Ámesos/Amesos

Por:

Mario de los Santos Sainz,
Raúl Fernández Guardia

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Desarrollo de Videojuegos
Facultad de Informática

Tutor: JoséJosé Luís Vázquez Poletti

Madrid, 2019-2020

Índice general

Agradecimientos

No podemos presentar este trabajo sin antes dar las gracias a nuestro director José Luis Vázquez Poletti por mantener una disponibilidad impecable y un entusiasmo asombroso durante todo el proyecto. Ha sido un gran guía capaz de despertar el interés por las redes a dos jóvenes estudiantes.

Resumen

Es indiscutible que el mercado de los videojuegos se han expandido con rapidez en los últimos años, y más concretamente el mercado de los videojuegos en línea.
La tendencia de estos está más que clara: la ejecución de los videojuegos se trasladará a la nube haciendo así que se solucionen problemas de hardware debido a que no será necesario poseer una consola u ordenador. Únicamente los jugadores se conectarán y jugarán con la misma potencia y calidad gracias a estas máquinas remotas.

¿Pero qué ocurre con la ***latencia***? Se tiene que recordar que se esta hablando de juegos en línea, por lo que la velocidad de conexión entre los jugadores es más que fundamental. Trasladar la ejecución a servidores en la nube no implica que estos servidores estén descentralizados haciendo así que algunos jugadores no puedan disfrutar de la experiencia de juego debido al excesivo ***ping***. Para ayudar a resolver este problema se ha desarrollado ***Ámesos***.

Ámesos es una aplicación capaz de desplegar y cerrar servidores de juego bajo demanda en centros de procesamiento de datos más cercanos a los clientes lo que provocará, cuando los clientes se conecten a estos servidores, una disminución de ***ping*** que desembocará en una mejor experiencia de juego para los jugadores.

Palabras clave

- Cliente
- Servidor
- Computación en la nube
- VirtualBox
- Bash
- Frontend
- Backend
- Joomla
- HTML5
- PHP

Summary

It is questionless that videogames market has expanded rapidly in the last few years, and more specifically the online videogames market.
The trend is very evident: videogames execution will be transfered to the cloud, solving hardware problems due to it will not be necessary to have a console or a computer. Players will just connect and play with the same power and quality thanks to these remote machines.

And what about ***latency***? We should remember that we are talking about online videogames, so speed connetivity is a fundamental part. Transfering execution to cloud servers does not involve that those servers are decentralized, meaning that some players cannot enjoy the pure game experience due to an excesive high ***ping***. To help solve this problem we have developed ***Ámesos***.

Ámesos is an app able to deploy videogame servers in data processing centers closer to clients so they can connect to them, providing a decrease of ***ping*** that will lead to a better gaming experience for the players.

Key words

- Client
- Server
- Cloud computing
- VirtualBox
- Bash
- Frontend
- Backend
- Joomla
- HTML5
- PHP

Capítulo 1

Introducción

En este capítulo se repasará brevemente la idea, justificación y motivos por los que se ha creado este proyecto, así como el *planning* pensado para su desarrollo.

1.1. Antecedentes

En la actualidad, las tecnologías en los videojuegos avanzan a un ritmo desmesurado año tras año, sobre todo de los juegos en línea [1]. Es cada vez más exigente en recursos, tecnologías, etc., y los jugadores siempre demandan la mejor experiencia de usuario posible [2]. Muchos juegos son capaces de, cada vez, incorporar y mantener a más jugadores conectados en la misma partida simultáneamente, y esto es un gran reto tecnológico a cumplir si se pretende ofrecer la mejor conectividad [3].

El desafío aumenta cuando sabemos que los jugadores conectados provienen de zonas completamente distanciadas unas de otras, lo cual evidentemente afecta a la latencia de los jugadores. Una alta latencia va a arruinar la experiencia deseada por los jugadores, por lo que es crucial disminuir los efectos negativos que puede producir [4].

Este tema ha despertado un gran interés porque, sin duda, es de los objetivos más fundamentales que los juegos en línea (sobre todo los masivos en línea) deben sobrellevar. Y es así puesto que un gran juego puede ser muy mal valorado si la experiencia ofrecida al jugador provoca que se sienta incómodo, desesperado o frustrado; o incluso piense que recibe una experiencia injusta con respecto a otros jugadores.

1.2. Motivación

La idea del trabajo surge del problema expuesto en el punto anterior, y es que es crucial encontrar métodos que impliquen una mejora en la conexión de los jugadores en los videojuegos en línea.

Es tarea complicada encontrar jugadores que no se hayan frustrado en algún momento por problemas de conectividad a un videojuego. Sobre todo en los más frenéticos que requieren de una respuesta casi instantánea en todo momento, y de manera simultánea para todos los jugadores.

Se sabe que la causa más fácilmente detectable la cual causa alta latencia es la simple y propia distancia. Jugadores que deben estar conectados a un servidor el cual está lo suficientemente alejado de ellos va a provocar un aumento de latencia significativo.
Pero, esto, al mismo tiempo que es fácilmente detectable, conlleva a que sea la principal causa a solucionar.

De este problema nació la idea de ***Ámesos***. Con el objetivo de analizar, gestionar y unir a los jugadores en la medida de lo posible para quitarse de en medio este gran problema de conectividad.

1.3. Objetivos y justificación

Con este proyecto se pretende crear una aplicación destinada a la administración de las conexiones de los clientes de un videojuego , desplegando servidores en centros de procesamientos más cercanos a estos clientes y conectándolos a ellos con el fin de reducir su latencia y que por tanto, disfruten de una mejor calidad de experiencia de juego. Para eso se necesitan lograr una serie de requisitos.

Lo primero es instalar en una máquina física un **hipervisor** para que esta funcione como centro de procesamiento de datos el cual se usará para lanzar maquinas virtuales que funcionen como servidores del juego. Por tanto, hay que encontrar una forma de configurar esas maquinas virtuales para que desplieguen un tipo de servidor específico.

Por otra parte, también se tiene que instalar y configurar una plataforma para poder comunicar esos clientes con esos centros de procesamiento de datos y con las máquinas virtuales que corren dentro de estos.

1.4. Plan de trabajo

A continuación, se detalla el plan de trabajo que se ha seguido para el desarrollo del proyecto.

En cuanto a la plataforma usada para la web, lo primero es buscar entre diferentes *frameworks* y sistemas de gestión de contenidos para desarrollar la comunicación cliente-servidor. Por otra parte, estudiar entre diferentes alternativas para poder crear el nodo físico que aloja los servidores.

Una vez se ha encontrado un sistema de gestión de contenidos y la forma de crear el nodo físico, se puede dividir el trabajo en dos enfoques. El primero es configurar el sistema de gestión de contenidos y la otra es crear maquinas virtuales que alojen los servidores de juego.

Referente a la configuración del sistema de gestión de contenidos, lo primero es inicializar el servidor local y la base de datos donde localizar el contenido de la web. Una vez se tiene esto, hay que estudiar el lenguaje **HTML** para usarlo de apoyo a la hora de desarrollar la aplicación web sobre el sistema de gestión de contenidos. Además, hay que estudiar también el lenguaje de **PHP** que se usa para creación de scripts que aporten las funcionalidades necesarias en el *backend* de la aplicación. Y por último, preparar un script en la aplicación web que permita

conexión vía protocolo *ssh* a una dirección y puertos específicos.

Una vez se tiene toda esta base en el sistema de gestión de contenidos, se pueden hacer modificaciones para mejorarlo como hacer pública la aplicación web de manera que pueda probarse la conectividad desde distintas redes, o crear y ahondar en los scripts de la aplicación para recoger más datos y aportar más funcionalidades. Ya por último se podría modificar el diseño del *frontend* de la web para ajustarse a las especificaciones requeridas y poder mostrar los datos obtenidos.

Y referente al centro de procesamiento de datos, lo primero es escoger la imagen virtual base usada para la creación de las máquinas virtuales, crearlas e instalar el servidor del videojuego en ellas. Una vez se ha escogido, hay que estudiar la **API** del **hipervisor** y aprender el lenguaje de programación **bash** para manipular las maquinas virtuales del nodo físico. Por otra parte, hay que configurar la redirección de los puertos del hipervisor para que las máquinas virtuales puedan ser accesibles desde el exterior. Y por último, hay que configurar la redirección de los puertos del router para poder realizar la comunicación desde redes diferentes.

Capítulo 2

Introduction

This chapter will review briefly the idea, justification and reasons why this project was created, as well as the intended planning for its developement.

2.1. Background

Nowadays, videogames technologies advance at an inordinate rate and year after year, especially, over online games [1]. It is increasingly demanding on resources, technologies, etc., and players always demand the best possible user experience [2]. Many games are capable of incorporating and keeping more players connected in the same game at the same time, and this is a great technological challenge to be met if it is to offer the best connectivity [3].

This challenge increases when we know that connected players come from completely distanced areas from each other, which obviously affects latency of the players. High latency can ruin the desired experience by players, so reducing any negative effects [4] this can trigger is crucial.

This topic has aroused great interest because, without a doubt, it is one of the most fundamental objectives that online games (especially massive online games) must meet. And it is so since a great game can be very poorly valued if the experience offered to the player causes them to feel uncomfortable, desperate or frustrated; or even think that they receive an unfair experience regarding other players.

2.2. Motivation

The idea of this project arises from the exposed problem in the previous point, and it is that it is crucial to find methods that imply an improvement in the connection of the players in online videogames.

It is a difficult task to find players who have not been found frustrated themselves at some point by connectivity problems in any videogame. Especially in the most frantic ones that require an almost instantaneous response from the server all the time, and simultaneously for all players.

It is known that the most easily detectable cause which causes high latency is the simple and proper distance. Players who must be connected to a server which is far enough from them will

cause a significant increase in latency, but this, while easily detectable, leads to it being the main cause to be solved.

From this problem, the idea of Ámesos was born. In order to analyze, manage and unite players as much as possible to get rid of this big connectivity problem.

2.3. Goals and justification

This project aims to create an application for managing the connections of clients in a particular videogame, deploying servers in processing centers the closer they can be to these clients and connecting them to those servers in order to reduce their latency and therefore enjoy of a better quality gaming experience. For this, a series of requirements must be met.

The first thing to do is to install a **hypervisor** on a physical machine so that it works as a data processing center which will be used to launch virtual machines that work as game servers. Therefore, a way has to be found to configure these virtual machines to display a specific type of server.

Besides, a platform to communicate clients with those data processing centers and with the virtual machines that run within them has to be installed and configured.

2.4. Workplan

Next, the work plan that has been followed for the project development is detailed.

Regarding the platform used for the web, the first thing is to search between different *frameworks* and content management systems to develop client-server communication. On the other hand, study between different alternatives to create the physical node that hosts the servers.

Once a content management system alongaside a way to create the physical node have been found, the work can be divided into two approaches. The first is to configure the content management system, while the other is to create virtual machines that host the game servers.

Concerning the configuration of the content management system, the first thing is to initialize the local server and the database where to locate the content of the website. Once this is done, the next thing to do is studying the **HTML** language to use it as a support when developing the web application on the content management system. In addition, it is also needed to study the language of **PHP** that is used to create scripts that provide the necessary functionalities in the *backend* of the application. And finally, prepare a script in the web application that allows connection via protocol *ssh* to a specific address and ports.

Once all this base is created in the content management system, some modifications can be made to improve it such as making the web application public so that connectivity from different networks can be tested, or create and delve into the application scripts to collect more data and provide more functionality. Finally, the design of the *frontend* of the web application could be modified to conform with the required specifications and to be able to display the data obtained.

Regarding the data processing center, the first thing to do is choosing the base virtual image used for the creation of the virtual machines, then create them and finally install the video game server on them. Once the image has been chosen, it is needed to study the **API** of the **hypervisor** and learn the **bash** programming language to manipulate the virtual machines of the physical node. On the other hand, it is necessary to configure the redirection of the ports inside the hypervisor so that the virtual machines can be accessible from the outside. And finally, you must configure the redirection of the router ports to be able to communicate from different networks.

Capítulo 3

Estado del Arte

En este capítulo se presenta el estudio de investigaciones y artículos actuales que están relacionadas en menor o mayor medida con las tecnologías empleadas en el proyecto. Dado que las tecnologías usadas pueden ser muy diferentes unas de otras, Dadose dividirá el capítulos en dos secciones temáticas las cuales serán **Sistemas de Gestión de Contenidos** y **Cloud/Edge/-Computing**.

3.1. Sistemas de Gestión de Contenidos (CMS)

En este apartado se estudia la aparición, evolución y uso de los CMS como herramientas tecnológicas.

El término *Content Management System* (CMS), aunque es cierto que está abierto a más de una definición en función de los distintos puntos de vista, se puede decir que hace referencia a aquel programa informático que permite crear un entorno de trabajo para la creación y administración de contenidos.

Puede dividirse en dos conceptos bien diferenciados: el **contenido** y el **diseño**.
Los CMS proporcionan una interfaz que actúa de capa intermedia para el control de los datos que maneja el sistema por debajo. Es decir, por un lado cuentan con la o las **bases de datos** donde se aloja el contenido y, por otro lado, y de manera independiente, la **interfaz** mostrada y que controla el usuario. [5]

A medida que en el mercado han evolucionado los productos marcados como CMS, también se ha ampliado el panorama con respecto al significado del término.
Y, es que, principalmente se usan estos sistemas sobre páginas web, por lo que han adoptado el término de *Web Content Management* (WCM). Es decir, sistemas de gestión de contenidos Web.

Partiendo de la definición dada al comienzo, la estructuración que tienen estos sistemas permiten mucha flexibilidad puesto que manejar el contenido no depende en absoluto del diseño. Por tanto, modificaciones del diseño no implican darle un formato al contenido de nuevo. Además, la distinción de permisos de edición y gestión del CMS permiten un control por capas ideal para la organización y esquematización del flujo de trabajo.

Como antes se mencionaba, hay distintos puntos de vista sobre los que se puede definir un CMS.

[6] *Bramscher y Butler* (2006) definen a los CMS según el siguiente esquema (fig. 3.1):

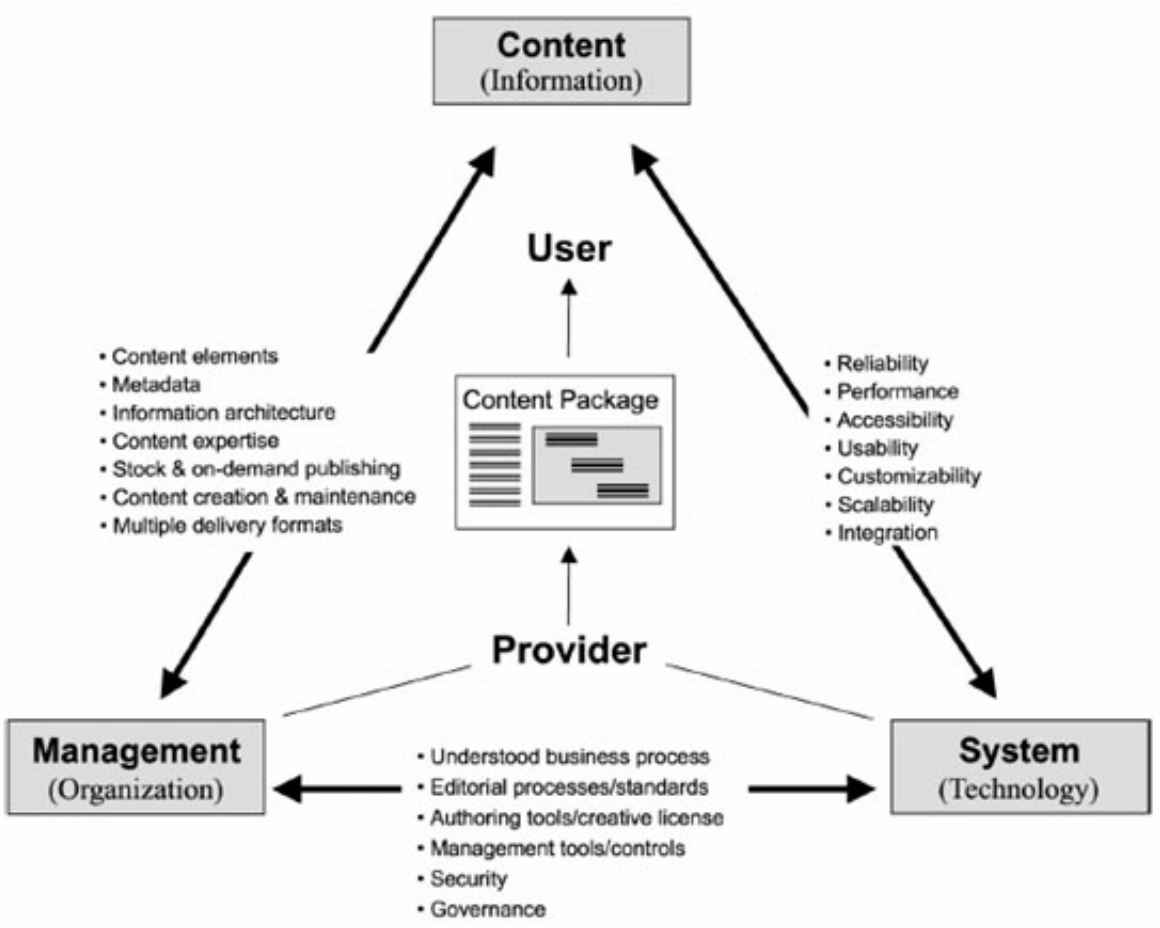


Figura 3.1: Funcionamiento de un CMS según *Bramscher y Butler* (2006)

Por lo que explica el autor, el CMS se define a través de la relación implícita entre tres estructuras: contenido (información), gestión (organización) y sistema (tecnología). Defiende que el CMS debe permitir la posibilidad de personalizar el contenido por personal no técnico, socializar las decisiones relativas a la arquitectura de la web y funciones de edición, y proporcionar los mecanismos para realizar dicho trabajo. Y es esta metodología la clave del éxito de un CMS como tecnología para satisfacer las necesidades de sus usuarios.

Según la definición ofrecida por Kinsta, un CMS es un software que ayuda a los usuarios a crear, administrar y modificar contenido en un sitio web sin la necesidad de conocimientos técnicos especializados.

Figura 3.2: Esquematización de un CMS y sus funcionalidades

Es decir, es una herramienta que ayuda a construir un sitio web sin la necesidad de escribir todo el código desde cero, o incluso sin saber codificar. Su ventaja esencial como herramienta es permitir manejar el diseño (desde una perspectiva visual) de manera independiente a la gestión de los componentes de contenido (el *backend*, como se le conoce).

3.1.1. Historia y evolución de los CMS

A principio de los noventa, el concepto de sistema de gestión de contenidos era desconocido. Las funciones que hoy en día sabemos que estos sistemas proporcionan se realizaban mediante aplicaciones independientes: editores de texto y de imágenes; bases de datos y programación a medida.[7]

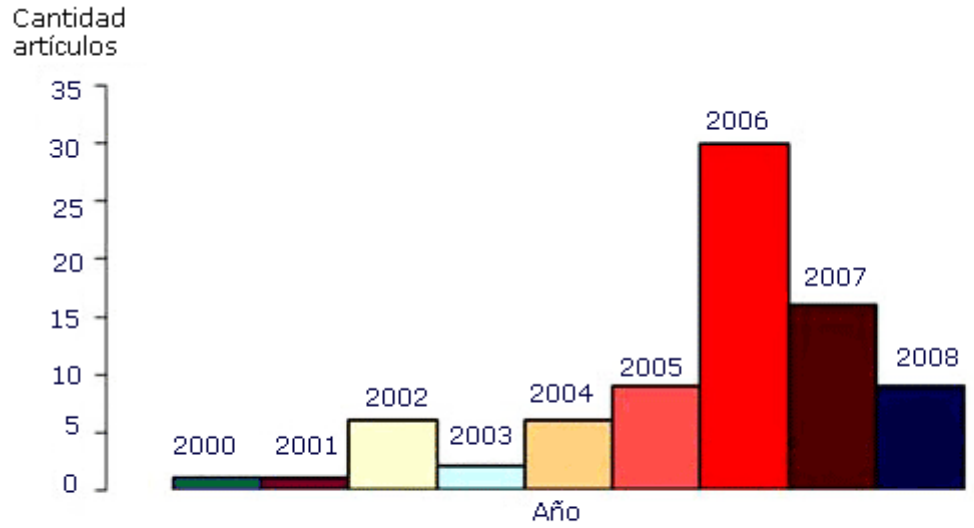
Sobre el año 1994, *Illustra Information Tecnology* comenzó a embarcarse en la idea de los CMS. Utilizaba una base de datos de objetos como repositorio de los contenidos de una web con la idea de poder reutilizar dichos objetos y proveer a sus usuarios un entorno de creación basado en patrones. Pero esta primera idea no llegó a convencer al público.

En el mismo año, *RedDot* fue una de las empresas pioneras que comenzó con el desarrollo de un gestor de contenidos. Al año siguiente presentaron su CMS basado en una base de datos.

Otro de los sistemas de gestión de contenidos con nacimiento en esta década fue *Typo 3*, el cual dio el salto al mercado en 1997, en palabras de su autor, Kasper Skårhøj, «antes de que el término gestión de contenidos fuera conocido sobradamente».

Y es que no fue hasta comienzos del nuevo siglo que los CMS comenzaron a popularizarse a partir de la creación de *PHPNuke*, una herramienta incialmente creada por Francisco Burzi. Nació de la administración de una web de noticias que, debido al crecimiento que tuvo, la necesidad de un sistema más potente le obligó a aprender y reescribir en código php su sitio. [8]

La popularidad de los CMS se hizo notar no solo en aplicaciones prácticas, sino también en el mundo editorial. Por ejemplo, las publicaciones en revistas hospedadas en el *Web of Science*. [9]



Fuente: Rosell Y. Impacto de los Sistemas Gestores de Contenido (CMS) en Centros de Educación Superior de Ciudad de La Habana [Tesis de maestría];(2009). p. 12.

Figura 3.3: Evolución de la frecuencia de publicaciones sobre los CMS

3.1.2. Presente y futuro de los CMS

En la actualidad, aparte de la ampliación de las funcionalidades de los CMS, uno de los campos más interesantes es la creación de estándares que mejoran sobre todo la compatibilidad de componentes, la estabilidad, y facilitan el aprendizaje al cambiar de sistema.

Algunos de estos estándares son CSS [1], un lenguaje de diseño gráfico muy usado para establecer el diseño visual de los documentos web; XML [2], un lenguaje de marcas que permite estructurar un documento; XHTML que es un subconjunto del anterior, pero orientado a la presentación de documentos vía web; los lenguajes PHP, Perl y Python [3]; y varios más.

No solo en los lenguajes, sino que también existen estándares en las aplicaciones que rodean los CMS, como pueden ser los servidores web Apache [4]; y las bases de datos MySQL [5].
La disponibilidad de para los principales sistemas operativos de estas aplicaciones y módulos supone que los CMS puedan funcionar y ser perfectamente compatibles en diversas plataformas sin necesitar muchas modificaciones.
Sobre el futuro de los CMS, Robertson (2003) comenta lo siguiente [10]:
Los CMS se convertirán en artículo de consumo, lo que provocará una disminución de los precios en productos comerciables y una mayor consistencia en las funcionalidades ofrecidas.

[1] CSS: https://www.w3.org/Style/CSS/Overview.en.html
[2] Definición de XML: https://es.wikipedia.org/wiki/Extensible_Markup_Language
[3] Estos lenguajes son muy usados en programación web, y orientados a la creación de scripts
[4] Apache: https://httpd.apache.org/
[5] MySQL: https://www.mysql.com/

Posiblemente, muchas empresas dedicadas a la implementación de webs tendrán que cerrar, y muchos otros proyectos fracasarán por no ajustarse a los estándares.

Otro concepto que va en camino de estandarizarse proviene de la unión de los CMS y el *e-learning* [11].
El e-learning tiene unas necesidades específicas que cubrir que un CMS general no es capaz de ofrecer, o en caso de hacerlo, no da las mismas facilidades que una herramienta creada específicamente para este trabajo.

En general, los sistemas de gestión de aprendizaje (Learning Management System, LMS) facilitan la interacción entre alumnos y profesores, aportando herramientas para la gestión de los contenidos académicos así como su seguimiento por parte de los alumnos. Uno de los más conocidos y usados de código abierto es *Moodle*.

Estos sistemas son diferentes a los sistemas de gestión de contenidos, tanto por el objetivo que busca cada uno como por las características que los componen, pero actualmente comienzan a incluir funcionalidades propias de los CMS. La fusión e integración de ambos conceptos da lugar a uno nuevo: los LCMS (Learning Content Management Systems).

Figura 3.4: Esquematización de un LCMS

Bien es cierto que, a pesar de ser una combinación de ambas herramientas, estos sistemas son nombrados como LMS.

3.2. Cloud/Edge/Fog Computing

En este apartado se investigan proyectos y artículos relacionados con los paradigmas de **Cloud Computing** , **Edge Computing** y **Fog Computing**. Para ello primero debemos definir algunos términos fundamentales para la comprensión del apartado, los cuales son los siguientes:

- **Latencia**
  En redes informáticas de datos, la latencia de red es la suma de retardos temporales dentro de una red. Algunos factores que influyen en la latencia son **la demora en la propagación y transmisión de paquetes dentro de la red , el tamaño de los paquetes transmitidos y el tamaño de los búferes dentro de los equipos de conectividad**

- **Cloud Computing**
  La **computación en la nube** (del inglés cloud computing), conocida también como **servicios en la nube** , **informática en la nube**, **nube de cómputo**, **nube de conceptos** o simplemente **«la nube»**, es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.
  La **computación en la nube** establece su arquitectura a partir de una fragmentación entre **aplicación informática**, **plataforma** y **hardware**, dando como resultado según NIST4 [6] los siguientes métodos de entrega: **software como servicio**, **plataforma como servicio** e **infraestructura como servicio**.

- **Edge Computing**
  El **cómputo de borde**, **edge computing**, es una arquitectura de tecnología de la información (TI) distribuida en la que los datos del cliente se procesan en la periferia de la red, lo más cerca posible de la fuente de origen.

- **Fog Computing**
  Computación de niebla o **Fog computing** puede parecer muy similar a **informática de punta** o **Edge computing** porque ambos implican acercar el procesamiento al lugar donde se recopilan los datos. Pero en computación de niebla, los datos se transmiten desde el punto de recopilación a una puerta de enlace para su procesamiento, luego se envían de vuelta al borde.

Una vez se han comprendido algunos de los términos a los que se harán referencia más adelante, se va a analizar el uso de estas tecnologías en proyectos y artículos diversos, que estarán dividimos de forma temática (**Optimización de sistemas para hacer descender la latencia y uso de arquitecturas cloud para el desarrollo de proyectos**)

3.2.1. Optimización de sistemas para reducir latencia

Existe una gran inmensidad de artículos que tratan el tema de la latencia, muchos de ellos se han dedicado a la creación de sistemas que optimizan recursos para hacer descender la latencia. Dado que **Ámesos** se usa en videojuegos, este apartado se centrará en estos últimos.

Para poder correr videojuegos en la **nube** es necesario que los proveedores de servicios de juegos en la **nube** empleen múltiples centros de datos distribuidos geográficamente para brindar

[6] The NIST Definition of Cloud Computing: https://csrc.nist.gov/publications/detail/sp/800-145/final

sus servicios (fig 3.5). Cuantos más centros de datos se usen, habrá una mayor calidad en la experiencia de juego ya que todos los clientes podrán disfrutar de una buena conexión. Dado que el aprovisionamiento de máquinas virtuales para juegos en la nube multijugador puede llegar a costar mucho dinero a una empresa, hay que encontrar un punto medio entre dos objetivos contradictorios: reducir los costos operativos de la infraestructura y aumentar la calidad de la experiencia del jugador.

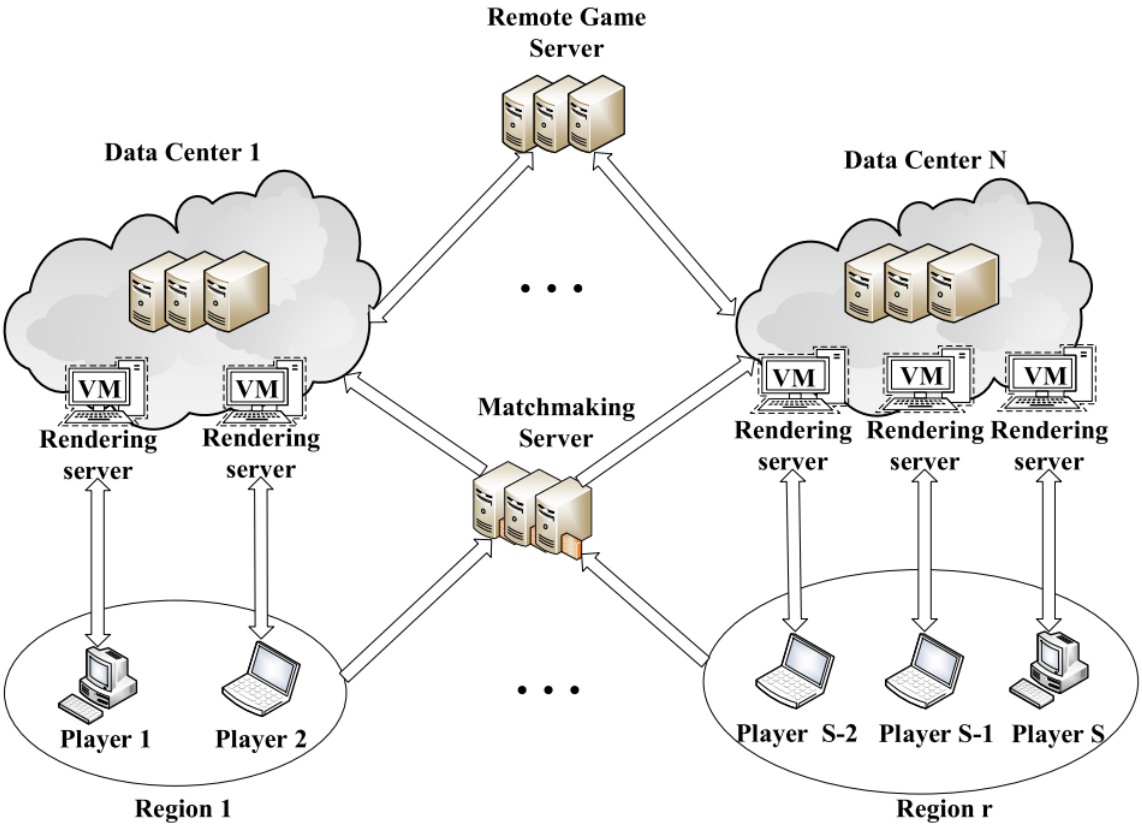


Figura 3.5: Tipica arquitectura Cloud en un videojuego multijugador, fuente:https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8852641

Fue en 2019 cuando Yongqiang Gao, Lin Wang y Jiantao Zhou elaboraron un documento en el cual abordaban este problema [12]. Formularon la cuestión como un problema de optimización multiobjetivo restringido y propusieron un algoritmo de lobo gris mejorado para resolverlo. El rendimiento de ese algoritmo se evalúa mediante experimentos de simulación basados en los parámetros del mundo real. Los resultados mostraron el rendimiento superior del enfoque que se propuso en comparación con los enfoques de vanguardia aplicados a problemas similares.

Ahora bien, está genial que se disponga de un sistema de optimización para calcular un número óptimo de centros de procesamiento de datos a usar, pero cada vez está mas claro que este aprovisionamiento de servicios no puede ser estático en lo referente a videojuegos y mucho menos el los **MMOGs (videojuego multijugador masivo en línea)** ya que la tasa de llegada de jugadores fluctúa continuamente y estos esperan que los servicios estén siempre disponibles con una calidad de servicio (QoS). Es por esto, que Mohammad Sadegh Aslanpour, Mostafa

Ghobaei-Arani, Morteza Heydari y Nader Mahmoudi recomiendan el aprovisionamiento dinámico de servicios proponiendo un enfoque de aprovisionamiento de recursos basado en el aprendizaje para los servicios MMOG que se basa en la combinación del paradigma de computación autónoma y el autómata de aprendizaje (LA) [13].

Ámesos no hace uso de algoritmos que prevean en mayor o menos exactitud cuantos son los servidores necesarios para alojar a todos los clientes en un momento dado. pero Ámesos no se ha quedado atrás en el aspecto de alojar servidores dinámicamente ya que es capaz de ampliar o reducir el número de servidores que se alojan en el centro de procesamiento de datos mediante el uso de un sistema de alertas.

3.2.2. Uso de arquitecturas cloud para el desarrollo de proyectos

Ya se ha hablado del uso del **Cloud/Edge/Fog Computing** en lo referente a la optimización de sistema para disminuir la latencia a los videojuegos que es lo mas revelante para Ámesos pero también se tiene que dar importancia al uso de estas arquitecturas para crear/mejorar proyectos varios que no tienen que ver con videojuegos. Este apartado estudiará algunos de estos ejemplos.

- **Mejora del CGM para los diabéticos**

Los pacientes con diabetes sufren niveles anormales de glucosa en la sangre, lo que puede causar diversos trastornos de salud que afectan los riñones, el corazón y la visión. Debido a estas condiciones, los pacientes con diabetes han controlado tradicionalmente los niveles de glucosa en sangre mediante técnicas de autocontrol de glucosa en sangre (SMBG) [7], como pincharse los dedos varias veces al día. Dichas técnicas implican una serie de inconvenientes que pueden resolverse mediante el uso de un dispositivo llamado Monitor de glucosa continuo (CGM) [8], que puede medir los niveles de glucosa en sangre continuamente durante todo el día sin tener que pinchar al paciente al realizar cada medición.

En 2019 Tiago M. Fernández-Caramés, Iván Froiz-Míguez, Oscar Blanco-Novoa y Paula Fraga-Lamas diseñaron e implementación un sistema que mejora las CGM comerciales al agregarles capacidades de Internet de las cosas (IoT) que les permiten monitorear a los pacientes de forma remota y, por lo tanto, advertirles sobre situaciones potencialmente peligrosas [14].

El sistema que propusieron utiliza teléfonos inteligentes para recopilar valores de glucosa en sangre de los GCM [8] y luego los envía a una **nube remota** o a **nodos de computación de niebla distribuidos**. Además, para intercambiar datos confiables y de seguridad cibernética con científicos médicos, médicos y cuidadores, el sistema incluye el despliegue de un **sistema de almacenamiento descentralizado que recibe, procesa y almacena los datos recopilados**.

- **Detector de caidas para personas mayores**

La vida asistida ambiental es un concepto que utiliza tecnología de información y comunicación para ayudar a la vida diaria de las personas. La detección de caídas en humanos es una

[7] The MANAGING BLOOD GLUCOSE Self Monitoring of Blood Glucose (SMBG): https://www.diabetes.co.uk/blood-glucose/blood-glucose-self-monitoring.html

[8] Continuous glucose monitor(CGM): http://www.endocrino.cat/es/diabetes.cfm/ID/4524/ESP/sensores-monitorizacion-continua-glucosa-cgm-.htm

subárea importante de la vida asistida ambiental. La caída humana ha sido vista como un problema crítico para las personas mayores. La detección de caídas es un enfoque que analiza los datos del sensor (sensores portátiles / sensores ambientales o sensores basados en la visión) para detectar la caída humana utilizando varios algoritmos de aprendizaje.

Este mismo 2020, más concretamente el 16 de Enero, Rashmi Shrivastava y Manju Pandey elaboraron un documento que presenta un método de detección de caídas que detecta y notifica la actividad de caídas en tiempo real utilizando la **computación de niebla** o **fog computing** [15].

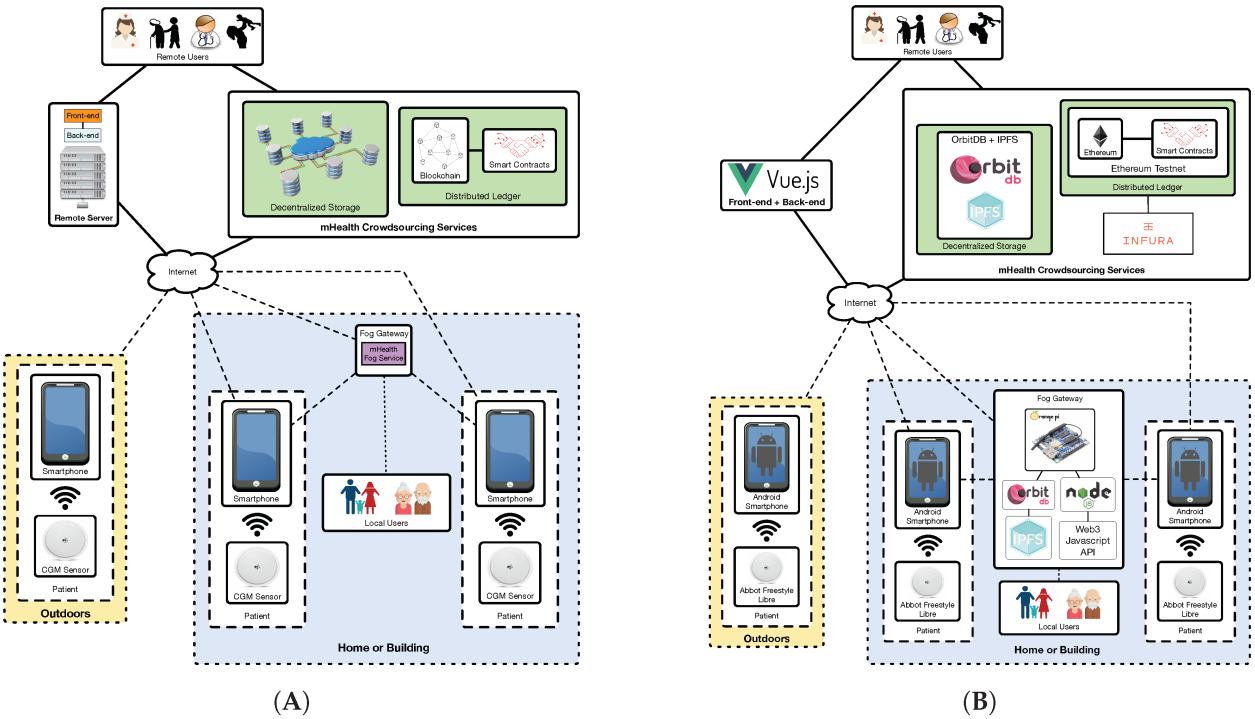


Figura 3.6: Arquitectura de comunicación propuesta (A) y arquitectura implementada (B), fuente:https://www.mdpi.com/1424-8220/19/15/3319

El modelo de detección de caídas esta construido por **SVMs (Support Vector Machines)** [9]. Calcularon cinco características a partir de los datos del acelerómetro de teléfonos inteligentes para construir el modelo de detección de caídas. Para implementar una clasificación de clase, propusieron un nuevo método para el cálculo de la matriz del núcleo. Este modelo de detección de caídas explotó el concepto de **cómputo de niebla** o **fog computing** para enviar notificaciones en tiempo real al cuidador, también es capaz de notificar al cuidador, en ausencia de nodo de niebla, a través del nodo en la nube. El uso del concepto de **computación de niebla** o **fog computing** redujo drásticamente la cantidad de datos transferidos a la nube de 900 valores (10,799 bytes) a 5 valores (59 bytes) cada 6 s.

[9]Support Vector Machine (SVM): https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

Capítulo 4

Arquitectura

En este capítulo se estudiará la arquitectura del proyecto al completo, desde una visión global de este hasta una visión detallada de cada una de las partes. Por tanto, parte de este capítulo se dividirá temáticamente en **centros de procesamiento de datos** y **sistemas de gestión de contenidos**.

4.1. Estructura del proyecto

En esta sección se mostrará y explicará la estructura global del proyecto para su mayor comprensión.

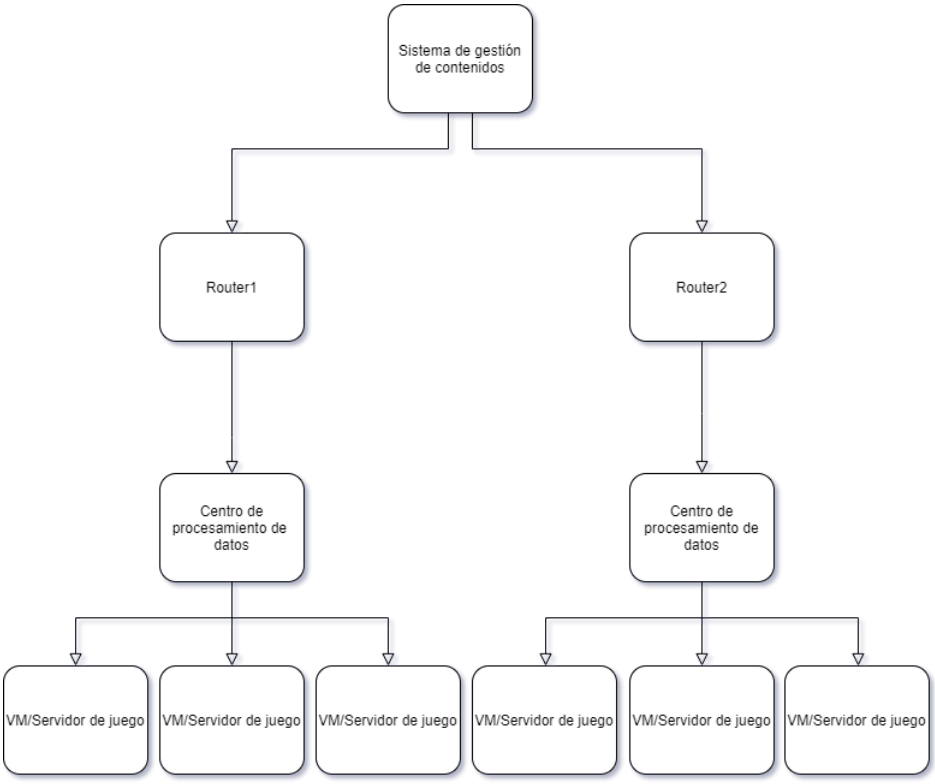


Figura 4.1: Estructura global del proyecto

Como se puede apreciar, se parte de sistema de gestión de contenidos. Este se conecta a la **IP pública** (del router) específica, el router a su vez redirigirá la conexión a un centro de pro-

cesamiento de datos donde se lanzarán máquinas virtuales que funcionarán como servidores de juego. Además, si es necesario, cada centro de procesamiento de datos redirigirá esa conexión a una máquina virtual específica.

Por tanto, es así entonces como se consigue que el sistema de gestión de contenidos (esté donde esté) se conecte con cualquier centro de procesamiento de datos al cual tenga acceso o a cualquier máquina virtual que corre dentro de esos centros.

4.2. Centro de procesamiento de datos

En esta sección se mostrarán todas aquellas partes que componen, modifican y monitorizan los centros de procesamiento de datos usados en el proyecto.

4.2.1. Tecnologías empleadas

En este apartado se mostrarán todas las plataformas y programas referentes a los **centros de procesamiento de datos**.

Antes de nada hay que hablar de donde se alojan estos centros de procesamiento de datos. Todo ellos usan **Linux** [1]como sistema operativo, mas concretamente la distribución de **Ubuntu**[2]

Una vez se sabe donde corren estos centros de procesamiento de datos, hay que centrarse en todo lo referente a las **máquina virtuales** que se usan en el proyecto. Por tanto, lo primero de lo que hay que hablar es del **hipervisor**[3] utilizado. Para este proyecto se ha usado **VirtualBox** [4] ya que es fácil de manejar y su **API**, como veremos más adelante, es muy completa.

4.2.2. Maquina virtual usada y su configuración

En este apartado se verá la maquina base utilizada así como su configuración para poder alojar un servidor de juego y para que se pueda conectar a él.

Lo primero es el tipo de máquina virtual usada. Para este proyecto se ha creado una máquina virtual base con especificaciones predeterminadas, pero se le ha añadido un **VDI**[5](versión 20.04) que se ha descargado desde la pagina oficial de **Ubuntu**[6].

Una vez ya se tiene la máquina virtual, hay que instalarle un servidor de juego. Para este proyecto, tras varias opciones, se ha escogido a **Minetest**[7] como ejemplo de servidor de juego que corre dentro de cada una de las máquinas virtuales.

[1]Linux: https://www.redhat.com/es/topics/linux
[2]Ubuntu: https://es.wikipedia.org/wiki/Ubuntu
[3]Hipervisor: https://www.redhat.com/es/topics/virtualization/what-is-a-hypervisor
[4]VirtualBox:https://www.virtualbox.org/
[5]Virtual Disk Image: https://www.techopedia.com/definition/10933/virtual-disk-image-vdi
[6]Pagina oficial de descargas de ubuntu:https://ubuntu.com/download/desktop
[7]Minetest: https://www.minetest.net/

Figura 4.2: Videojuego utilizado para el proyecto

Ahora hay que configurar el centro de procesamiento de datos para que pueda recibir tramas del exterior y redirigir esas tramas a las máquinas virtuales. Para esto se ha usado la herramienta **iptables**[8]. Hay que recordar que se buscan dos cosas: que los clientes se puedan conectar al servidor que corre dentro de la máquina virtual, y que el sistema de gestión de contenidos se pueda conectar a cada una de las máquinas virtuales que corren en estos centros de procesamiento de datos.

Para la primera parte, debido que a que el servidor de **Minetest** funciona con **UDP**[9], hay que abrir en el centro de procesamiento de datos aquellos puertos dispuestos en el router para que acepten tramas UDP. Para ello se ha ejecutado el siguiente comando:

```
1   $ iptables -A INPUT -p udp -m udp --dport 5001:5034 -j
      ACCEPT
```

Figura 4.3: Regla UDP con iptables para la conexión con el servidor de Minetest

Ahora, para poder realizar la conexión SSH desde el sistema de gestión de contenidos a cada una de las máquinas virtuales (más el propio centro de procesamiento de datos), hay que abrir en este los puertos dispuestos en el router para dicha conexión, aunque esta vez será con **TCP** dado que SSH usa TCP en vez de UDP. Para ello se ha usado el comando que se ve a continuación.

[8]Iptables: https://wiki.archlinux.org/index.php/Iptables
[9]UDP: https://es.wikipedia.org/wiki/Protocolo_de_datagramas_de_usuario

```
$ iptables -A INPUT -p tcp -m tcp --dport 6000:6034 --tcp-
    flags FIN,SYN,RST,ACK SYN -j ACCEPT
```

Figura 4.4: Regla TCP con iptables para la conexión con las máquinas virtuales

Con esto ya se consigue que el reenvío de puerto vaya a funcionar perfectamente ya que se abren los puertos necesarios para ello.

4.3. Sistema de gestión de contenidos

En esta sección se mostrarán, explicarán y analizarán todas las partes que componen el CMS, así como su funcionamiento.

El CMS aportará al cliente el manejo necesario sobre el despliegue de los servidores del juego en cuestión. La estructura del CMS permite mostrar un *frontend* sencillo para el cliente mientras el *backend* realiza todo el trabajo por detrás.
Esta estructura se sitúa en una máquina individual en la cual se encuentra el servidor web, la base de datos asignada, y los *scripts* del *backend* que serán ejecutados localmente.

El funcionamiento aportado al cliente es simple y sencillo. Mediante *clicks* a los botones pertinentes se realizan todas las funciones.

Para empezar, se permite **desplegar un servidor** del juego en cuestión en cualquiera de los **centros de procesamiento de datos** disponibles.

De la misma manera se pueden **apagar dichos servidores**, pero en este caso se tendrán en cuenta los **menos usados** y se mantendrán aquellos que estén siendo utilizados.

Por último, el CMS otorga al cliente la posibilidad de obtener en formato JSON los **datos más relevantes** de todos los servidores ya desplegados para poder conectarse a los mismos.

4.3.1. Tecnologías empleadas

En esta sección se mostrarán y todas las plataformas, lenguajes, librerías y scripts usados en el CMS y por el CMS.

CMS
Como plataforma de Sistema de Gestión de Contenidos se ha usado **Joomla** [10]. Ha servido como base en conjunto con LAMP [11] para la creación de la aplicación web desde la que se maneja Ámesos en sí.

Joomla permite construir de manera sencilla y eficaz los dos grandes bloques que componen la aplicación: la **interfaz** para el usuario así como su diseño, y la **gestión de datos** que se realiza

[10] Joomla: https://www.joomla.org/
[11] LAMP es el acrónimo formado en base a un grupo de tecnologías que combinadas definen la infraestrcutura de un servidor web. **L**inux, **A**pache, **M**ySQL y **P**HP

internamente.

La **estructura** de la web se define por la que ofrece Joomla. Las páginas de la web se nombran como **artículos**, que a su vez están contenidos en **menús**. Siguiendo este modelo, en Ámesos cada página de cada centro de procesamiento de datos se muestra en un artículo individual, y luego estos se agrupan en un menú global que los contiene.

El **diseño** de la web viene, en gran parte, definido por los *templates* de Joomla. Es una característica muy interesante que permite cambiar por completo el diseño gráfico de la interfaz web sin necesidad de cambiar uno a uno los elementos que la componen, ni entrar en el propio código de la web. El template utilizado en este caso es el **J51 - Renovate**[12].

No obstante, Joomla y sus templates permiten añadir o modificar su código con código propio si se requiriera, como es el caso de Ámesos. De esta manera, se ofrece libertad total de diseño, pero usando una base sólida y que permite mantener por separado la interfaz de los datos puros de la aplicación.

La **gestión de datos** que ofrece Joomla se basa en componentes donde el usuario obtiene el control de las bases del CMS mediante variables propias fijas que ofrece Joomla. Sin embargo, y gracias a plugins y componentes creados por la comunidad de Joomla, también se permite crear contenido base mediante el uso de diversos lenguajes de programación web. Este contenido (HTML, PHP, Javascript, etc.) se puede incrustar en los artículos deseados y corren en las propias páginas sin necesidad de crear componentes originales de Joomla.

Otra de las funciones que permite Joomla es crear **módulos**, pequeñas extensiones que se añaden a las páginas selecionadas. La ventaja que tienen es que pueden asignarse directamente a los elementos de un mismo menú o categoría, por lo que funcionan y se muestran igual en todas las páginas fácilmente.

Este modelo es con el que se ha creado y con el que funciona Ámesos para el control, gestión y muestra de los datos y conexiones que funcionan por debajo de la aplicación. El backend está pensado para, con la misma estructura y sin necesidad de repetir código, funcionar de la misma manera en cada página que corresponde a un centro de procesamiento de datos.

Paquetes usados

- **Apache**
  Utilizado para el lanzamiento del servidor web en conjunto al resto de tecnologías de LAMP. Concretamente, para el uso en local de la aplicación web.
- **MySQL**
  Usado como sistema de gestión de la base de datos que forma la aplicación web.

Lenguajes usados

- **HTML**
  Usado como base para la creación de los elementos web, principalmente mediante la ejecución de scripts en las páginas correspondientes.

[12] Template J51 - Renovate: https://www.joomla51.com/joomla-templates/j51-renovate

La idea es simple: los scripts, en función de los datos obtenidos de cada servidor, crean contenedores de elementos HTML ( ) donde se almacenan dichos datos. Luego, estos objetos simplemente son mostrados en la web vía código HTML.

```
<html>
<div>
        <div id="boxParent"></div>
</div>

<!--Scripts code below-->
```

El objeto 'boxParent' almacena como objetos hijos las respectivas cajas creadas por cada servidor.

- **CSS**
  Usado para crear y otorgar un diseño propio a los contenedores de datos de cada servidor. Estos contenedores son cajas donde se muestra la información requerida de cada servidor del centro de procesamiento de datos en cuestión que se está observando.

```
.box {
background: #FFFFF;
border: black 1px solid;
width: 50%;
height: 50%;
cursor: pointer;
display: grid;
text-align: center;
white-space: pre-line;
}

.row {
margin: auto;
display: block;
float:left;
width:50%;
}
```

Gracias a la funcionalidad que ofrecen los templates de Joomla explicadosexplicada anteriormente, crear nuevos objetos .CSS es muy sencillo y no implica modificar el resto de elementos que componen la interfaz ni el código base de la misma. Simplemente se puede añadir nuevo código .CSS fácilmente, y éste se aplica y se fusiona con el ya existente.

- **PHP**
  Usado como lenguaje de programación para el desarrollo web. Concretamente, se ha usado para los scripts que requieren, o bien de comunicación entre máquinas, o ejecución de

comandos/*shell scripts* [13]. Dentro de la aplicación web, es necesaria la conexión a los centros de procesamiento para lanzar servidores u obtener datos de los mismos, y esto se realiza mediante los distintos **botones** de la web.

```php
<?php
function buttonController()
{
    if ($_GET['info']) {

        $salida=shell_exec('./scripts/info/jsonInformativo.sh');
        echo 1;
        return;
       }
       elseif ($_GET['encender']) {
        $salida=shell_exec('./scripts/info/lanzarServidor.sh Centro1');
        echo 1;
        return;
       }
       elseif($_GET['apagar']) {
        $salida=shell_exec('./scripts/info/apagarServidor.sh Centro1');
        echo 1;
        return;
       }

    echo 0;
    return;
}
?>
```

Los botones añaden una variable al enlace de la página, y en función del valor de ésta el script ejecutará un shell script concreto. Las acciones a realizar son **encender** un nuevo servidor, **apagar** el servidor menos usado y **obtener los datos** necesarios para conectarse a los mismos.

```
    {
  "Servidores": [
    {
      "public_ip": "83.37.48.162",
      "port": 5003
    },
    {
      "public_ip": "83.37.48.162",
      "port": 5005
```

[13] Un shell script es un programa designado para ejecutarse en la línea de comandos, con la ventaja de poder automatizar procesos programando los comandos a ejecutar, además de programar cómo y cuándo se lanzan.

```
    }
  ]
}
```

Para obtener los datos del centro y de los servidores lanzados en el mismo se ejecuta un script al acceder a la página correspondiente a cada centro.

```
<?php
function updateData()
{
    shell_exec('./scripts/info/datos.sh Centro1');
    shell_exec('./scripts/info/actualizarTiempoYCoste.sh Centro1');
}
?>
```

Lanza shell scripts que actualizan dichos datos para poder ser recogidos y, seguidamente, ser mostrados en la propia interfaz web.

- **Javascript**
  Usado a su vez junto con HTML y PHP para crear los scripts que forman y controlan la base de la aplicación web.

  El primer script controla si se ha pulsado un botón (y por tanto se ha realizado cualquiera de las acciones ya mencionadas), estos no se ejecuten infinitamente eliminando la variable que es leída del enlace de la página.

```
<script>
async function controller()
{
    var buttonController= <?php buttonController();?>

    if(buttonController == 1)
        window.location = window.location.href.split("?")[0];
}

controller();
</script>
```

Por otra parte está el grueso de la creación de la información de cada página de los centros de procesamiento de datos.

Lo primero que realiza el script es, tanto realizar la llamada para actualizar los datos referentes al centro que está procesando, como seguidamente abrir el archivo que los contiene.

```
<script>
async function readTextFile(file, callback) {
    <?php updateData(); ?>

    var len = 0;
    var rawFile = new XMLHttpRequest();

    rawFile.overrideMimeType("application/json");
    rawFile.open("GET", file, true);


    // Para que no lo almacene en caché
    // y lea el archivo en busca de cambios constantemente
    rawFile.setRequestHeader('Cache-Control', 'no-cache');
```

La información del centro de procesamiento de datos y de los servidores contenidos en el mismo se guardan en un archivo JSON, el cual es *parseado* seguidamente.
Una vez se tiene dicho objeto, se llama a la función que va a construir los elementos HTML de la página junto con los datos leídos.

```
rawFile.onreadystatechange = function() {
        if (rawFile.readyState === 4 && rawFile.status == "200") {

            var data = JSON.parse(rawFile.responseText);
            len = callback(data, len);

        }
    }

    rawFile.send(null);
}
```

Con estos datos construye un objeto HTML, en concreto una caja que contiene la información del servidor. Este proceso se realiza por cada servidor activo y creará tantos de estos elemento como servidores haya en ese momento. Luego, simplemente se mostrará en la página del centro de procesamiento el objeto padre que las contiene.

```
readTextFile("/amesos/scripts/info/datos.json", function(data, len){
var new_len = Object.keys(data.Centres[0].Servers).length;

for (var i = len; i < new_len; i++) {
    var row = document.createElement('div');
    row.className = "row";

    var box = document.createElement('div');
```

```
9       box.className = "box";
10
11      box.textContent += "CPU: " + data.Centres[0].Servers[i].CPU;
12      box.textContent += "Memory: " + data.Centres[0].Servers[i].Mem;
13      box.textContent += "Time up: " + data.Centres[0].Servers[i].TimeUp;
14      box.textContent += "Cost: " + data.Centres[0].Servers[i].Cost;
15
16      row.appendChild(box);
17
18      document.getElementById('boxParent').appendChild(row);
19  }
20
21  return new_len;
22  });
```

Librerías usadas

- **Jumi**
  Jumi [14] es una extensión de Joomla que permite incluir código PHP, HTML, Javascript en los artículos propios de Joomla. En el caso de este proyecto, se ha usado para así poder incluir scripts directamente en el código que compone la página web, y hacer las funciones de backend conrrespondientes sin necesidad de crear componentes propios de Joomla.

4.3.2. Interfaz web

En este apartado se observarán los elementos gráficos que se le mostrarán al cliente y con los que manejará toda la aplicación web.

El frontend de la aplicación es bastante sencillo, así como la navegación y uso de las funcionalidades ofrecidas por la misma.

[14] Librería Jumi: https://extensions.joomla.org/extension/jumi

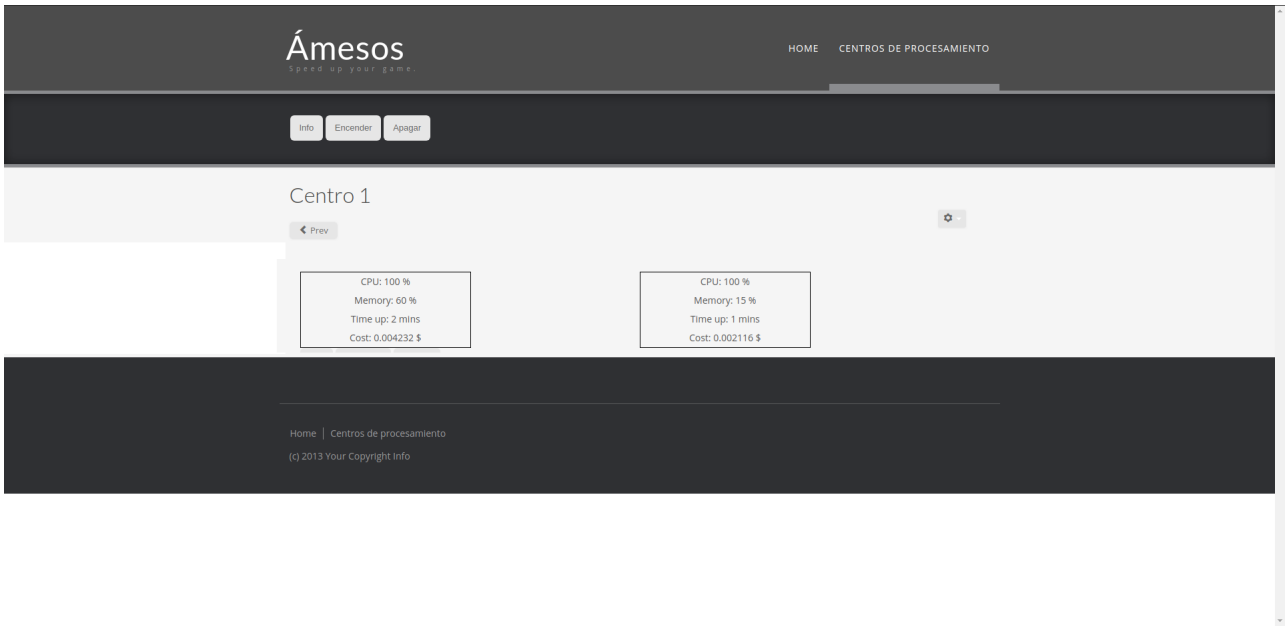


Figura 4.5: Vista general de la interfaz web

Por un lado, se encuentra la primera fila donde a la izquierda se muestra el nombre de la aplicación, y a la derecha los distintos menús por donde la web permite navegar.

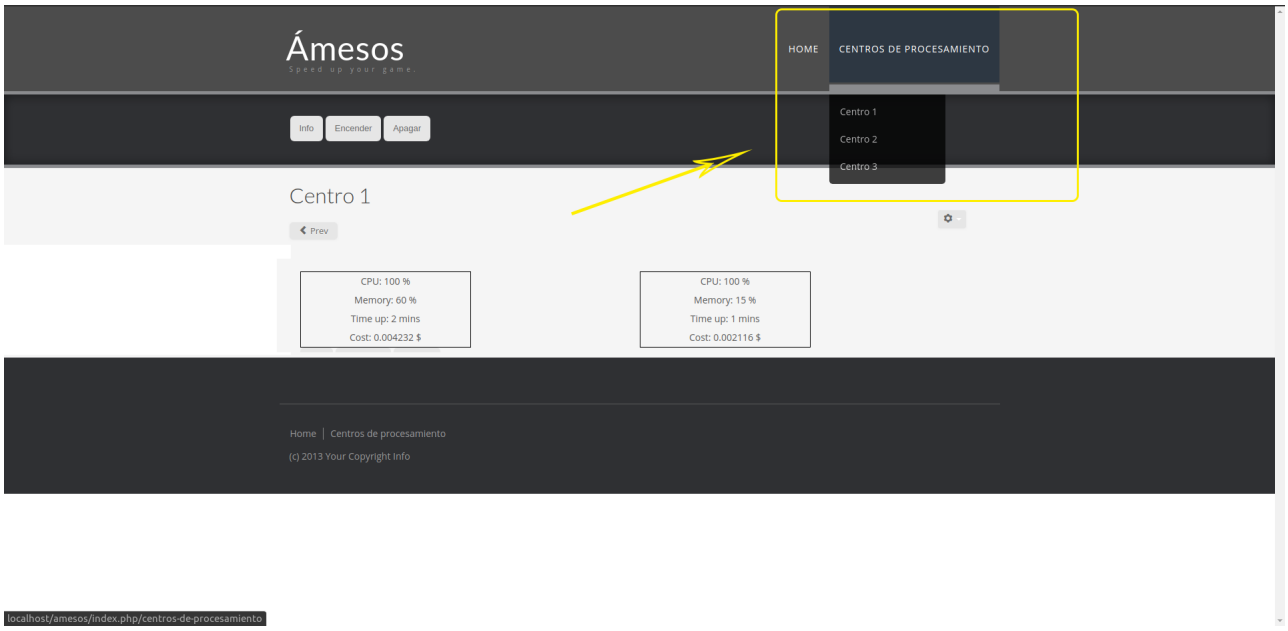


Figura 4.6: Menús desplegables de la interfaz web

Estos menús son desplegables, más concretamente el menú de **Centros de procesamiento de datos**. Es aquí donde el cliente puede escoger a cuál de las páginas navegar fácilmente.

No obstante, por comodidad, la URL para acceder a cada centro de procesamiento sigue una

misma estructura y solo se diferencian entre ellas por el número del centro al que se relaciona. Por tanto, se puede modificar esto en caso de que al cliente le fuera más cómodo.

Figura 4.7: Primer plano de los botones de la interfaz web

En la fila siguiente se encuentran los botones con los que el cliente controla las funciones ofrecidas por la aplicación web.

Están separados del resto de elementos de la interfaz y fijados en una misma posición independientemente de lo mostrado en la página de cada centro. De esta manera su uso sigue siendo igual de intuitivo independientemente del tamaño y número de servidores que contenga la página de cada centro.Está, además, situado en una fila de distinto estilo para resaltar y generar contraste con tal de resultar más intuitivo al cliente.

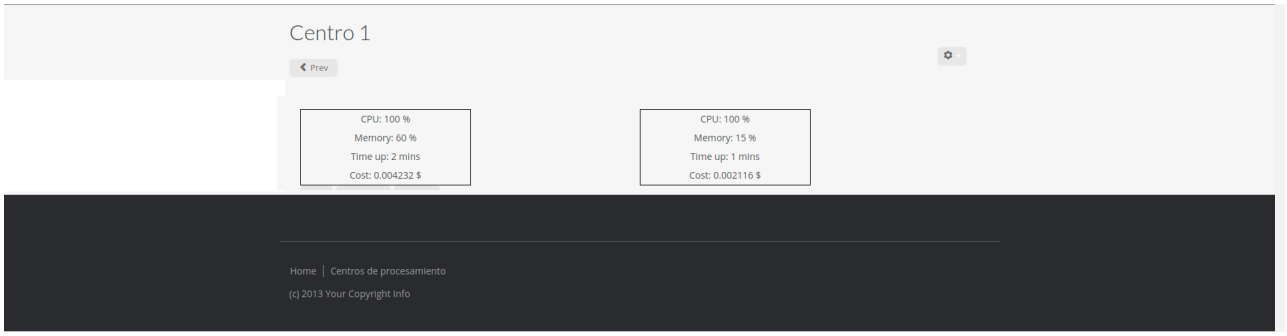


Figura 4.8: Ejemplo del contenido mostrado dentro de la interfaz web

Por último, en el centro de la interfaz es donde aparecerá el contenido como tal de cada centro de procesamiento de datos.

En función de los servidores lanzados en el centro que se está observando, la página crecerá e irá incorporando las cajas correspondientes a cada servidor donde se muestra la información más relevante de los mismos.

Dichas cajas tratarán de colocarse en orden buscando ajustarse al tamaño de la página. Según aparezcan más servidores, la web irá creciendo verticalmente, pero, como ya se menciona anteriormente, sin modificar el resto de la estructura de interfaz web.

4.4. Comunicación del sistema de gestión de contenidos con los centros de procesamiento de datos

En esta sección se hablará de todo lo referente a la hora de comunicar el **sistema de gestión de contenidos** con el cada uno de los **centros de procesamiento de datos**.

4.4.1. Tecnologías empleadas

En este apartado se mostrarán todas las plataformas, programas y scripts que sean referentes a la comunicación del sistema de gestión de contenidos con cada uno de los centros de procesamiento de datos.

Lo primero que hay que conocer es cómo se va a poder conectar el sistema de gestión de contenidos a los centros de procesamiento de datos y a cada una de las máquinas virtuales que se alojan en ellos. Para esto se ha hecho uso de dos técnicas: el uso del **protocolo SSH** [15] y el uso de la herramienta de **reenvío de puertos** en los **routers**(herramienta que existe en la configuración avanzada de practicamente todo los **routers**).

Entonces, como se puede predecir, se usará SSH junto al uso de reenvío de puertos para poder conectar rápidamente el sistema de gestión de contenidos con el centro de procesamiento de datos y con cada una de las máquinas virtuales que corren en ellos.

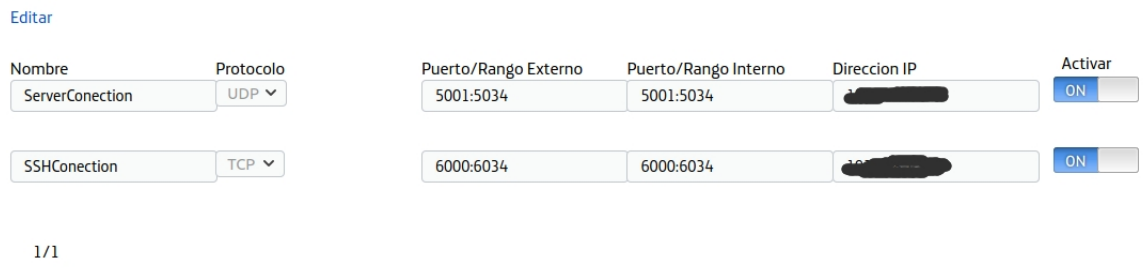


Figura 4.9: Ejemplo de un router configurado para usar el reenvío de puertos

Pero, ¿Cómo sabe el **sistema de gestión de contenidos** a que **IP** y **puerto** hay que conectarse? ¿O cómo sabe cual es el usuario y contraseña del sistema destino?. Como se ve claramente, es necesario contar con una información previa para poder realizar la conexión con éxito. Entonces, ¿cómo se le muestra toda esta información al sistema de gestión de contenidos?

En el caso de este proyecto, se usa un archivo con un formato de texto tipo **JSON** [16] debido a que su forma de estructurar la información encaja perfectamente con el modelo deseado. Este archivo contendrá todo lo que el sistema de gestión de contenidos tiene que saber para poder conectarse a cada uno de los centros de procesamiento de datos, así como a cada una de las máquinas virtuales que se alojan en los mismos. Además, contendrá datos de los todos los centros

[15] SSH (Secure Shell): https://es.wikipedia.org/wiki/Secure_Shell
[16] JSON (JavaScript Object Notation): https://www.json.org/json-es.html

y las máquinas que se usarán posteriormente para mostrar información relevante de estos en la página web.

A continuación, se va a mostrar y explicar por encima el archivo JSON base que contiene todos los datos relevantes para realizar la conexión, así como de dónde se partiría para colocar toda esta **información** sobre cada una de las **máquinas virtuales** de cada uno de los **centros de procesamiento de datos**.

```
{
  "Centres": [
    {
      "public_ip": *.*.*.*,
      "private_ip":*.*.*.*,
      "port": 6000,
      "user": *****,
      "password": *****,
      "name": "Centro1",
      "CPU": 0,
      "NextAvaliablePort": 5001,
      "NextSSHPort": 6001,
      "TimeUp": 0,
      "Mem": 0,
      "UnusedPorts": [
      ],
      "Servers": [
      ]
    }
  ]
}
```

Figura 4.10: Ejemplo de JSON base con un solo centro de procesamiento de datos y ninguna máquina virtual abierta en él.

Ahora se van a explicar brevemente algunos de los campos de ese archivo **JSON**

- **public ip**
  IP pública del centro de procesamiento de datos a la cual se conectará el sistema de gestión de contenidos
- **private ip**
  IP privada del centro de procesamiento de datos. Cuando el sistema de gestión de contenidos se conecte a la IP pública del centro, si se ha conectado al puerto correcto, la conexión será redirigida al la IP privada del centro.
- **port (Siempre es 6000)**
  El puerto 6000 se ha configurado en los routers de los centros de procesamiento de datos

para que redirija cualquier conexión **TCP** [17] a la IP privada de estos centros. Por lo tanto, cuando el sistema de gestión de contenidos se conecte vía **SSH** a la IP pública y puertos dados en el JSON, la conexión se realizará con éxito.

- **user**
  Usuario del sistema destino del centro de procesamiento de datos (necesario para poder realizar la conexión vía SSH).

- **password**
  Contraseña del usuario destino del centro de procesamiento de datos (necesario para poder realizar la conexión vía SSH).

- **CPU**
  Porcentaje de CPU que está usando el centro de procesamiento de datos.

- **NextAvaliablePort**
  Siguiente puerto del centro de procesamiento de datos que se le asignará a la **nueva máquina virtual** lanzada que funcionará para redirigir las conexiones exteriores al servidor que corre dentro de esa nueva máquina virtual. Para este proyecto, el **rango de puertos disponibles** con este fin es de **5001-5034**. Es decir, **que pueden llegar a alojarse 34 servidores de juego en un solo centro de procesamiento de datos**.

- **NextSSHPort**
  Siguiente puerto del centro de procesamiento de datos que se le asignará a la nueva máquina virtual lanzada que funcionará para que sistemas externos puedan conectarse mediante SSH a cada una de las maquinas virtuales, las cuales corren en ese centro de procesamiento de datos. Existirá tantos puertos con el fin conectarse mediante SSH como puertos usados para conectarse al servidor que corre dentro de cada máquina. Por tanto el rango de puertos para SSH será de 6001-6034, siendo así un total de 34.

- **TimeUp**
  Tiempo que lleva el centro de procesamiento de datos activo (en minutos).

- **Mem**
  Porcentaje de memoria RAM que esta usando el centro de procesamiento de datos.

- **UnusedPorts**
  Array de puertos que han quedado disponibles tras destruir alguna máquina virtual que estaba activa.

- **Servers**
  Array de máquinas virtuales que están corriendo en el centro de procesamiento de datos.

Bien, ahora ya se conoce cómo es capaz el sistema de gestión de contenidos de conectarse al centro de procesamiento de datos. Pero ahora llega la pregunta, ¿Cómo es capaz de encender o apagar una máquina virtual en el centro? ¿O cómo es capaz de conocer a que IP y puerto se tiene que conectar alguien para entrar en el juego que corre en esa máquina virtual? ¿ Y cómo monitoriza sus características (CPU, Memoria, Tiempo que lleva levantada, etc.)?

[17] TCP/IP:https://openwebinars.net/blog/que-es-tcpip/

Para todas estas funcionalidades se han creado una gran serie de **scripts**[18] implementados en diferentes lenguajes. El primero de ellos es **Bash**[19] y el otro es **PHP**[20]. Además para poder manejar archivos JSON dentro de los **Shell Scripts** se ha usado **jq**[21] que es un procesador JSON de línea de comandos ligero y flexible.

A continuación se van a listar todos los *scripts* implementados, incidiendo en aquellos que sean relevantes para su explicación.

Primero se verán aquellos que son capaces de **crear y destruir** máquinas virtuales.

- **lanzarServidor.sh**
  Este **script** crea una copia enlazada de una máquina virtual base que contiene un servidor de juego, configura su **reenvío de puertos** y por último la arranca automáticamente. Recibe como argumento el nombre del centro de procesamiento de datos donde quiere lanzar una nueva máquina virtual. Dado que este script es probablemente el más complejo de todos, se analizará mas profundamente.

  Lo primero que hace es buscar en el **JSON** el nombre del centro de procesamiento de datos que coincida con el del argumento del programa, una vez encontrado, tiene que elegir los puertos que se le asignarán a ese centro que funcionarán para redirigir el cliente al servidor que corre dentro de esa máquina virtual y para poder conectar el sistema de gestión de contenidos a esa máquina. La forma de encontrar esos puertos disponibles es la siguiente:

  Primero se mira si existe algún puerto disponible en el *array* de ***UnusedPorts*** mencionado anteriormente. En caso de que no sea así, se usan los valores almacenados en los campos ***"NextAvaliablePort"*** que se usará para el servidor de juego y el campo ***"NextSSHPort"*** que se usará para realizar la conexión **SSH** a la **máquina virtual**. Tras esto, **se actualizarán ambos campos sumándoles +1**, haciendo así que apuntes a los siguientes puertos disponibles.

```
1  #ELEGIMOS EL SIGUIENTE PUERTO
2      puertosDesuso=$(jq --arg var $i
3      '.Centres[$var | tonumber ].UnusedPorts | length  ' $jsonPath)
4      numero=1
5      puertoMinetest=0
6      puertoSSH=0
7
8     #Miramos si exite algun puerto en el array UnusedPorts
9      if (("$puertosDesuso" < "$numero"));
10
11       #Si no es así , asignamos los valores guardados
12       #en NextAvaliablePort y NextSSHPort
```

[18]Scripts:https://es.wikipedia.org/wiki/Script
[19]Bash:https://es.wikipedia.org/wiki/Bash
[20]PHP(Hypertext Preprocessor):https://www.php.net/manual/es/intro-whatis.php
[21]jq: Preprocessor):https://stedolan.github.io/jq/

```
13          puertoMinetest=$(jq .Centres[$i].NextAvaliablePort $jsonPath)
14          puertoSSH=$(jq .Centres[$i].NextSSHPort $jsonPath)
15
16          #Actualizamos el siguiente puerto disponible del servidor de juego
17          NextPortMinetest=$(jq .Centres[$i].NextAvaliablePort $jsonPath)
18          temp=$((NextPortMinetest+1))
19          echo "puerto $temp"
20
21          jq .Centres[$i].NextAvaliablePort=$temp $jsonPath > temp.json
22          cat temp.json > $jsonPath
23          rm temp.json
24
25          #Y por ultimo actualizamos el siguiente puerto
26          #disponible para la conexión SSH
27          NextPortSSH=$(jq .Centres[$i].NextSSHPort $jsonPath)
28          echo $NextPortSSH
29
30          #Le quitamos las comillas para poder sumar
31          temp2=$((NextPortSSH+1))
32
33          jq .Centres[$i].NextSSHPort=$temp2 $jsonPath > temp.json
34          cat temp.json > $jsonPath
35          rm temp.json
```

Pero en caso de que no sea así, el siguiente puerto disponible para conectarse al servidor de juego será el primero encontrado en el *array* de ***UnusedPorts*** mencionado anteriormente , y por suspuesto, se borrará esa entrada del *array*. El siguiente puerto para la conexión **SSH** a esa máquina será el puerto anterior mencionado **+1000** dada la correlación existente entre estos. Ejemplo (5001-6001,5002-6002...etc)

```
1       else
2          puertoMinetest=$(jq .Centres[$i].UnusedPorts[0].number $jsonPath)
3          nombreMaquina=$(jq .Centres[$i].UnusedPorts[0].name $jsonPath)
4          #Borramos el server en desuso
5          indice_servidor=0
6          jq --argjson indice $indice_servidor
7          'del(.Centres[0].UnusedPorts[$indice])' $jsonPath > temp.json
8          cat temp.json > $jsonPath
9          rm temp.json
10         puertoSSH=$((puertoMinetest+1000))
```

Una vez ya se saben los puertos que se van a usar para la configuración, se llama a un *script* tipo **PHP** que se encargará de establecer la conexión con el centro de procesamiento de datos pertinente.

```php
$connection_string = ssh2_connect($public_ip, $port);

if (@ssh2_auth_password($connection_string, $user, $password))
{
        echo "Authentication Successful!\n";
        $stream = ssh2_exec($connection_string,  $(Aqui iria el
           comando o el Script se quiere ejecutar dentro del nodo));

        stream_set_blocking ($stream, true);
        echo stream_get_contents($stream);
}
else
{
        throw new Exception("Authentication failed!");
}
```

Figura 4.11: Ejemplo de archivo PHP que se usa para conectar el sistema de gestión de contenidos con el nodo físico (esta estructura se repite en todos los archivos PHP usados con este fin)

Una vez conectados, se ejecutará en ese centro otro *script* tipo **Bash** llamado **encender.sh** que creará una copia enlazada de la máquina base, la configurará y después, la lanzará automáticamente. Todo esto gracias al uso de **la API de VirtualBox**.

```
#Primero creamos una copia enlazada de la maquina base del servidor de juego
vboxmanage clonevm Minetest-Server  --name=$nombreMaquina
--options=Link --snapshot=Minetest --register

#Ahora añadimos la regla de redireccion de puerto de Minetest
vboxmanage modifyvm "$nombreMaquina" --natpf1
"TFG,udp,$private_ip,$puertoMinetest,10.0.2.15,30000"

#Ahora añadimos la regla de redireccion de puerto de SSH
vboxmanage modifyvm "$nombreMaquina" --natpf1
"SSH,tcp,$private_ip,$puertoSSH,10.0.2.15,22"

#Y una vez este configurada, la iniciamos
vboxmanage startvm $nombreMaquina --type headless
```

- **apagarServidor.sh**
  Este *script* se encarga de apagar y destruir automáticamente una máquina virtual de un centro de procesamiento de datos que recibe como argumento. La estructura es muy parecida al del **lanzarServidor.sh**, pero tiene una peculiaridad. Y es que, apaga y destruye aquella máquina virtual que esté consumiendo menos CPU, para así, ahorrar gastos al cliente.

  Por tanto tiene que escoger aquella máquina virtual que use menos CPU.

```
numero_servidores=$(jq --arg var $i
'.Centres[$var | tonumber ].Servers | length -1' $jsonPath)
indice_servidor=0
CPU_final=$(jq .Centres[$i].Servers[0].CPU $jsonPath)
for j in $( seq 1 $numero_servidores ); do
    #Recorremos todos los servidores buscando aquel que use menos CPU
    nueva_CPU=$(jq .Centres[$i].Servers[$j].CPU $jsonPath)
    if (("$nueva_CPU" < "$CPU_final")) ; then
        CPU_final=$nueva_CPU
        indice_servidor=$j
    fi
    echo $CPU_final
    echo $indice_servidor
done
```

Tras saber qué máquina tiene que destuir, llama a un *script* tipo **PHP** al igual que en **lanzarServidor.sh**. Dentro de este, se establece la conexión con el centro de procesamiento de datos pertinente y se ejecuta otro *script* tipo **Bash** llamado **apagar.sh** que, con ayuda de la **API de Virtualbox**, apaga y destruye la máquina.

```
#!/bin/bash
#Apagamos la maquina cuyo nombre sea igual al del parametro recibido
VBoxManage controlvm $1 poweroff
#La borramos
VBoxManage unregistervm $1 --delete
```

Una vez se ha hablado de aquellos *scripts* que sirven para crear y destruir maquinas virtuales en centros de procesamiento concretos, se va a hablar de todos aquellos *scritps* usados para monitorizar los centros de procesamiento y las máquinas virtuales que corren en ellos.

- **actualizarTiempoYCoste.sh**
  Este *script* calcula cuanto tiempo lleva cada máquina virtual encendida (en minutos) y por tanto, cuanto lleva el cliente pagando por ella. En cuanto al precio por minuto de una máquina virtual, se ha cogido como referencia el coste de una **r5.large de Amazon Web Services**[22] que es de 0,126 dolares/hora lo que es igual 0,0021 dolares/minuto. Además, también calcula el tiempo que lleva cada **centro de procesamiento de datos** activo.

  La estructura es idéntica a los otros *scripts* mencionado anteriormente, la única diferencia es que ahora dentro del archivo **PHP** (el cual se llama **actualizar.php)** en vez de ejecutar otro script dentro de la **maquina virtual**, ahora simplemente se ejecuta un comando. Para calcular el tiempo se ha hecho uso del comando **uptime**[23], y el comando para que devuelta el tiempo en minutos es el siguiente.

[22] Precios de las maquinas virtuales de Amazon Web Services): https://aws.amazon.com/es/ec2/pricing/on-demand/
[23] Uptime: https://es.wikipedia.org/wiki/Uptime

```
uptime | awk -F ',' ' {print $1} ' |
awk ' {print $3} ' | awk -F ':' ' {hrs=$1; min=$2; print hrs + min} '
```

- **datos.sh**
  Este script **calcula el porcentaje de CPU y de memoria RAM** que está usando cada máquina virtual y cada centro de procesamiento de datos. Una vez más la estructura es idéntica a las demás, pero en este caso se invocan dos archivos **PHP** en vez de uno ya que se necesita ejecutar un comando diferente para calcular una u otra cosa. Estos dos ***scripts*** **PHP** se llaman **Mem.php** y **CPU.php**. Para ambos se hace uso de la **herramienta de monitorización vmstat**[24].

  Para calcular el porcentaje de uso de **memoria RAM**, se usa el siguiente comando.

```
vmstat -s | { read a b ; read c d ; echo $((100*c/a)) ; } ;
```

  Y para calcular el porcentaje de uso de **CPU**, se usa este comando.

```
echo $[100-$(vmstat 1 2|tail -1|awk '{print $15}')]
```

Tras la explicación de todos estos *scripts* se va a mostrar como se vería afectado el archivo **JSON** comparado con el que se ha puesto anteriormente. Hay que recordar que ese archivo JSON se utiliza para mostrar la información en el sistema de gestión de contenidos.

[24] Vmstat: https://en.wikipedia.org/wiki/Vmstat

```
{
  "Centres": [
    {
      "public_ip": "*.*.*.*",
      "private_ip": "*.*.*.*",
      "port": 6000,
      "user": *****,
      "password": *****,
      "name": "Centro1",
      "state": true,
      "CPU": 40,
      "NextAvaliablePort": 5003,
      "NextSSHPort": 6003,
      "TimeUp": 110,
      "Mem": 30,
      "UnusedPorts": [
        {
          "name": "Servidor0",
          "port": 5001
        },
      ],
      "Servers": [
        {
          "name": "Servidor1",
          "port": 5002,
          "SSHPort": 6002,
          "user": *****,
          "password": *****,
          "ultimaCPU": 1,
          "ultimaMem": 62,
          "CPU": 60,
          "Mem": 62,
          "TimeUp": 26,
          "Cost": 0.055016
        },
      ]
    }
  ]
}
```


Figura 4.12: Ejemplo de JSON modificado tras ejecutar varias veces todos *scripts* que crean, destruyen y monitorizan maquinas virtuales

4.5. Tecnologías descartadas

Esta sección agrupa todas aquellas tecnologías que se han estudiado y que por distintas razones, se han acabado desechando.

4.5.1. Sistema de gestión de contenidos

Sistema Operativo

En un comienzo, el desarrollo se comenzó usando el S.O. **Windows**. Por comodidad de los sistemas, tecnologías y, en general, recursos en posesión de los desarrolladores, se trató de construir la base del CMS en un sistema Windows.

Al comienzo, esta opción fue la más cómoda y se llegó a crear la base de datos, el servidor web local y una primera versión de la aplicación web utilizando WAMP [25] y Joomla.

Sin embargo, esta opción acabó siendo descartada debido a problemas de incompatibilidad con los scripts que requería la aplicación web. La conectividad entre máquinas requerida en el backend de la aplicación, entre máquinas Linux es mucho más viable y sencillo, sobre todo mediante comandos.

El protocolo SSH usado por Ámesos para conectar con las respectivas máquinas remotas no es viable en un sistema Windows. Además, la interconexión entre los distintos scripts que corren por debajo es mucho más viable usando solamente sistemas Linux.

Lenguajes de programación

Durante el proceso de análisis e investigación del trabajo se observaron diversos lenguajes de programación para hacer a mano la aplicación del panel de control.

- **Python**
  Fue uno de los primeros lenguajes observados debido al framework **Kivy** [26]. Este framework, en principio, nos serviría para realizar la aplicación del panel de control y construir el GUI.

  Aun así, fue uno de nuestros primeros descartes para esta tarea puesto que dicho framework pertence al lenguaje Python, al cual no estamos tan familiarizados como a los demás observados.

- **C++**
  Es el lenguaje al que estamos más familiarizados, pero no era nuestra mejor opción puesto que los frameworks observados para realizar el GUI no nos interesaron tanto como los que podían haber en otros lenguajes.
  Por lo tanto, también lo descartamos rápidmente y nos planteamos opciones mejores.

- **C#**
  Ésta era nuestra opción principal con el objetivo de crear la aplicación del panel de control a base de programación pura.
  Es uno de los lenguajes que mejor controlamos, y al igual que en el lenguaje Python

[25] WampServer: https://sourceforge.net/projects/wampserver/
[26] Framework Kivy: https://kivy.org

estudiamos diversos frameworks de este lenguaje que nos ayudaran a crear y programar el GUI.

Los frameworks observados más interesantes fueron tanto **Xander UI** [27], un framework simple de utilizar para crear una interfaz interactiva; como **Bunifu UI** [28], más orientado a las estadísticas.

Librerías

En este apartado reunimos aquellas librerías que observamos y que, o bien no llegamos a utilizar, o bien reemplazamos o dejamos de usar porque no eran necesarias.

- **Sourcerer**
  Sourcerer [29] es una librería de Joomla que permite añadir código PHP y cualquier estilo de código HTML en los artículos, secciones, componentes, etc., de Joomla.
  Su uso estaba destinado a incluir scripts en código PHP junto con el código HTML de la propia página web del panel de control.

  Se descartó ya que por desgracia no correlacionaba bien con los scripts y comandos que estaban pensados ejecutarse, y se acabó sustituyendo por la librería **Jumi**[30].

- **Quix**
  Quix [31] es una extesión de Joomla que permite crear páginas web con un entorno gráfico sencillo de manejar. Con ella se evita tener que programar la interfaz desde 0, y así crearla de manera más directa y visual.

  Fue descartada debido a que la interfaz acabó siendo creada a mano y mediante scripts. Quix permite también añadir código HTML personalizado, pero al no usar ninguno de los elementos que ofrece la librería porque todos están incluidos a mano, se decidió usar la librería Jumi para crear los módulos correspondientes que forman las páginas web.

4.5.2. Centro de procesamiento de datos

Lo primero de lo que hay que hablar es del sistema operativo, ¿Por qué se usó **Linux** para correr el **centro de procesamiento de datos** y no otro sistema operativo? Ciertamente, la primera opción para este proyecto fue usar **Windows**.

Para **Windows**, la mejor forma de conectarse a las **máquinas virtuales** que corren dentro de **VirtualBox** es mediante la combinación de dos herramientas. La primera es el uso de la herramienta **VirtualBox Remote Desktop Protocol (VRDP)**[32], esto hace que **VirtualBox** sea capaz de convertirse en un servidor RDP[33]. La forma de activar esta herramienta es la siguiente:

[27] Xander UI: https://kivy.org/#home
[28] Bunifu UI: https://bunifuframework.com/
[29] Librería sourcerer: https://extensions.joomla.org/extension/sourcerer/
[30] Jumi: https://extensions.joomla.org/extension/jumi/
[31] Librería Quix: https://www.themexpert.com/quix-pagebuilder
[32] VirtualBox Remote Desktop Protocol (VRDP): https://www.virtualbox.org/manual/ch07.html
[33] Remote Desktop Protocol(RDP): https://es.wikipedia.org/wiki/Remote_Desktop_Protocol

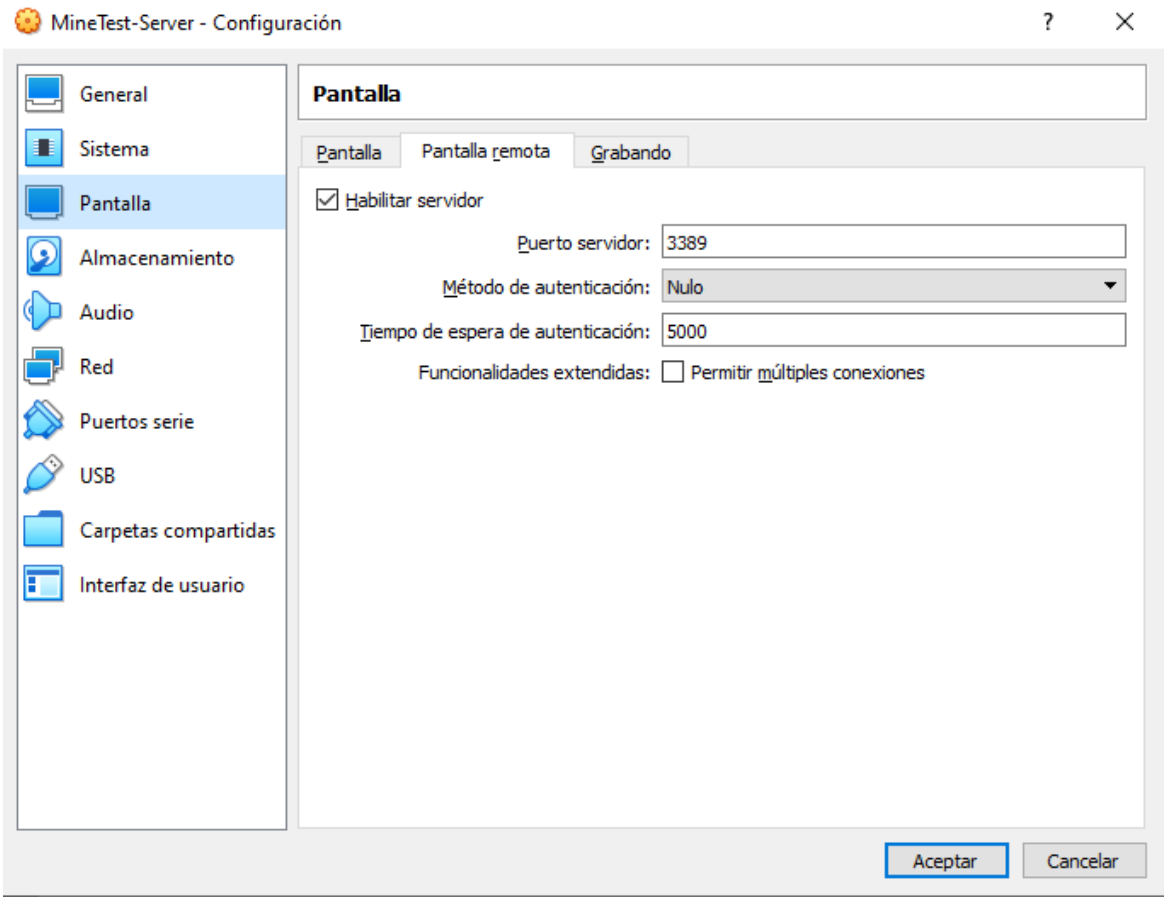


Figura 4.13: Activación de la herramienta VRDP de VirtualBox

Como se ve en la imagen, se puede escoger el **puerto** usado para la conexión con esa **máquina virtual** específica. También hay otros parámetros como el tiempo de autenticación (para que no intente conectarse indefinidamente), el método de autenticación o si quiere que se admita mas de una conexión.

Bien, ahora que ya se tiene a **VirtualBox** corriendo como un servidor **RDP** se puede usar cualquier programa que funcione como **cliente RDP**. Lo más común para **Windows**, es usar una herramienta que viene con el propio sistema operativo que es **mstsc**[34]

Lo normal en el uso de mtstsc es usar la interfaz gráfica para conectase al sistema remoto tal y como se ve en la siguiente imagen.

[34]mstsc: https://docs.microsoft.com/es-es/windows-server/administration/windows-commands/mstsc

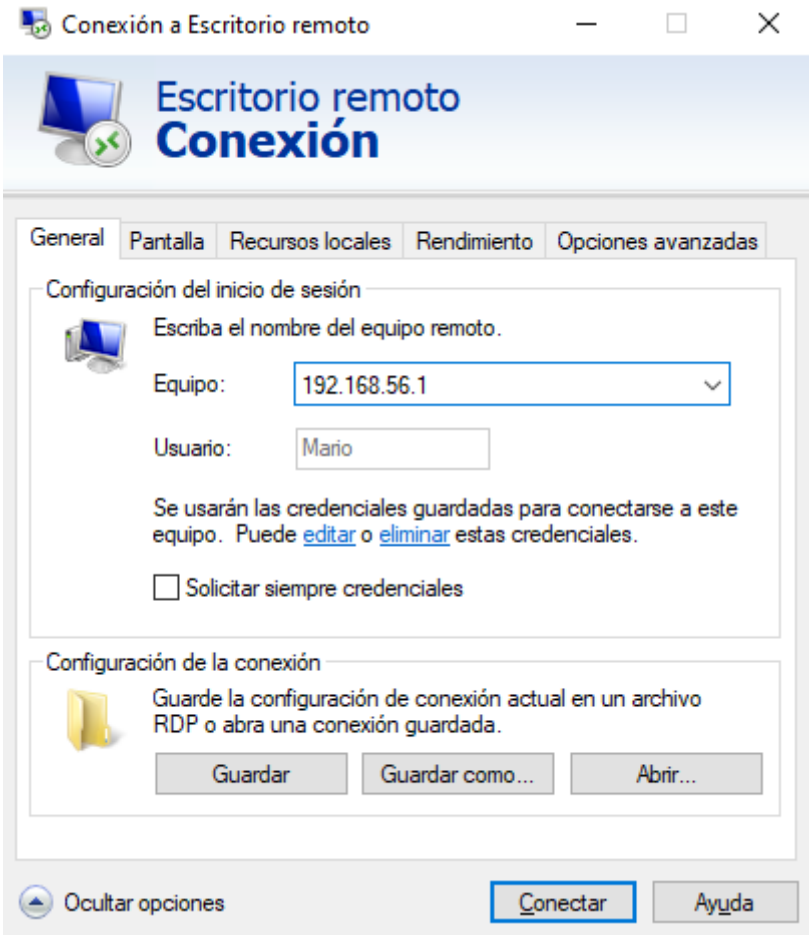


Figura 4.14: Interfaz gráfica de la aplicación ”mtstc”de Windows

Pero también se puede manejar esta conexión mediante el símbolo del sistema. Estos son todos los tipos de argumentos que se le pueden pasar a la aplicación.

- **/v:<server>[:<port>]**
  Especifica el equipo remoto y, opcionalmente, el número de puerto al que desea conectarse.
- **/admin**
  Le conecta a una sesión para administrar el servidor.
- **/f**
  Inicia Conexión a Escritorio remoto en modo de pantalla completa.
- **/w<width>**
  Especifica el ancho de la ventana de Escritorio remoto.
- **/h<height>**
  Especifica el alto de la ventana Conexión a Escritorio remoto.
- **/public**
  Ejecuta Escritorio remoto en modo público. En modo público, las contraseñas y los mapas de bits no se almacenan en caché.

- **/span**
  Coincide con el ancho y el alto de Escritorio remoto con el escritorio virtual local, lo que abarca varios monitores si es necesario.

- **/Edit<connectionfile>**
  Abre el archivo. RDP especificado para su edición.

- **/migrate**
  Migra los archivos de conexión heredados que se crearon con el administrador de conexiones de cliente a los nuevos archivos de conexión. RDP.

- **/?**
  Muestra la Ayuda en el símbolo del sistema.

Como se ve, existen numerosos tipos de argumentos pero ninguno de ellos posibilita la conexión a la **maquina virtual** indicada sin abrir el **GUI**. Además para este proyecto se necesita pasarle a la conexión el **script o comando** que se quiere ejecutar en el sistema remoto. Por todo esto se decidió desechar esta idea del **VRDP Suppor de VirtualBox** y **mtstc** para realizar la conexión entre el sistema de gestión de contenidos y el centro de procesamiento de datos. Y en su cambio se uso el **protocolo SSH**, al principio se pensó en la idea de usarlo con **Putty**[35] pero luego se pensó que sería mejor idea instalar directamente una partición de **Linux** para poder usar el **protocolo SSH** original. Gracias a **SSH** se puede conectar a un sistema remoto únicamente para ejecutar un *script* o comando dentro de él **(ssh usuario@MachineB 'bash -s' <localscript.sh)** o (**ssh -p puerto usuario@ip comando**).

Una vez claro el porque se usó **Linux** para correr los centros de procesamiento de datos, hay que preguntarse ¿Y por qué se ha usado **VirtualBox** como hipervisor?. Se miró **VMware** como posible candidato pero como ya se había usado en el pasado **VirtualBox** y además su **API** es muy completa y funcional, se desecho la idea de usar **VMware**.

Ahora, ya pasando al interior de la **máquinas virtuales** usadas, se miraron por encima varios juegos como **Smokin' Guns**, **FreeCiv** o **Battle for Wesnoth** para que funcionasen como servidor de juego para el proyecto.

Figura 4.15: FreeCiv

Figura 4.16: Smokin' Guns

[35]Putty: https://es.wikipedia.org/wiki/PuTTY

Figura 4.17: Battle for Wesnoth

Pero debido a que ya se había manejado antes el juego de **Minetest**, este fue elegido para funcionar como el servidor de juego del proyecto y por tanto, todos los demás fueron descartados.

Por otra parte, se vieron varias herramientas para monitorizar tanto los centros de procesamiento de datos como las máquinas virtuales que corren en ellos.

- **top**
  top[36] da información acerca del uso de la **CPU**, de la **memoria RAM**, de los **procesos en ejecución**...etc en tiempo real. Y fue por este motivo, porque es en tiempo real, que se desechó la idea de esta herramienta ya que se buscaba una cuyo *output* fuera puntual y no cada cierto tiempo. También se pensó en usar esta herramienta para mirar el tiempo que llevaba una máquina encendida.

- **iostat y mpstat**
  Tanto **iostat**[37] como **mpstat**[38] dan el uso de la **CPU** o **memoria RAM** pero debido a que no se encontró el comando en específico que diese los valores que se estaban buscando, se desechó la idea de usarlas.

- **w**
  Se miró el usar **w**[39] para devolver el tiempo que llevaba una máquina encendida, pero debido a que no se encotró el comando específico para devolver ese tiempo en minutos, se decidió no usarla.

[36]top: https://geekytheory.com/funcionamiento-del-comando-top-en-linux
[37]iostat: https://docs.oracle.com/cd/E24842_01/html/E23086/spmonitor-4.html
[38]mpstat: https://linux.die.net/man/1/mpstat
[39]w: https://francisconi.org/linux/comandos/w

Capítulo 5

Validación del proyecto

En este capítulo se abordará un caso práctico paso a paso para la mayor comprensión del proyecto, viendo como se modifican el **sistema de gestión de contenidos** y los **centros de procesamiento de datos** y como se puede un cliente conectar a los servidores que se van creando. **Por seguridad se han tapado la IP publica del CPD así como su usuario y contraseña**.

Lo primero es mostrar el estado inicial de cada una de las partes, el sistema de gestión de contenidos y el del centro de procesamiento de datos.

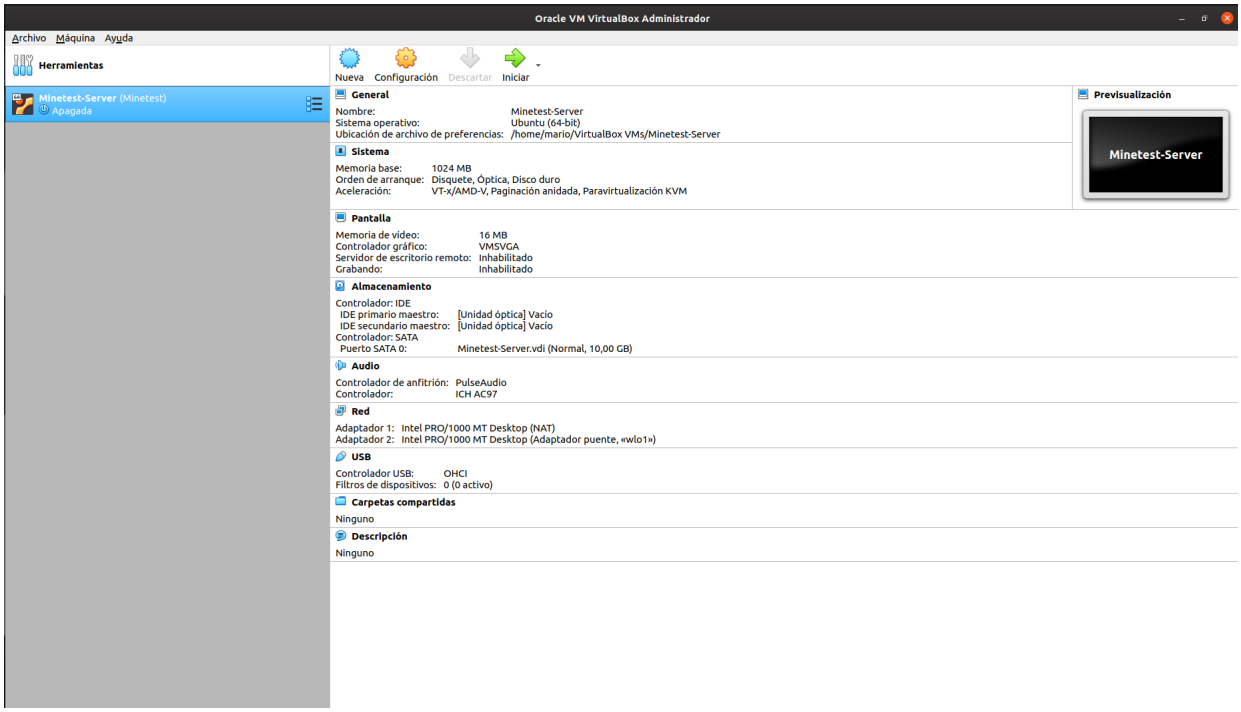


Figura 5.1: Centro de procesamiento de datos vacío

Como se puede apreciar, se parte del centro de procesamiento de datos totalmente vacío, únicamente contiene la **máquina virtual base** de las que partirán todas las demás. Como se dijo anteriormente, el servidor de juego utilizado es el de **Minetest**.

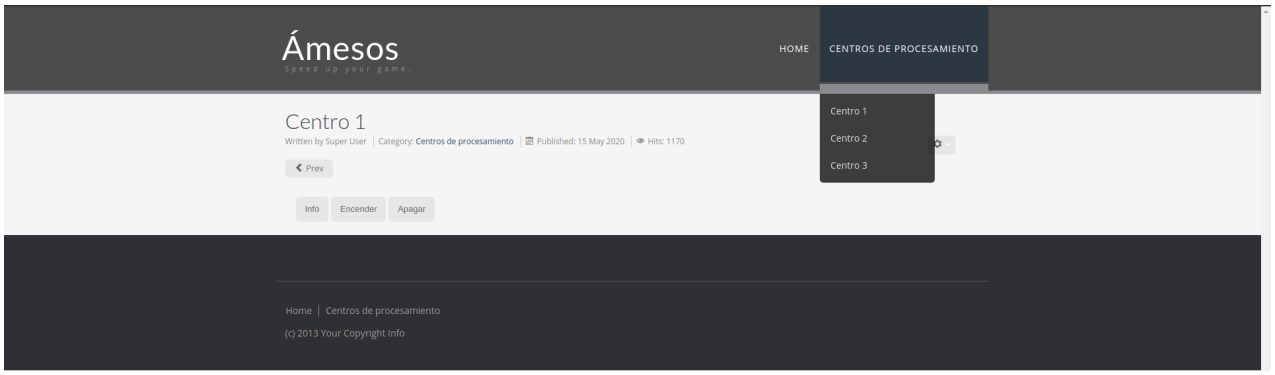


Figura 5.2: Cómo escoger centro de procesamiento de datos en el sistema de gestión de contenidos

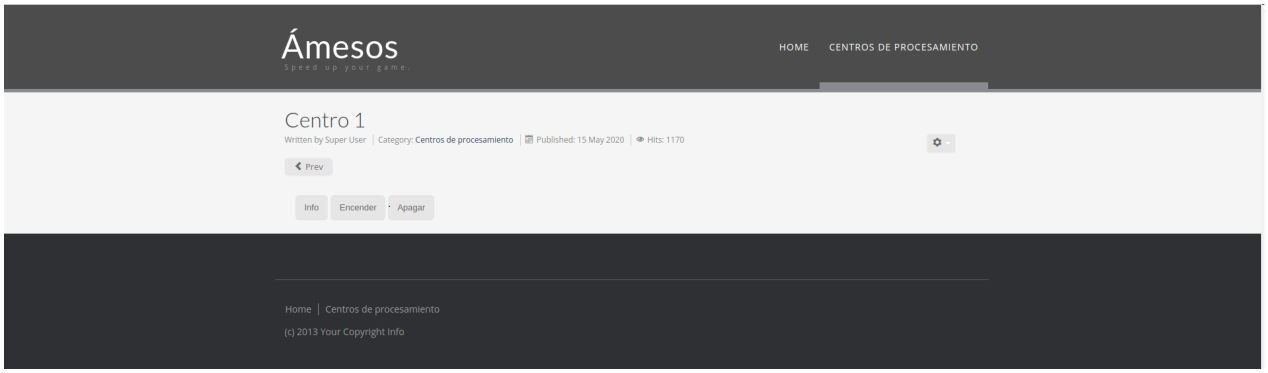


Figura 5.3: Página web base vacía para el Centro1

Ahora se abre el primer servidor en el Centro1 dando al **botón de encender**.

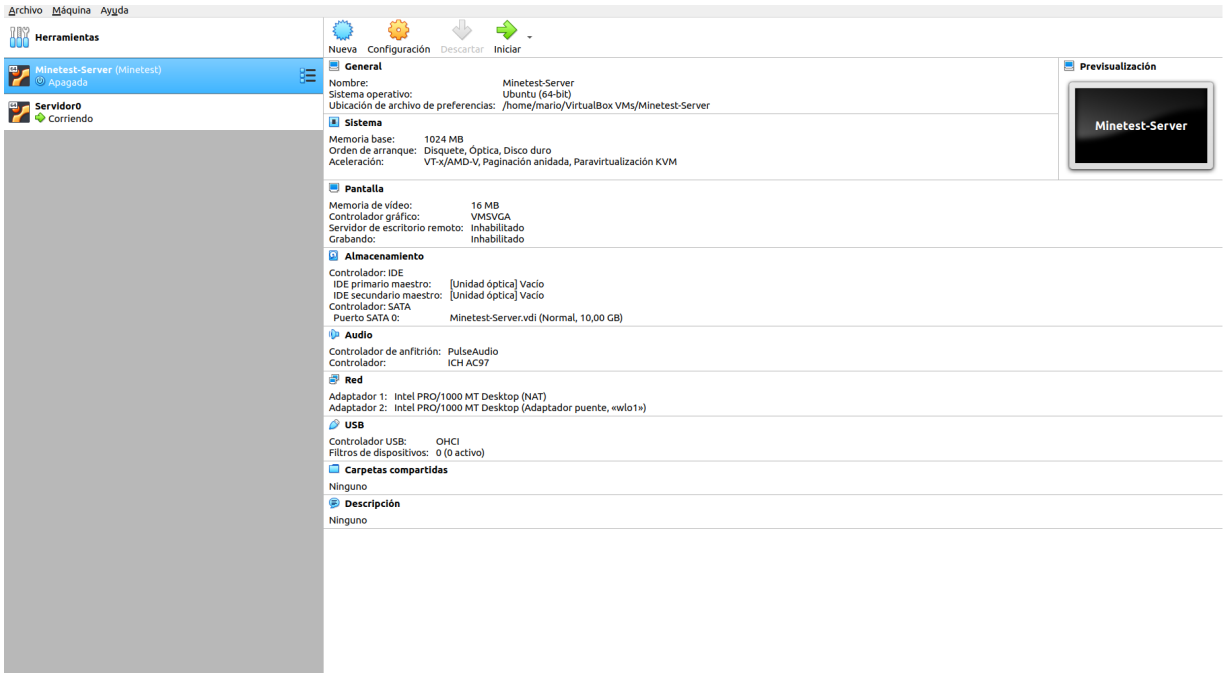


Figura 5.4: Primer servidor arrancado en el Centro1

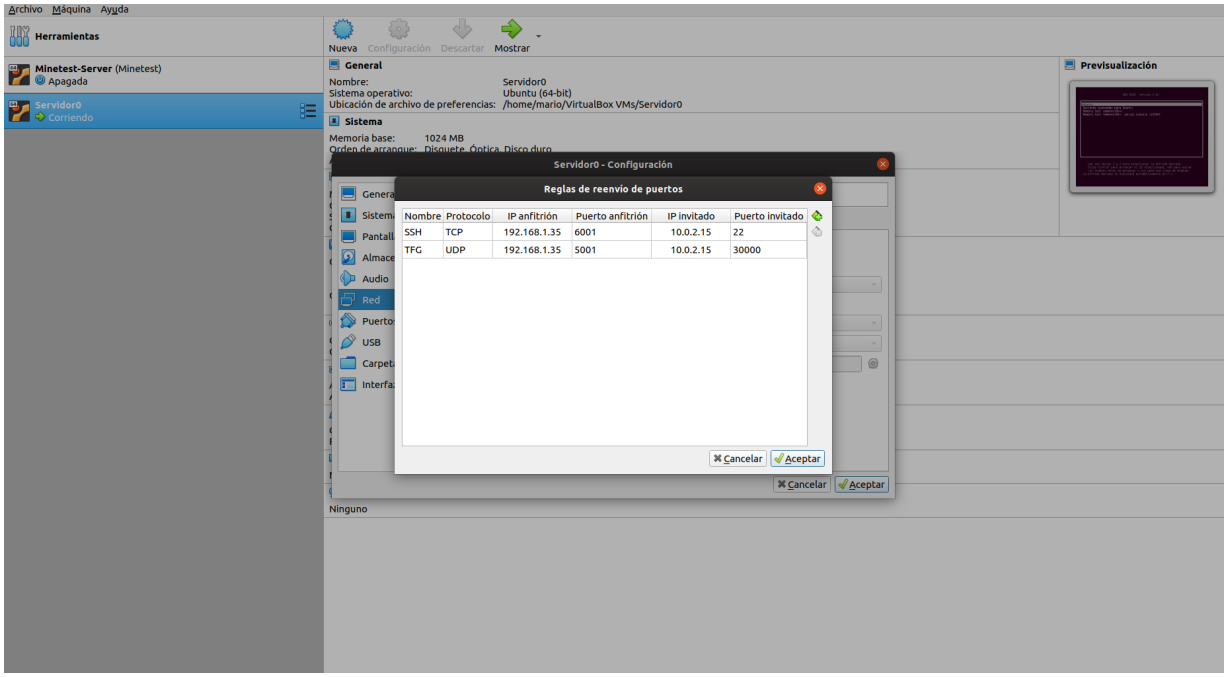


Figura 5.5: Configuración automática del primer servidor arrancado

Como se ve se abre el primer servidor **llamado Servidor0** que es una copia enlazada de la máquina virtual base, y además se configuran automáticamente sus reglas de **reenvío de puertos**. La primera corresponde a la conexión SSH para que el sistema de gestión de contenidos se pueda conectar directamente a la máquina virtual y la segunda corresponde a la conexión al servidor de juego que corre dentro de la máquina virtual.

Figura 5.6: Creación del Servidor0 en el JSON

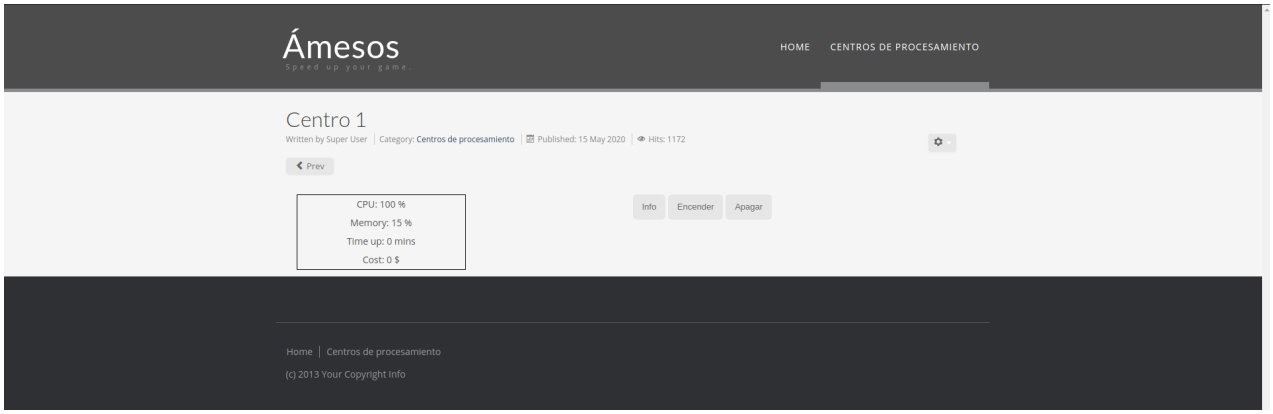


Figura 5.7: Servidor0 mostrado en la pagina Web del Centro1

Como es lógico, la creación del Servidor0 en el centro de procesamiento de datos se ve reflejada en **JSON** que usa el sistema de gestión de contenidos. Y por tanto, el propio sistema de gestión de contenidos actualiza la página web con la información del servidor creado. Dado que el servidor que se acaba de arrancar, este usa **toda su CPU** y lleva **0 minutos corriendo**.

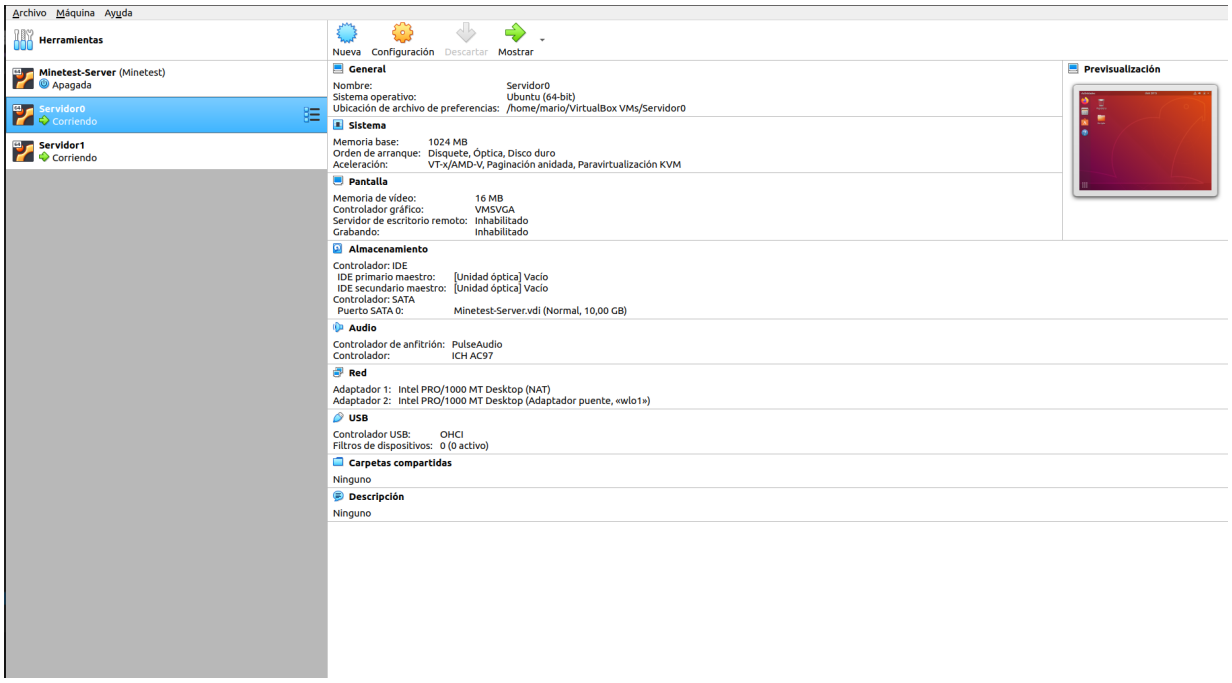


Figura 5.8: Creación del Servidor1 en el centro de procesamiento de datos

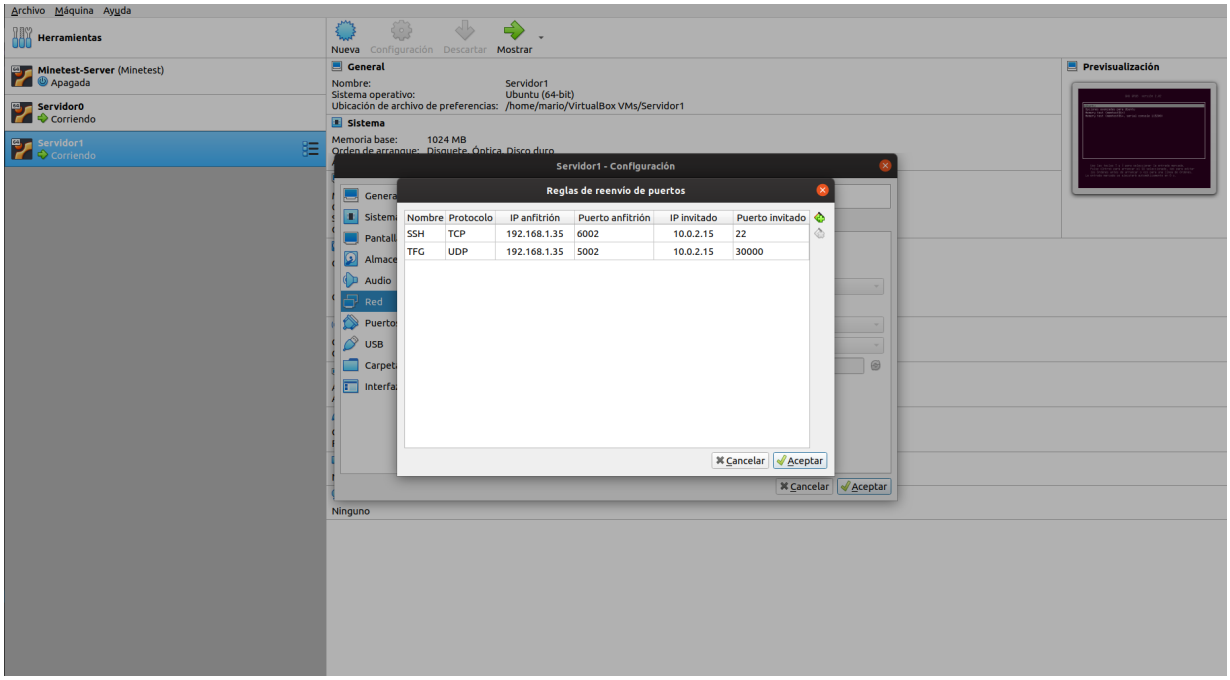


Figura 5.9: Configuración automática del Servidor1

Ahora se abre un segundo servidor **llamado Servidor1** y como se ve, se crean a la perfección las reglas necesarias para usar el reenvío de puertos apuntando a los siguientes puertos disponibles. Es decir, **los puertos usados en el Servidor0 +1**.

Figura 5.10: Creación del Servidor1 en el JSON

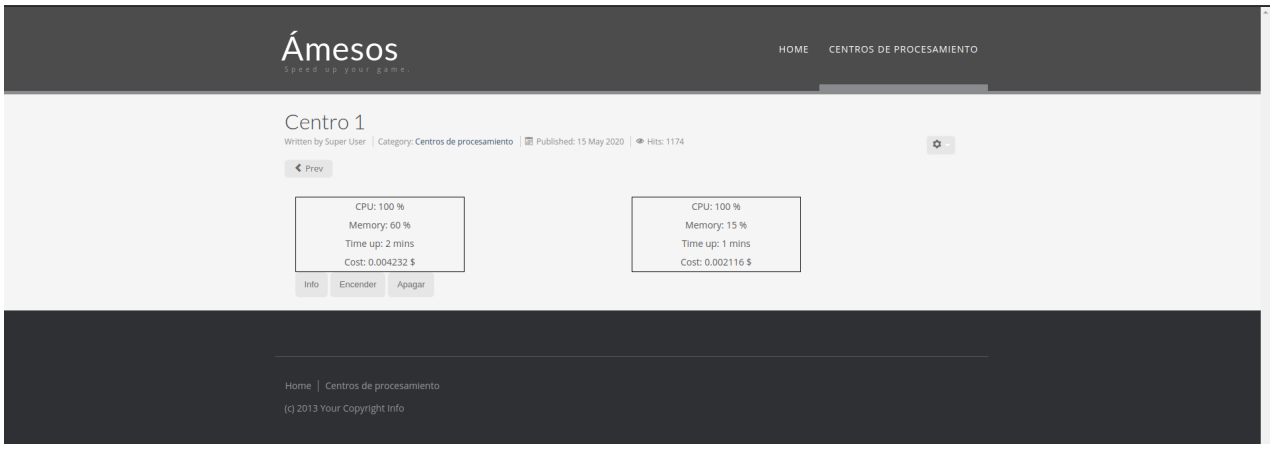


Figura 5.11: Servidor1 mostrado en la página Web del Centro1

Por tanto ahora, este nuevo servidor se ve reflejado tanto el **JSON** como en el sistema de gestión de contenidos. Se ve que esta prueba se ha hecho todo de una vez ya que el servidor creado anteriormente **lleva 2 minutos corriendo** y este nuevo servidor **lleva únicamente 1 minuto corriendo**.

Figura 5.12: Creación del JSON informativo con 2 servidores abiertos

Ahora, pulsando **el botón de info** se crea este **JSON informativo**, el cual contiene la **dirección IP y el puerto a donde se tiene que conectar para entrar al servidor que corre en cada una de las máquinas virtuales**. **Este JSON es el que el cliente distribuye a sus clientes para que estos puedan jugar en los servidores creados.**

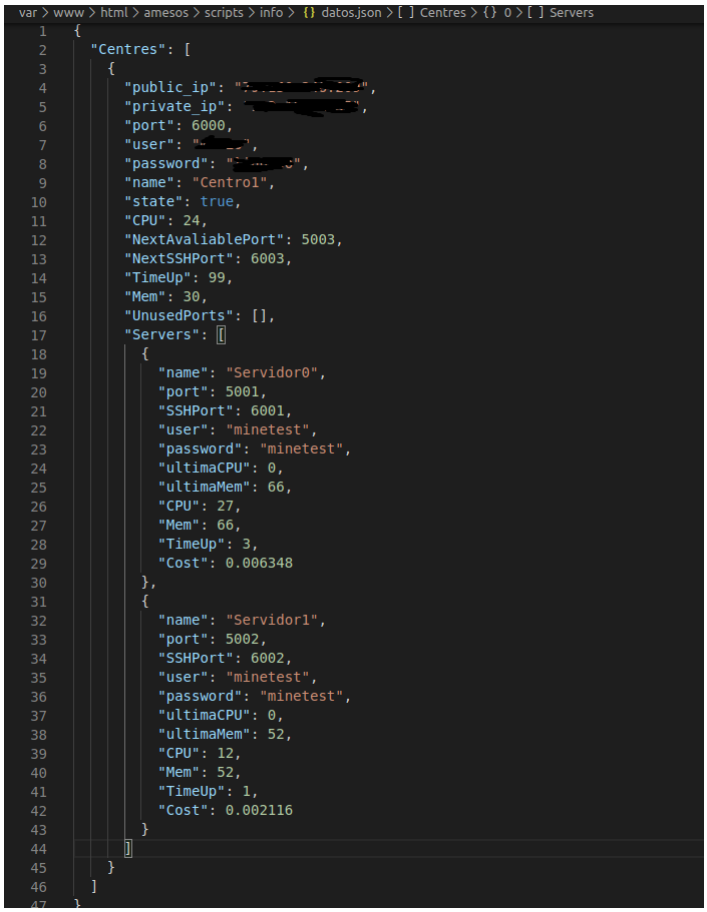


Figura 5.13: Monitorización de los servidores

Ahora, como se aprecia, se ha esperado unos minutos para que se vayan monitorizando esos 2 servidores creados y que, por tanto, actualicen sus atributos. Se observa que el primero de ellos está usando un 27 por ciento de CPU mientras que el segundo solo está utilizando un 12.

Figura 5.14: Modificación en el JSON tras destruir un servidor

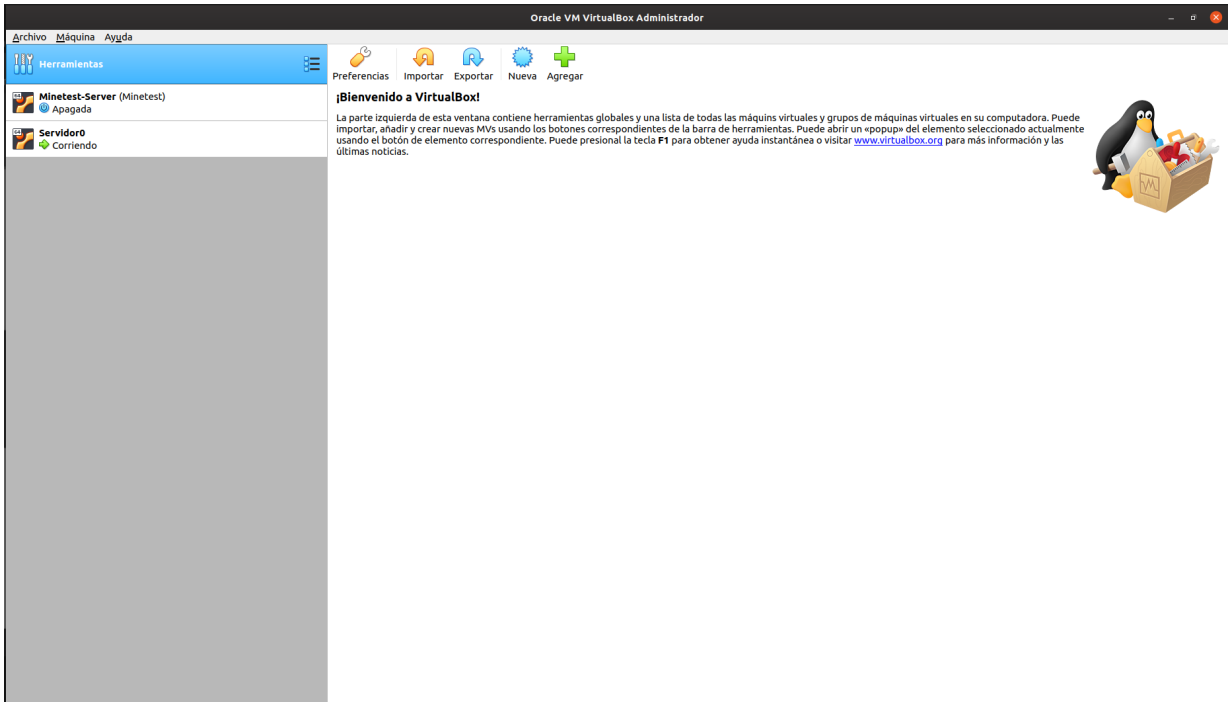


Figura 5.15: Modificación en el centro de procesamiento de datos tras destruir un servidor

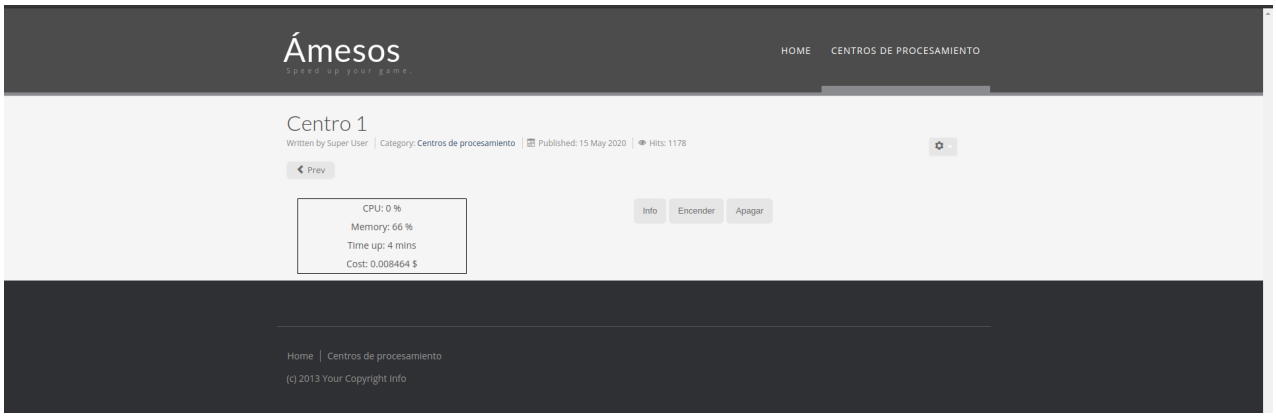


Figura 5.16: Modificación en el sistema de gestión de contenidos tras destruir un servidor

Se ha pulsado el **botón de apagar** de la página web y **debido a que el segundo servidor estaba usando menos CPU, este ha sido el que se ha apagado y borrado.** Como se ve en las imágenes de arriba, esa modificación se ha actualizado tanto en el JSON (donde se ha guardado ese servidor destruido en *UnusedPorts*), en el centro de procesamiento de datos y en el sistema de gestión de contenidos.

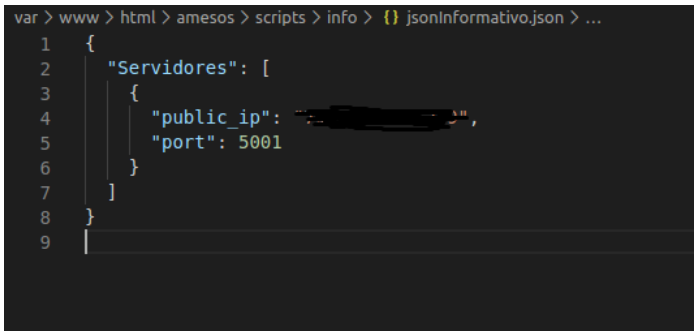


Figura 5.17: JSON informativo con un solo servidor

Se pulsa de nuevo el **botón de info** para crear un nuevo JSON informativo, donde como se ve ahora, solo muestra un único servidor disponible.

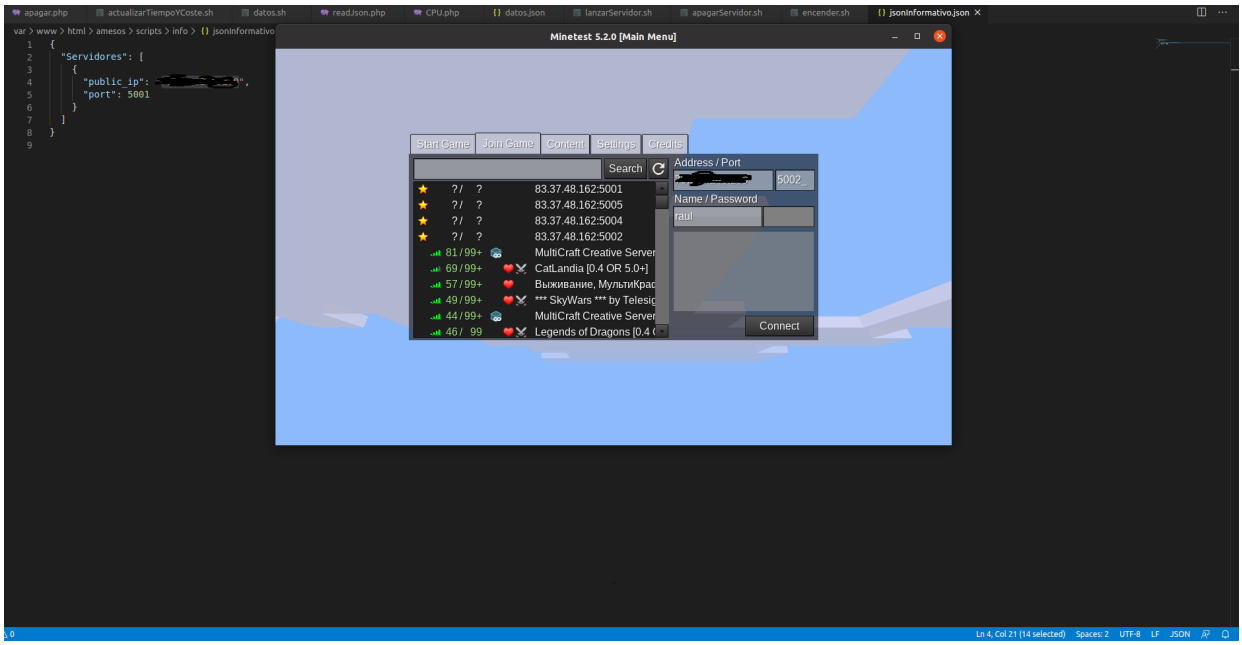


Figura 5.18: Conexión a servidor inexistente

Seguidamente se va a arrancar **el cliente de Minetest** y se va a intentar conectar a la IP dada por el JSON informativo pero a un puerto incorrecto **(5002 en vez de 5001)**, para que a propósito falle y, por tanto, mostrar que no se puede conectar a ningún servidor que no existe.

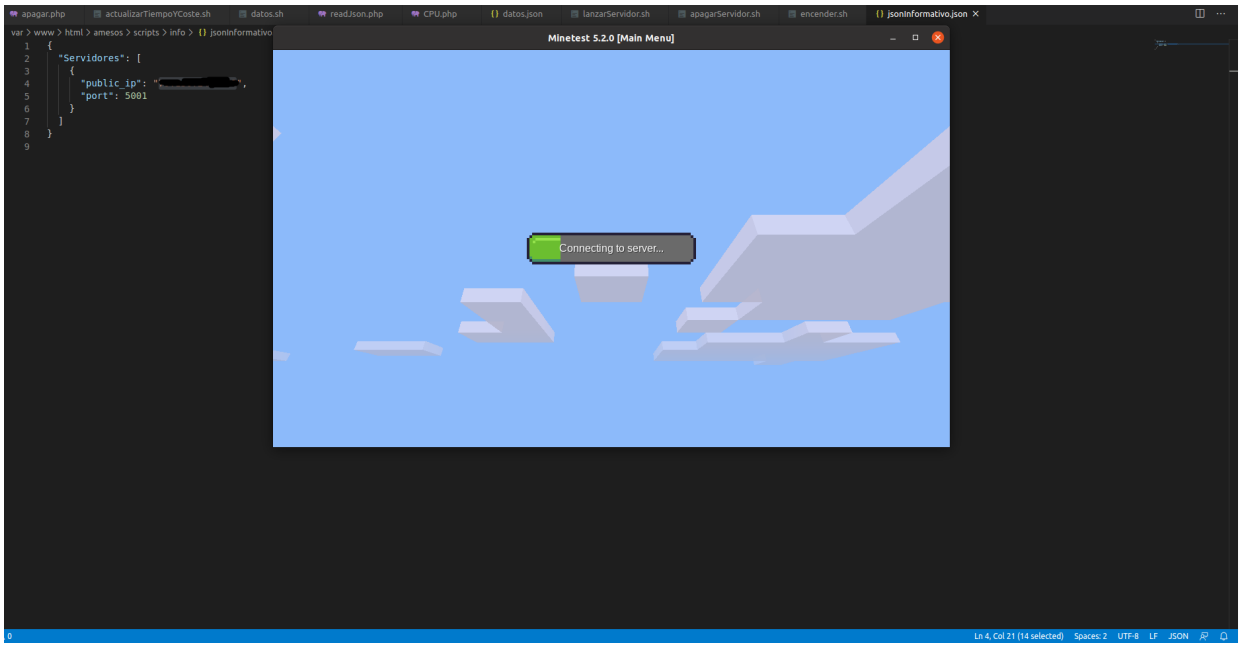


Figura 5.19: Cargando conexión

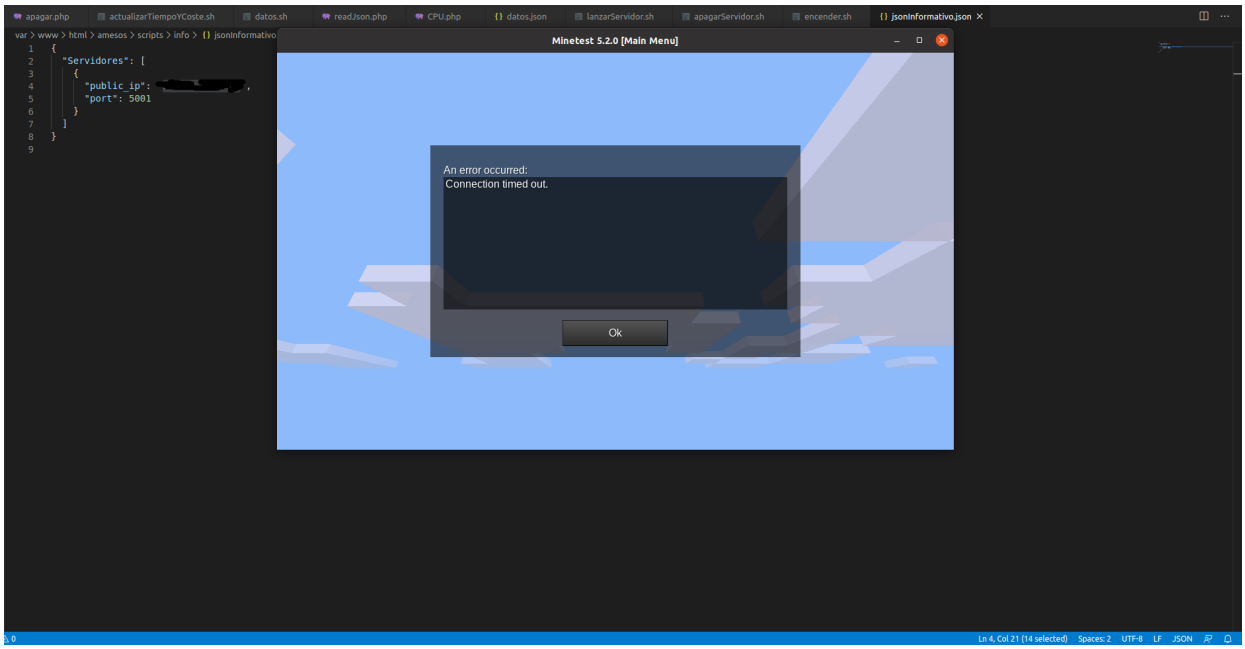


Figura 5.20: Conexión fallida

Se ve, como es lógico, que no se ha podido establecer conexión con ningún servidor.

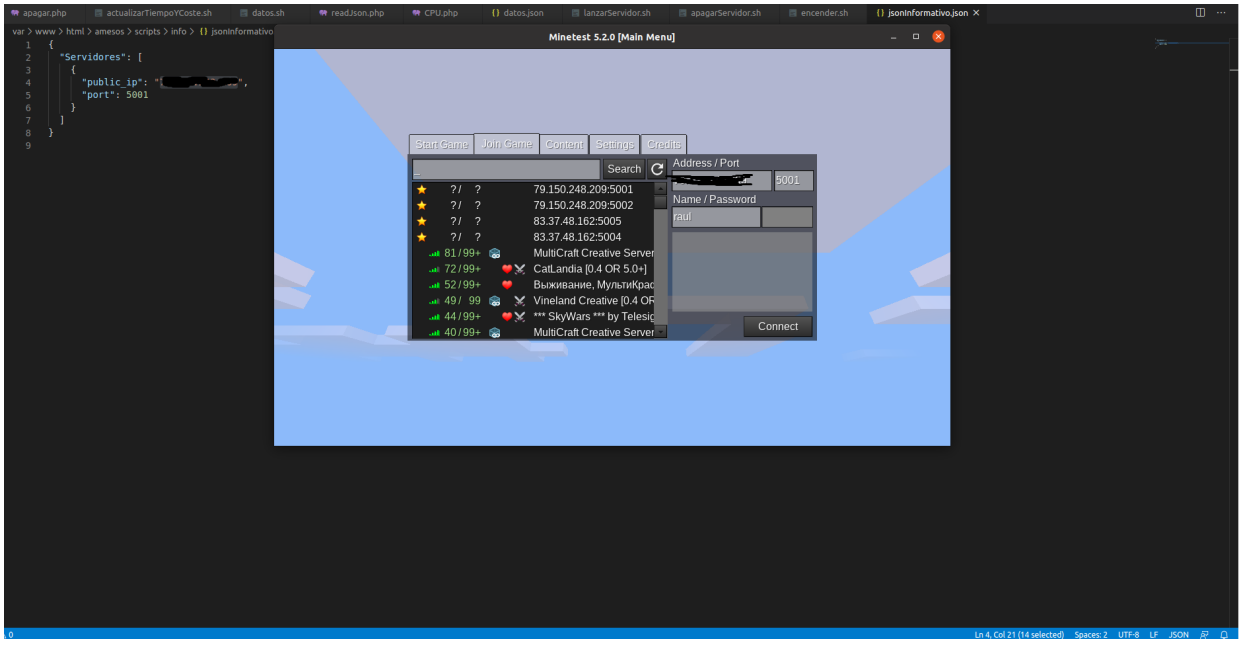


Figura 5.21: Conexión a un servidor que corre dentro del centro de procesamiento de datos Centro1

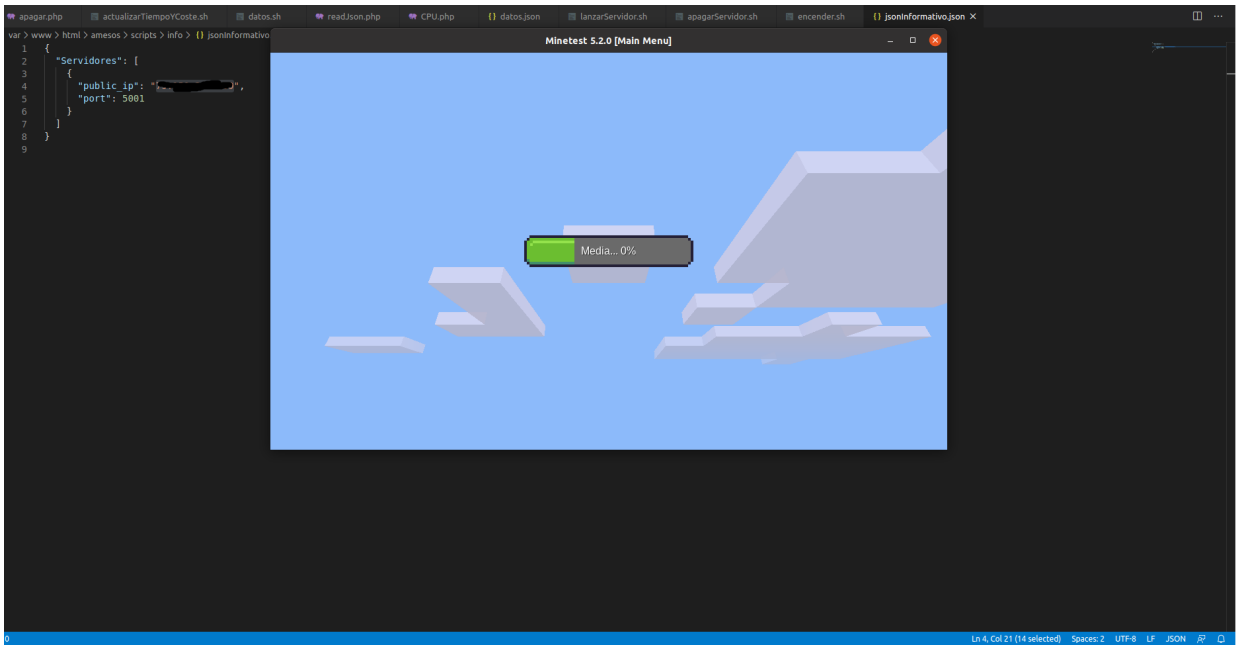


Figura 5.22: Conexión cargando con éxito

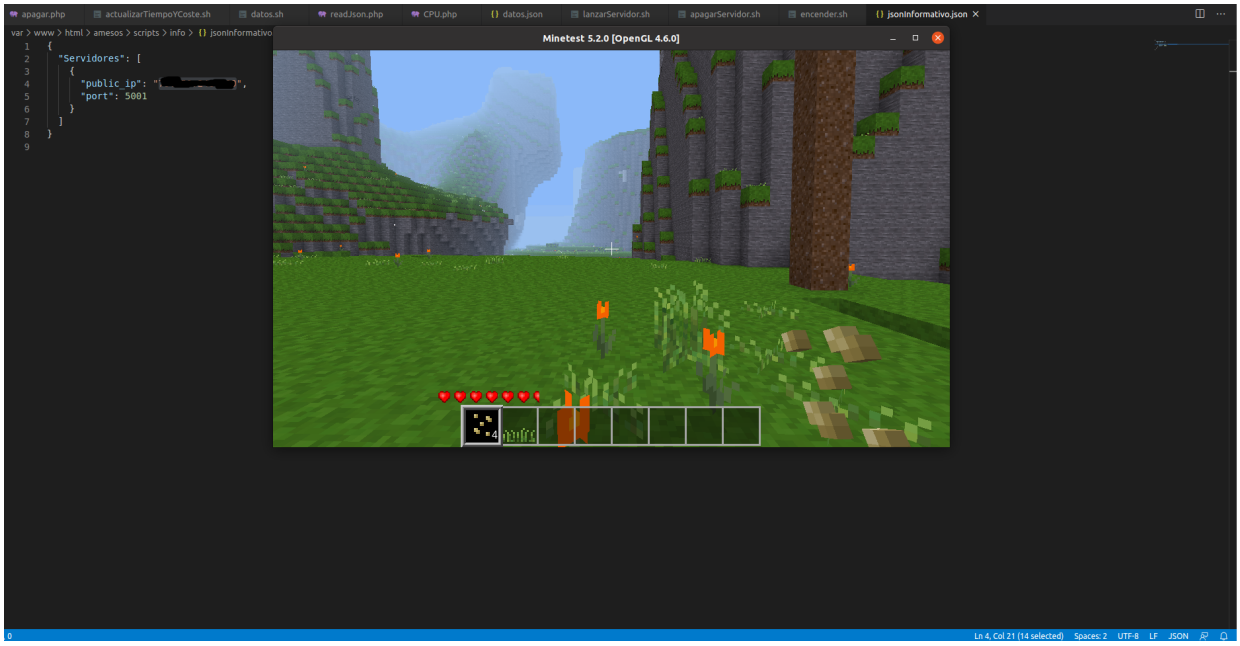


Figura 5.23: Conexión establecida con el servidor

Ahora, se intenta conectar a la IP y puertos dados por el JSON informátivo y **el cliente se puede conectar perfectamente a un servidor que corre dentro de la máquina virtual Servidor0 del centro de procesamiento de datos Centro1**, por lo que la prueba ha sido todo un éxito.

Capítulo 6

Conclusiones

En este capítulo comentaremos de manera personal la evaluación del proyecto, el trabajo dedicado al mismo y las aportaciones obtenidas por los miembros del grupo, así como sus posibles ampliaciones y mejoras de cara al futuro.

Nos gustaría, además, compartir todo el software desarrollado el cual se puede encontrar en un repositorio de GitHub[1] accediendo al siguiente enlace: https://github.com/MarioDeLosSantos/Amesos

6.1. Valoración personal

Lo primero a mencionar es que desde un inicio la idea de realizar un proyecto basado en redes nos pareció muy atractiva. A pesar de cursar un grado más relacionado puramente con los videojuegos, queríamos hacer algo distinto, pero útil en torno a esta rama.

Por norma general, cada empresa de videojuegos utiliza sistemas propios para gestionar y monitorizar el uso de sus servidores. De aquí nació Ámesos.
La idea principal se ha basado en construir un sistema estandarizado de gestión y monitorización de servidores de videojuegos genéricos, reduciendo así tiempo y costes a las empresas debido a que no tienen que implementar sus propios modelos independientes.

No obstante, es importante mencionar que nuestros conocimientos sobre redes, al no cursar un grado más orientado a estas tecnologías, son relativamente limitados. A pesar de esto y aunque supusiera un gran reto, personalmente estamos muy satisfechos con el resultado obtenido ya que hemos construido desde 0, con el uso de diferentes tecnologías, una aplicación web capaz de desplegar servidores de juego funcionales en centros de procesamiento de datos remotos bajo demanda.

Por otro lado, y referente al apartado tecnológico, hemos ampliado enormemente nuestros conocimientos sobre redes y sobre el funcionamiento de los hipervisores. Y, además, debido a que el proyecto exigía una estructura funcional en la web, hemos obtenido abundantes fundamentos tanto de programación web como del manejo y estructura de los CMS.

[1] GitHub: https://github.com/

6.2. Futuro trabajo

Bien es cierto que el proyecto cumple con las expectativas impuestas inicialmente, pero también da pie a que se expanda enormemente en muchos aspectos.

Para empezar, se podría añadir una herramienta de geolocalización para los centros de procesamiento de datos. De esta manera, el cliente podría conocer con mayor precisión aquellos nodos en los que reunir a los jugadores más cercanos a los mismos. Así, junto con los datos ya obtenidos por la aplicación, mejoraría globalmente la experiencia ofrecida a los jugadores del cliente.

Por otra parte, debido a que la conexión del sistema de gestión de contenidos con todos los centros de procesamiento de datos se basa únicamente en el protocolo **SSH** y no en la API concreta de un hipervisor, se puede ampliar el proyecto para que use otros hipervisores distintos a VirtualBox donde alojar las máquinas virtuales tales como **VMware**, **bhyve**, etc.

Además, como se ha mostrado anteriormente, para este proyecto únicamente se usa **Minetest** como servidor de juego. Pero sería fácilmente ampliable ya que solo se requeriría instalar más máquinas virtuales base donde se alojasen nuevos servidores de juego. En estas máquinas base solo sería necesario incluir la imagen del servidor del juego en cuestión de la empresa del cliente, y luego esta se clonará en función de las necesidades tal y como ofrece la aplicación de Ámesos.

Ahora mismo la aplicación de Ámesos usa siempre el formato **JSON** para mostrar a nuestro cliente los servidores que están disponibles en nuestros centros de procesamiento de datos. Pero debido a que es muy probable que este cliente distribuya estos servidores a sus jugadores en otro tipo de formato, Ámesos podría adapartarse a ese formato específico de la empresa para que así nuestro cliente no tuviera que hacer conversiones a la hora de distribuirlo a sus clientes.

En cuanto al *frontend* de la aplicación, se podría mejorar la interfaz de la aplicación web añadiendo más elementos visuales como iconos o imágenes. Además, se podrían usar colores para resaltar algunos elementos como un uso bajo o alto de cpu o memoria.

Capítulo 7

Conclusions

In this chapter we will personally comment on the evaluation of the project, the work dedicated to it and the contributions obtained by the members of the group, as well as its possible extensions and improvements for the future.

Also, we would be pleased to share all of the software that can be found under a GitHub[1] repository accessed by clicking on the following link: https://github.com/MarioDeLosSantos/Amesos

7.1. Personal assessement

The first thing to mention is that from the beginning the idea of carrying out a network-based project seemed very attractive to us. Despite studying a degree more related purely to videogames, we wanted to do something different, but useful around this branch.

As a general rule, each videogame company uses their own systems to manage and monitor the use of their servers. This is where Ámesos was born.
The main idea has been based on building a standardized management and monitoring system for generic videogame servers, thus reducing time and costs for companies because they do not have to implement their own independent models.

However, it is important to mention that our knowledge about networking, as we do not study a degree more oriented to these technologies, is relatively limited. Despite this and although it was a big challenge, we are personally very satisfied with the results obtained since we have built from scratch, with the use of a wide variety of technologies, a web application capable of deploying on demand functional game servers in remote data processing centers.

On the other hand, and referring to the technology section, we have vastly expanded our knowledge of networking and operation of hypervisors. And also, cause the project required a functional structure on the web, we have earn abundant foundations both for web programming and for management and structuring of CMS.

[1] GitHub: https://github.com/

7.2. Future work

It is true that the project meets the expected requirements initially imposed, but due to how it is structured, this also leeds to a potencial wide expansion in multiple aspects.

To begin with, a geolocation tool could be added for the data processing centers. This way, the client could know more precisely those nodes in which to bring the closest players together. Thus, along with the data already gathered by the application, the client users experience would be greatly improved.

Moreover, due to connectivity between the Content Management System and every Data Processing Center is based exclusively in SSH protocol and not particulary in the hypervisor, the proyect could be expanded also in this branch as different hypervisors other than VirtualBox could be used to allocate the virtual machines. Some of them such as **VMware**, **bhyve**, etc.

Furthermore, as it has been shown previously, **Minetest** is the only server image used for this project. Nonetheless, this feature is easily expandable as the only requierement to meet is installing more base virtual machines to allocate the new game servers. This base machines just should incorporate the game server images in question from the client's company. Then, this image will just get cloned based on the particular needs just as Ámesos application offers.

Also, as demonstrated above, **Minetest** is used as the game server for this specific project. But it would be easily expandable since it would only be necessary to install more base virtual machines where new game servers are hosted. On these base machines it would only be necessary to include the image of the game server in question of the client's company, and then it will clone depending on the needs as offered by the Ámesos application.

Right now the Ámesos application always uses the **JSON** format to show our client the servers that are available in our data processing centers. But because this client is very likely to distribute these servers to their players in another type of format, Ámesos could adapt to a specific format so that our client does not have to make any conversions when distributing to its clients.

As for the *frontend* of the application, we could improve the interface of the web application by adding more visual elements such as icons or images. In addition, we can use colors to highlight some elements such as low or high use of cpu or memory.

Bibliografía

[1] Kevin Webb. The 120$ billion gaming industry is going through more change than it ever has before, and everyone is trying to cash in. *Business Insider*, 2019. https://www.businessinsider.com/video-game-industry-120-billion-future-innovation-2019-9?IR=T.

[2] Keshav Bhat. Call of duty: Warzone features 150 players. *Charlie Intel*, 2020. https://charlieintel.com/call-of-duty-warzone-features-150-players/59526/.

[3] Elisabeth C. World of tanks sets world record for most players concurrently online on a single server. *Engadget*, 2013. https://www.engadget.com/2013-03-12-world-of-tanks-sets-world-record-for-most-players-concurrently-o.html.

[4] Pete Mastin. How latency is killing online gaming. *Venturebeat*, 2016. https://venturebeat.com/2016/04/17/how-latency-is-killing-online-gaming/.

[5] Yorbelis Rosell León. Sistemas gestores de contenidos: una mirada desde las ciencias de la información. *ACIMED*, 2011. http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352011000100002.

[6] P. Bramscher and J. Butler. Libdata to libcms: One library's evolutionary pathway to a content management system. *Library Hi Tech*, 2006. https://www.researchgate.net/publication/243461557_LibData_to_LibCMS_One_library's_evolutionary_pathway_to_a_content_management_system.

[7] Xavier Garcia Cuerda. Introducción a los sistemas de gestión de contenidos (cms) de código abierto. *Mosaic*, 2004. https://mosaic.uoc.edu/2004/11/29/introduccion-a-los-sistemas-de-gestion-de-contenidos-cms-de-codigo-abierto/.

[8] Claudio Erba. Php-nuke: Management and programming. 2003. https://www.tldp.org/HOWTO/PHP-Nuke-HOWTO/index.html.

[9] Rosell Y. Impacto de los sistemas gestores de contenido (cms) en centros de educación superior de ciudad de la habana [tesis de maestría]. 2009.

[10] Robertson J. Looking towards the future of cm. 2003. http://www.steptwo.com.au/papers/cmb_future/index.html.

[11] Eleni Zoe. Lms vs cms: What's the difference? *Talentlms*, 2020. https://www.talentlms.com/blog/a-comparison-between-lms-and-cms/.

[12] Yongqiang Gao, Lin Wang, and Jiantao Zhou. Cost-efficient and quality of experience-aware provisioning of virtual machines for multiplayer cloud gaming in geographically

distributed data centers. *IEEE Access*, 2019. https://ieeexplore.ieee.org/document/8852641.

[13] Mohammad Sadegh Aslanpour, Mostafa Ghobaei□Aran, Morteza Heydari, and Nader Mahmoudi. Larpa: A learning automata□based resource provisioning approach for massively multiplayer online games in cloud environments. *Wiley Online Library*, 2019. https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4090.

[14] M. Fernández-Caramés Tiago, Iván Froiz-Míguez, Oscar Blanco-Novoa, and Paula Fraga-Lamas. Enabling the internet of mobile crowdsourcing health things: A mobile fog computing, blockchain and iot based continuous glucose monitoring system for diabetes mellitus research and care. *MDPI*, 2019. https://www.mdpi.com/1424-8220/19/15/3319.

[15] Rashmi Shrivastava and Manju Pandey. Real time fall detection in fog computing scenario. *Springer Link*, 2019. https://link.springer.com/article/10.1007/s10586-020-03051-z.

Apéndice A

Manual de instalación

Para futuros administradores del sistema se ha creado este apartado que recopilará paso a paso todos los elementos necesarios a instalar y configurar para el funcionamiento de la aplicación en su totalidad. Se ha dividido temáticamente en centro de procesamiento de datos y sistemas de gestión de contenidos debido a que son bloques bien diferenciados entre ellos.

A.1. Centro de procesamiento de datos

Sistema operativo

Dado que todo este entorno corre en un sistema operativo **Linux**, esto es lo primero que hay que instalar. En el caso de este proyecto se ha creado una partición en un portátil para la cual se ha usado un USB que gracias a la aplicación **Universal USB Installer** [1], contenía la configuración necesaría para instalar dicho sistema operativo desde la **BIOS**.

VirtualBox

Como ya se ha mencionado anteriormente, el hipervisor usado ha sido VirtualBox. Pero no vale únicamente con instalar dicho programa, también hay que instalar el **Extension Pack de VirtualBox**[2], localizado en la página de descargas oficial es Oracle, para poder usar su API mediante comandos.

Máquina virtual

La máquina virtual base usada se ha creado de forma natural con VirtualBox con los parámetros prederminados de la aplicación. Tras eso, se ha abierto la configuración de la máquina virtual creada y en el apartado de **Almacenamiento ->Controlador:Sata** se ha añadido un archivo de escritorio virtual (VDI), más concretamente la **versión 20.04**[3].

[1]Universal USB Installer: https://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/
[2]Extension Pack: https://www.virtualbox.org/wiki/Downloads
[3]Archivo VDI: https://www.osboxes.org/ubuntu-server/

Minetest y configuración del servidor

La forma mas común de instalar un servidor de Minetest es con el siguiente comando:

```
$ sudo apt-get install minetest-server
```

Este comando, a menos que actualicemos la versión que podemos descargar, descarga la **versión 0.4.16**, mientras que el siguiente comando, el cual es para instalar el cliente, instala la **versión 5.0.1**.

```
$ sudo apt-get install minetest
```

Dado que las versiones del cliente y el servidor son incompatibles, hay que buscar una forma de instalar la última versión del servidor. Para ello, hay que seguir los siguientes pasos.

Lo primero es añadir un repositorio donde se encuentre dicha versión y actualizar el sistema para que lo descargue desde ahí.

```
$ sudo add-apt-repository ppa:minetestdevs/stable
$ sudo apt update
```

Ahora ya podemos instalar Minetest

```
$ sudo apt-get install minetest
```

Una vez instalado, se abre el puerto donde escucha dicho servidor que **por defecto es el 30000**.

```
$ sudo ufw allow 30000
```

Ahora se configura el servidor de Minetest. Lo primero es descargar el archivo de configuración del servidor el cual tiene un lista enorme de posibles parámetros a configurar o habilitar.

```
$ cd ~/.minetest
$ sudo wget https://raw.githubusercontent.com/minetest/minetest/master/
minetest.conf.example
$ mv minetest.conf.example minetest.conf
```

Una vez instalado el archivo de configuración, se pueden modificar algunos parámetros para personalizar el servidor como los siguientes.

```
server_name = Minetest server
server_description = Mi servidor de Minetest
bind_address = IP privada de la máquina virtual
port = 30000
```

Bien, ya se tiene el servidor configurado, ahora hay que arrancarlo cada vez que se arranque el sistema que en el caso de este proyecto, cuando se arranque cada una de las máquinas virtuales. Por tanto tenemos que crear y modificar el servicio de Minetest dentro de la carpeta de **/etc/systemd/system/** con el siguiente comando.

```
$ sudo nano /etc/systemd/system/minetest.service
```

Ahora hay modificarlo con los parámetros adecuados.

```
[Unit]
Description=Mi servidor de Minetest
After=network.target

[Service]
Type=simple
User=minetest
Group=minetest
WorkingDirectory=/home/minetest
ExecStart=/usr/bin/minetest --server
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

Ya por último, una vez creado el servicio, solo hay que habilitar dicho servicio y arrancarlo.

```
$ sudo systemctl enable minetest.service
$ sudo systemctl start minetest.service
```

Con todo esto hecho, cada vez que se arranque la máquina virtual base o alguna copia enlazada suya, se abrirá automáticamente un servidor que **escuchará en su IP privada y en el puerto 30000**.

Configuración del router

Para que el centro de procesamiento de datos pueda ser visible desde el exterior hay que configurar el reenvío de puertos del router. Este herramienta puede variar dependiendo de la empresa a la que pertenezca el servicio, pero normalmente se encuentra en **configuración avanzada->Reevío de puertos**. En el caso de este proyecto, esta en la interfaz de configuración **fig:4.9**.

Hay que abrir tanto puertos como servidores se quiera que aloje el centro de procesamiento de datos. Tiene que haber dos tipos, uno de tipo **UDP** para la conexión con el servidor de Minetest y otro tipo **TCP** para la conexión con la propia máquina virtual

Configuración del centro de procesamiento de datos

Como se ha mencionado en apartados anteriores, hay que abrir los puertos usados para que se puedan redirigir las conexiones del sistema de gestión de contenidos con cada una de las máquinas virtuales para cada uno de los centros de procesamiento de datos. Usando **iptables** se abrirán tantos tipos de rango de puertos como tipos de conexiones haya configuradas en el apartado de reenvío de puertos del router. En el caso de este proyecto, estas son las reglas implementadas **fig:4.3** y **fig:4.4**.

Por otra parte, debe existir una correlación entre donde se encuentran los scripts **encender.sh** y **apagar.sh** en el centro de procesamiento de datos y cuál es el path que se pone en los demás scripts que se ejecutan desde el sistema de gestión de contenidos.

A.2. Sistema de gestión de contenidos

Configración de LAMP

Lo primero de todo es **configurar la estructura LAMP** en el sistema Linux. Ejecute los siguientes comandos desde la terminal para la instalación del servidor LAMP.

Primero, añada el repositorio PPA [4] en su sistema.

```
$ sudo apt-get install -y python-software-properties
$ sudo apt add-repository ppa:ondrej/php -y
```

Ahora instale los paquetes de LAMP: Apache, MySQL y PHP. Use los siguientes comandos para cada uno de ellos.

```
$ sudo apt-get install apache2 apache2-data apache2-utils
$ sudo apt-get install php php-mcrypt php-curl php-mysql php-gd php-cli php-json
php-xml php-zip libapache2-mod-php
$ sudo apt-get install mysql-server mysql-client
```

Descarga del CMS Joomla

Hay que descargar la última versión de Joomla desde el repositorio oficial. Ejecute los siguientes comandos, y recuerde añadir la última versión de Joomla en el momento de instalar (en este caso, la 3.9.12).

```
$ cd /tmp
$ wget https://github.com/joomla/joomla-cms/releases/download/3.9.12/
Joomla_3.9.12-Stable-Full_Package.tar.gz
```

[4] Los PPA (Personal Packages Archives) permiten subir paquetes, software, etc. a un repositorio y que este, una vez está instalado en el sistema, se pueda actualizar de manera sencilla gracias al control de versiones.

Luego se extrae Joomla bajo la raíz del *Apache VirtualHost* para el servidor. Se añade el nombre deseado a la carpeta. En este caso será 'Amesos'.

```
$ mkdir -p /var/www/html/Amesos
$ cd /var/www/html/Amesos
$ tar xzf Joomla_3.9.12-Stable-Full_Package.tar.gz
$ chown -R www-data:www-data .
$ chmod -R 755 .
```

Configuración del VirtualHost de Apache

Se puede acceder al servidor de Joomla usando la URL como **http://localhost/Amesos**. En cambio, si se necesitara Joomla en el dominio principal, es posible configurarlo de la siguiente manera y añadiendo los datos correspondientes.

```
<VirtualHost *:80>
  ServerName joomla.example.com
  ServerAdmin webmaster@example.com
  DocumentRoot /var/www/html/joomla
  <Directory /var/www/html/joomla>
  Allowoverride all
  </Directory>
</VirtualHost>
```

Creación de la base de datos MySQL

El siguiente paso es crear tanto la base de datos correspondiente como el usuario que tendrá todos los privilegios sobre la misma.

```
mysql> CREATE DATABASE amesos;
mysql> CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'password';
mysql> GRANT ALL PRIVILEGES ON * amesos * TO 'usuario'@'localhost';
mysql> FLUSH PRIVILEGES;
```

Instalador de Joomla

Joomla proporciona un instalador web desde el cual acabar la propia instalación más cómodamente. Hay que acceder a la URL que ha configurado anteriormente para el servidor Apache (o si no se ha modificado, a http://localhost/Amesos).

En la primera página se deberá añadir los datos correspondientes para crear el usuario con acceso al panel de administrador de Joomla para la página web.

Figura A.1: Instalador de Joomla - datos del superusuario

En la siguiente página hay que introducir los detalles correspondientes a la base de datos creada anteriormente.

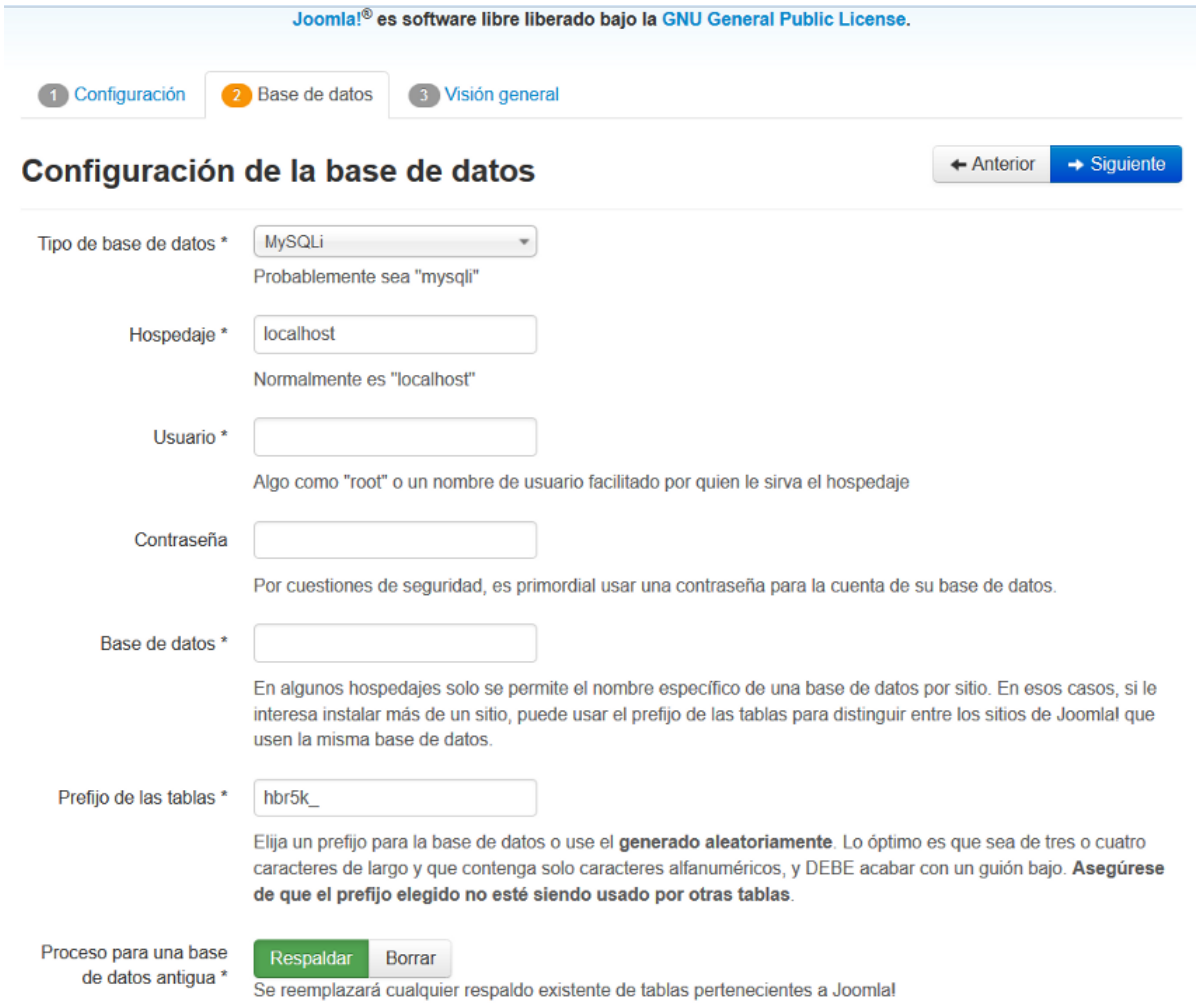


Figura A.2: Instalador de Joomla - info. de la base de datos

Joomla ofrece la opción de incluir distintas plantillas predefinidas para montar la página web inicial. se tiene que seleccionar una de ellas si interesa, o en caso contrario hay que dejar marcada la opción de ’Ninguno’.

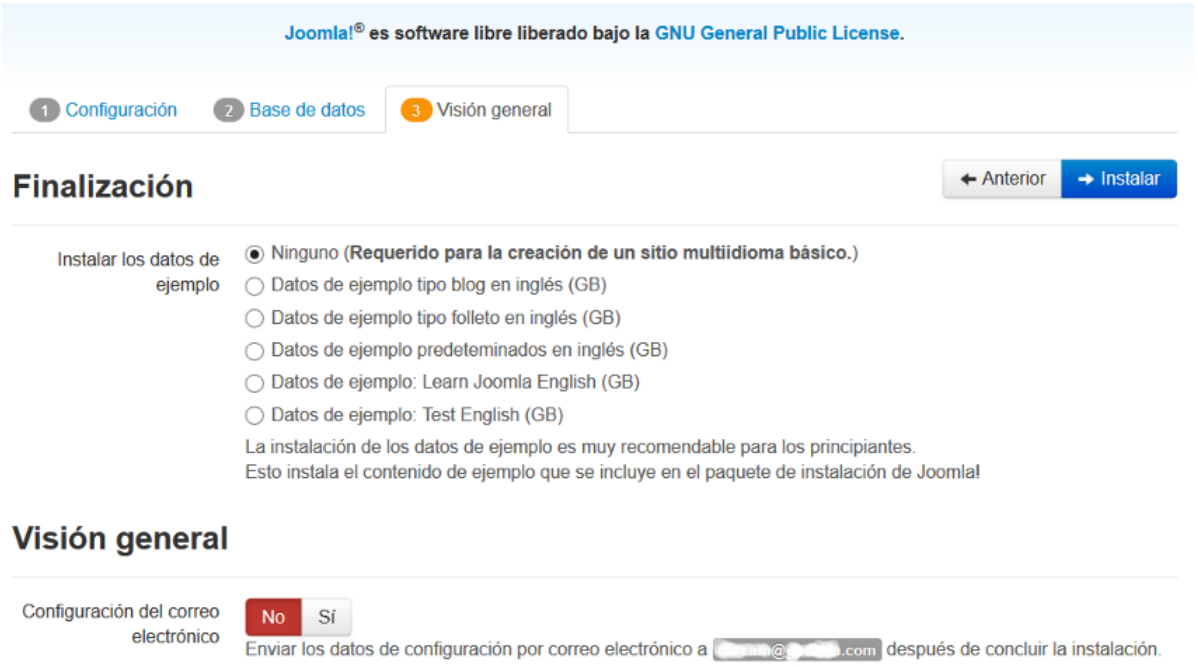


Figura A.3: Instalador de Joomla - selector de plantillas predefinidas

En la siguiente página se observará un resumen de los datos y opciones escogidas, y una comprobación general del estado y soportes del sistema con respecto a Joomla. Si no hay ningún error, se podrá continuar sin problema.

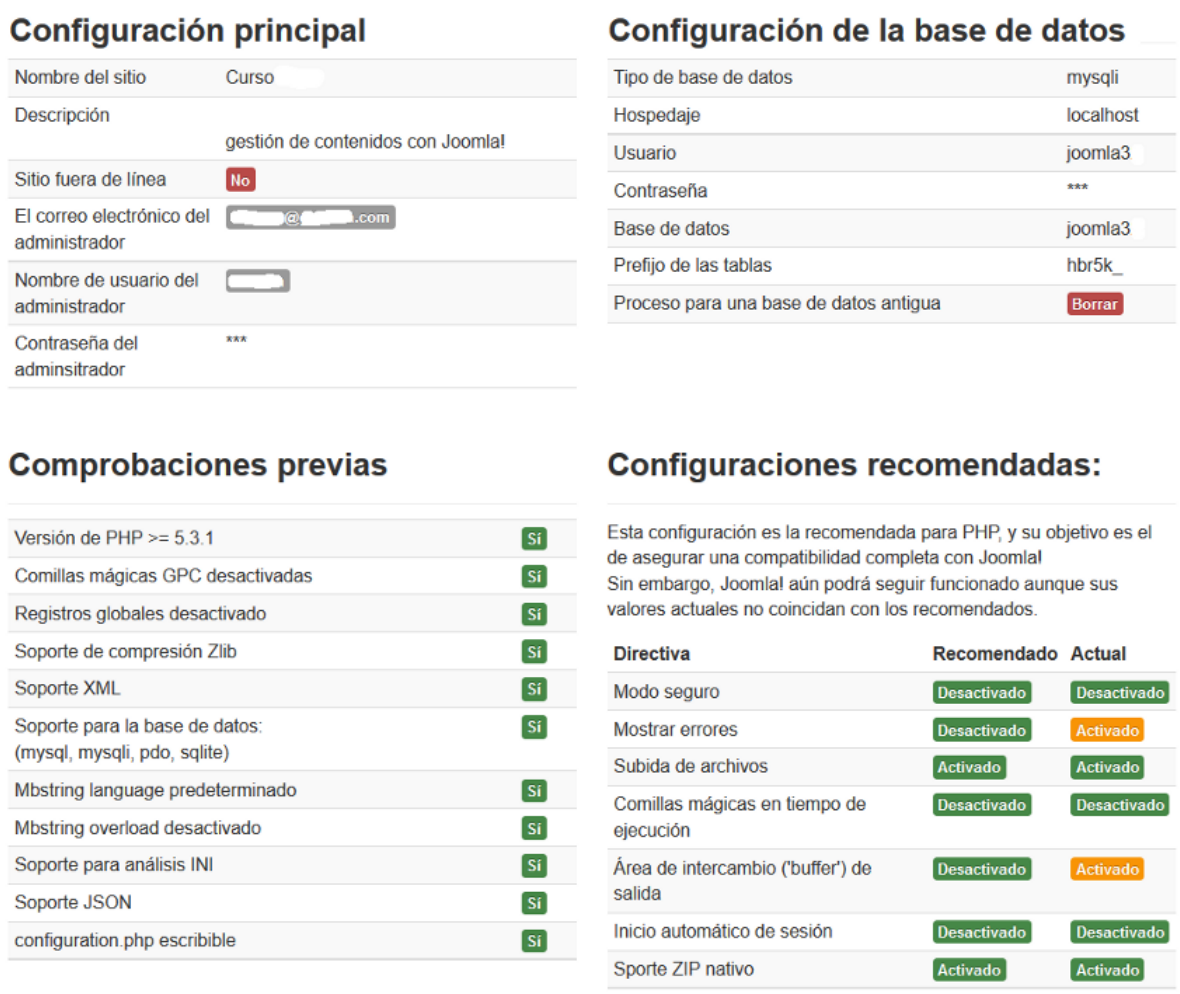


Figura A.4: Instalador de Joomla - resumen de la instalación

Por último, Joomla comenzará a instalarse como tal y ya se tendrá acceso al panel de administrador de la página. hay que eliminar la carpeta de instalación y acceder al sitio con normalidad.

Figura A.5: Instalador de Joomla - finalización

Configuración e instalación de paquetes y librerías

Una vez en el panel de administración, lo primero a modificar (aunque es opcional) por comodidad es el tiempo de vida de cada sesión antes de que se realice un *log out* automático. En la opción resaltada en la parte inferior de la imagen, añadir un tiempo de 360 debería ser suficiente.

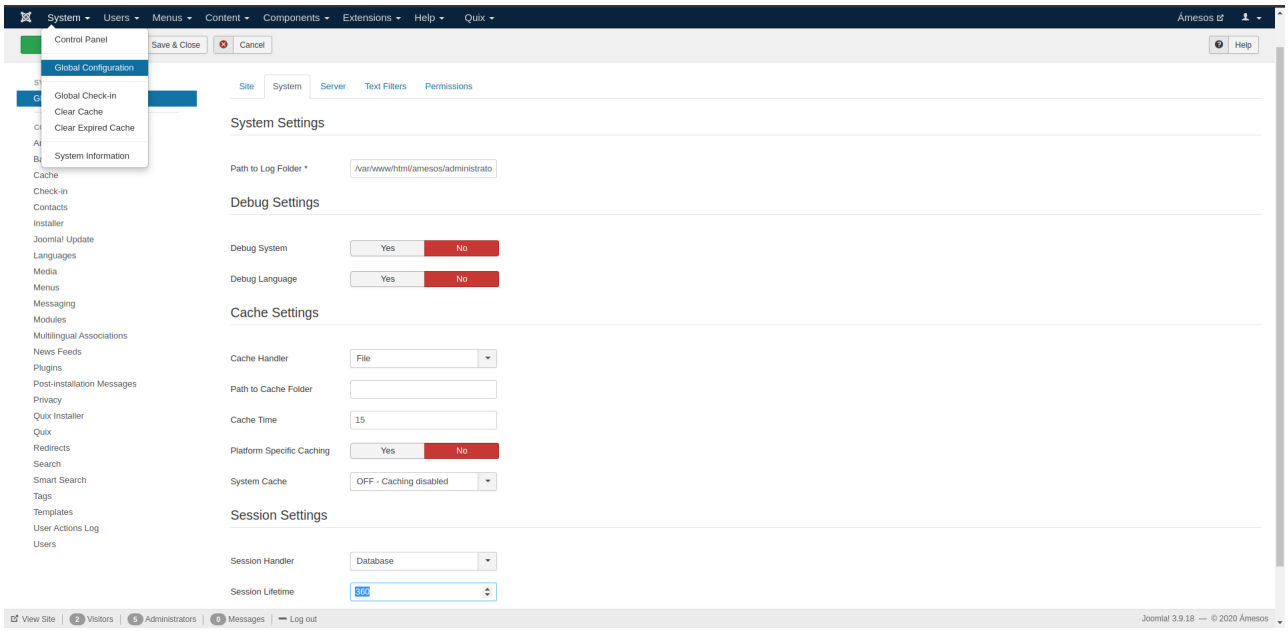


Figura A.6: Configuaración del tiempo de sesión en Joomla

Ahora, el siguente paso fundamental es la instalación de paquetes y librerías. En el caso de Ámesos, se ha instalado una plantilla para cambiar el diseño base de la web, pero es un paso opcional.
En cambio, las librerías que utiliza la aplicación sí son fundamentales, como es la libería Jumi en este caso.

Ambos paquetes se instalan de la misma manera en el apartado mostrado en la imagen una vez han sido descargados previamente de sus respectivos orígenes donde se obtienen, y el mismo proceso se realizaría para otros paquetes, librerías, plantillas, etc.

Simplemente se arrastra el archivo comprimido descargado a la zona indicada y comienza la instalación del paquete. En el caso de que el paquete tenga un instalador propio, bastará con seguir los pasos mostrados en el mismo escogiendo las opciones ofrecidas. En caso contrario, el paquete ya habrá sido instalado y quedará listo para su uso.

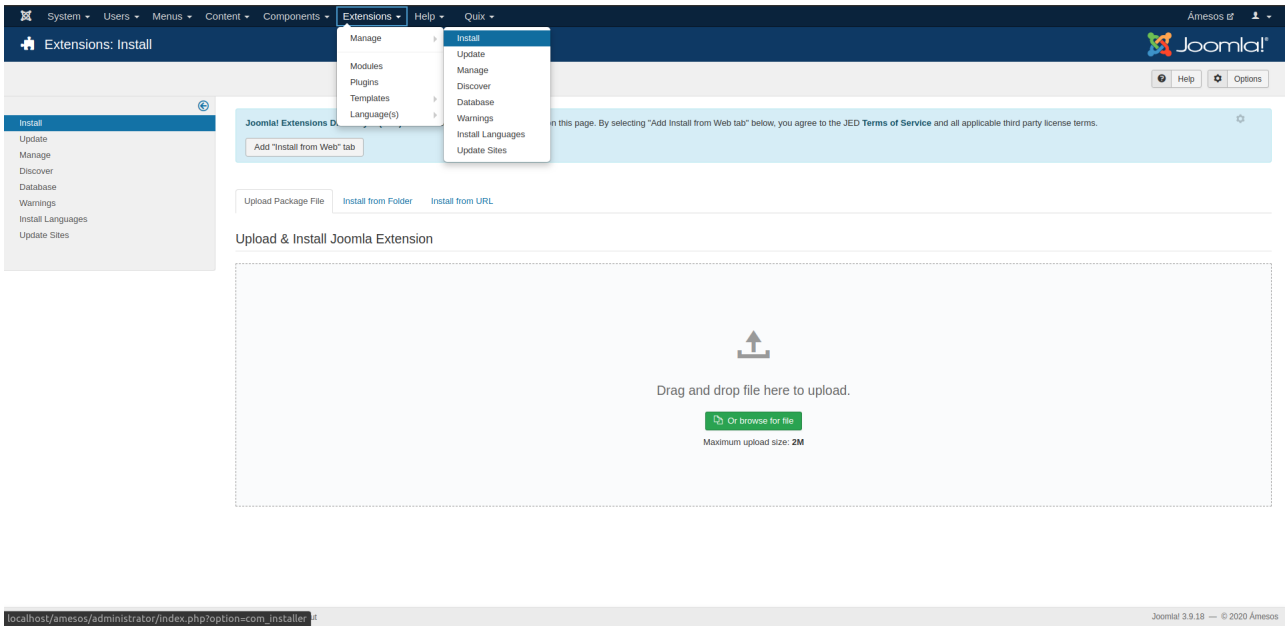


Figura A.7: Instalación de paquetes en Joomla

Scripts y permisos

Los scripts y archivos generados por la aplicación se ejecutan localmente, por lo que es necesario situarlos en una localización adecuada.

Esto es apto al usuario, pero en el caso de Ámesos, por organización, se han seguido los siguientes pasos:

El primer paso es crear una carpeta 'scripts' dentro del directorio del servidor local de Ámesos (/var/www/html/amesos/scripts). Será importante tener en cuenta la dirección relativa desde Ámesos a dicha carpeta, puesto que es la carpeta raíz de la aplicación (la dirección relativa sería /scripts).

Seguidamente, y por cada script creado, es **imprescindible otorgar permisos 'rwx'**[5]. Siguiendo la estructura de directorios mencionada y los permisos otorgados inicialmente en la creación del servidor, se pueden ejecutar los siguientes comandos desde una terminal abierta en el directorio del servidor.

```
1 $ sudo chown -R www-data scripts
2 $ sudo chmod -R 777 scripts
```

[5]Los permisos 'rwx' hacen referencia a los permisos de lectura, escritura y ejecución (**r**ead, **w**rite y e**x**ecute)

Apéndice B

Anexo II: Manual de usuario

Para explicar el uso y funcionamiento de la aplicación se tendrá en cuenta que los centros de procesamiento de datos están previamente configurados para su uso, y que los datos requeridos para conectar a ellos (IP, rango de puertos, etc.) están a punto. En caso contrario, lo primero será realizar dicha configuración para poder ponerse en funcionamiento.

Acceso a la aplicación

Para entrar a la aplicación, lo primero es acceder a su enlace web. Como es un servidor local, se accede mediante la URL **http://localhost/amesos**.

El siguiente paso es hacer *login*. Debe hacerse como superusuario, es decir, con unas credenciales que te acrediten como superusuario. De lo contrario, no podrá accederse a la aplicación web como medida de seguridad.

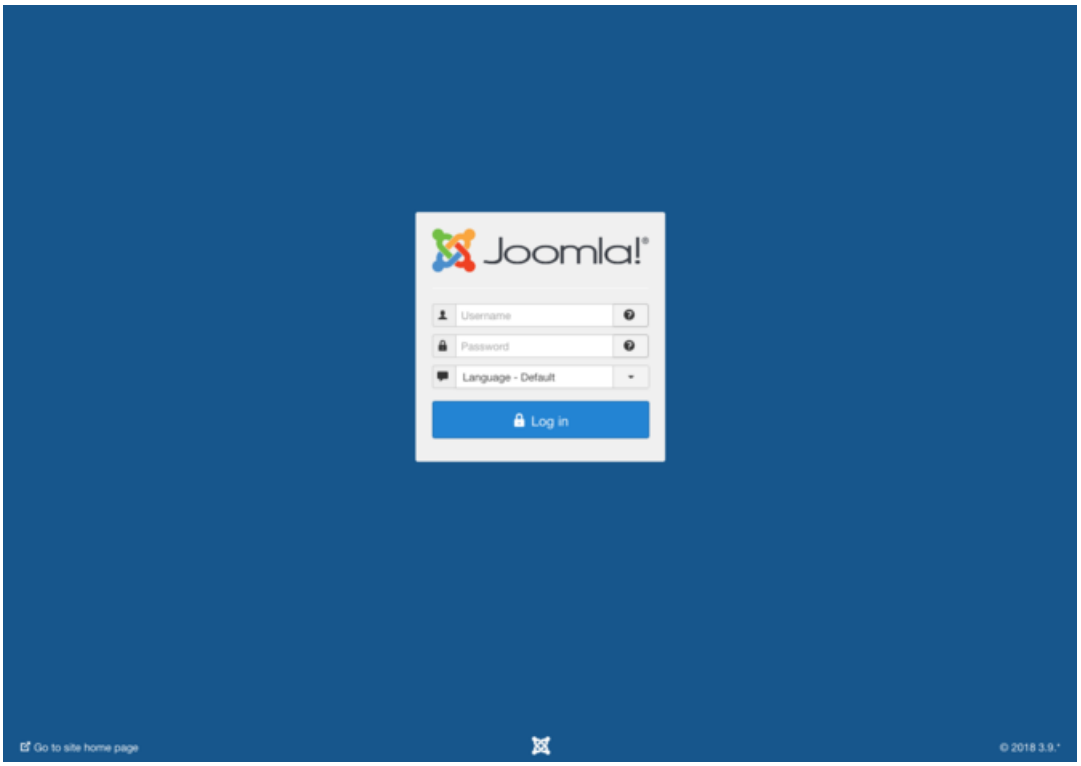


Figura B.1: Pantalla de login de Ámesos

Navegación por la interfaz web

Una vez dentro de la aplicación, el único acceso disponible es el desplegable en el que se hace referencia a los centros de procesamiento de datos disponibles **fig:4.6**.

Pulsar en cualquiera de los botones del desplegable llevará al usuario a la página correspondiente donde controlar y gestionar el centro de procesamiento de datos correspondiente.

Gestión del centro de procesamiento de datos

Dentro de las páginas que referencian cualquiera de los centros de procesamiento de datos la interfaz da al usuario toda la información necesaria a simple vista **fig:4.5**.

Los menús desplegables se siguen manteniendo en la parte superior de la ventana para poder navegar entre los distintos centros.

Justo debajo se encuentran los botones con los que se realiza todo el funcionamiento de la aplicación. Esta es la función que desempeña cada uno:

- **Info**
  Genera (o actualiza) un archivo JSON en el que se encuentra un listado tanto de IP como puerto de los servidores encendidos actualmente en el centro en cuestión. Con estos datos es posible conectarse a dichos servidores.

- **Encender**
  Enciende un nuevo servidor dentro del centro de procesamiento de datos.
- **Apagar**
  Apaga el servidor cuyo uso sea menor. Es decir, de esta manera comenzará apagando aquellos servidores actualmente inactivos y que no están realizando ningún trabajo.

Información de los servidores

En la parte central y en contraste con el resto de elementos gráficos se encuentran los datos de cada servidor encendido **fig:4.8**.

Cada caja hace referencia a un servidor. Por cada servidor se muestran los siguientes datos:

- **CPU:** Uso o carga de la CPU de la máquina en ese momento.
- **Memory:** Uso o carga de la memoria RAM de la máquina en ese momento.
- **Time up:** Tiempo (en minutos) que lleva encendida la máquina.
- **Cost:** Coste total acumulado (en dólares) por el uso de la máquina.

Apéndice C

Anexo III: Aportaciones de cada miembro

Como se ve claramente, este proyecto se puede seccionar de manera sencilla en dos bloques muy diferenciados, siendo uno de ellos todo lo referente a los **centros de procesamiento de datos** y el otro a todo lo referente **al sistema de gestión de contenidos**.

Debido a que este proyecto lo componen dos estudiantes y es fácilmente dividible, cada uno de ellos se ha encargado de un bloque en concreto. **Mario de los Santos Sainz** del bloque del centro de procesamiento de datos y **Raúl Fernández Guardia** del bloque del sistema de gestión de contenidos. A continuación se van a detallar las aportaciones de cada uno de ellos.

C.1. Aportaciones de Mario de los Santos Sainz

Para empezar, investigué cual sería el sistema operativo óptimo donde alojar todos estos centros de procesamiento de datos. Tras finalizar con la idea de que **Linux**, más concretamente la partición de **Ubuntu** era la mejor opción, creé una partición en mi portátil que funcionaría como centro de procesamiento de datos donde instalé este sistema operativo.

Tras esto, tenía que escoger un hipervisor que usaría para la creación de las máquinas virtuales. Después de investigar varias opciones, la plataforma que usé fue **VirtualBox**.

Bien, ahora ya tenía tanto el sistema como la plataforma que alojarían nuestros servidores, por tanto me dispuse a investigar que tipo de máquina virtual usaría para alojar el servidor de juego. Tras varias pruebas, me quedé con **la imagen virtual de Ubuntu 20.04**.

El siguiente paso era buscar un juego válido cuyo servidor se podía alojar fácilmente en la máquina virtual instalada anteriormente. Aprovechando que estábamos en un sistema Linux, busqué un juego que viniese con el propio sistema operativo, el juego escogido fue **Minetest**.

Lo siguiente era conseguir que tanto los centro de procesamiento de datos como la máquinas virtuales que corren en ellos fueran visibles para el sistema de gestión de contenidos. Por tanto, por una parte modifiqué la configuración de mi router para abrir una serie de puertos que se usarían para comunicar ambas plataformas y por otra parte, configuré tanto el **firewall** del CPD como el de la máquina virtual para que la conexión fuera posible.

Más tarde y conjuntamente a mi compañero Raúl, me encargué de conectar la aplicación web con el centro de procesamiento de datos mediante protocolo SSH.

Después, creé un archivo formato JSON llamado **datos.json** que usaría el sistema de gestión de contenidos para conocer todas las características de todos los centro de procesamiento de datos así como de las máquinas virtuales que corrían en ellos.

Con toda esta base, el sistema de gestión de contenidos ya podía mostrar características de los centros de procesamiento de datos. Ahora tenía que buscar una forma de poder modificar ese centro creando y destruyendo máquinas virtuales.
Lo primero fue establecer la máquina virtual instalada anteriormente como máquina base de la cual las demás partirían. Tras esto, busqué una forma de poder crear y destruir copias enlazadas de esa máquina base. Tras investigar, supe que el lenguaje **bash** era la mejor opción, por tanto implementé los scripts llamados **lanzarServidor.sh**, **encender.sh**, **apagarServidor.sh** y **apagar.sh**.

El siguiente paso fue la monitorización de estas máquinas. Lo primero fue investigar que atributos serían relevantes para el cliente. Al final estos fueron el porcentaje de CPU y memoria RAM así como el tiempo que llevaba la máquina arrancada y el precio de esta. Por tanto implementé los scripts llamados **datos.sh** y **actualizarTiempoYCoste.sh**.

Para finalizar, había que pensar una forma para mostrar los servidores disponibles y los datos requeridos para conectarse a los mismos a nuestros clientes, por lo que implementé el script **jsonInformativo.sh**.

C.2. Aportaciones de Raúl Fernández Guardia

En un comienzo, investigué los frameworks o plataformas más óptimas para la construcción de la aplicación base. Tras concretar que la mejor opción era utilizar un **CMS**, analicé entre los más destacados del mercado para decantarme por **Joomla**.

Para construir las bases del CMS, analicé el **S.O.** más adecuado sobre el que realizar el montaje pertinente, quedándome con el sistema **Linux** por concordancia con el resto de partes del proyecto.

Seguidamente, construí el **servidor local** y la **base de datos** necesaria para el funcionamiento de la aplicación web.

Una vez construidas las bases de la web, investigué acerca de las funcionalidades ofrecidas por Joomla más adaptadas a nuestros objetivos. Busqué **librerías** que facilitaran tanto la construcción de la interfaz como la implementación del *backend* que correría por debajo.

Al mismo tiempo, me informé sobre diversos fundamentos de programación web y de los lenguajes usados por esta rama. Adapté la búsqueda lo más concretamente posible a lo requerido en nuestro proyecto, por lo que me centré en aprender sobre lenguaje **HTML** y **CSS** para las bases del diseño, y en **Javascript** y **PHP** para la implementación más pura de los scripts que serían ejecutados en el servidor.

Para la construcción de scripts funcionales ejecutados localmente, creé la carpeta 'scripts'

dentro de los datos del servidor y configuré los **permisos** necesarios sobre los archivos ahí contenidos para poder ejecutarse correctamente por la propia aplicación.

A continuación, y conjuntamente a mi compañero, me encargué de conectar la aplicación web con el centro de procesamiento de datos mediante protocolo SSH. Gracias a la investigación y uso de librerías externas, realicé scripts PHP capaces de ejecutar comandos en la máquina del centro de procesamiento de datos de manera remota mediante el protocolo mencionado. Estos scripts son **CPU.php**, **Mem.php**, **actualizar.php**, **encender.php** y **apagar.php**.

Ya alcanzado el punto en el que teníamos una conectividad remota funcional, comencé con el **diseño** de la interfaz web y el añadido de los elementos gráficos esenciales. Creé botones para las funciones requeridas, así como los menús correspondientes por los que el usuario sería capaz de navegar dentro de la aplicación.

Ya con la estructura definida, implementé los scripts base de la aplicación para realizar toda funcionalidad especificada. Creé scripts de control para los botones y la ejecución de sus funciones concretas, a su vez que scripts para la lectura, actualización y gestión de los datos de cada servidor lanzado. Estas funciones son **buttonController()**, **updateData()** y **readTextFile()**, incluidas en el código HTML del módulo que corre en cada página web de los centros de procesamiento de datos.

Por último, realicé los últimos acabados de diseño para la muestra de información de los datos de cada centro de procesamiento y reestructuré la interfaz para comodidad de visualización del cliente de los elementos gráficos mostrados.

Índice de figuras

PASCAL
ENERO 2018
Ult. actualización 23 de junio de 2020
LaTeX lic. LPPL & powered by TEFLON CC-ZERO