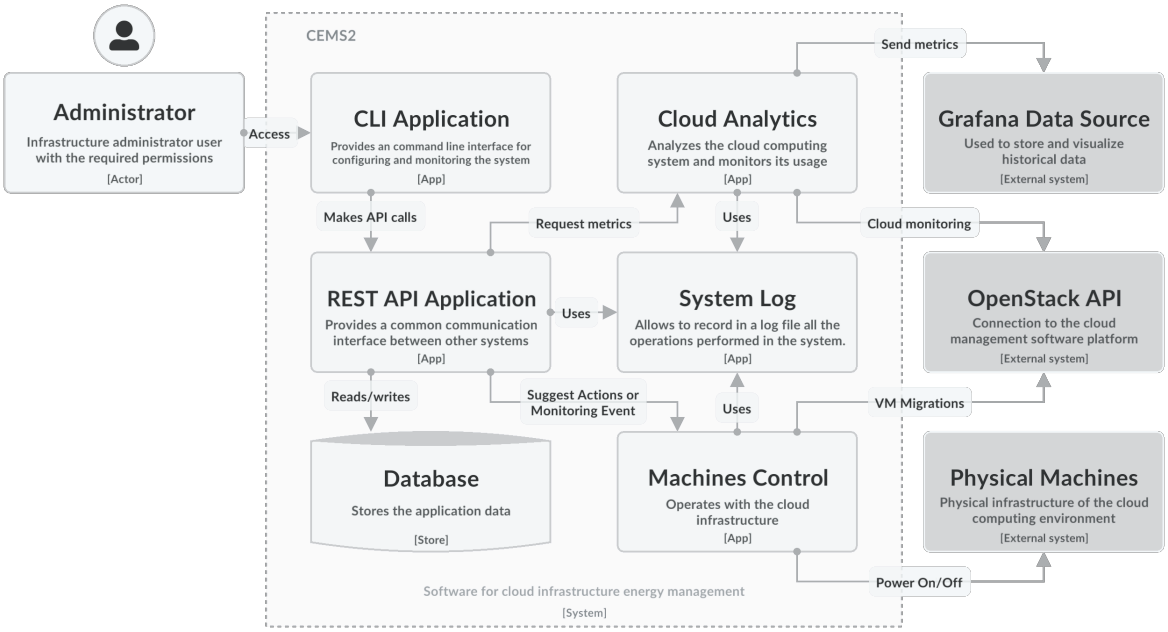


Figura 4.3: Diagrama de contenedores del sistema de gestión energética

El sistema se ha diseñado para tener una arquitectura cliente-servidor, de forma que el servidor se ejecute indefinidamente y gestione e interactúe las máquinas físicas de forma dinámica. El cliente es una utilidad que se le ofrece al administrador para consultar el estado y configurar el sistema de forma remota.

4.2.3. Diagramas de componentes

4.2.3.1. Componentes de la API REST

Como se ha comentado anteriormente, la **REST API Application** de **CEMS2** proporciona al *frontend* una interfaz de comunicación común para todos los contenedores del *backend*, e internamente, la comunicación entre cada uno de los contenedores y sistemas externos. Esta se compone de tres componentes que se comunican entre sí de forma horizontal, como se muestra en la figura 4.4, y cada uno de ellos, con cada uno de los contenedores del sistema de forma individual y vertical. Dicho de otro modo, cada componente se encarga de comunicar únicamente a un contenedor con el resto del sistema a través de sus respectivos componentes. De esta forma se reducen el número de dependencias y relaciones entre los distintos contenedores, agrupando toda la comunicación dentro de la propia API.

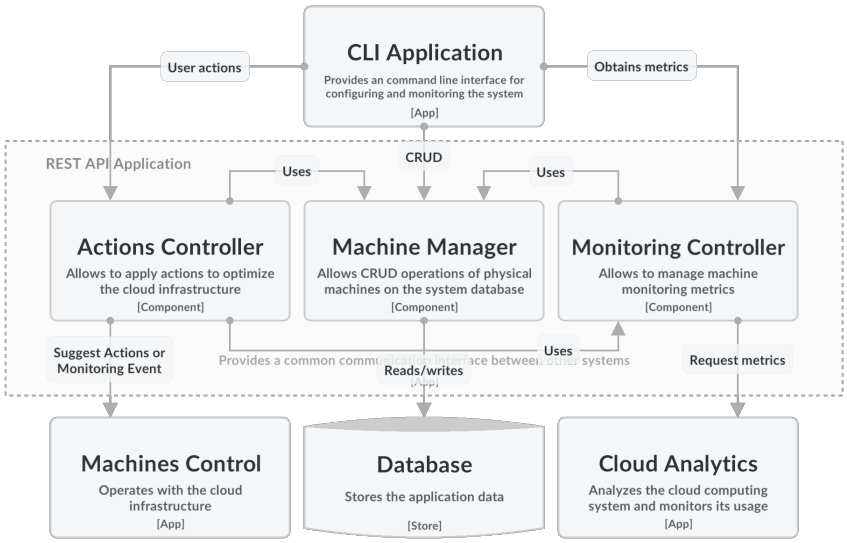


Figura 4.4: Diagrama de componentes de la API REST

Cada uno de los componentes proporcionará los *end-points* a la **CLI Application** necesarios, así como funciones internas para la comunicación entre los distintos **controllers** y **managers**.

Las comunicaciones internas entre los tres componentes son las siguientes:

- Tanto el componente **Actions Controller**, como el **Monitoring Controller**, obtienen por medio del **Machine Manager** las máquinas físicas disponibles. Para el primero de ellos, serán las que deberá controlar y gestionar y, para el segundo componente, serán las cuales deba monitorizar.
- El sistema de control necesita las métricas obtenidas por el sistema de monitorización para generar optimizaciones a partir de los valores de estas, por tanto, el **Actions Controller** necesitará obtenerlas por medio del componente nombrado como **Monitoring Controller**.

4.2.3.2. Componentes de la CLI

Este contenedor implementa el *frontend* del sistema de gestión energética **CEMS2**, a través de una interfaz por linea de comandos (*Command-line interface* (CLI)). Consta de tres componentes análogos a los tres de la **REST API Application**. De esta manera cada componente proporcionará al administrador un grupo de comandos. Cada grupo tendrá varios subcomandos con sus

respectivas opciones. Cada componente, por tanto, realizará las peticiones a los *end-points*, parametrizados de acuerdo a los subcomandos y opciones, que se ofrecen en cada uno de los respectivos componentes la **REST API Application**, tal y como se muestra en la figura 4.5.

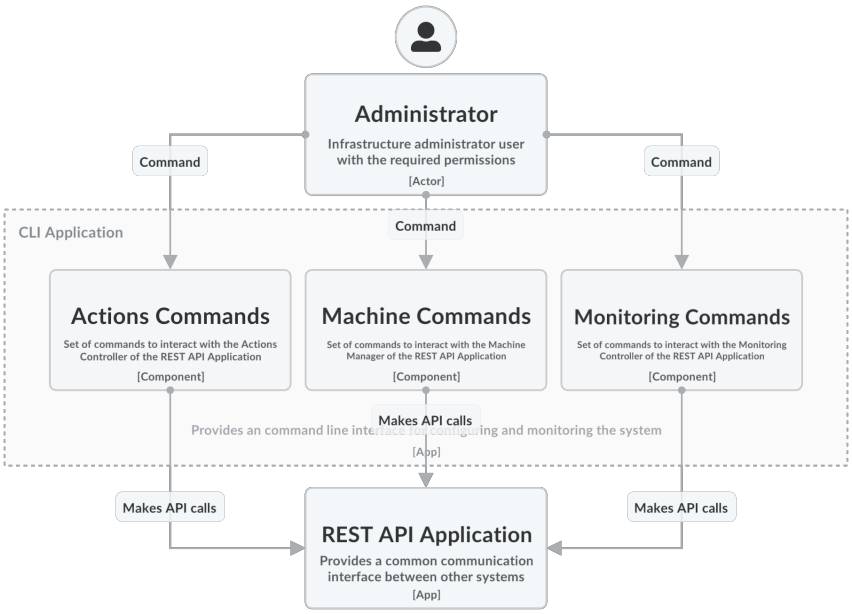


Figura 4.5: Diagrama de componentes de la *CLI Application*

4.2.3.3. Componentes del sistema de análisis

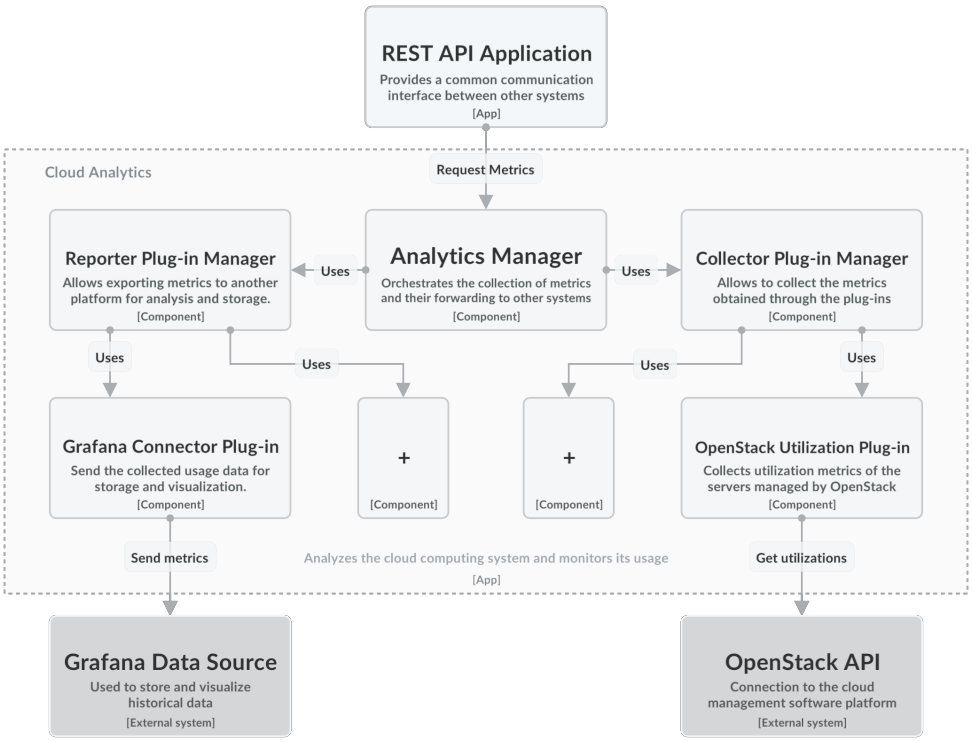


Figura 4.6: Diagrama de componentes del sistema de análisis y monitorización del *cloud*

El contenedor **Cloud Analytics** se encarga de realizar la monitorización de la infraestructura de computación *cloud*. El núcleo se compone de un **Analytics Manager** que, además de comunicarse con la **REST API Application**, coordina la recolección de métricas del CPD y su envío a otros sistemas externos para su posterior visualización y estudio. Este mánager emplea otros 2 *managers*, uno enfocado en la tarea de obtención de métricas (**Collector Plug-in Manager**) por medio de extensiones, y el otro centrado en el envío y reporte de las métricas obtenidas (**Reporter Plug-in Manager**), a través de **plug-ins**.

Tanto como para el componente **Collector Plug-in Manager**, como para el **Reporter Plug-in Manager**, se les podrá instalar nuevos *plug-ins*, representados en el diagrama de la figura 4.6 por los componentes nombrados como **+**, para así ampliar su funcionalidad y permitir obtener otro tipo de métricas y enviarlas a otras plataformas o servicios. En este caso, en el diagrama de componentes se han representado los siguientes *plug-ins*: **OpenStack Utilization**, en la parte de recolección de métricas y el conector con **Grafana** para el envío de las mismas, ya que son los necesarios para realizar una primera versión del sistema y así comunicarlo con los sistemas actuales presentes en los servicios de computación del Instituto de Física de Cantabria.

4.2.3.4. Componentes del sistema de control de máquinas

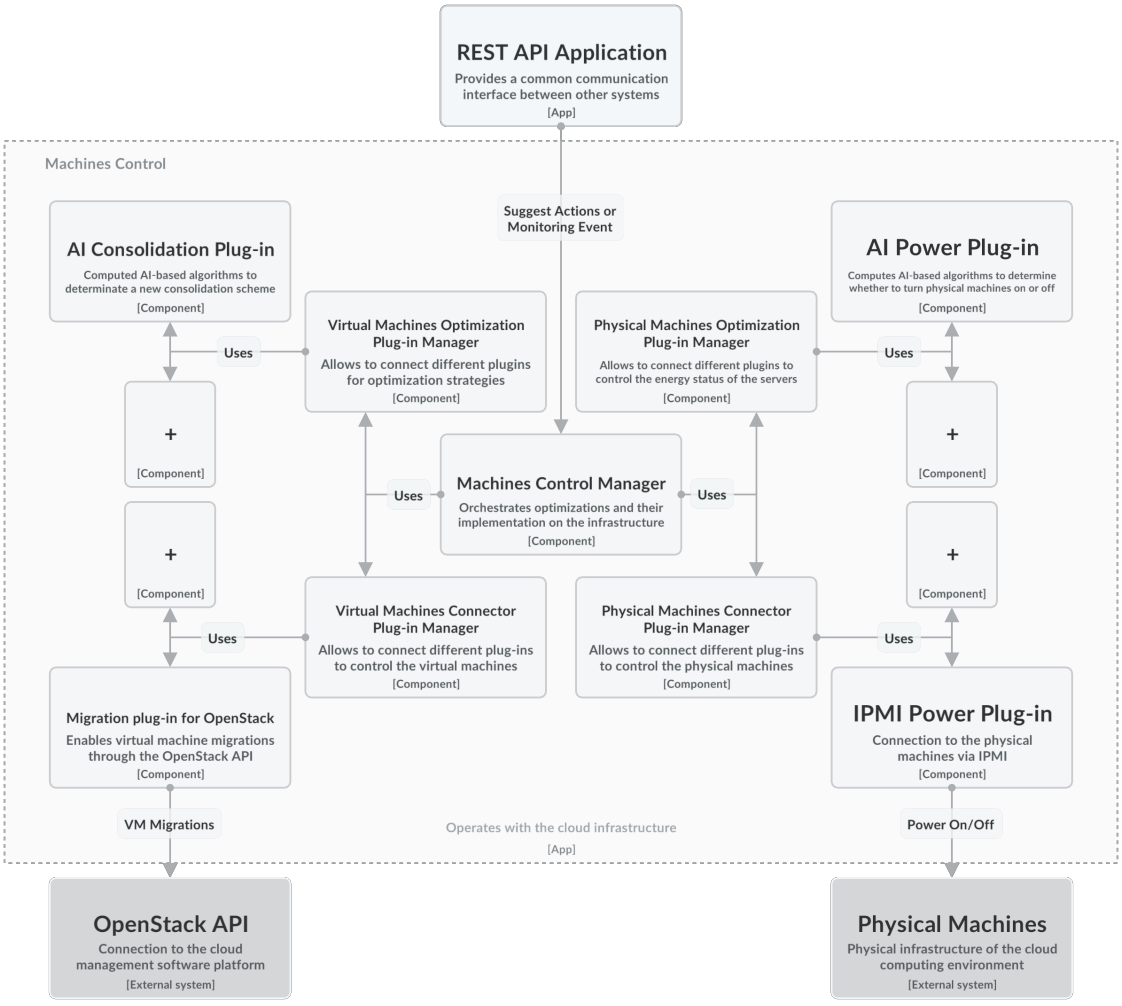


Figura 4.7: Diagrama de componentes del sistema de control de máquinas

Este contenedor, modelado en la figura 4.7, implementa el control de las máquinas por parte del sistema de gestión energética. Al igual que el contenedor de análisis, el núcleo está formado por un componente llamado **Machines Control Manager**, el cual coordina las optimizaciones y su aplicación, tanto de las máquinas virtuales como de las máquinas físicas, sobre la infraestructura cloud del CPD. Además, permite la comunicación con el resto del sistema a través de la API y su *controller* correspondiente.

Al igual que el sistema de análisis del *cloud*, la funcionalidad se descompone en un conjunto de plug-ins accesibles en este caso mediante cuatro *managers*: dos dedicados a la lógica de optimización de recursos, uno para máquinas virtuales y su análogo para las máquinas físicas. Los otros 2 *managers* permiten la conexión y aplicación de las optimizaciones obtenidas en cada uno de los dos tipos de máquinas del centro de procesamiento de datos.

A los componentes **Virtual Machine Optimization Plug-in Manager**, **Virtual Machine Connector Plug-in Manager**, **Physical Machine Optimization Plug-in Manager** y **Physical Machine Connector Plug-in Manager**, se les podrá instalar nuevos *plug-ins*, representados en el diagrama de la figura 4.7 por los componentes nombrados como **+**, para así ampliar su funcionalidad y permitir utilizar otro tipo de optimizaciones y nuevas formas o mecanismos para aplicarlas sobre la infraestructura de computación. En este caso, para la infraestructura del CPD del IFCA, se necesitan implementar dos *plug-ins* específicos para la conexión con la infraestructura. Por un lado, **Migration plug-in for OpenStack**, en la parte de conexión con las máquinas virtuales (VMs), cuyo propósito es, como su propio nombre indica, migrar las VMs. Por otro lado, para la conexión con las máquinas físicas, es necesario desarrollar el *plug-in* llamado **IPMI Power Plug-in**, para poder interactuar con las mismas, ya sea apagando o encendiendo los servidores según sea necesario.

4.2.3.5. Componentes del sistema de Log

Este contenedor no ha sido necesario desarrollarlo en mayor profundidad, con un diagrama de componentes, proporcionando así un mayor nivel de detalle, puesto que la función de este es clara y concisa, y no es necesario para entender su cometido dentro del sistema. Es uno de los principios del método de modelado C4: Se deben realizar únicamente los diagramas que aporten y faciliten la compresión del sistema a desarrollar, en parte porque será más fácil de mantener si hay cambios debido a la metodología de desarrollo ágil.

4.2.4. Diagramas de código

Al igual que para el diagrama de componentes del sistema de Log, tampoco se ha profundizado más en ninguno de los componentes que conforman los diagramas del mismo nombre, ya que, en primer lugar, requiere de mayor complejidad a la hora de realizarlo, ya que se tratan de diagramas de más bajo nivel, en lenguaje UML, que no siempre va de la mano de una mayor compresión del sistema. Además, hoy en día existen muchas herramientas para obtener de forma automática este tipo de diagramas, como son los diagramas de clases, estados, secuencia o actividades, por ejemplo, una vez realizado una primera versión de la implementación. Estos podrían servir para verificar que a bajo nivel se mantiene la estructura definida con los diagramas C4 de un nivel superior.

Finalmente, también se verificó que no era necesario tanto nivel de detalle, ya que cada componente de diagrama de componentes únicamente se descomponía en una clase. Por lo tanto, no hay la posibilidad de establecer relaciones internas al componente, con uno de estos diagramas.

CAPÍTULO 5

Implementación

En este capítulo se detalla el proceso de transformar el diseño y las especificaciones del software en un producto funcional y listo para su uso. Para ello, se explica la implementación, abordándola desde un punto de vista de acuerdo a su estructura, ya que tiene su importancia a la hora de implementar el sistema y el flujo de ejecución del mismo.

El desarrollo íntegro del proyecto software se encuentra disponible en un repositorio en *GitHub*, accesible a través del siguiente enlace: https://github.com/jaimeib/CEMS2

5.1. Desarrollo del *backend*: `cems2`

Como se ha comentado previamente, el backend del sistema de gestión energética está implementado como un paquete *Python*, llamado `cems2`. Este contiene 3 módulos, 4 paquetes y un fichero de configuración. A continuación, en la tabla 5.1, se muestra la estructura y una pequeña descripción de su propósito y utilidad dentro del sistema.

Tabla 5.1: Estructura del paquete[1] `cems2`

Tipo:	Nombre:	Descripción:
Módulo	`__main__.py`	Implementa el *launcher* del *backend* del sistema.
Módulo	`config_loader.py`	Establece la ruta al fichero de configuración y permite la carga de los parámetros desde otros módulos del sistema.
Módulo	`log.py`	Implementa y configura el sistema de log del *backend*.
Paquete	`API`	Implementa la *REST API* de `cems2`.
Paquete	`cloud_analytics`	Implementa el sistema de monitorización.
Paquete	`machines_control`	Implementa el sistema de control y operación tanto de las máquinas físicas como las máquinas virtuales.
Paquete	`schemas`	Proporciona un conjunto de clases para realizar la comunicación en base a estos objetos predefinidos.
Fichero	`config.ini`	Fichero de configuración del *backend* del sistema.

5.1.1. *Launcher*

Este módulo, como su propio nombre indica, se encarga de lanzar la ejecución de todos los contenedores que conforman el *backend* del sistema de gestión energética. Es el *entry-point* de ejecución del paquete `cems2`.

Su finalidad es ejecutar de manera asíncrona, mediante tres corrutinas, los tres flujos de ejecución concurrente que presenta el software: el servidor que atiende peticiones de la

[1] Se obvia el módulo `__init__.py` en este y en todos los paquetes descritos, ya que no contiene ninguna funcionalidad, simplemente es para indicar al intérprete de *Python* que el directorio que lo contiene es un paquete.

API, el sistema de monitorización del entorno cloud y, por último, el sistema que controla y aplica determinadas acciones sobre las máquinas físicas y sobre las máquinas virtuales de la infraestructura. Dentro de cada corrutina se tiene en cuenta que el código que se ejecuta es síncrono, ya que, por defecto, este también debe ser asíncrono.

Antes de ejecutarlas, también se encarga de establecer las dependencias que existen entre los distintos objetos, como son: el sistema de monitorización (`cloud_analytics`) con su *controller* (`monitoring_controller`) de la API, el sistema de control (`machines_control`), con su respectivo *controller* (`actions_controller`), y también las relaciones entre los propios *controllers* dentro de la API, para poder comunicarse entre ellos, como se establecía en el diagrama de componentes de la figura 4.4.

Por último, se disponen los mecanismos necesarios para poder detener el sistema utilizando este sistema de corrutinas asíncronas, ya que por defecto no captura las señales de interrupción (`SIGTERM` y `SIGINT`) del sistema operativo.

5.1.2. Gestión de configuración del sistema

La configuración del sistema se realiza por medio de un fichero de configuración, en formato `.INI`, como el que se encuentra disponible en el anexo D.2. Se encuentra dividido en varias secciones en función de la parte del sistema que utilice cada parámetro. La lectura del fichero se realiza a través de los mecanismos que proporciona el paquete externo `configparser`[2]. Este se configura, en el módulo `config_loader.py`, para indicar la ruta del fichero de configuración y se le añade la opción de poder leer valores en formato de lista, ya que por defecto solo lee valores escalares.

5.1.3. Sistema de Log

El módulo de *log* es bastante sencillo, ya que simplemente emplea y configura el paquete estándar de log que incluye *Python* por defecto. Se configura de acuerdo con los parámetros definidos en el fichero de configuración del sistema: Permite indicar el canal por el que mostrar los mensajes, por consola, a un fichero, o ambos. En el caso del fichero deberá también indicarse en otro parámetro la ruta del fichero en cuestión. A su vez, permite configurar el nivel de *logging* que el administrador desee, con un total de cinco niveles ordenados en orden de la gravedad de la siguiente forma: `DEBUG`, `INFO`, `WARNING`, `CRITICAL` y `ERROR`.

El formato de los mensajes de log se ha definido de la siguiente manera: Se indica una marca de tiempo, la gravedad del mensaje, el módulo que lo ha generado, y el texto descriptivo del mensaje como tal. Además, los mensajes mostrados por consola se imprimen de manera contextual usando una selección de colores[3], para así identificar más fácilmente la importancia de los mensajes. En la figura 5.1, se muestra un ejemplo de cada tipo de mensaje:

```
22-06-2023 12:03:02 | DEBUG    | cems2.cloud_analytics.manager | Waiting for the monitoring interval (240) to finish
22-06-2023 12:01:05 | INFO     | cems2.machines_control.manager | Baseline set to 2 machines
22-06-2023 12:03:06 | WARNING  | cems2.machines_control.pm_connector.manager | Turning off cloudC04
22-06-2023 12:03:08 | CRITICAL | cems2.machines_control.pm_connector.plugins.test | Powering off: 10.128.3.14
22-06-2023 12:01:41 | ERROR    | cems2.cloud_analytics.reporter.manager | Timeout reached for the reporter 'test2'
```

Figura 5.1: Impresión por consola de los mensajes de log de `cems2`

[2] `configparser`: https://pypi.org/project/configparser/
[3] Códigos de escape ANSI: https://en.wikipedia.org/wiki/ANSI_escape_code#Colors

El resto de los módulos deberán importar el objeto que se define dentro de este para utilizar el log con la configuración realizada.

5.1.4. Implementación de la *REST API*

La RESTful API desarrollada para `cems2`, está dividida en un módulo principal, y en tres paquetes que agrupan toda la funcionalidad en función del papel que desempeñan en la misma.

Tabla 5.2: Estructura del paquete `API`

Tipo:	Nombre:	Descripción:
Módulo	`api.py`	Implementa un servidor que atiende las peticiones de la API.
Paquete	`database`	Incluye los módulos para la gestión y conexión con la base de datos.
Paquete	`models`	Incluye las clases que representan las tablas en la base de datos.
Paquete	`routes`	Incluyen los módulos que implementan los *endpoints*.

El módulo principal se encarga de crear la API a partir del objeto proporcionado por la librería *FastAPI*, e incluir las rutas definidas en el paquete homónimo. Este objeto es el que utiliza la primera corrutina, para a través de un servidor web ASGI, llamado `Uvicorn`[4], ejecutar la API del sistema que permite comunicar sus dos paquetes principales, ejecutados en las otras dos corrutinas asíncronas.

5.1.4.1. Conexión con la base de datos

La conexión con la base de datos se realiza a través de los paquetes `database` y `models`. Este último contiene un módulo con una clase llamada `Machines`, cuya utilidad es establecer el esquema que debe tener la tabla del mismo nombre en la base de datos. El mapeo entre la clase y la tabla se realiza a través de lo que se conoce como *Object Relational Mapper* (ORM), en concreto por medio de `SQLAlchemy`[5], puesto que el sistema trabaja con una base de datos SQL.

El paquete `database` contiene dos módulos: `config.py` permite establecer una conexión con la base de datos *MySQL* del sistema, a través del paquete externo `PyMySQL`[6]. Los parámetros de conexión con la base de datos se obtienen del fichero de configuración. El segundo módulo, llamado `loader.py`, permite cargar el fichero de datos, en formato `yaml`, como el que se encuentra en el anexo D.1, el cual indica las máquinas disponibles en el centro de procesamiento de datos (CPD). Este módulo se encarga de parsear el fichero y cargarlo en la base de datos del sistema, manteniendo la consistencia de los datos del fichero y la integridad con los datos ya existentes en la base de datos. La lógica de este módulo es compleja, ya que requiere parsear un fichero con una estructura variable, por necesidad de los administradores del *datacenter*.

5.1.4.2. Rutas de la API

El paquete `routes` contiene los *endpoints* o rutas que ofrece la API, distribuidas en tres módulos (`machine.py`, `monitoring.py` y `actions.py`), uno por cada componente dentro de la API

[4] `Uvicorn`: https://www.uvicorn.org/
[5] `SQLAlchemy`: https://pypi.org/project/SQLAlchemy/
[6] `PyMySQL`: https://pypi.org/project/pymysql/

según el modelo de la figura 4.4, agrupadas por el ámbito del sistema con el que interactúan.

Todos los módulos tienen la misma estructura. Contienen una serie de métodos privados que implementan los *endpoints*, con el uso de decoradores, según la nomenclatura de *FastAPI*. Estos métodos realizan toda la lógica para procesar la petición. Además, los módulos tienen definida una clase, con el respectivo nombre del componente del diagrama, que contiene como atributos los objetos (componentes en el diagrama) con los que interactúa de forma directa. Estos atributos son los que han sido inicializados por el *launcher*, como se explicó en el apartado 5.1.1. Por último, la clase también contiene un conjunto de métodos públicos que pueden ser llamados desde otros componentes para comunicarse. Estos métodos simplemente hacen la función de *wrapper*, para reducir dependencias entre módulos.

Se puede consultar la documentación de las rutas o *endpoints* desarrollados en el anexo C.

La API que da acceso a `cems2` se ofrece sin autenticación implementada internamente. La autenticación se delega en el servidor web que ejecute el sistema en la versión en producción.

Por último, estos módulos emplean el paquete `schemas`, presente en el directorio raíz, para definir el formato de los datos que se intercambian en las peticiones. Son un conjunto de clases con atributos formateados usando el paquete externo `Pydantic`[7], el cual se emplea para la validación de datos. Se encuentran en el anexo C.2 dentro de la especificación completa de la API.

5.1.5. Monitorización de la infraestructura

El paquete `cloud_analytics` se encarga de gestionar la monitorización de la infraestructura *cloud*. El módulo principal, llamado `manager.py`, es el que realiza toda la gestión de la monitorización por medio de los paquetes `collector` y `reporter`. Este módulo, contiene una clase homónima con un método síncrono `run()`. Este es el método el cual se ejecuta desde el *launcher* de `cems2`, como la segunda corrutina asíncrona.

Tabla 5.3: Estructura del paquete `cloud_analytics`

Tipo:	Nombre:	Descripción:
Módulo	`manager.py`	Implementa el mánager del sistema de monitorización.
Módulo	`plugin_loader.py`	Permite cargar los *plug-ins* instalados para la recolección y exportación de métricas a través de *Stevedore* [8].
Paquete	`collector`	Implementa la funcionalidad de recolección de métricas.
Paquete	`reporter`	Implementa la funcionalidad de exportación de métricas.

El flujo de ejecución de este mánager de monitorización, mostrado en la figura 5.2, se compone de los siguientes procesos: En primer lugar, carga los objetos que representan los mánager de recolección y exportación, presentes en los dos paquetes. Obtiene el intervalo de monitorización establecido en el fichero de configuración. Cabe destacar que es un periodo absoluto. En segundo lugar, el mánager se quedará esperando a que se active el *flag* que tiene para controlar su ejecución (`running`) y que la lista de máquinas para monitorizar tenga alguna máquina

[7] `Pydantic`: https://pypi.org/project/pydantic/
[8] La gestión de *plug-ins* se desarrolla en el apartado 5.1.7, puesto que es una parte común en todo el sistema.

registrada. Las máquinas para monitorizar son las que están disponibles en la base de datos, que tengan activado el *flag* de monitorización, que lo establece el administrador manual, a través de la API o el cliente de *frontend*, y que la máquina física esté encendida. Si se cumplen todas estas condiciones, entonces se ejecuta lo que se ha denominado *sprint* de monitorización.

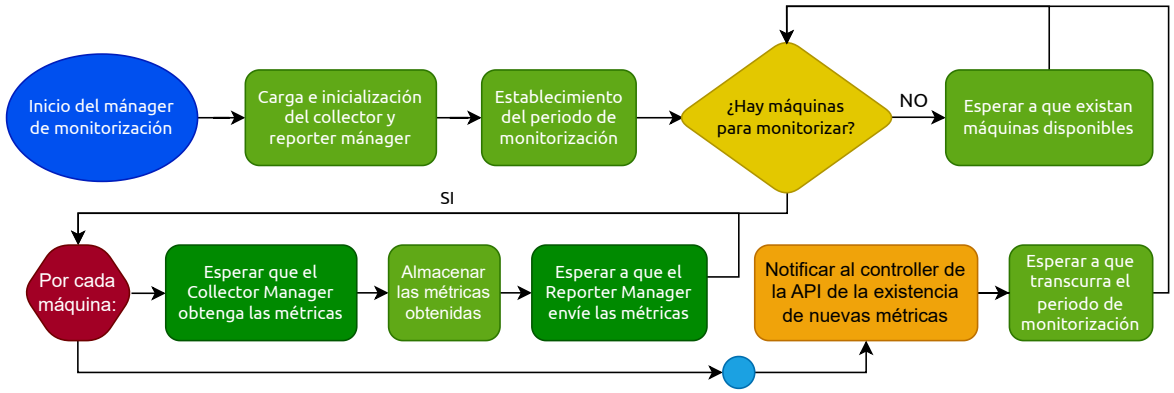


Figura 5.2: Diagrama de flujo de ejecución del mánager del sistema de monitorización

El *sprint* de monitorización consiste en crear, por medio de *Trio*, una corrutina asíncrona que procese cada máquina de la lista. Cada corrutina espera a que el `CollectorManager` devuelva todas las métricas para esa máquina. A continuación, las registra internamente en un diccionario, cuya llave es el `hostname` de cada máquina y el valor contiene la lista de las métricas recogidas, de forma que se almacenan únicamente las últimas métricas para cada máquina registrada. Posteriormente, las manda por medio del `ReporterManager`. Una vez completado el *sprint* de monitorización, se notifica al `controller` de la API emparejado, la existencia de nuevas métricas, las cuales se envían, por medio de esta, a los sistemas que generan las optimizaciones. Por último, se espera el intervalo definido hasta el siguiente *sprint*.

5.1.5.1. Recolección de métricas

El paquete `collector` implementa el sistema de *plug-ins* para la recolección de métricas del entorno computacional. Tiene una estructura como la que se muestra en la tabla 5.4, la cual es común a todos los sistemas de *plug-ins* que se expliquen en adelante.

Tabla 5.4: Estructura del paquete que implementa un sistema de *plug-ins*

Tipo:	Nombre:	Descripción:
Módulo	`manager.py`	Implementa el mánager que gestiona el sistema de *plug-ins*.
Módulo	`base.py`	Implementa una clase abstracta que heredan los *plug-ins*.
Paquete	`plugins`	Agrupa los módulos que implementan los *plug-ins* instalados.

En concreto, el mánager de este paquete contiene el método asíncrono que permite obtener la lista de métricas para una máquina dada. Este método es el que se llama por medio del `CollectorManager` en el mánager general de monitorización.

Como se indica en el diagrama de flujo de la figura 5.3, este método permite lanzar, de forma asíncrona, una corrutina para obtener la métrica que proporciona cada uno de los *plug-ins* de recolección cargados en el sistema, por lo que se generara una lista con todas las métricas

capturadas para una máquina física concreta. Particularmente, como la corrutina depende de la respuesta de un sistema externo, en el caso del IFCA, *OpenStack*, las corrutinas son lanzadas con un *timeout*, para garantizar que el sistema no se quede bloqueado en el caso de que algún *plug-in* para una máquina concreta no responda. Se notifica, por medio del *log*, si el *timeout* ha expirado para un *plug-in* recolector especifico en una máquina determinada y así se pueda solucionar por parte del administrador en la mayor brevedad posible.

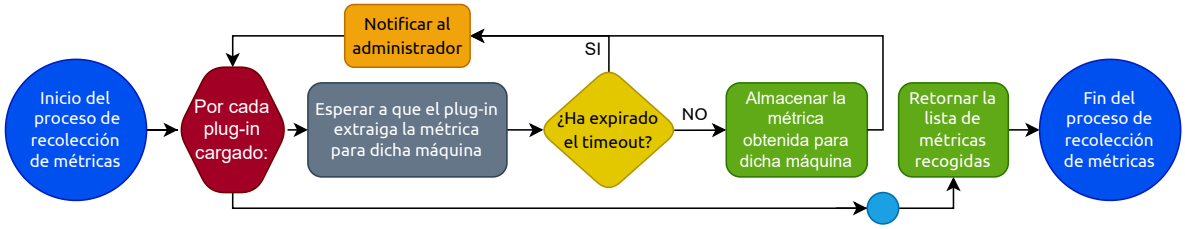


Figura 5.3: Diagrama de flujo de ejecución del proceso de recolección de métricas

Por último, el módulo `base.py`, es el esquema que deben seguir y heredar las extensiones del sistema de recolección de métricas. Contiene dos métodos: uno síncrono de inicialización, para realizar las labores especificas con cada *plug-in*. Por ejemplo, para un *plug-in* que se conecta a través de la API de *OpenStack*, se necesita obtener una sesión por medio de su API de autenticación, presente en el componente llamado *Keystone*. El segundo método, de característica asíncrona, es el que permite como tal obtener una métrica, siguiendo el *schema* proporcionado para ello (`Metric`), dado un identificador de la máquina (`hostname`).

5.1.5.2. Exportación de métricas

El paquete `reporter` al tratarse de un sistema de *plug-ins* tiene la misma estructura que la de la tabla 5.4. Además, el funcionamiento interno es homólogo al del sistema de recolección.

El mánager contiene el método asíncrono que permite enviar la lista de métricas desde la corrutina creada para cada una de las instancias de máquinas físicas, en el mánager de monitorización. Al igual que en caso anterior, se crea otra corrutina para enviar la lista de métricas de cada máquina a cada uno de los *plug-ins* de exportación que estén cargados en ese preciso momento. Cada una de ellas también tendrá un *timeout* definido, con el valor de otro parámetro en el fichero de configuración. En la figura 5.4 se describe el flujo de ejecución de este método.

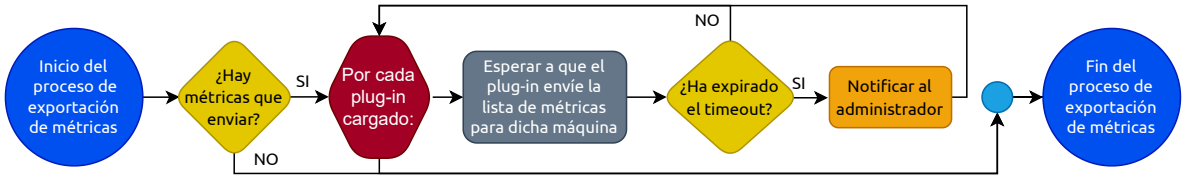


Figura 5.4: Diagrama de flujo de ejecución del proceso de exportación de métricas

El módulo `base.py` define la estructura que han de heredar los *plug-ins* de exportación. En este caso, contiene su respectivo método de inicialización para, por ejemplo, en el *plug-in* que implemente dicha funcionalidad, establecer una conexión al *data-source* configurado en *Grafana*, y otro método asíncrono para poder enviarle la lista de métricas de cada una de las máquinas físicas, desde el mánager presente en este paquete.

5.1.6. Control de la infraestructura

El propósito del paquete `machines_control` es el control y operación con la infraestructura del *datacenter*. Se compone de un módulo principal (`manager.py`) que coordina todo el proceso de optimización y aplicación sobre esta. Lo realiza a través de cuatro paquetes específicos, dos de ellos para obtener las optimizaciones, a partir de las métricas recogidas, y los otros dos para plasmarlas sobre el entorno de computación. Cada paquete contiene un sistema de *plug-ins* propio.

Tabla 5.5: Estructura del paquete `machines_control`

Tipo:	Nombre:	Descripción:
Módulo	`manager.py`	Implementa el mánager del sistema de control.
Módulo	`plugin_loader.py`	Permite cargar los *plug-ins* instalados para obtener tanto las optimizaciones, como los conectores a las máquinas a través de *Stevedore*.
Paquete	`vm_optimization`	Implementa la gestión de las optimizaciones sobre las máquinas virtuales (VMs) del entorno de computación.
Paquete	`vm_connector`	Implementa la conexión con la plataforma de gestión de las máquinas virtuales (VMs).
Paquete	`pm_optimization`	Implementa la gestión de las optimizaciones sobre las máquinas físicas del entorno de computación.
Paquete	`pm_connector`	Implementa la conexión con la plataforma de gestión de las máquinas físicas.

De forma análoga al sistema de monitorización, en módulo principal contiene una clase `Manager`, en el cual se establecen como atributos en su inicialización, las dependencias tanto con su *controller* de la API, como con los mánager presentes en cada uno de los paquetes ya descritos. De igual manera, contiene un método síncrono `run()`, el cual es ejecutado como la tercera corrutina por el *launcher* de `cems2`.

Como se muestra en la figura 5.5, el flujo de ejecución comienza por cargar los mánager de optimización y conexión para poder tener disponibles desde el primer momento los *plug-ins* de conexión con las máquinas, ya que la siguiente tarea que se ha de realizar será, obtener la totalidad de la lista de máquinas disponibles (*flag* `available` activo) en el sistema, para inicializar su estado energético actual. Para realizarlo, obtiene la lista de máquinas físicas disponibles registradas en el sistema a través de su *controller* de la API (`actions_controller`), que se comunicará con el `machine_manager` para obtener la lista de máquinas desde la base de datos. Una vez obtenida la lista, por cada máquina física se creará una corrutina asíncrona para procesar la petición de obtener el estado energético actual. Como depende de un sistema externo, conectado a través de un *plug-in*, cada corrutina tiene asociado un *timeout* para que el sistema no se bloqueé por algún problema de conexión con alguna de las máquinas. Dicha corrutina esperará a que el mánager del sistema de conexión con las máquinas físicas obtenga el estado actual la máquina.

El flujo de ejecución del mánager de control diverge en dos flujos, ejecutados de forma concurrente mediante dos corrutinas asíncronas a través de la librería *Trio*:

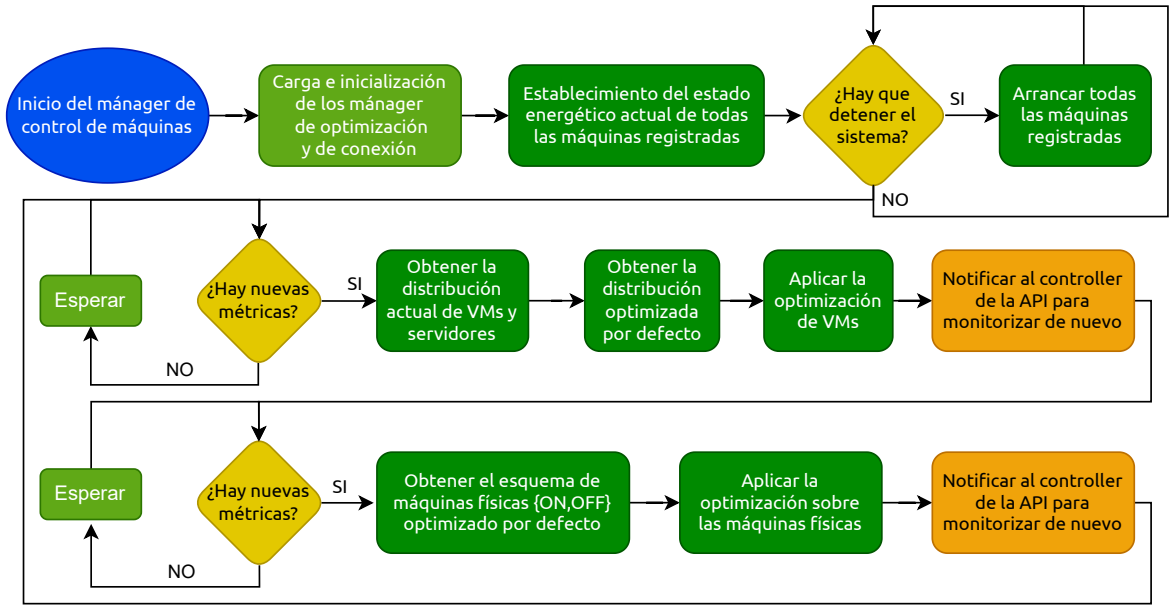


Figura 5.5: Diagrama de flujo de ejecución del mánager del sistema de control de máquinas

La primer de ellas es una rutina de control de la ejecución de dicho mánager. La funcionalidad de esta rutina es que, si en algún momento el administrador de la infraestructura decide detener el sistema por labores de mantenimiento o actualización, por ejemplo, el mánager deberá arrancar todas las máquinas disponibles en el sistema, ya que, por ejemplo, cuando se aplica una actualización, se aplica a todas las máquinas, y para ello tienen que estar encendidas. Al igual que para obtener el estado energético, se crea una corrutina asíncrona para cada una de ellas con un *timeout* establecido. Se realiza el apagado utilizando el mánager del sistema de conexión con las máquinas físicas, incluido en el paquete `pm_connector`. Por último, notificará al resto de componentes del sistema, por medio del *controller* de la API, el cambio realizado, para que se actualice en la base de datos, así como en el sistema de monitorización.

El segundo flujo de ejecución realiza lo que se ha denominado *sprints* de optimización y aplicación y está dividido en dos fases. Cada una de las fases del *sprint* se ejecutará cada vez que active un evento definido para la llegada de nuevas métricas, recogidas por el sistema de monitorización. Cuando se reciben las métricas por medio del `controller` de la API, se distribuirán entre los mánager de optimización, ya que son los que las necesitan para optimizar el sistema. A continuación, se activará el evento y comenzará el *sprint*.

La primera acción que se realiza es obtener desde el mánager de optimización de máquinas virtuales, la optimización por defecto, definida en el fichero de configuración del sistema. El resultado la optimización es un diccionario, donde cada llave es el `hostname` de la máquina física, y el valor contiene una lista de las VMs que deben ejecutarse en la correspondiente máquina física. A continuación, a través del mánager del conector con las máquinas virtuales, se realizarán las migraciones de máquinas que correspondan en función de su disposición actual y el esquema de consolidación generado. Para terminar esta primera fase, notificaremos al sistema de monitorización que la infraestructura se ha modificado y, por tanto, es necesario volver a ejecutar el *sprint* de monitorización con la nueva disposición de máquinas virtuales. Por último, es necesario esperar a que termine la monitorización, así que, se reinicia el evento y se espera.

La segunda fase del *sprints* de optimización y aplicación se centra en optimizar las máquinas físicas. Una vez se ha activado el evento de la recepción de las nuevas métricas, se obtendrá la

optimización por defecto, indicada por medio del fichero de configuración. La optimización se representa con un diccionario con dos llaves: "on" y "`off`". El valor de cada una de estas llaves es una lista con las máquinas físicas que hay que encender o dejar encendidas en el caso de que ya estén en ese estado, o, por lo contrario, apagar o dejar en ese estado en el caso de que ya estén apagadas. Una vez obtenida la optimización, se aplica mediante el mánager del conector con las máquinas físicas. Al igual que en la fase anterior, se notifica por medio del *controller*, que el estado del entorno ha cambiado y por tanto el sistema de monitorización debe volverse a ejecutar. Para finalizar, se reinicia el evento para volver a esperar al inicio de la primera fase.

5.1.6.1. Optimización de máquinas virtuales

El paquete `vm_optimization` implementa un sistema de *plug-ins* para la obtención optimizaciones, en forma de consolidaciones de máquinas, a través del conjunto de *plug-ins* instalados. La estructura del paquete es una vez más como la indicada en la tabla 5.4.

Contiene un mánager que ejecuta las labores de comunicación entre el mánager principal de control y los *plug-ins*. Este mismo, al inicializarse, creará una corrutina asíncronas, por medio de la librería *Trio*, para ejecutar el *plug-in* de optimización definido por defecto por el administrador en el fichero de configuración. En este momento, el *plug-in* en cuestión deberá esperar a recibir, desde este mánager, las siguientes métricas que se obtenidas por medio del sistema de monitorización. El *plug-in* de optimización deberá decidir por sí solo, si descartar la optimización lograda hasta el momento, puesto que el estado del sistema ha podido variar, y por tanto ya no ser válida, o adaptarla a las nuevas métricas recibidas. Del mismo modo, cada *plug-in* disponible deberá decidir que métricas utilizar en función de su estrategia de optimización. En este caso, para implementar las extensiones que obtengan los esquemas de consolidación se necesitarán disponer de métricas de utilización de las máquinas físicas, lo que incluye un listado de las mismas y las VMs en ejecución. Las máquinas virtuales que se están ejecutando en una máquina física se trata como si fuera una métrica más, instalada a partir de un *plug-in* en el sistema de monitorización.

Por último, por medio de este mánager, se podrán obtener los resultados de las optimizaciones para cada uno de los *plug-ins* de optimización instalados. Para ello, el mánager creará una corrutina para cada uno de los *plug-ins* existentes, a excepción del establecido por defecto, debido a que ya se encuentra en ejecución y esperará a que el *plug-in* le proporcione un resultado válido.

El módulo `base.py` define el esquema, en forma de clase, que deben heredar los *plug-ins* de optimización. Define tres métodos públicos: el primero de ellos para inicialización y de forma síncrona, a diferencia de los restantes, de tipología asíncrona, para la ejecución del mismo, y la obtención del resultado de la optimización desde el mánager.

5.1.6.2. Conexión con las máquinas virtuales

A través del paquete `vm_connector` se pueden realizar las migraciones de máquinas virtuales necesarias, según el esquema de consolidación generado por la extensión de optimización de máquinas virtuales por defecto en el sistema. La estructura de este corresponde con la mostrada en la tabla 5.4, al tratarse también de un sistema de *plug-ins*.

Por medio del mánager presente en este sistema de *plug-ins*, se calcula cuáles son las migraciones necesarias para pasar de la distribución actual a la distribución optimizada. Las migraciones se realizan de forma concurrente por medio de corrutinas asíncronas, con un *timeout* asociado a cada una de ellas, por si el *plug-in* de conexión o la plataforma que gestiona las

VMs no respondiesen. La plataforma de gestión de las máquinas virtuales se obtiene a partir de los metadatos presentes en las métricas del tipo "máquina virtual". Realiza una serie de comprobaciones para cerciorarse de la plataforma que aprovisiona de máquinas virtuales sobre la infraestructura física, y que se dispone del *plug-in* de conexión para dicho sistema.

Cada corrutina asíncrona recibe un *hostname* de máquina física destino junto con una lista de máquinas virtuales. Para ello se obtiene el conector necesario y se ejecuta el método implementado para dicho propósito.

El módulo `base.py` contiene la clase que deben extender cada uno de los *plug-ins* de este sistema. Contiene dos métodos: el primero de ellos para la inicialización, para en este caso, como la plataforma de gestión es *OpenStack* establecer una conexión con el componente encargado de proporcionar el servicio de aprovisionamiento de máquinas virtuales. El según método es el que permite la migración de una VMs, identificada por su *uuid*, a una máquina física, definida por su *hostname*.

5.1.6.3. Optimización de máquinas físicas

El paquete `pm_optimización` contiene un tercer sistema de *plug-ins*, está vez para la obtención de optimizaciones sobre la infraestructura hardware del *datacenter*. Como se ha comentado previamente, las optimizaciones consisten en obtener un esquema de representan los conjuntos de máquinas que hay que apagar y que hay que encender. También se estructura como el resto de los sistemas de *plug-ins*.

Al igual que su análogo para las optimizaciones de las máquinas virtuales, el mánager lanza una corrutina asíncrona, para el *plug-ins* de optimización por defecto. Cuando esté en ejecución, esperará a recibir nuevas métricas desde el mánager, las cuales deberán ser autogestionadas por cada extensión, en función de la estrategia que empleé. Del mismo modo, el mánager podrá obtener la última optimización que el propio *plug-in* consideré válida.

Para implementar este tipo de optimizaciones, cada *plug-in* necesitará disponer de métricas de utilización de las máquinas físicas, y el mánager le proporcione el *baseline* de máquinas en reserva, respecto del número total de máquinas con las que puede operar el sistema, para suplir la posible demanda, para tenerlo en cuenta a la hora de calcular cuantas máquinas es necesario apagar o encender. Las extensiones deben heredar la clase definida en el módulo `base.py`, con el mismo esquema que para la homologa en las optimizaciones de VMs.

Al igual que el sistema equivalente para las máquinas virtuales, se puede obtener el resultado de cualquiera de las optimizaciones implementadas en cada uno de los *plug-ins* instalados en el sistema, paralelizando la ejecución y obtención de resultados de los mismos, para reducir el tiempo de respuesta de la petición.

5.1.6.4. Conexión con las máquinas físicas

Por medio del último paquete (`vm_connector`) del sistema de control de la infraestructura, se aplican las optimizaciones obtenidas por el sistema descrito anteriormente. Su estructura se compone, como el resto de sistemas de *plug-ins*.

El mánager permite obtener para cada máquina, la extensión necesaria para su conexión, a partir del atributo `connector`, definido en cada instancia. Con el *plug-in* seleccionado, por medio del mánager, se podrán apagar o encender cada una de las máquinas físicas, según lo indique el esquema generado por la optimización por defecto. Para ejecutar la aplicación de la optimización sobre la infraestructura, y puesto que el control energético de las máquinas depende de un

sistema externo, el mánager, crea una corrutina asíncrona por cada máquina del esquema de optimización, asociada a la misma un *timeout* para no bloquear el sistema en el caso de que la comunicación con las máquinas no sea la esperada. Una vez lanzado el comando para modificar el estado de la máquina, se obtendrá el estado actual para comprobarlo y se notificará al mánager principal del sistema de control sobre la infraestructura, para que mediante el *controller* asociado, actualice los cambios en el resto del sistema.

5.1.7. Implementación de los *plug-ins*

El patrón de carga de *Stevedore* elegido para todos los sistemas de *plug-ins* del software de gestión energética ha sido el de extensiones, debido a que permite una mayor flexibilidad y posibilidad de ampliación en un futuro. La forma de elegir que *plug-ins* están disponibles para cada espacio de nombres se ha implementado a través de parámetros en el fichero de configuración, del anexo D.2, que permiten definir, en formato de lista, los nombres de los *plug-ins* que se quieren cargan cuando el sistema se inicie.

Realmente los *plug-ins* no tienen que estar en ningún paquete concreto, se ha realizado así para agruparlos según el tipo del sistema al que pertenecen, ya que la forma de instalarlos es indicando su *entry-point* en las *setuptools* del proyecto (`setup.py`), indicando los nombres de los *plug-ins* a instalar para cada *namespace* definido, y el propio *entry-point*, como se muestra a continuación:

Código 5.1: Instalación de los *plug-ins* en las *setuptools* del proyecto

```
[entry_points]
cems2.cloud_analytics.collector =
    test = cems2.cloud_analytics.collector.plugins.test:Test
    test2 = cems2.cloud_analytics.collector.plugins.test2:Test2

cems2.cloud_analytics.reporter =
    test = cems2.cloud_analytics.reporter.plugins.test:Test
    test2 = cems2.cloud_analytics.reporter.plugins.test2:Test2
```

Para la carga dinámica de estos *plug-ins* en tiempo de ejecución a través de *Stevedore*, se utilizan los distintos módulos denominados `plugin_loader.py`. Contienen definido en forma de constante, cada uno de los espacios de nombre (*namespaces*) que contiene `cems2`. Se definen seis *namespaces*, uno por cada sistema de *plug-ins* presente en el sistema.

El módulo utiliza *Stevedore* por medio de dos métodos privados, que tienen como parámetro el `namespace` correspondiente a cada sistema de *plug-ins*. El primero obtiene un conjunto de los *plug-ins* instalados para dicho espacio de nombres, a través del objeto `ExtensionManager` de la librería, puesto que es el patrón de carga seleccionado. El segundo de los métodos permite obtener las clases que implementan los *plug-ins* como tal, a través de un diccionario que mapea cada nombre del *plug-in* (*entry-point*) con su respectiva clase.

Por último, este módulo dispone de parejas de métodos públicos que permiten obtener desde el módulo que gestiona la carga y llamada de los *plug-ins* (los mánager de cada uno de los sistemas de *plug-ins*) la lista con los nombres, así como el diccionario con los *plug-ins* para cada *namespace* definido en sistema de *plug-ins*.

5.2. Desarrollo del *frontend*: `cems2-cli`

El *frontend* del sistema de gestión energética (**CEMS2**), está implementado en un paquete *Python*, de nombre `cems2cli`, el cual contiene una estructura como la descrita en la tabla 5.6.

Tabla 5.6: Estructura del paquete `cems2cli`

Tipo:	Nombre:	Descripción:
Módulo	`__main__.py`	Implementa el *launcher* del *frontend* del sistema.
Módulo	`config_loader.py`	Establece la ruta al fichero de configuración y permite la carga de los parámetros desde otros módulos del sistema.
Paquete	`command`	Incluye los módulos que implementan los comandos.
Fichero	`config.ini`	Fichero de configuración del *frontend* del sistema.

El módulo de carga de la configuración es idéntico al desarrollado en el *backend*. Sin embargo, el fichero de configuración es distinto, ya que únicamente contiene la ruta de acceso a la API. Se encuentra disponible en el anexo D.3.

5.2.1. *Launcher*

Este módulo, como su propio nombre indica, se encarga de lanzar la ejecución del paquete que proporciona la interfaz del *frontend*, en forma de *Command-line interface* (CLI). Es el *entry-point* de ejecución del paquete `cems2cli`.

Como se ha comentado en la sección 2.6, para la implementación de esta se utiliza la librería *Typer*. Por ello, en este módulo se crea una instancia del objeto del mismo nombre, el cual tendrá la función de ejecutar toda la interfaz de comandos. Por último, se añaden los comandos presentes en los módulos del paquete llamado `command`, como se explica en el siguiente apartado.

5.2.2. Comandos disponibles

La *Command-line interface* (CLI) contiene tres comandos principales: `machine`, `monitoring` y `actions`. Cada comando tiene un módulo dentro del paquete `command`, y como se explicó en el capítulo 4, tiene una analogía a la arquitectura interna de la API, con la cual se comunica. Es por esto, que cada comando contendrá una serie de subcomandos y opciones que permitirán comunicarse con los *endpoints* de esta, agrupados de igual manera en estos tres grupos.

5.2.2.1. Comando `machine`

El comando `machine` permite realizar las llamadas a los *endpoints* disponibles desde el módulo `machine.py` de la API, para la gestión de las máquinas físicas. Implementa 2 subcomandos:

- **`inventory`:** Permite obtener toda la información de las máquinas físicas registradas en el sistema. No quiere argumentos, pero ofrece una serie de opciones de filtrado:
  - **`-h`** o **`--host`** para filtrar por el *hostname* de la máquina.
  - **`-i`** o **`--id`** para filtrar por el ID de la máquina en la base de datos.

- **`-g`** o **`--group`** para filtrar por el nombre del grupo de la máquina.
- **`-b`** o **`--brand`** para filtrar por la marca o modelo de la máquina.
- **`-c`** o **`--connector`** para filtrar por el nombre del conector de la máquina.
- **`-e`** o **`--energy`** para filtrar por el estado energético actual de la máquina. El parámetro junto a la opción debe ser de tipo booleano (ON=TRUE | OFF=FALSE)
- **`-m`** o **`--monitoring`** para filtrar por estado de monitorización, es decir, si la máquina está siendo monitorizada o no. El parámetro junto a la opción debe ser de tipo booleano.
- **`-a`** o **`--available`** para filtrar por la disponibilidad de la máquina en el sistema. El parámetro junto a la opción debe ser de tipo booleano.

- **`monitor`:** Permite establecer que máquinas físicas han de ser monitorizadas. Requiere un argumento booleano para indicar si debe o no debe ser monitorizada. Para la identificación de la máquina en cuestión existen 2 opciones: Por el *hostname* (**`-h`** o **`--host`**) o por el ID interno del sistema de gestión energética (**`-i`** o **`--id`**).

A continuación, en la figura 5.6 se muestra el resultado utilizando uno de los comandos desarrollados en este módulo, para mostrar el inventario detallado de las máquinas físicas registradas. Para la salida por consola en formato de tabla se utiliza la librería externa para *Python* llamada `Rich`[9].

```
❯ cems2cli machine inventory
```

CEMS2 Machines Information

ID	Hostname	Groupname	Brand Model	Management IP	Management Username	Management Password	Connector Name	Energy Status	Monitoring Flag	Available Flag
1	cloudA001	cloudA	IBM Power 8	10.128.1.1	admin1	admin1	test	ON	OFF	ON
2	cloudA002	cloudA	IBM Power 8	10.128.1.2	admin2	admin2	test	OFF	ON	ON
3	cloudA003	cloudA	IBM Power 8	10.128.1.3	admin	admin	test	OFF	ON	ON
4	cloudA011	cloudA	IBM Power 8	10.128.1.11	admin	admin	test	OFF	ON	ON
5	cloudA021	cloudA	IBM Power 8	10.128.1.21	admin	admin	test	ON	ON	ON
6	cloudA022	cloudA	IBM Power 8	10.128.1.22	admin	admin	test	ON	ON	ON
7	cloudB01	cloudB	Lenovo	10.128.2.1	LenovoAdmin	LenovoPass	test	OFF	OFF	OFF
8	cloudB02	cloudB	Lenovo	10.128.2.10	LenovoAdmin	LenovoPass	test	OFF	OFF	OFF
9	cloudB03	cloudB	Lenovo	10.128.2.3	LenovoAdmin	LenovoPass	test	OFF	OFF	OFF
10	cloudB04	cloudB	Lenovo	10.128.2.4	LenovoAdmin	LenovoPass	test	OFF	OFF	OFF
11	cloudB05	cloudB	Lenovo	10.128.2.5	LenovoAdmin	LenovoPass	test	OFF	OFF	OFF
12	cloudC01	cloudC	Dell	10.128.3.11	DellAdmin	DellPass	test	ON	ON	ON
13	cloudC02	cloudC	Dell	10.128.3.12	DellAdmin	DellPass	test	ON	ON	ON
14	cloudC03	cloudC	Dell	10.128.3.13	DellAdmin	DellPass	test	ON	ON	ON
15	cloudC04	cloudC	Dell	10.128.3.14	DellAdmin	DellPass	test	ON	ON	ON
16	cloudC05	cloudC	Dell	10.128.3.15	DellAdmin	DellPass	test	ON	ON	ON
17	cloudD1	cloudD	HP	10.128.4.25	HPAdmin	HPPass	test2	ON	ON	ON
18	cloudD2	cloudD	HP	10.128.4.26	HPAdmin	HPPass	test2	ON	ON	ON
19	cloudD3	cloudD	HP	10.128.4.27	HPAdmin	HPPass	test2	OFF	ON	ON
20	cloudD4	cloudD	HP	10.128.4.28	HPAdmin	HPPass	test2	OFF	ON	ON
21	cloudD5	cloudD	HP	10.128.4.29	HPAdmin	HPPass	test2	ON	OFF	ON

Figura 5.6: Impresión por consola de `cems2cli` utilizando el comando "`machine inventory`"

5.2.2.2. Comando `monitoring`

EL comando `monitoring` realiza las llamadas de la API implementadas en el *Monitoring Controller*. Ofrece 6 subcomandos para el control y supervisión del sistema de monitorización:

- **`metrics`:** Permite obtener las últimas métricas recogidas. No requiere de argumentos. Admite una serie de filtros, como son el *hostname* de la máquina (**`-h`** o **`--host`**), su ID (**`-i`** o **`--id`**) o el nombre de la métrica que se quiere obtener (**`-n`** o **`--name`**).
- **`plugins`:** Con este subcomando se obtiene un listado de los *plug-ins* instalados en todo el sistema de monitoreo. Tampoco requiere de argumentos. Permite el filtrado a través del tipo (**`-t`** o **`--type`**), que equivale al sistema de plug-ins con el que interactúa (*collector* o *reporter*), y su estado (**`-s`** o **`--status`**), es decir, si está instalado o cargado.

[9] `Rich`: https://pypi.org/project/rich/

- **`status`:** Permite saber el estado de ejecución del sistema de monitorización de la infraestructura *cloud*. No tiene ni argumentos ni opciones.
- **`stop`:** Permite detener este sistema. Implícitamente también detiene el sistema de control, ya que sin métricas este último no puede operar. No tiene ni argumentos ni opciones.
- **`start`:** Posibilita la acción de reanudar el sistema de monitoreo.
- **`restart`:** Una combinación de los dos anteriores de forma consecutiva.

5.2.2.3. Comando `actions`

Este último comando interactúa con el sistema de control de máquinas mediante de su *controller* de la API. Ofrece también 6 subcomandos:

- **`optimizations`:** Permite obtener los resultados de los dos tipos de optimizaciones en el sistema. El tipo de optimización se selecciona a través de un argumento: VM o PM. Permite una opción para filtrar según el nombre (**`-n`** o **`--name`**) del *plug-in* que realiza dicha optimización.
- **`plugins`:** Con este subcomando se obtiene un listado de los *plug-ins* instalados en todo el sistema de control. Tampoco requiere de argumentos. Permite el filtrado a través del tipo (**`-t`** o **`--type`**), que equivale al sistema de plug-ins con el que interactúa (*vm_optimization*, *vm_connector*, *pm_optimization* o *pm_connector*), y su estado (**`-s`** o **`--status`**).
- **`status`:** Permite saber el estado de ejecución del sistema de control de la infraestructura *cloud*. No tiene ni argumentos ni opciones.
- **`stop`:** Permite detener este sistema. No tiene ni argumentos ni opciones.
- **`start`:** Posibilita la acción de reanudar este sistema.
- **`restart`:** Una combinación de los dos anteriores de forma consecutiva.

A continuación, en la figura 5.7 se muestra el resultado utilizando uno de los comandos desarrollados en este módulo, para obtener el resultado de la optimización por defecto de las máquinas físicas.

```
❯ cems2cli actions optimizations PM
```

CEMS2 Machine Control System PM Optimizations

Optimization Name	Physical Machines ON	Physical Machines OFF
test3	['cloudD1', 'cloudC03', 'cloudA011', 'cloudC04', 'cloudD2', 'cloudC05', 'cloudD4', 'cloudA021']	['cloudA003', 'cloudA002', 'cloudC01', 'cloudA022']
test2	['cloudD1', 'cloudC03', 'cloudA011', 'cloudC04', 'cloudD2', 'cloudC05', 'cloudD4', 'cloudA021']	['cloudA003', 'cloudA002', 'cloudC01', 'cloudA022']
test	['cloudD1', 'cloudC03', 'cloudA011', 'cloudC04', 'cloudD2', 'cloudC05', 'cloudD4', 'cloudA021']	['cloudA003', 'cloudA002', 'cloudC01', 'cloudA022']

```
Waiting for optimization... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100%
```

Figura 5.7: Impresión por consola de `cems2cli` utilizando el comando "`actions optimizations VM`"

CAPÍTULO 6

Evaluación

En este capítulo se describe el proceso de pruebas realizado, una fase fundamental en cualquier proyecto de desarrollo de un producto software. Se centran en evaluar y mejorar la calidad de un producto de software mediante la identificación y corrección de defectos. Estas pruebas se realizan para garantizar que el software cumple con los requisitos, funcione correctamente, sea confiable en su funcionamiento y se ajuste a unos estándares de calidad.

6.1. Pruebas del software

Toda la batería de pruebas se ha desarrollado con la librería *Pytest* y se puede ejecutar a través de la herramienta de pruebas automatizadas Tox[1]. Además, a través de la cual también se ejecutan formateadores de código como *flake8* y *black*.

6.1.1. Pruebas unitarias

Las pruebas unitarias comprueban el funcionamiento aislado de partes del software que se pueden probar independientemente. Normalmente, las pruebas de unidad se realizan con acceso al código fuente y con el soporte de herramientas de depuración.

Se han desarrollado este tipo de pruebas para módulos concretos del proyecto que tienen cierta complejidad lógica como son los *plug-ins* de optimización o la carga de datos a partir del fichero. Las pruebas realizadas emplean en pruebas de caja negra, basadas en la especificación de cada módulo. Para establecer los casos de prueba se han usado técnicas de partición equivalente.

6.1.2. Pruebas de integración

El propósito de las pruebas de integración es verificar la correcta interacción entre los distintos componentes del software. Estás están dirigida por la arquitectura, lo que conlleva a integrar los componentes basándose en los caminos o flujos definidos e identificados en los diagramas, mostrados en el apartado 4.2. Se trata de un modelo de pruebas de flujo continuo, ya que han de realizarse cada vez que se realizan modificaciones o incrementa el número de componentes.

En este caso se han realizado este tipo de pruebas con los componentes que conforman la API, para verificar su integración con el resto de sistemas con los que tiene relación. Posteriormente se han realizado un conjunto de *plug-ins* de prueba, algunos que funcionan de manera correcta y otro que no, para verificar la integración de estos con el mánager asociado. Los sistemas de *plug-ins* para extender las funcionalidades del sistema presentan una gran ventaja, ya que una vez probado que el sistema se integra bien con uno determinado, el resto de *plug-ins* que se implementen a posteriori, también lo estarán, ya que la interfaz de comunicación es común.

6.1.3. Pruebas de sistema

Las pruebas de sistema verifican comportamiento del sistema al completo. Están también orientadas a validar los requisitos no funcionales. En este sistema, es un proceso laborioso y

[1] Tox: https://pypi.org/project/tox/

complejo, ya que parte del comportamiento recae en la lógica que implementan los distintos *plug-ins*, es por ello por lo que no se han realizado de forma íntegra y simplemente se ha evaluado el sistema utilizando los *plug-ins* de prueba explicados en la sección anterior.

6.1.4. Pruebas de aceptación

Este tipo de pruebas verifican si el sistema satisface los requisitos establecidos en la especificación de requisitos de software (ERS). Su desarrollo se basa en la ejecución del sistema e ir replicando las conductas comunes que seguirán los usuarios para uso, las cuales están reflejadas en dichos requerimientos, en este caso a través de las historias de usuario.

En la realización de dichas pruebas, se hizo una demostración del sistema simulado, frente a los miembros del grupo de Computación Avanzada y e-Ciencia del IFCA, para confirmar que el comportamiento del software se ajustaba a lo acordado en la especificación de requisitos de software (ERS), en el inicio del proyecto. Únicamente se ha quedado sin validar los requisitos relacionados con el acceso y autenticación de los usuarios al sistema, debido a que, como se ha comentado en la sección 5.1.4.2, esto se realiza en el propio despliegue del sistema en producción, mediante los sistemas de autenticación que proporcione el servidor web que permita el acceso al sistema de gestión energética.

6.2. Calidad del software

Para analizar la calidad del software se ha utilizado la herramienta *SonarQube* para únicamente evaluar la calidad del código de manera estática. La cobertura total del código no ha sido evaluada ni configurada, ya que tampoco se han realizado pruebas del software completo, por lo que no es relevante medir que porcentaje se ha cubierto respecto del total.

A continuación, en las figuras 6.1 y 6.2, se muestra un resumen proporcionado por la herramienta, para cada uno de los paquetes que conforman el software:

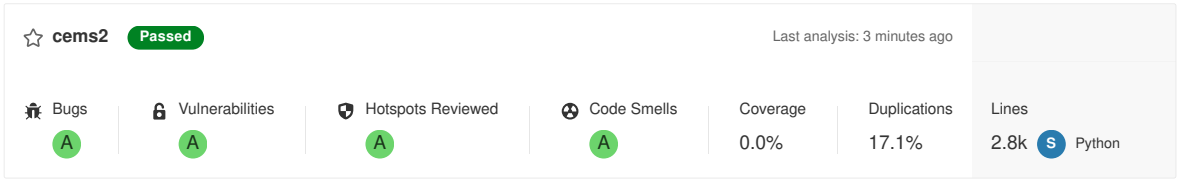


Figura 6.1: Resumen del panel de mandos de la herramienta *SonarQube* [25] para el paquete `cems2`

El código duplicado en el paquete `cems2` se debe a que los *plug-ins* están duplicados para probar que realmente el sistema permite la ejecución múltiple de las extensiones y de forma correcta, ya que devuelven el mismo resultado, pese a la posibilidad de que se ejecuten en instantes de tiempo distintos, por lo que se valida la sincronización entre los *plug-ins* y el mánager.

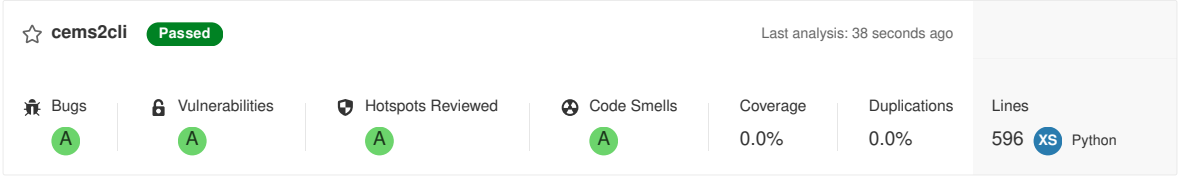


Figura 6.2: Resumen del panel de mandos de la herramienta *SonarQube* [25] para el paquete `cems2cli`

CAPÍTULO 7

Conclusión

En este capítulo se exponen las conclusiones obtenidas una vez terminado el trabajo, comentando los objetivos logrados durante el desarrollo y el trabajo futuro que se presenta a raíz de la realización de este.

7.1. Objetivos conseguidos

A lo largo de la realización de este Trabajo Fin de Grado, se han ido cumpliendo los objetivos propuestos para el trabajo y para el desarrollo del sistema software con las premisas descritas en el apartado 1.2.

En primer lugar, se ha realizado un análisis del entorno de computación donde se enmarcaba el tema del trabajo. Se ha estudiado el modo de operación del *cloud computing*, con el propósito de enfocar el software de gestión energética a dicho paradigma. Además, se ha estudiado diversas fuentes de ineficiencias energéticas presentes en la infraestructura, con soluciones válidas para su posterior implementación.

En segundo lugar, se ha investigado acerca de las tecnologías ya presentes en el entorno de computación donde está propuesto desplegar el sistema, así como las tecnologías empleadas en la implementación del mismo. Del mismo modo, se ha abordado todo el proceso de análisis y especificación de requisitos con los miembros del grupo de Computación Avanzada y e-Ciencia del Instituto de Física de Cantabria para ajustarse a las necesidades propias del centro.

En tercer lugar, se ha realizado una propuesta de diseño arquitectural para el sistema, de forma que cumpla con todos los requisitos marcados. El diseño se ha orientado claramente a una estructura modular con un núcleo, y varios sistemas de *plug-ins* que permiten extender la funcionalidad del sistema. Fruto de ello se refleja en el apartado 7.2, donde explica el trabajo futuro en relación a este trabajo.

En cuarto lugar, se ha implementado el núcleo del sistema de gestión energética (**CEMS2**), compuesto por una RESTful API, un sistema de monitorización de la infraestructura, un sistema de log, un sistema de control de las máquinas y una interfaz para el usuario en forma de línea de comandos. Este núcleo es altamente escalable, debido a su grado de concurrencia dinámica en función de la cantidad de plug-ins cargados, así como claramente compartimentado, por lo que debería ser sencillo de mantener y mejorar. Tanto el sistema de monitorización como el de control de máquinas permiten la flexibilidad necesaria para adaptarlo a diferentes infraestructuras y sistemas, ya que la heterogeneidad es una de las características de este tipo de entornos de computación. Además, a través del sistema de control se permite disponer de varios algoritmos de optimización para realizar distintas políticas de gestión, tanto para la ejecución de consolidaciones de máquinas, como para el apagado y encendido dinámico bajo demanda.

Por último, para poder evaluar el sistema se han implementado un conjunto de *plug-ins* de prueba, que sirven como base a futuros *plug-ins* finales, como los que se indican en la sección 7.2. Además, se ha realizado una fase de evaluación y pruebas del software, la cual puede mejorarse, tal y como se describe en el siguiente apartado.

7.2. Trabajo futuro

Tras finalizar este Trabajo Fin de Grado, queda pendiente de completar los siguientes aspectos:

En primer lugar, es necesario desarrollar una serie de *plug-ins* específicos para la integración del **CEMS2** con toda la infraestructura del centro de procesamiento de datos del Instituto de Física de Cantabria (IFCA). Por un lado, para el sistema de monitorización es necesario que al menos existan tres extensiones, tal y como está planteado el sistema actualmente. Estás son: un recolector de métricas de utilización de máquinas físicas por medio de *OpenStack*, un *plug-in* para la obtención de información sobre las máquinas virtuales (VMs) desplegadas en cada servidor, también a través de *OpenStack*, y por último, una extensión para el reporte de métricas a un *data-source* de *Grafana*. Para está última funcionalidad, también será necesario crear un *data-source*, desplegar en *Grafana* un panel de mandos y realizar la configuración necesaria. Por otro lado, para la integración del sistema de control de máquinas hay que desarrollar dos *plug-ins* enfocados en la conexión con la infraestructura virtual y física. El primero de ellos deberá permitir migrar VMs entre los distintos servidores gestionados por *OpenStack*. El segundo *plug-in* se encargará de la conexión con las máquinas físicas a través de IPMI. Mediante esta plataforma se gestionarán los apagados y encendido de los servidores cuando el sistema de gestión energética lo considere.

En segundo lugar, para que el sistema funcione correctamente, en el sentido de que cumpla su objetivo de aumentar la eficiencia energética del *datacenter*, se necesita disponer de unos *plug-ins* de optimización que implementen algún algoritmo que minimice el uso de recursos. Estos pueden estar basados en técnicas de inteligencia artificial o heurísticas.

En tercer lugar, para realizar el proceso de pruebas de sistema del software completo de una forma exhaustiva, teniendo en cuenta que se trata de un sistema complejo y crítico que modifica la composición de la infraestructura de gran escala sobre la que se ofrecen varios servicios de vital importancia, es necesario desarrollar un simulador, para que, por medio de este, se supervise y evalúe el comportamiento del sistema de gestión energética **CEMS2**. El simulador deberá implementar unos *plug-ins*, coordinados entre sí, de forma que pueda simular un flujo de ejecución coherente y similar al que se obtendría ejecutando el sistema en producción, en el propio CPD.

Por último, se presentan una serie de mejoras para el futuro desarrollo que tenga el sistema. En la versión actual del software para la presentación de este TFG, únicamente se aplican las optimizaciones establecidas por defecto por el administrador, para cada uno de los dos tipos de optimizaciones. En un futuro, se podría extender la implementación para permitir ejecutar varios algoritmos de optimización de forma simultánea y realizar las migraciones o apagados/encendidos de forma dinámica en base al valor de mejora obtenido al aplicar dicha optimización. Adicionalmente, los *plug-ins* que realizan los apagados y encendidos de máquinas físicas deben tener en cuenta la heterogeneidad de la infraestructura para decidir qué máquinas físicas dejar encendidas como baseline y cuáles apagar, seleccionando las que más relación rendimiento/eficiencia posean.

Completando el software y aplicando esta serie de mejoras se obtendría un sistema software modular para la gestión energética de las infraestructuras de computación *cloud*, una herramienta fundamental para optimizar el consumo de energía y promover la sostenibilidad en los centros de procesamiento de datos. Al usar este sistema, las organizaciones pueden obtener beneficios económicos, ambientales y de rendimiento, asegurando un uso más eficiente y responsable de la energía en el ámbito de la computación en la nube.

Referencias

[1] Massoud Pedram. «Energy-Efficient Datacenters». En: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.10 (oct. de 2012), págs. 1465-1484. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2012.2212898. URL: http://ieeexplore.ieee.org/document/6303941/ (visitado 31-05-2023).

[2] Eric Masanet y col. «Recalibrating global data center energy-use estimates». en. En: *Science* 367.6481 (feb. de 2020), págs. 984-986. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aba3758. URL: https://www.science.org/doi/10.1126/science.aba3758 (visitado 18-06-2023).

[3] Anders Andrae y Tomas Edler. «On Global Electricity Usage of Communication Technology: Trends to 2030». en. En: *Challenges* 6.1 (abr. de 2015), págs. 117-157. ISSN: 2078-1547. DOI: 10.3390/challe6010117. URL: http://www.mdpi.com/2078-1547/6/1/117 (visitado 18-06-2023).

[4] Luiz André Barroso, Urs Hölzle y 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 (visitado 31-05-2023).

[5] Kenneth S. Rubin. *Essential Scrum: a practical guide to the most popular agile process*. Upper Saddle River, NJ: Addison-Wesley, 2012. ISBN: 978-0-13-704329-3.

[6] «The Scrum Guide». en. En: *Software in 30 Days*. Ed. por Ken Schwaber y Jeff Sutherland. Hoboken, NJ, USA: John Wiley & Sons, Inc., oct. de 2015, págs. 133-152. ISBN: 978-1-119-20327-8 978-1-118-20666-9. DOI: 10.1002/9781119203278.app2.

[7] Anthony T. Velte, Toby J. Velte y Robert C. Elsenpeter. *Cloud computing: a practical approach*. en. New York: McGraw-Hill, 2010. ISBN: 978-0-07-162695-8.

[8] Peter Mell y Timothy Grance. *The NIST Definition of Cloud Computing*. Inf. téc. 800-145. Gaithersburg, MD: National Institute of Standards y Technology (NIST), sep. de 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[9] Dinesh G. Harkut y col. *Cloud Computing - Technology and Practices*. en. Ene. de 2019. ISBN: 978-1-78984-916-5. DOI: 10.5772/intechopen.72088. URL: https://www.intechopen.com/books/6696 (visitado 27-02-2023).

[10] Rafael Moreno-Vozmediano, Montero e Ignacio M. Llorente. «IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures». En: *Computer* 45.12 (dic. de 2012), págs. 65-72. ISSN: 0018-9162. DOI: 10.1109/MC.2012.76. URL: http://ieeexplore.ieee.org/document/6165242/ (visitado 05-06-2023).

[11] Alvaro López García. *Computing Resources - IFCA Computing - Confluence*. Jun. de 2021. URL: https://confluence.ifca.es/display/IC/Computing+Resources (visitado 19-02-2023).

[12] Miguel Angel Nuñez Vega. *Hardware Resources - IFCA Computing - Confluence*. Mar. de 2023. URL: https://confluence.ifca.es/display/IC/Hardware+Resources (visitado 17-03-2023).

[13] Thomas Krenn. *IPMI Basics - Thomas-Krenn-Wiki*. URL: https://www.thomas-krenn.com/en/wiki/IPMI_Basics#cite_note-ipmispec-1 (visitado 29-05-2023).

[14] Intel Corporation. *Intelligent Platform Management Interface Specification v2.0 rev. 1.1*. en. Oct. de 2013. URL: https://www.intel.com/content/www/es/es/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html (visitado 22-06-2023).

[15] *Intelligent Platform Management Interface*. en. Page Version ID: 1147406833. Mar. de 2023. URL: https://en.wikipedia.org/w/index.php?title=Intelligent_Platform_Management_Interface&oldid=1147406833 (visitado 29-05-2023).

[16] OpenInfra Project Fundation. *OpenStack - Open Source Cloud Computing Platform Software*. URL: https://www.openstack.org/software/ (visitado 26-02-2023).

[17] *Grafana: The open observability platform*. URL: https://grafana.com/ (visitado 01-06-2023).

[18] *Python.org*. en. Mayo de 2023. URL: https://www.python.org/ (visitado 05-06-2023).

[19] Sebastián Ramírez. *FastAPI*. URL: https://fastapi.tiangolo.com/ (visitado 22-03-2023).

[20] *stevedore – Manage Dynamic Plugins for Python Applications — stevedore 5.1.0.dev4 documentation*. URL: https://docs.openstack.org/stevedore/latest/ (visitado 16-04-2023).

[21] *Trio: a friendly Python library for async concurrency and I/O — Trio 0.21.0+dev documentation*. URL: https://trio.readthedocs.io/en/stable/ (visitado 29-05-2023).

[22] *SQLAlchemy*. en. URL: https://www.sqlalchemy.org (visitado 09-06-2023).

[23] Sebastián Ramírez. *Typer*. en. URL: https://typer.tiangolo.com/ (visitado 04-06-2023).

[24] *pytest: helps you write better programs — pytest documentation*. URL: https://docs.pytest.org/en/7.3.x/ (visitado 09-06-2023).

[25] *Code Quality Tool & Secure Analysis with SonarQube*. en. URL: https://www.sonarsource.com/products/sonarqube/ (visitado 09-06-2023).

[26] Alexander Menzinsky, Gertrudis López, Juan Palacio, Miguel Ángel Sobrino,Rubén Álvarez y Verónica Rivas. *Historias de usuario - Ingeniería de requisitos ágil*. Versión 3.01 – Agosto 2022. Historias de usuario 1804036442990. Scrum Manager®, sep. de 2020. URL: https://www.safecreative.org/work/2009135322450-historias-de-usuario-ingenieria-de-requisitos-agil (visitado 29-04-2023).

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

[28] Ron Jeffries. *Essential XP: Card, Conversation, Confirmation*. Ago. de 2001. URL: https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/ (visitado 29-04-2023).

[29] Pierre Bourque y Richard E. Fairley, eds. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Version 3.0. Los Alamitos, CA: IEEE Computer Society, 2014. ISBN: 978-0-7695-5166-1. URL: http://www.swebok.org/.

[30] *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 (visitado 06-03-2023).

[31] Simon Brown. *The C4 model for visualising software architecture*. URL: https://c4model.com/ (visitado 06-02-2023).

[32] Victor Leach y Jacob Shadbolt. *IcePanel | Explain complex software systems*. en. URL: https://icepanel.io/ (visitado 24-03-2023).

ANEXOS

Esta página se ha dejado en blanco intencionadamente.

ANEXO A

Servidores de la infraestructura *cloud*

Servicio de computación *cloud*:

ucloud: 32 máquinas *SuperMicro MicroCloud 5037MC-H8TRF*, con un *Intel® Xeon® CPU E31260L @ 2.40GHz* de 8 cores y 16 GB de memoria RAM.

acloud: 24 máquinas *IBM dx360M4 idataplex*, con 2x *Intel® Xeon® CPU E5-2670 @ 2.60GHz*, con 16 cores por chip (cpc) y 64 GB de RAM.

icloud: 90 máquinas *IBM dx360M4 idataplex*, con 2x *Intel® Xeon® CPU E5-2670 @ 2.60GHz*, con 16 cores por chip (cpc) + *Infiniband FDR* y 64 GB de RAM.

fcloud: 5 máquinas *Lenovo ThinkSystem SR630*, con 2x *Intel® Xeon® Gold 6230 CPU @ 2.10GHz*, con 40 cores por chip (cpc) y 775 GB de RAM.

mcloud: 8 máquinas *DELL*, con un *Intel Xeon Gold 5220R 2.20GHz* de 96 cores y 192 GB de RAM.

aitken: 20 máquinas *IBM system x3850 X5*, con 4x *Intel® Xeon® CPU E7- 4870 @ 2.40GHz*, con 20 cores por chip (cpc) y 1 TB de RAM.

Servicio *AI/Machine Learning*:

gpumad: 20 máquinas *Lenovo ThinkSystem SR650*, con un 2x *Intel® Xeon® Gold 6230 CPU @ 2.10GHz*, con 40 cores por chip (cpc), 2x *NVIDIA® Corporation GV100GL 32Gb*, *Infiniband EDR* y 188 GB de memoria RAM.

gpumax0x: 1 máquina *Supermicro 418-20*, con un *Intel® Xeon® CPU E5-2603 v4 @ 1.70GHz* de 12 cores, 10 GPUs *NVIDIA® 1080Ti* y 192 de RAM.

gpumax1x: 10 máquinas *Supermicro 418*, con un *Intel® Xeon® Gold 5220R CPU @ 2.20GHz* de 96 cores, 8 GPUs *NVIDIA® T4* y 384 GB de RAM.

Servicio *Grid (HTC)*:

cms: 18 máquinas *Fujitsu PRIMERGY BX920 S2*, con 2x *Intel® Xeon® CPU X5670 @ 2.93GHz*, 6 cores por chip (cpc) y 48 GB de memoria RAM.

cmfj: 18 máquinas *Fujitsu PRIMERGY BX924 S4*, con 2x *Intel® Xeon® CPU E5-2697 v2 @ 2.70GHz*, 12 cores por chip (cpc) y 96 GB de RAM.

cmshp: 6 máquinas *HP ProLiant BL460c Gen8*, con 2x *Intel® Xeon® CPU E5-2670 @ 2.60GHz*, 8 cores por chip (cpc) y 128 GB de RAM.

cmsln: 20 máquinas *Lenovo ThinkSystem SD530*, con 2x *Intel® Xeon® Gold 6252 CPU @ 2.10GHz*, 48 cores por chip (cpc) y 192 GB de RAM.

Esta página se ha dejado en blanco intencionadamente.

ANEXO B

Historias de usuario

En este anexo se presentan las historias se usuario utilizadas, ya que describen las necesidades y requerimientos del usuario de manera detallada y comprensible, lo que ayudará a la hora de diseñar y desarrollar el sistema o aplicación de manera efectiva y eficiente. Además, las historias de usuario pueden ser utilizadas como base para la creación de casos de prueba y para la evaluación del éxito del proyecto una vez que se complete.

ID:	US-01
Nombre:	Registrar máquinas físicas en el sistema.
Descripción:	Como administrador quiero poder añadir, modificar, ver y eliminar cada una de las máquinas físicas o conjuntos de estas.
Criterios de aceptación:	1. El sistema deberá utilizar un fichero que establecerá los datos de las máquinas físicas del CPD disponibles. 2. Los datos para cada máquina son: *hostname*, grupo, modelo, dirección IP de gestión, usuario y contraseña de gestión, e interfaz de conexión a cada máquina. 3. Se deberá permitir definir grupos de máquinas físicas indicando los *hostnames* del grupo, al igual que las direcciones IP, en forma de rangos. 4. El sistema deberá validar los datos, por ejemplo, el *hostname* y la dirección IP de gestión deben ser únicos en todo el sistema. 5. Se deberá notificar al administrador los errores en el caso de que los datos no sean válidos, o incoherencias entre el fichero y las máquinas ya registradas en el sistema de forma persistente. 6. El sistema proporcionará el resto de los datos necesarios, como el estado energético actual o la fecha de creación y modificación. 7. Por defecto, toda máquina física añadida al sistema deberá registrarse con la opción de monitorización deshabilitada, hasta que el administrador indique lo contrario. 8. El sistema informa al administrador si la nueva máquina física se ha añadido correctamente.
Comentarios:	■ En el fichero se deberán especificar todos los campos, ya sea de manera individual o por grupos y rango de valores. ■ Cada máquina deberá tener 2 *flags* para indicar si la máquina física tiene que ser monitorizada y si está disponible en el CPD. ■ El estado energético actual de cada máquina se obtendrá a través de *Intelligent Platform Management Interface* (IPMI).

ID:	US-02
Nombre:	Habilitar o deshabilitar la monitorización de las máquinas físicas.
Descripción:	Como administrador quiero habilitar o deshabilitar cada una de las máquinas físicas, para que el sistema no las tenga en cuenta a la hora de realizar la monitorización u optimizar su uso.
Criterios de aceptación:	1. El sistema permitirá mostrar en su interfaz el listado de máquinas para guiar al administrador. 2. El administrador podrá seleccionar las máquinas físicas que desea habilitar o deshabilitar la opción de monitorización. 3. El sistema procederá a realizar la actualización, tanto en la base de datos, como en memoria en tiempo de ejecución.
Comentarios:	Para poder habilitar la monitorizar una máquina o un grupo de máquinas deberá estar habilitada su disponibilidad.

ID:	US-03
Nombre:	Configurar el *baseline* de máquinas de reserva.
Descripción:	Como administrador quiero poder fijar el número de máquinas de reserva (*baseline*) de tal modo que siempre disponga de ese número de máquinas arrancadas para poder satisfacer la demanda a futuros usuarios en el menor tiempo posible.
Criterios de aceptación:	1. El administrador podrá introducir un valor en formato de porcentaje [0, 100] por medio del parámetro definido para ello en el fichero de configuración. 2. El sistema calculará cuantas máquinas deben reservarse en función de la disponibilidad de máquinas actual, y se mostrará en la interfaz. En sistema redondeará al alza siempre que sea necesario. 3. En el próximo reinicio, en el caso de que el valor sea válido, el sistema actualizará su configuración. 4. En el próximo reinicio, en el caso de que el valor sea erróneo, el sistema mostrará un mensaje de error.
Comentarios:	El porcentaje establecido será un valor fijo. Sin embargo, el número real de máquinas con el que el sistema trabaja se modificará en tiempo de ejecución cada vez que el número de máquinas disponibles se varíe.

ID:	US-04
Nombre:	Habilitar o deshabilitar la disponibilidad de las máquinas físicas.
Descripción:	Como administrador quiero habilitar o deshabilitar la disponibilidad cada una de las máquinas físicas, para que no formen parte del sistema de gestión energética.
Criterios de aceptación:	1. El administrador eliminará del fichero de datos las máquinas o grupos de máquinas físicas que desee deshabilitar su disponibilidad. 2. El sistema deberá actualizar los datos al reiniciarse e indicar los cambios realizados al administrador.
Comentarios:	Al deshabilitar la disponibilidad una máquina o un grupo de máquinas también el sistema deberá deshabilitar la opción de monitorización de las mismas.

ID:	US-05
Nombre:	Monitorizar la utilización de las máquinas.
Descripción:	Como administrador quiero poder monitorizar el nivel de utilización de los distintos grupos de máquinas de manera que se pueda supervisar el uso del *cloud* del CPD, y comprobar el propio funcionamiento de este sistema de gestión energética.
Criterios de aceptación:	1. El sistema deberá mostrar en su interfaz el nivel de utilización de cada grupo de máquinas, así como de cada una de ellas. 2. El sistema deberá mostrar en su interfaz el estado de cada máquina física (encendida o apagada).
Comentarios:	Se hará uso de la API de *OpenStack*, para obtener los datos de utilización de las máquinas.

ID:	US-06
Nombre:	Configurar el tiempo de monitorización de recursos.
Descripción:	Como administrador quiero poder definir el periodo de monitorización del sistema, de tal forma que pueda controlar el grado de monitorización del sistema, y por tanto obtener o no una eficiencia energética más precisa.
Criterios de aceptación:	1. El administrador podrá definir un valor a través del fichero de configuración del sistema. 2. En el caso de que el valor sea válido, el sistema actualizará su configuración en el próximo reinicio. 3. En el caso de que el valor sea erróneo, el sistema mostrará un mensaje de error en el próximo reinicio.

ID:	US-07
Nombre:	Apagado dinámico de máquinas.
Descripción:	Como administrador quiero que el sistema de gestión energética apague máquinas dinámicamente para poder obtener un ahorro energético en la infraestructura.
Criterios de aceptación:	Cuando el número de máquinas que no están siendo utilizadas supera el *baseline* definido por el administrador, el sistema apagará dinámicamente un número de máquinas de tal forma que vuelva a haber el mismo número de máquinas vacías, definido como *baseline*.
Comentarios:	Se hará uso de algún protocolo estándar de gestión hardware como *Intelligent Platform Management Interface* (IPMI).

ID:	US-08
Nombre:	Encendido dinámico de máquinas.
Descripción:	Como administrador quiero que el sistema de gestión energética arranque máquinas dinámicamente para poder satisfacer la demanda de los usuarios, en el caso de que esta aumente.
Criterios de aceptación:	Cuando el número de máquinas vacías disminuye del *baseline* marcado por el administrador, el sistema arrancará dinámicamente un número de máquinas de tal forma que se vuelva a alcanzar el mismo número de máquinas en reserva.
Comentarios:	Se hará uso de algún protocolo estándar de gestión hardware como *Intelligent Platform Management Interface* (IPMI).

ID:	US-09
Nombre:	Desactivar el sistema de gestión energética.
Descripción:	Como administrador quiero poder desactivar el sistema de gestión energética de tal modo que se queden encendidas todas las máquinas para labores de mantenimiento y actualización.
Criterios de aceptación:	1. El sistema se desactivará y detendrá la monitorización, así como el resto de las funciones. 2. El sistema deberá encender todas las máquinas registradas en el sistema y notificar al administrador. 3. Al administrador se le dará la opción de reanudar el sistema, volviendo a activar todas las funciones activas previamente, manteniendo la configuración anterior.

ID:	US-10
Nombre:	Configurar el tiempo de operación del sistema (*timeout*).
Descripción:	Como administrador quiero poder fijar el tiempo que debe transcurrir para que el sistema cancele una determinada operación, en el caso de que está no responda correctamente.
Criterios de aceptación:	1. El administrador podrá introducir un valor por medio del fichero de configuración. 2. En el caso de que el valor sea válido, el sistema actualizará su configuración cuando se reinicie. 3. En el caso de que el valor sea erróneo, el sistema mostrará un mensaje de error en el próximo reinicio.

ID:	US-11
Nombre:	Extensión mediante plug-ins.
Descripción:	Como administrador quiero que el sistema sea extensible en un futuro mediante el uso de *plug-ins* de tal modo que se puedan incorporar nuevas funcionalidades.
Criterios de aceptación:	1. El administrador incorpora los módulos necesarios para la correcta ejecución del *plug-in* 2. El sistema cargará de forma dinámica las extensiones indicadas a través del fichero de configuración. 3. El administrador del sistema podrá consultar a través de la interfaz el estado de los *plug-ins* instalados.

ID:	US-12
Nombre:	Extensión: Consolidación de máquinas.
Descripción:	Como administrador quiero que el sistema permita realizar migración de máquinas virtuales entre las máquinas físicas arrancadas en el sistema, de tal modo que así se puedan optimizar totalmente el uso de recursos.
Criterios de aceptación:	1. El administrador podrá definir el algoritmo de consolidación. 2. El sistema generará un esquema de consolidación, a partir de las métricas obtenidas de la monitorización, el cual indicará para cada máquina física, que máquinas virtuales deben ejecutarse para minimizar el número de servidores utilizados. 3. El sistema realizará las migraciones necesarias para alcanzar el estado propuesto por el esquema de consolidación.
Comentarios:	La migración de máquinas virtuales entre las máquinas físicas del sistema se hará utilizando la API de *OpenStack*.

Esta página se ha dejado en blanco intencionadamente.

ANEXO C

Especificación de la REST API

C.1. Documentación de los *endpoints*

Machine Manager

GET `/machines` Get all the registered machines information

Permite obtener todo el inventario de las máquinas registradas en la base de datos. Retorna un listado de máquinas según su *schema* definido en el apartado C.2. El código de estado HTTP de la respuesta de la petición con valor 200 (*OK*). Las máquinas pueden filtrarse por los siguientes parámetros: *groupname*, *brand_name*, *connector*, *energy_status*, *monitoring* y *available*.

GET `/machines/id={id}` Get the machine information identified by its ID

Permite obtener una máquina del inventario, de las registradas en la base de datos, identificadas con su ID. Devuelve una única máquina con los campos definidos en el *schema* `Machine`, y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Eleva la excepción HTTP con el código de error 404 (*Not Found*) si la máquina no existe por el ID indicado en la petición.

PATCH `/machines/id={id}` Set if the machine has to be monitored identified by its ID

Permite actualizar si la máquina tiene que ser supervisada o no en el sistema identificada por su ID. Devuelve la máquina actualizada y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*) Eleva las excepciones HTTP con el código de error 404 (*Not Found*) si la máquina no existe por el ID indicado en la petición y con el código de error 400 (*Bad Request*) si la máquina no está disponible, pero existe.

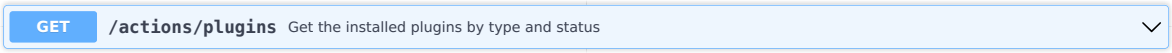
GET `/machines/hostname={hostname}` Get the machine information identified by its hostname

Permite obtener una máquina del inventario, de las registradas en la base de datos, identificadas con su *hostname*. Devuelve una única máquina con los campos definidos en el *schema* `Machine`, y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Eleva la excepción HTTP con el código de error 404 (*Not Found*) si la máquina no existe por el *hostname* indicado en la petición.

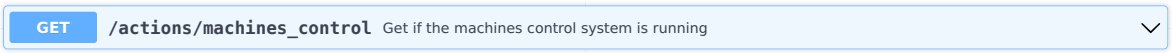
PATCH `/machines/hostname={hostname}` Set if the machines has to be monitored identified by hostname

Permite actualizar si la máquina tiene que ser supervisada o no en el sistema identificada por su *hostname*. Devuelve la máquina actualizada y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Eleva las excepciones HTTP con el código de error 404 (*Not Found*) si la máquina no existe por el *hostname* indicado en la petición y con el código de error 400 (*Bad Request*) si la máquina no está disponible, pero existe.

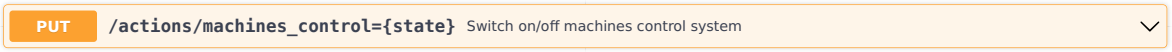
Actions Controller



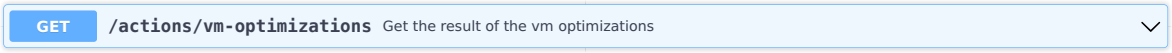
Permite obtener los *plug-ins* instalados en el sistema de control de máquinas. Ofrece las opciones de filtrado por tipo (*vm_optimization*, *pm_optimization*, *vm_connector* y *pm_connector*), así como su estado (*installed*, *loaded* y *default*). Devuelve una lista de *plug-ins* según el respectivo *schema*, y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Eleva la excepción HTTP con código de error 404 (*Not Found*) si no se encuentra ningún plug-in disponible en el sistema. No quita que por las opciones de filtrado establecidas en una petición concreta la lista de plug-ins este vacía.



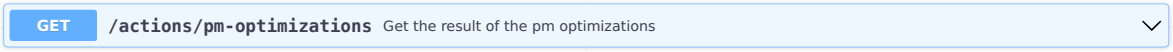
Permite obtener si el sistema de control de las máquinas está funcionando correctamente. Devuelve un valor booleano que representa el estado de ejecución del sistema: TRUE = *Running* | FALSE = *Not Running*. El código de estado HTTP de la respuesta de la petición con valor 200 (*OK*).



Permite modificar el estado de ejecución del sistema de control de máquinas. Devuelve un mensaje con el resultado de la operación, ya que, si se ordena detener y la estaba detenido, se comunica que ya lo estaba, al igual que si ya está en ejecución y se ordena ejecutar. El código de estado HTTP de la respuesta de la petición con valor 200 (*OK*).

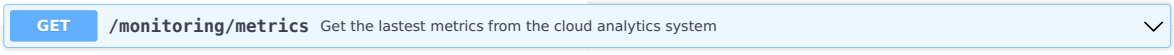


Permite obtener el resultado de las optimizaciones sobre las máquinas virtuales (VMs). Devuelve un diccionario con llave, el nombre del *plug-in* que computa la optimización y valor el propio resultado, en este caso también un diccionario con el par *hostname* de la máquina física y valor una lista de máquinas virtuales, y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Permite filtrar como parámetro de la petición por el nombre del *plug-in*. Lanza una excepción HTTP con código de error 404 (*Not Found*), si no se encuentra el *plug-in* indicado.



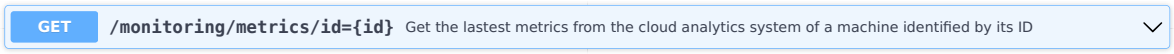
Permite obtener el resultado de las optimizaciones sobre las máquinas físicas (PMs). Devuelve un diccionario con llave, el estado energético de la máquina (ON|OFF) y como valor la lista de *hostnames* de las máquinas, y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Permite filtrar como parámetro de la petición por el nombre del *plug-in*. Lanza una excepción HTTP con código de error 404 (*Not Found*), si no se encuentra el *plug-in* indicado.

Monitoring Controller

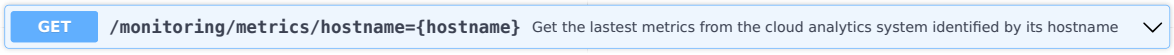


Permite obtener las últimas métricas recogidas para todas las máquinas en estado de monitorización, por el sistema de monitorización de la infraestructura. Retorna un diccionario de métricas con su respectivo *schema*, y con código de estado HTTP de respuesta de la petición

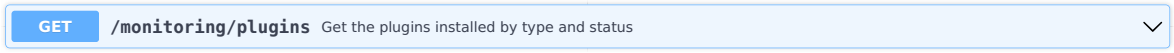
con valor 200 (*OK*). El diccionario está compuesto del *hostname* de la máquina física con llave y como valor una lista de métricas. Permite filtrar por el nombre de la métrica.



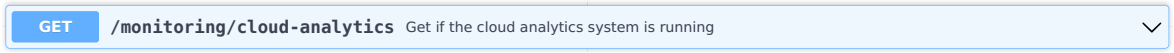
Permite obtener las últimas métricas recogidas por el sistema de monitorización de la infraestructura, para una máquina concreta, identificada por su ID. Retorna una lista de métricas según su *schema*, y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Puede elevar las excepciones HTTP con código de error 404 (*Not Found*) si la máquina indicada por ese ID no está registrada o el código de error 400 (*Bad Request*) si la máquina no está siendo monitorizada.



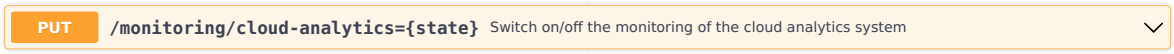
Permite obtener las últimas métricas recogidas por el sistema de monitorización de la infraestructura, para una máquina concreta, identificada por su *hostname*. Retorna una lista de métricas según su *schema*, y con código de estado HTTP de respuesta de la petición con valor 200 (*OK*). Puede elevar las excepciones HTTP con código de error 404 (*Not Found*) si la máquina indicada por ese *hostname* no está registrada o el código de error 400 (*Bad Request*) si la máquina no está siendo monitorizada.



Permite obtener los *plug-ins* instalados en el sistema de monitorización de la infraestructura. Ofrece las opciones de filtrado por tipo (*collector* y *reporter*), así como su estado (*installed* y *loaded*). Devuelve una lista de *plug-ins* según el respectivo *schema*. El código de estado HTTP de la respuesta de la petición con valor 200 (*OK*). Al igual que en el *endpoint* análogo, eleva la excepción HTTP con código de error 404 (*Not Found*) si no se encuentra ningún plug-in disponible en el sistema y no quita que por las opciones de filtrado establecidas en una petición concreta la lista de plug-ins este vacía.



Permite obtener si el sistema de monitorización de la infraestructura está funcionando correctamente. Devuelve un valor booleano que representa el estado de ejecución del sistema: TRUE = *Running* | FALSE = *Not Running*. El código de estado HTTP de la respuesta de la petición con valor 200 (*OK*).



Permite modificar el estado de ejecución del sistema de monitorización de la infraestructura. Devuelve un mensaje con el resultado de la operación, ya que, si se ordena detener y la estaba detenido, se comunica que ya lo estaba, al igual que si ya está en ejecución y se ordena ejecutar. El código de estado HTTP de la respuesta de la petición con valor 200 (*OK*).

C.2. Documentación de los *schemas*

Machine ›

Contiene los siguientes atributos:

- **id:** El ID de la máquina.
- ***groupname*:** El nombre del grupo de la máquina.
- ***hostname***: El nombre de la máquina (único).
- ***brand_name***: El nombre comercial de la máquina.
- ***management_ip***: La dirección IP para gestionar la máquina (única).
- ***management_username***: La dirección IP para gestionar la máquina (única).
- ***management_password***: La contraseña para gestionar la máquina.
- ***connector***: El nombre del *plug-in* de conexión con la máquina.
- ***energy_status***: Verdadero si la máquina está encendida, Falso en caso contrario.
- ***monitoring***: Verdadero si la máquina está siendo monitorizada, Falso en caso contrario.
- ***available***: Verdadero si la máquina está disponible en el sistema, Falso en caso contrario.
- ***created_at***: Fecha y hora de registro de la máquina.
- ***updated_at***: Fecha y hora de actualización de la máquina.

Message ›

Simplemente contiene un atributo donde se introduce el propio contenido del mensaje.

Metric ›

Contiene los siguientes atributos:

- ***name*:** El nombre de la métrica.
- ***payload***: El valor de la métrica en formato de *json*.
- ***hostname***: El nombre de la máquina de donde se han recogido la métrica.
- ***timestamp***: Fecha y hora de recogida la métrica.
- ***collected_by***: Nombre del plug-in usado para recolectar la métrica.

Plugin ›

Contiene los siguientes atributos:

- ***name*:** El nombre del *plug-in*.
- ***type***: El tipo de plug-in (nombre del sistema de *plug-ins* al que pertenece).
- ***status***: El estado del *plug-in* (*installed*, *loaded* o *default*)

ANEXO D

Ficheros de configuración

D.1. Fichero de configuración de máquinas cloud

Se muestran una variedad de casos distintos, puesto que el sistema soporta varios formatos para que sea más fácil para el administrador definir las máquinas disponibles en el sistema.

Código D.1: Fichero de configuración de las máquinas cloud disponibles

```
- groupname: cloudA
  brand_model: IBM Power 8
  hostname_range: [cloudA001, cloudA003]
  management_ip_range: [10.128.1.1, 10.128.1.3]
  management_username: admin
  management_password: admin
  connector: test
  hosts:
      - hostname: cloudA001
        management_username: admin1
        management_password: admin1
      - hostname: cloudA002
        management_username: admin2
        management_password: admin2
        connector: test

- groupname: cloudA
  brand_model: IBM Power 8
  hostname_range: [cloudA011]
  management_ip_range: [10.128.1.11]
  management_username: admin
  management_password: admin
  connector: test

- groupname: cloudB
  brand_model: Lenovo
  hostname_range: [cloudB01, cloudB05]
  management_ip_range: [10.128.2.1, 10.128.2.5]
  management_username: LenovoAdmin
  management_password: LenovoPass
  connector: test
  hosts:
      - hostname: cloudB02
        management_ip: 10.128.2.10

# - groupname: cloudC
  # brand_model: Dell
  # hostname_range: [cloudC01, cloudC05]
  # management_ip_range: [10.128.3.11, 10.128.3.15]
  # management_username: DellAdmin
  # management_password: DellPass
  # connector: test
```

D.2. Fichero de configuración de cems2

El fichero está dividido por secciones, para así tener organizados los distintos parámetros según el ámbito del sistema donde se utilizan.

Código D.2: Fichero de configuración de cems2

```
[data]
file=/etc/cems2/CPD_IFCA.yaml

[log]
handlers=console,file
level=DEBUG
file=/var/log/cems2/CPD_IFCA.log

[database]
name=CPD_IFCA
user=root
pass=dbpassword
host=localhost
port=3306

[cloud_analytics]
interval=10
collector_timeout=10
reporter_timeout=10

[cloud_analytics.plugins]
collectors=test,test2,test_utilization,test_VMs
reporters=test,test2

[machines_control]
baseline=10
pm_connector_timeout=10
vm_connector_timeout=10

[machines_control.plugins]
default_vm_optimization=AI_consolidation
vm_connectors=test,openstack_migration
default_pm_optimization=time_utilization
pm_connectors=test,IPMI

[plugins.X]
; Specific plugin 'X' parameter
```

D.3. Fichero de configuración de cems2cli

Código D.3: Fichero de configuración de cems2cli

```
[API]
URL = http://localhost:8000
```

Acrónimos

API *Application Programming Interfaces*. 9, 10, 14, 25, 30–36, 40–43, 45, 55, 57

ASGI *Asynchronous Server Gateway Interface*. 31

BMC *Baseboard Management Controller*. 8

CEPH *Ceph File System*. 7

CLI *Command-line interface*. 16, 20, 23, 25, 40

CPD Centro de Procesamiento de Datos. 1, 2, 6–10, 12, 18, 23, 27, 28, 31, 46, 53, 55

DCEE *Data Center Energy Efficiency* . 12, 13

ERS Especificación de Requisitos de Software. 19, 44

GPFS *IBM's General Parallel File System*. 7

GPU *Graphics Processing Unit*. 8, 51

HPC *High Performance Computing*. 7

HTC *High Throughput Computing*. 7, 8

IaaS *Infrastructure as a Service*. 6, 7, 9

IFCA Instituto de Física de Cantabria. 4, 7–10, 17, 23, 27, 28, 34, 44–46

IPMI *Intelligent Platform Management Interface*. 8, 14, 18, 46, 53, 56

NIST *National Institute of Standards and Technology*. 5

ORM *Object Relational Mapper*. 15, 31

PaaS *Platform as a Service*. 6

PUE *Power Usage Effectiveness* . 11–13

RES Red Española de Supercomputación. 7

RF Requisitos Funcionales. 19

RNF Requisitos No Funcionales. 20, 43

SaaS *Software as a Service*. 6

SSC Servicio Santander de Supercomputación. 7

TFG Trabajo Fin de Grado. 4, 5, 11, 23, 45, 46

VMs Máquinas Virtuales. 6, 8, 9, 13, 24, 28–30, 35–38, 46, 60

Esta página se ha dejado en blanco intencionadamente.