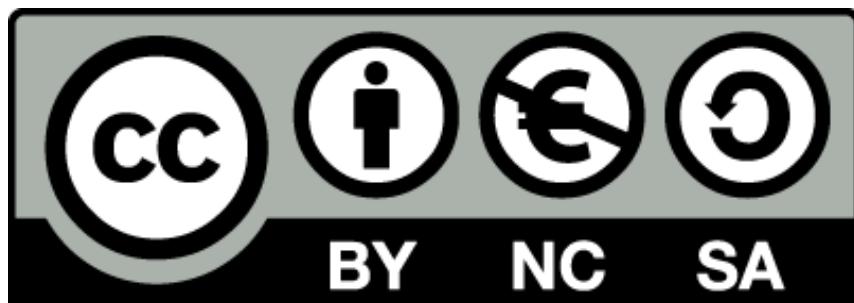


LUIS DIÉGUEZ BARRO

GRADO SUPERIOR ADMINISTRACIÓN DE SISTEMAS
INFORMÁTICOS Y EN RED

LICENCIA

Esta obra está bajo una licencia Creative
Commons Reconocimiento Internacional Público 4.0 ©
Atribución-NoComercial-CompartirlIgual
CC BY-NC-SA



ÍNDICE

1. Introducción	5
2. Funcionamiento	7
3. Tecnologías	9
4. Realización	
4.1. Instalación de Proxmox	13
4.1.2 Seguridad en Proxmox	23
4.2. Instalación de VPN	34
4.3. Instalación de Rancher	39
4.4. Instalación de Kubernetes	46
4.5. Creación de Docker	57
4.5.1 Servicio gestor Gcloud	62
4.5.2 Servicio bot Instagram	65
4.5.3 Servicio bot Twitter	67
4.5.4 Servicio Minecraft	68
4.5.5 Servicio Nextcloud	70
4.6. Automatizar GitHub Actions	72
4.6.1 Añadir secretos a GitHub	76
4.6.2 Actions gestor Gcloud	77
4.6.3 Actions bot Instagram	79
4.6.4 Actions bot Twitter	81
4.6.5 Actions juego Minecraft	83
4.6.6 Actions Nexcloud y MySQL	85
4.6.7 Resumen GitHub Actions	87
4.7. Levantando en Kubernetes	88
4.7.1 Kubernetes Ingress Traefik	89
4.7.2 Kubernetes gestor Gcloud	90
4.7.3 Kubernetes juego Minecraft	92
4.7.4 Kubernetes bot Instagram	99
4.7.5 Kubernetes bot Twitter	101
4.7.6 Kubernetes Nexcloud y MySQL	105
4.8. Comprar dominio	115
4.8.1 Utilizar CDN y Anti-DDOS	116
4.8.2 Almacenamiento página web	118
4.9. Páginas web	119
4.9.1 Subdominio con Wordpress	120
4.9.2 Plugins Wordpress	121

4.9.3 Código Wordpress	122
4.9.4 Dominio principal NodeJS	137
4.9.5 Código NodeJS	138
5. Presupuesto	151
6. Conclusiones	153
7. Bibliografía	155

1. INTRODUCCIÓN

"La Web como la imaginé, aún no la hemos visto. El futuro es aún mucho más grande que el pasado."

Tim Berners-Lee, inventor de la World Wide Web.

Y es que en Internet cada año, mes e incluso diariamente cambia. Desde 2015 el estándar de desarrollo de software se basa en **microservicios** en vez de las clásicas aplicaciones **monolíticas**.

Los **sistemas monolíticos** son los que agrupan el código de toda la aplicación en un mismo servicio o lugar. Esto da lugar a que el desarrollo inicial sea rápido y también fácil de desplegar.

Voy a describir la vida de una aplicación usual: Las aplicaciones no suelen ser estáticas, sino dinámicas. Una vez las creas les vas añadiendo poco a poco funcionalidades extra que la mejoran. Con el tiempo una aplicación crece en número de usuarios y eso significa tener que aumentar los recursos. También si avanza correctamente necesitarás mayor número de desarrolladores. Lo que acabara en una aplicación compleja. Lo que no es para nada negativo.

Pero este tipo de desarrollo de software tiene varias desventajas importantes. Con este estilo de desarrollo monolítico va a ser muy complicado añadir nuevo código. Al estar todo en un mismo lugar los distintos desarrolladores, especializados cada uno en un área distinta de la aplicación, van a estar editando lo mismo. Esto hace que el desarrollo sea muy lento. Y sobre todo puede desembocar en graves errores que van a ser difícil de identificar y subsanar.

La arquitectura para diseñar aplicaciones monolíticas puede parecer positiva a corto plazo pero a largo plazo se hace insostenible.

Los **sistemas de microservicios** se basan en dividir una aplicación en servicios individuales y totalmente independientes de unos de otros pero al mismo tiempo que se puedan comunicar entre ellos. Así, con esta arquitectura cada desarrollador o equipo puede trabajar en su propio servicio y luego intercomunicarlos. Incluso puede tener componentes con diferentes lenguajes de programación. Aunque es un gran avance respecto a la anterior arquitectura, permiten una gran flexibilidad, agilidad y posibilidad de crecimiento a gran escala, tiene una gran complejidad y no es sencillo de implementar.

Por ello es necesario que los desarrolladores conozca las prácticas de desarrollo ágil. Estos métodos brindan soluciones a través de la colaboración entre equipos auto-organizados y multifuncionales. Y que el administrador de sistemas tenga amplios conocimientos sobre DevOps ya que es el encargado del despliegue, actualizaciones, mantenimiento y monitorización de la aplicación.

LUDIBA GROUP es una empresa desarrolladora de servicios. Cada uno de los servicios se van a basar en la arquitectura de microservicios para que puedan ser fácilmente escalable según pase el tiempo. Me voy a centrar en el desarrollo de servicios orientados al *hosting*. El *hosting*, es una palabra inglesa que se traduce como hospedaje, siendo el servicio en Internet más utilizado ya que cualquier recurso en la Web necesita ser almacenado para que posteriormente los usuarios puedan tener acceso a él.

En LUDIBA GROUP ofrecemos los siguientes servicios especializados e individuales:

1. Hostería de páginas web completas
2. Hostería de servidores de juegos
3. Hostería de vídeo
4. Hostería de bots de redes sociales
5. Hostería de archivos

Todos los servicios necesitan de servidores para funcionar. En este proyecto he elegido utilizar los servidores en la nube (Cloud). Utilizar estos servidores tiene grandes ventajas para la empresa:

- **Pago por consumo:** No necesitas comprar todos los componentes de un servidor, solo pagas por el uso.
- **Obsolescencia:** Los ordenadores tienen una vida útil. Cuando ese tiempo pasa hay que comprar nuevos equipos. Al utilizar Cloud eso no ocurre ya que no compras equipo, si no horas de computo.
- **Flexibilidad:** Al no tener que comprar el equipo físico no estás limitado por los propios recursos. Si necesitas más memoria RAM, direcciones IP, CPU u otro recurso lo tienes al instante.
- **Seguridad:** Los servidores en Cloud tienen estrictas reglas de seguridad y permiten configurar y limitar los permisos de cada persona.
- **Refrigeración:** No te tienes que preocupar de crear una sala acondicionada específicamente para los servidores.
- **Escalabilidad vertical:** Puedes aumentar el hardware individual en el momento.
- **Escalabilidad horizontal:** Permite la creación de clusters en el momento.
- **Failover:** Podemos añadir alta disponibilidad tanto a las redes como a las máquinas individuales o a los clusters.

En la empresa contamos con un pequeño servidor local el cual voy a utilizar. Primero lo virtualizaré y posteriormente instalaré un sistema de administración para poder controlar los servidores del Cloud. También creare un sistema de acceso privado para aumentar la seguridad general.

Por último, todo el código va a estar en un repositorio. Esto es para que si en algún momento hay más desarrolladores puedan acceder, crear más código y realizar cambios de forma sencilla.

2.

FUNCIONAMIENTO

Como en el anterior punto he especificado, **LUDIBA GROUP** es una empresa de desarrollo especializada en servicios. En este proyecto voy a realizar todo el ciclo de una aplicación desde cero.

Comienza con el desarrollo del código de la plataforma desde la que los usuarios pueden acceder, visualizar e interactuar con los distintos servicios que LUDIBA GROUP según se van desplegando.

La **plataforma** esta dividida en dos partes diferenciadas. Ambas partes de la plataforma están alojadas en BanaHosting, una empresa de servidores, en la que he alquilado un servidor dedicado.

La primera parte es la **página principal** (ludiba.org) y las páginas accesibles desde el menú. Esta basada en **NodeJS**, un entorno basado en **ECMAScript**. He creado tanto la infraestructura de la aplicación web con **Express**, el esqueleto de la página en **HTML5**, el diseño de la página en **CSS3** y el dinamismo e interacción con las APIs con **Javascript**.

La segunda parte es el **blog** (blog.ludiba.org). En el se van a ir subiendo noticias relacionadas con la empresa. Esta basada en **Wordpress**. He creado desde 0 con **PHP** un tema y sus estilos. Esta conectada con una base de datos **MySQL** para guardar cada entrada.

La plataforma almacenada en los servidores de BanaHosting esta protegida con una **CDN** (Content Delivery Network). Esta red lo que permite es replicar automáticamente nuestros archivos de la plataforma en servidores de todo el mundo. Así, aunque nuestro servidor esté en Europa, cualquier usuario por muy lejos del servidor que esté pueda acceder a la plataforma de forma rápida y segura. Para implementar la CDN he utilizado **CloudFare**.

Los **servicios** anteriormente mencionados estarán alojados en **Google Cloud**.

El despliegue de estos servicios va a ser con **Kubernetes**. Los servicios van a estar creados en contenedores utilizando **Docker**.

Luego la **administración** del cluster va a ser desde el panel de control de **Rancher**. Este panel va a estar en el servidor local virtualizado con **Proxmox**. Al cual solo es posible acceder con una **VPN** creada en una **Raspberry Pi**.

Por último, la **automatización** de la creación de las imágenes para los contenedores se va a realizar desde **Github Actions**. Todo el código esta público y disponible en el repositorio de **Github**. Una vez se crean las imágenes automáticamente el día 1 de cada mes, se suben a **Docker Hub**.

3.

TECNOLOGÍAS

Al ser un proyecto tan extenso, ya que trata todo el ciclo de creación, es muy importante definir correctamente las tecnologías a utilizar. También es importante definir las herramientas para gestionar el trabajo: en este caso he utilizado **Trello** y **Todoist** para hacer la administración de tareas y **Visual Studio Code** como IDE por su gran cantidad de extensiones, lo que facilita el trabajo.

Como mencione anteriormente he utilizado dos tipos de tecnologías diferentes para la plataforma por distintas razones.

La parte principal esta creada en **NodeJS** ya que a diferencia de **PHP**, **NodeJS** es asíncrono. Que sea asíncrono mejora el tiempo de respuesta de la página ya que no bloquea la conexión esperando un recurso y los va cargando según se van ejecutando. Esto reduce considerablemente el consumo de recursos del servidor, y menos recursos consumidos significa menos dinero que hay que pagar por conexión. Además NodeJS tiene una gran comunidad. A día de hoy hay disponible de forma Open Source 1.3 millones de paquetes que podemos utilizar en el proyecto. Cada paquete lo podemos utilizar en el proyecto como una extensión para añadir y desarrollar nuevas funcionalidades.

La parte del blog esta creada en **PHP**. Este lenguaje de programación tiene una mejor comunicación con las bases de datos SQL como **MySQL**. Y soporta varios CMS (Sistemas de gestión de contenidos) como **Wordpress**, el cual he utilizado. **Wordpress** es un motor de creación de páginas dinámicas. Este CMS guarda los datos en la base de datos SQL y luego realiza llamadas a esos datos en las páginas. Es una ventaja que guarde todos los datos en una base de datos en vez en texto plano.

Cualquier página necesita una **CDN**. Y **CloudFlare** es la empresa que ofrece mejor CDN de forma gratuita. Las claves de que 26 millones de clientes la utilicen son las siguientes. Utiliza inteligencia artificial y aprendizaje automático para detectar la procedencia del usuario y redirigirla al servidor más cercano, así el usuario tiene una carga rápida y muy baja latencia en su conexión. Tienen más de 200 centros de datos a lo largo de todo el mundo. Ahorra ancho de banda a nuestro propio servidor ya que los archivos se guardan en su propio caché. Al ser un sistema de red en la nube, nos permite redirigir nuestro tráfico hacia su propio certificado SSL, así nunca más tendremos que pagar un certificado o renovarlo cada cierto tiempo con Let's Encrypt. Nos permite utilizar los últimos estándares de red como HTTP/2 y TLS 1.3.

En los **servicios** he creado una arquitectura real. Para ello me he basado en las últimas tecnologías que las empresas utilizan. El resultado es que el 89.7% de las empresas utilizan contenedores según la analítica 2019 de portworx.

Pero, ¿qué son los **contenedores**?

Un contenedor se basa en la virtualización ligera. Esto es cuando no necesitas un sistema operativo completo. Solo una aplicación en concreto. Como por ejemplo MySQL o Apache. **Docker** automatiza la creación de esos contenedores.

Como ejemplo: Una máquina virtual tiene que replicar todo un sistema operativo para utilizar una aplicación. Con Docker solo necesitas las librerías necesarias para esa aplicación.

Una **máquina virtual** en la que queremos instalar Apache, primero necesitamos instalarnos todo Debian + las librerías propias de Apache. El total da más de 2GB. Y eso si nos descargamos la versión Lite, que si tenemos la versión Full serían 5GB.

Un **contenedor Docker** de Apache ocupa menos de 100 MB ya que solo instala las librerías. Y es un Apache totalmente funcional y completo.

Para crear un contenedor de Docker hay que crear un archivo llamado **Dockerfile** donde van a estar las características, comandos y especificaciones del propio contenedor. Luego creamos el contenedor y podemos subir esa imagen que acabamos de crear a un repositorio de imágenes. **Docker** ofrece un repositorio público llamado **Docker Hub** el cual voy a utilizar.

Bien. Ya tenemos nuestro servicio en un contenedor. Ahora necesitamos una tecnología que nos permita administrar esos contenedores. Esa tecnología se llama **Kubernetes**. Kubernetes nos va a permitir abstraer todos los elementos necesarios para los contenedores. Además nos va a permitir definir el despliegue de los contenedores, el tipo de escalado y en general el manejo. Kubernetes está integrado por todos los principales proveedores de servidores cloud. Y también se puede instalar en cualquier otro tipo de servidor.

He elegido utilizar el proveedor **Cloud Google** (GCP) por cuatro motivos. El primer motivo es que dan una prueba gratuita de 200\$ en todos sus servicios, a diferencia de otros proveedores cloud como Azure que solo dura 30 días o Amazon Web Services que te limita a los servicios de su capa gratuita. El segundo motivo es que su SDK y panel son muy intuitivos. El tercer motivo es que Google fue el desarrollador de Kubernetes, así que siempre va a estar actualizado con las últimas novedades. El cuarto motivo es que Google Cloud utiliza un 100% de energía renovable, a diferencia de Amazon Web Services que es de las empresas que mayor huella de carbono tiene en el planeta.

La administración como ya mencioné, la voy a hacer con **Rancher**. Rancher es un proyecto Open Source que nos permite administrar multi-cluster y con lo que vamos a poder desplegar una nube híbrida.

Nos proporciona su propio panel desde el cual podemos administrar permisos para los diferentes niveles de usuarios que puedan administrar el cloud. Y tiene integración con Active Directory así que no habría que crear nuevos usuarios, solo conectarlos.

El controlador de Rancher no va a estar en ningún servidor cloud. Va a estar en un servidor local que he virtualizado con **Proxmox**. Proxmox es un sistema operativo Open Source que nos permite virtualizar máquinas virtuales y contenedores LXC. Es un hipervisor de tipo 1, lo que significa que es nativo, que se va a ejecutar directamente sobre el hardware.

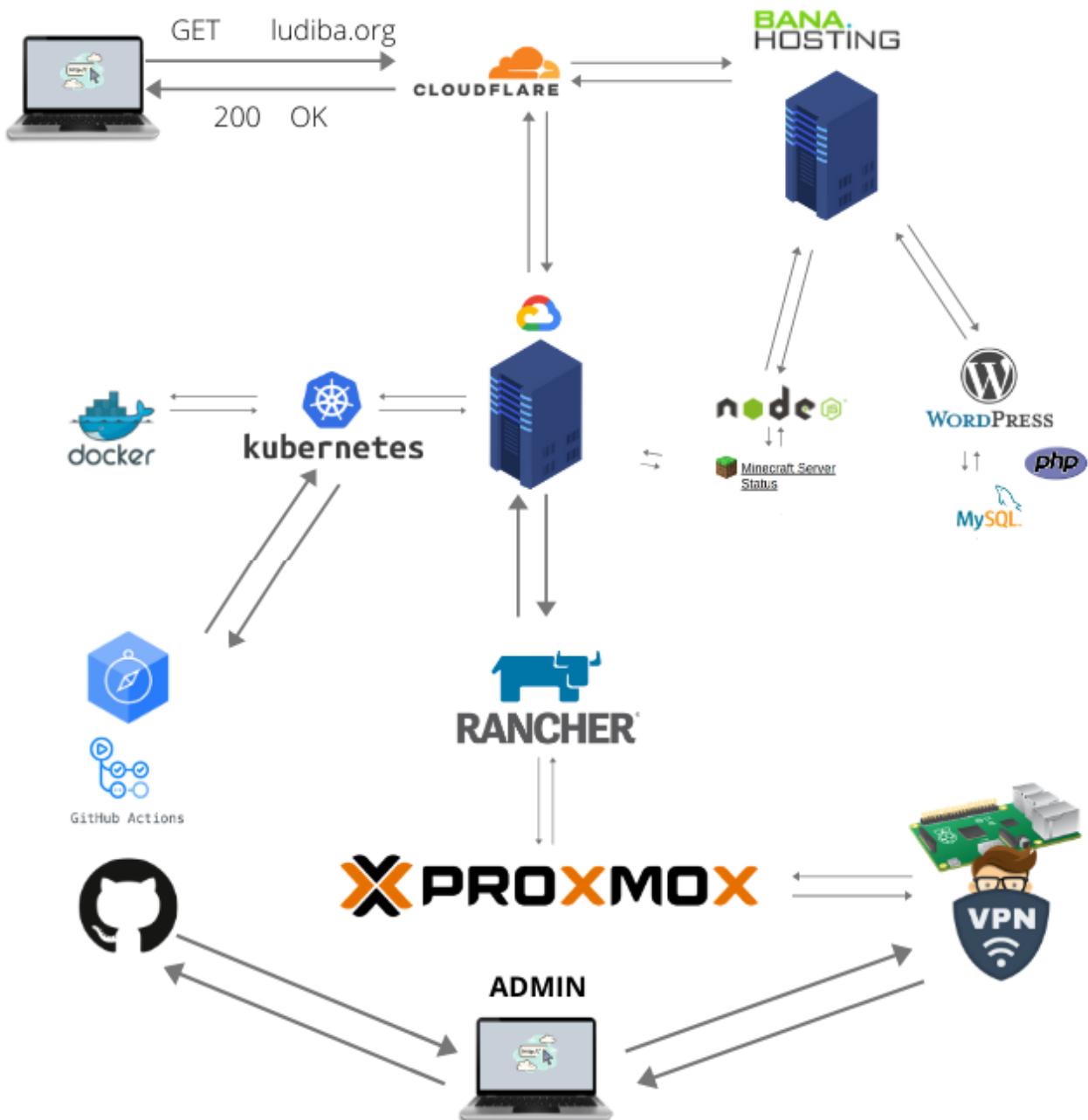
A diferencia de los hipervisores de tipo 2 como VirtualBox, los de tipo 1 son los recomendados por su mayor rendimiento ya que actúan directamente como sistema operativo y no como aplicación.

El acceso a la red donde va a estar el servidor local virtualizado con Proxmox que va a contener Rancher lo voy a realizar con una **Raspberry Pi** que va a crear una **VPN** (red privada virtual). He elegido esta opción por ser la más segura ya que no hay que exponer la red entera, solo crear un acceso privado. El servidor VPN va a ser **OpenVPN**, que al igual que todas las tecnologías usadas en este proyecto es Open Source.

Este protocolo de VPN puede configurarse tanto para que la conexión se realice por el protocolo de capa de transporte TCP como por el protocolo de capa de transporte UDP. He elegido TCP ya que este protocolo espera la confirmación antes de enviarlo nuevamente y aunque no es tan rápido como UDP es más importante que la transferencia de datos sea completa. OpenVPN usa encriptación OpenSSL de 256 bits.

Por último todo el código, tanto de cada servicio individual, como de la página principal como del blog va a estar público en un repositorio de **Github**. He elegido este almacenamiento de código ya que permite tener ilimitados repositorios, tanto privados como públicos. Además recientemente incluyó una herramienta llamada **Github Actions** que nos permite automatizar la creación de las imágenes para los contenedores desde el propio repositorio, sin necesidad de recurrir a herramientas de terceros como Jenkins o CircleCI. Además este tipo de repositorios permiten la interacción con otros usuarios. Cualquier persona puede realizar su propia rama y realizar modificaciones. La creación de las imágenes las he programado con un comando CRON para que se creen y se suban al repositorio de imágenes de DockerHub cada primer día de cada mes. Lo podía haber programado para que solo se creen las imágenes una vez yo actualice manualmente el repositorio pero la idea del proyecto es que por mucho tiempo que pase, el proyecto siga funcionando sin necesidad de atención.

Como todo esto puede sonar un poco abstracto, ya que así es, he creado una imagen explicativa para mostrar el ciclo del proyecto.



4.1 INSTALACIÓN DE PROXMOX

Proxmox VE es una plataforma de virtualización completa, de nivel empresarial, que está desarrollada íntegramente en código abierto. Está basado en el hipervisor KVM y en contenedores LXC, el almacenamiento y la funcionalidad de red se definen por software. Además, permite administrar fácilmente clusters para alta disponibilidad e incluye herramientas para la recuperación de desastres desde una cómoda interfaz web.

En la empresa tengo un servidor local con las siguientes características físicas:

- Placa madre (motherboard): Supermicro X8DT3
- Procesador (CPU): Intel Xeon X5680 (x2 Sockets)
- Memoria (RAM): 16GB DDR3 ECC REG
- Disco Duro (SSD): KINGSTON SA400S3 240GB
- Fuente de alimentación (PSU): EVGA 750 G3 Gold
- Ventiladores: ARTIC Alpine 11 PLUS (x2)
- Gabinete: NEO R810 4U

Aplicando los conocimientos de la asignatura de Fundamentos de hardware monté el servidor en el gabinete correctamente. Luego aplicando los conocimientos de la asignatura Administración de sistemas operativos instalé Proxmox. Por último apliqué los conocimientos de Seguridad y alta disponibilidad.

Para la instalación del sistema operativo seguí los siguientes pasos:

El primer paso es descargar la imagen (ISO) del sistema operativo de Proxmox. La imagen está en la página oficial de Proxmox: <https://www.proxmox.com/en/downloads>

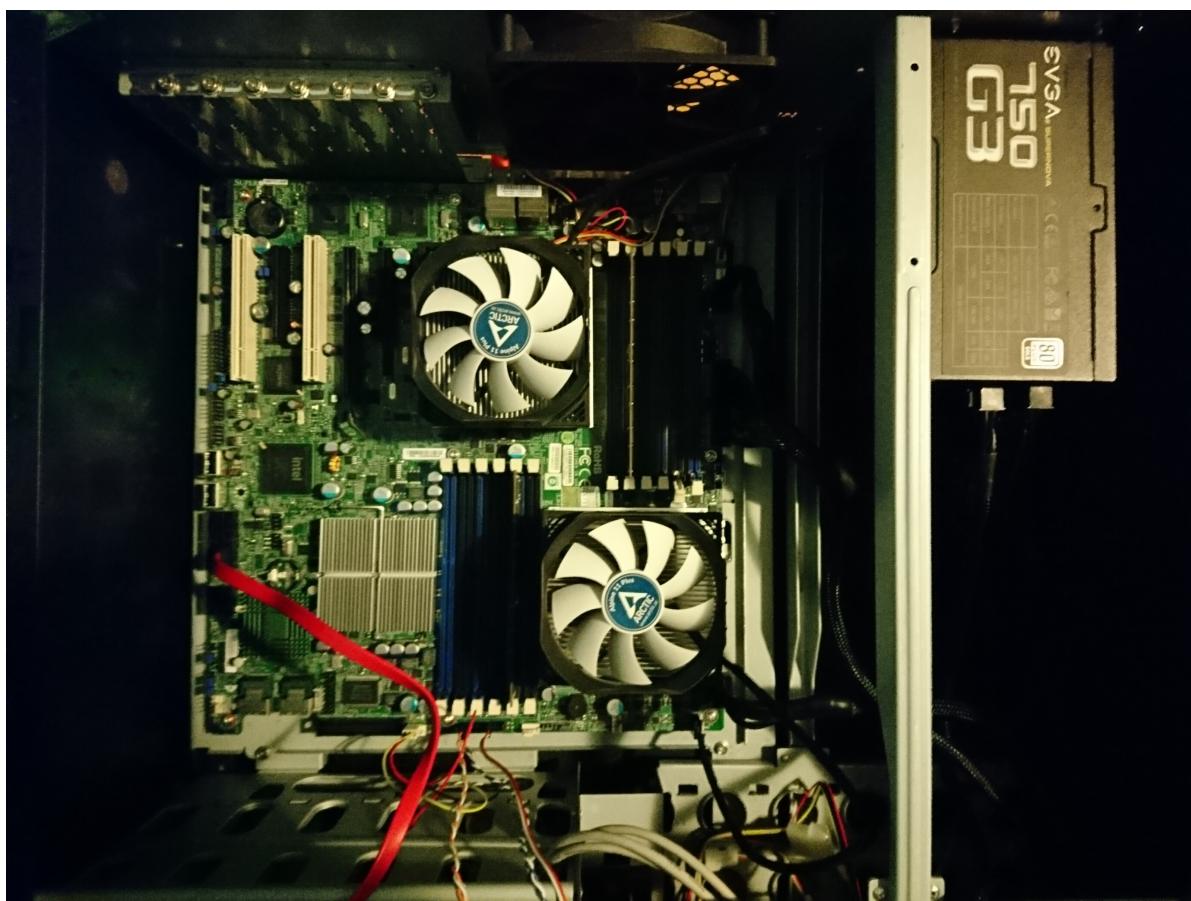
La última versión actualmente es 6.1-2. La imagen tiene un peso de 850 MB.

Una vez descargado el segundo paso es grabar la imagen del sistema operativo en un periférico externo, como por ejemplo una memoria USB. Para grabar la imagen es necesario utilizar un programa de creación de dispositivos de arranque. En esta ocasión he elegido ETCHER ya que aumenta considerablemente la velocidad de creación. Así que en su página oficial lo descargamos: <https://www.balena.io/etcher/>

Seguimos el menú de instalación. Una vez se instale podemos utilizarlo. Su uso es muy sencillo. Tan solo tenemos que seleccionar la imagen ISO de Proxmox que previamente he descargado y seleccionar el USB para crear el dispositivo de arranque. Cuando termine el proceso de creación ya tendremos listo nuestro USB de arranque con Proxmox.

El tercer paso es cambiar la BIOS del servidor para que podamos ejecutar el USB antes que el disco duro. También comprobar en la BIOS que tenemos activada la aceleración por hardware para poder virtualizar.

Resultado del montaje:



Una vez este todo listo, el último paso es instalar Proxmox. Solo queda introducir el USB y proceder a la instalación. Lo primero que aparecerá será el menú para seleccionar el tipo de instalación. Da a elegir entre cuatro modos.

- **Install Proxmox VE**: Inicia la instalación normal.
- **Install Proxmox VE (Debug mode)**: Inicia la instalación en modo de depuración. Se abre una consola en varios de los pasos de la instalación. Esto se usa para reparar si algo sale mal, por ejemplo para reparar el gestor de arranque de una configuración existente de Proxmox VE.
- **Rescue Boot**: Con esta opción, inicia una instalación existente. Busca en todos los discos duros adjuntos.
- **Test Memory**: Esto es útil para verificar si la memoria es funcional y libre de errores. Utiliza memtest86+

Seleccioné **Install Proxmox VE** ya que es la instalación normal.



Una vez seleccionado, aparecerá el acuerdo de Licencia con el Usuario Final (EULA). Es un contrato entre el fabricante y/o autor del software, y el usuario final del mismo. Los EULA protegen y comprometen a ambas partes (usuario y productor y/o autor).

Tiene siete puntos. En el primer punto Proxmox se compromete a dar licencia de su programa de forma permanente y global basándose en *GNU General Public License*, versión 3. Esta licencia permite la modificación del código fuente pero esa persona tendrá que hacer su código fuente también público. En el segundo punto expone que Proxmox no tiene garantía de ningún tipo. En el tercer punto que ni la empresa ni sus afiliados son responsables de daños en los equipos. En el cuarto punto declara sus propios derechos intelectuales, que todo aquello bajo su marca registrada Proxmox no permite su redistribución. El quinto punto aclara que puede incluir programas de terceros y que ellos están sujetos a sus propias licencias. En el sexto punto expone que el país de desarrollo es Austria y que el programa esta sujeto a la administración y regulación de dicho país. Por último el séptimo punto menciona que si hay alguna disputa debe ser juzgado de acuerdo a las leyes del gobierno austriaco.



END USER LICENSE AGREEMENT (EULA)

END USER LICENSE AGREEMENT (EULA) FOR PROXMOX VIRTUAL ENVIRONMENT (PROXMOX VE)

By using Proxmox VE software you agree that you accept this EULA, and that you have read and understand the terms and conditions. This also applies for individuals acting on behalf of entities. This EULA does not provide any rights to Support Subscriptions Services as software maintenance, updates and support. Please review the Support Subscriptions Agreements for these terms and conditions. The EULA applies to any version of Proxmox VE and any related update, source code and structure (the Programs), regardless of the delivery mechanism.

1. License. Proxmox Server Solutions GmbH (Proxmox) grants to you a perpetual, worldwide license to the Programs pursuant to the GNU Affero General Public License V3. The license agreement for each component is located in the software component's source code and permits you to run, copy, modify, and redistribute the software component (certain obligations in some cases), both in source code and binary code forms, with the exception of certain binary only firmware components and the Proxmox images (e.g. Proxmox logo). The license rights for the binary only firmware components are located within the components. This EULA pertains solely to the Programs and does not limit your rights under, or grant you rights that supersede, the license terms of any particular component.

2. Limited Warranty. The Programs and the components are provided and licensed "as is" without warranty of any kind, expressed or implied, including the implied warranties of merchantability, non-infringement or fitness for a particular purpose. Neither Proxmox nor its affiliates warrants that the functions contained in the Programs will meet your requirements or that the operation of the Programs will be entirely error free, appear or perform precisely as described in the accompanying documentation, or comply with regulatory requirements.

3. Limitation of Liability. To the maximum extent permitted under applicable law, under no

Abort

Previous

I agree

Las siguientes pantallas ya son específicamente sobre la instalación de Proxmox.

En la próxima pantalla determiné el disco duro donde se instalará Proxmox VE. Aparece una lista desplegable donde pude elegir entre todos los discos duros conectados al servidor. También como opciones estaban especificar el sistema de archivos, dejar una parte de espacio libre en el disco, especificar el tamaño de la memoria SWAP, el tamaño máximo de la partición inicial LVM, el espacio libre para la partición inicial LVM y el tamaño máximo para los datos de la partición inicial LVM.



Proxmox Virtualization Environment (PVE)

The Proxmox Installer automatically partitions your hard disk. It installs all required packages and finally makes the system bootable from hard disk. All existing partitions and data will be lost.

Press the Next button to continue installation.

- **Please verify the installation target**

The displayed hard disk is used for installation.
Warning: All existing partitions and data will be lost.

- **Automatic hardware detection**

The installer automatically configures your hardware.

- **Graphical user interface**

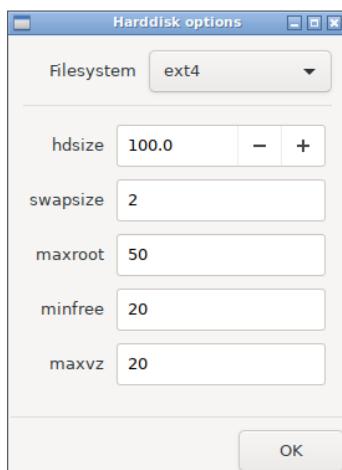
Final configuration will be done on the graphical user interface via a web browser.

Target Harddisk: /dev/sda (100GB, VBOX HARDDISK) Options

Abort

Previous

Next



Después, pude elegir la localización del servidor. Especifiqué el país, la zona horaria y el lenguaje del teclado. La zona horaria es importante ya que va a determinar la fecha y hora exactas del servidor y es vital para determinar las horas menos transitadas en las que se hagan las copias de seguridad.



Location and Time Zone selection

The Proxmox Installer automatically makes location based optimizations, like choosing the nearest mirror to download files. Also make sure to select the right time zone and keyboard layout.

Press the Next button to continue installation.

- **Country:** The selected country is used to choose nearby mirror servers. This will speedup downloads and make updates more reliable.
- **Time Zone:** Automatically adjust daylight saving time.
- **Keyboard Layout:** Choose your keyboard layout.

Country	Spain
Time zone	Europe/Madrid
Keyboard Layout	Spanish

Abort Previous Next

A continuación procedí a la creación del usuario root (administrador). Especifiqué la contraseña y un email de respaldo en caso de que se olvide la contraseña, ocurra otro tipo de error o alerta. Al ser la contraseña del usuario de administración del servidor es obligatorio que sea segura, por ello debe tener como mínimo 8 caracteres y combinar letras, números y símbolos.



Administration Password and E-Mail Address

Proxmox Virtual Environment is a full featured highly secure GNU/Linux system based on Debian.

Please provide the *root* password in this step.

- **Password:** Please use a strong password. It should have 8 or more characters. Also combine letters, numbers, and symbols.

- **E-Mail:** Enter a valid email address. Your Proxmox VE server will send important alert notifications to this email account (such as backup failures, high availability events, etc.).

Press the Next button to continue installation.

This screenshot shows the "Administration Password and E-Mail Address" configuration screen. It includes fields for "Password" (containing masked text), "Confirm" (also containing masked text), and "E-Mail" (set to "luisdieguezbarrio@gmail.co"). Below the fields are "Abort", "Previous", and "Next" buttons.

El último paso sería la configuración de la red en el servidor. Especifíqué la tarjeta de red que va a tener acceso al exterior e Internet, el nombre de dominio (*hostname FQDN*) local, la IP local, la mascara de red (*Netmask*), la puerta de enlace (*Gateway*) y los servidores DNS.



Management Network Configuration

Please verify the displayed network configuration. You will need a valid network configuration to access the management interface after installation.

Afterwards press the Next button. You will be shown a list of the options that you chose during the previous steps.

- **IP address:** Set the IP address for your server.
- **Netmask:** Set the netmask of your network.
- **Gateway:** IP address of your gateway or firewall.
- **DNS Server:** IP address of your DNS server.

This screenshot shows the "Management Network Configuration" screen. It displays network settings: Management Interface (selected as "enp0s3 - 08:00:27:c8:86:82 (e1000)"), Hostname (FQDN) ("luisdieguez.local"), IP Address ("192.168.1.21"), Netmask ("255.255.255.0"), Gateway ("192.168.1.1"), and DNS Server ("1.1.1.1"). Below the form are "Abort", "Previous", and "Next" buttons.

La siguiente pantalla es un resumen en el que podré comprobar todos los datos previamente introducidos.



Summary

Please verify the displayed informations. Once you press the **Install** button, the installer will begin to partition your drive(s) and extract the required files.

Option	Value
Filesystem:	ext4
Disk(s):	/dev/sda
Country:	Spain
Timezone:	Europe/Madrid
Keymap:	es
E-Mail:	luisdieguezbarrio@gmail.com
Management Interface:	enp0s3
Hostname:	luisdieguez
IP:	192.168.1.21
Netmask:	255.255.255.0
Gateway:	192.168.1.1
DNS:	1.1.1.1

Abort

Previous

Install

Una vez ya terminado el proceso de configuración tardará sobre diez minutos en instalar completamente el sistema operativo Proxmox VE. Al terminar, pulsaré sobre el botón reiniciar y ya podré acceder al menú web desde cualquier otro ordenador que este en la misma red y rango.



Virtualize your IT Infrastructure

Proxmox VE is ready for enterprise deployments.

The role based permission management combined with the integration of multiple external authentication sources is the base for a secure and stable environment.

Visit www.proxmox.com for more information about commercial support subscriptions.

- Commitment to Free Software**

The source code is released under the GNU Affero General Public License.

- RESTful web API**

Resource-oriented architecture (ROA) and declarative API definition using JSON Schema enable easy integration for third party management tools.

- Virtual Appliances**

Pre-installed applications - up and running within a few seconds.

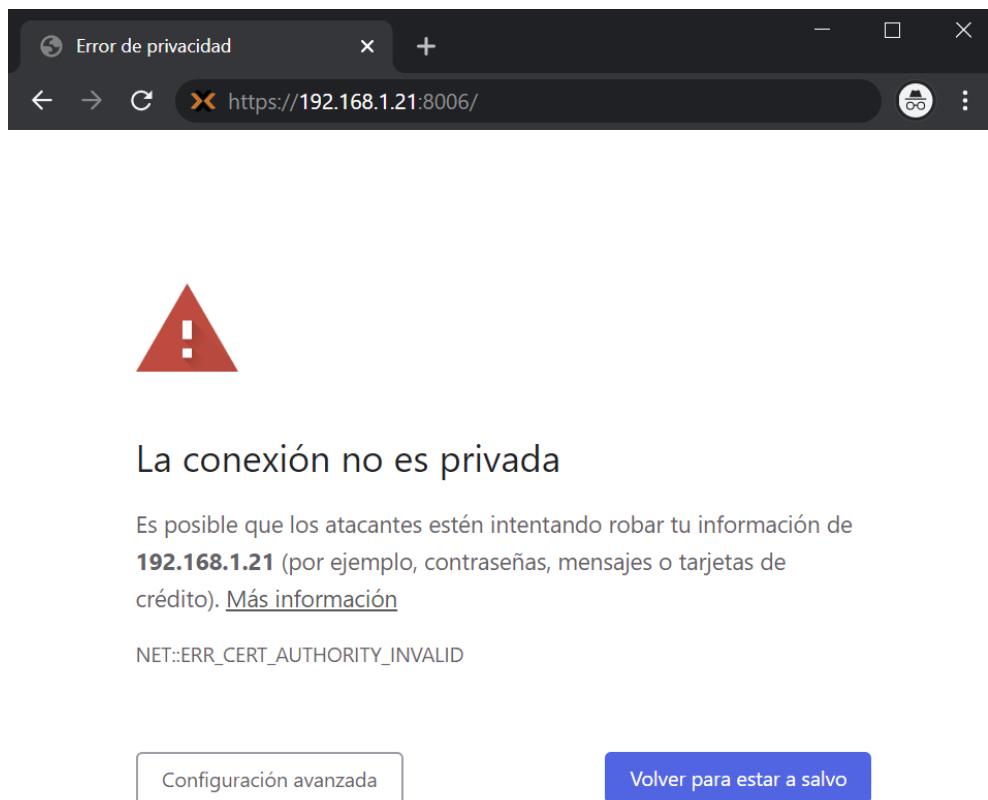
extracting base system

28%

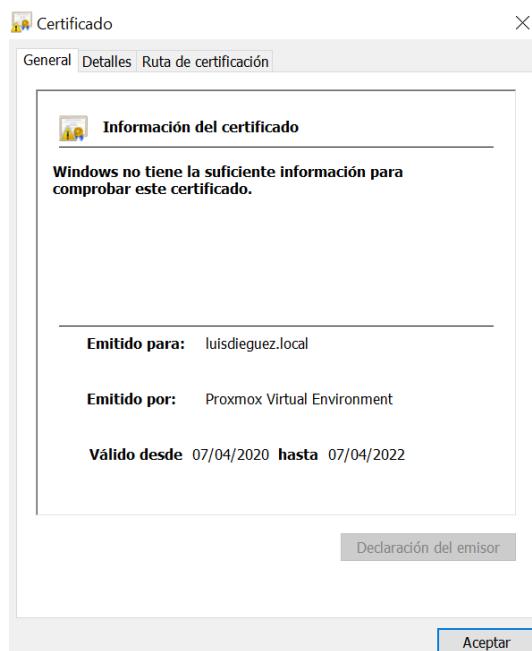
Abort

Install

En el servidor ya puedo quitar todos los periféricos, no es necesario que tenga ni ratón, ni teclado ni pantalla ya que toda la próxima configuración y gestión la voy a realizar desde su panel web. Accedo al panel web desde la IP especificada anteriormente y el puerto ocho mil seis. Utilizamos HTTPS ya que el propio Proxmox genera un certificado SSL. Al acceder puedo ver que el certificado al ser autofirmado y no validado por una tercera entidad de confianza por lo tanto nuestro navegador nos va a mostrar una advertencia.

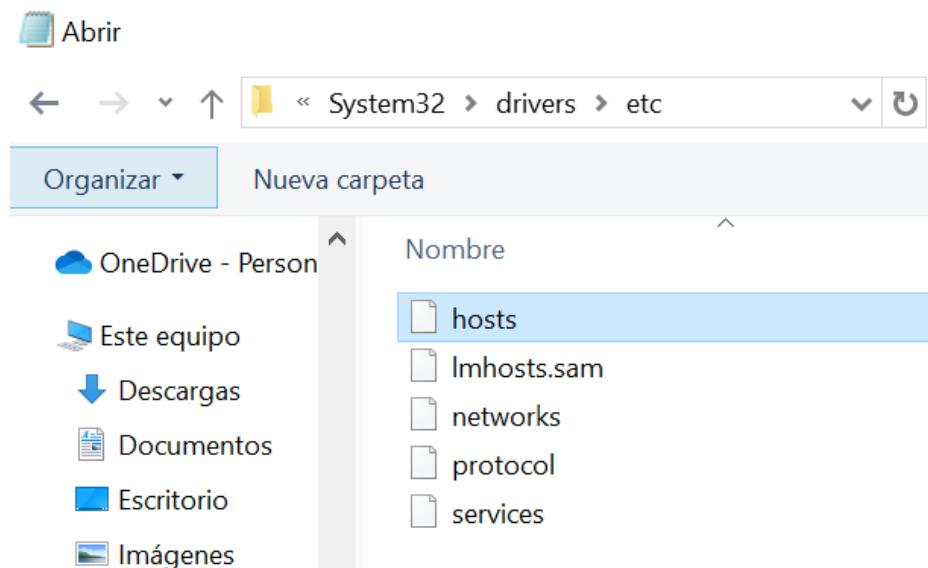


Al abrir el certificado podemos ver que es correcto pero no validado. Y está apuntando a nuestro dominio local.



Pero todavía tengo que escribir la IP para poder acceder. Para poder utilizar el dominio local y no necesitar recordar la IP del servidor voy a utilizar dominios locales. Utilizar este tipo de dominios además de no tener que memorizar direcciones IP, ahorra la necesidad de tener un servidor DNS dedicado y también de comprar un dominio.

Para hacer dominios locales hay que tener en cuenta el sistema operativo del usuario. En el caso de tener el sistema operativo Windows: Ejecuto en modo administrador el Bloc de Notas u otra herramienta similar de tratado de texto. Abro la carpeta C:\Windows\System32\drivers\etc y selecciono mostrar todos los archivos. Localizo el archivo hosts y pulso para modificarlo.



Ahora añado la IP seguido del nombre de dominio y extensión a utilizar.

Si utilizo la siguiente estructura: 192.168.1.21 luisdieguez.local
Luego podría acceder con: <https://luisdieguez.local:8006>

Así quedaría el archivo hosts:

```
hosts: Bloc de notas
Archivo Edición Formato Ver Ayuda
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97      rhino.acme.com      # source server
#      38.25.63.10      x.acme.com          # x client host
#
# localhost name resolution is handled within DNS itself.
#      127.0.0.1      localhost
#      ::1            localhost
192.168.1.21  luisdieguez.local|
```

En Linux el proceso es muy similar. Tan solo edito el archivo de configuración hosts con un editor de texto con permisos de administrador, en este caso utilizo *nano*, y añado la dirección IP junto al nombre de mi dominio local.

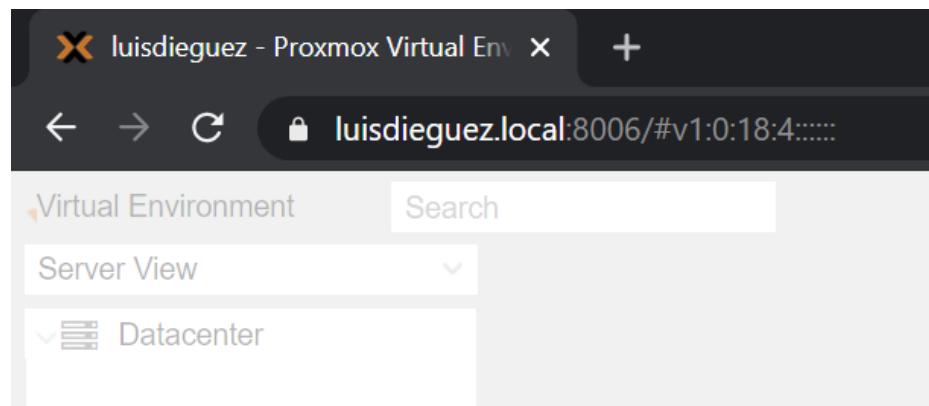
```
sudo nano /etc/hosts
```

Por lo que quedaría así:

```
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.9.3                               /etc/hosts
127.0.0.1      localhost
127.0.1.1      luis-UX370UAR

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.1.21 luisdieguez.local
```

Ahora en cualquier dispositivo tanto Linux como Windows en el que haya modificado el archivo host podré acceder a la URL luisdieguez.local con un certificado valido.



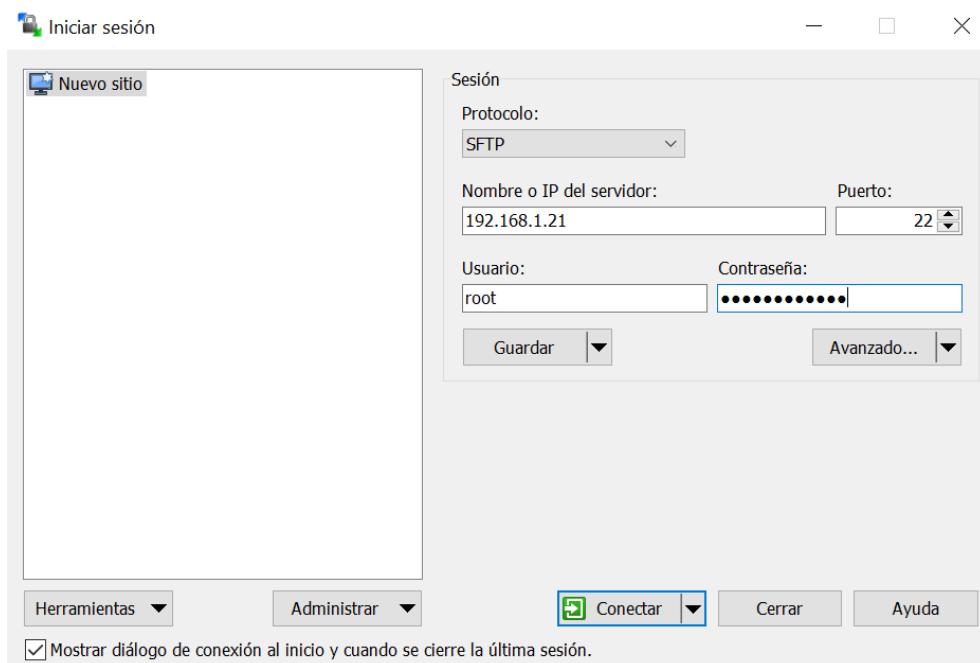
4.1.2

SEGURIDAD EN PROXMOX

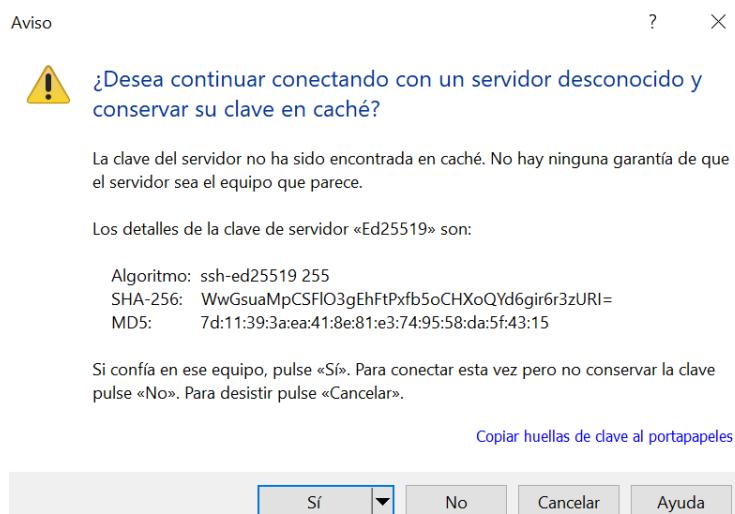
El proceso de instalación ya ha finalizado. Pero al ser un servidor empresarial un punto muy importante es la seguridad. Por ello voy a añadir varias capas de seguridad.

La primera es añadir nuestro certificado autofirmado. Voy a utilizar el programa WinSCP para poder acceder de forma sencilla a los archivos del servidor. Lo descargué de su página oficial: <https://winscp.net/eng/download.php>

Una vez descargado introduzco el usuario del administrador (root) y la contraseña del servidor junto con su IP para acceder por SSH.



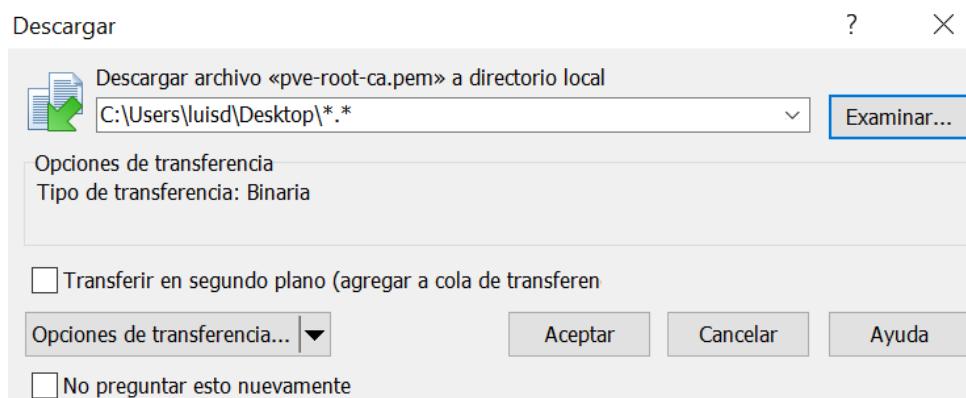
Al pulsar en conectar va a aparecer el mensaje de advertencia de servidor desconocido. Esto es por no tener la clave previamente. Al pulsar en Sí, su clave se nos guardará en nuestro ordenador para futuras conexiones.



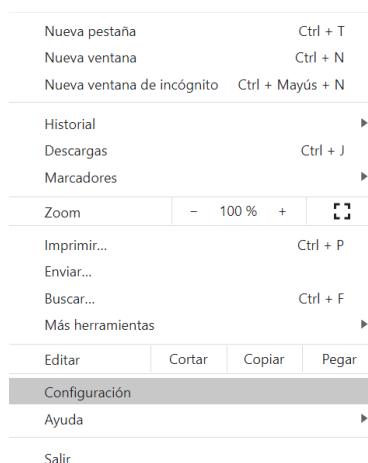
Una vez aceptada la clave ya puedo navegar por el árbol de directorios. La ruta donde están almacenados el certificado es /etc/pve y el nombre del certificado es pve-root-ca.pem.

/etc/pve/					
Nombre	Tamaño	Modificado	Permisos	Propietario	...
..		08/04/2020 11:01:14	rwxr-xr-x	root	
local		01/01/1970 1:00:00	rwxr-xr-x	root	
lxc		01/01/1970 1:00:00	rwxr-xr-x	root	
nodes		08/04/2020 11:01:28	rwxr-xr-x	root	
openvz		01/01/1970 1:00:00	rwxr-xr-x	root	
priv		08/04/2020 11:01:28	rwx-----	root	
qemu-server		01/01/1970 1:00:00	rwxr-xr-x	root	
authkey.pub	1 KB	08/04/2020 11:01:28	rw-r-----	root	
datacenter.cfg	1 KB	08/04/2020 10:58:54	rw-r-----	root	
pve-root-ca.pem	3 KB	08/04/2020 11:01:30	rw-r-----	root	
pve-www.key	2 KB	08/04/2020 11:01:28	rw-r-----	root	
storage.cfg	1 KB	08/04/2020 10:58:54	rw-r-----	root	
user.cfg	1 KB	08/04/2020 10:58:54	rw-r-----	root	
vzdump.cron	1 KB	08/04/2020 11:01:30	rw-r-----	root	

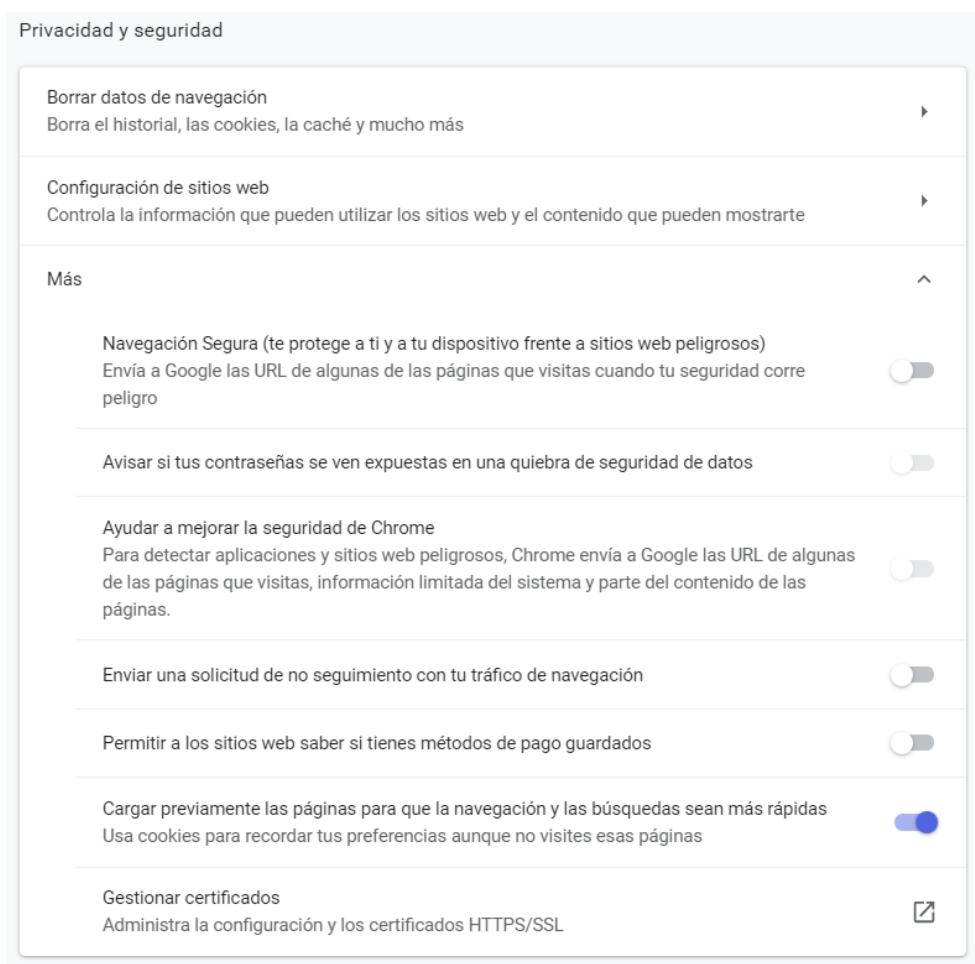
Una vez encontrado al pulsar sobre él tenemos la opción de descargarlo y elegir la localización. Elegí guardarlo en el Escritorio para posteriormente exportarlo a la ruta de certificados raíz.



Ahora fui a mi navegador usual, Google Chrome. En la parte superior derecha hay tres puntos que nos abren el menú inicial. Al pulsar sobre configuración se abre el menú de configuración.



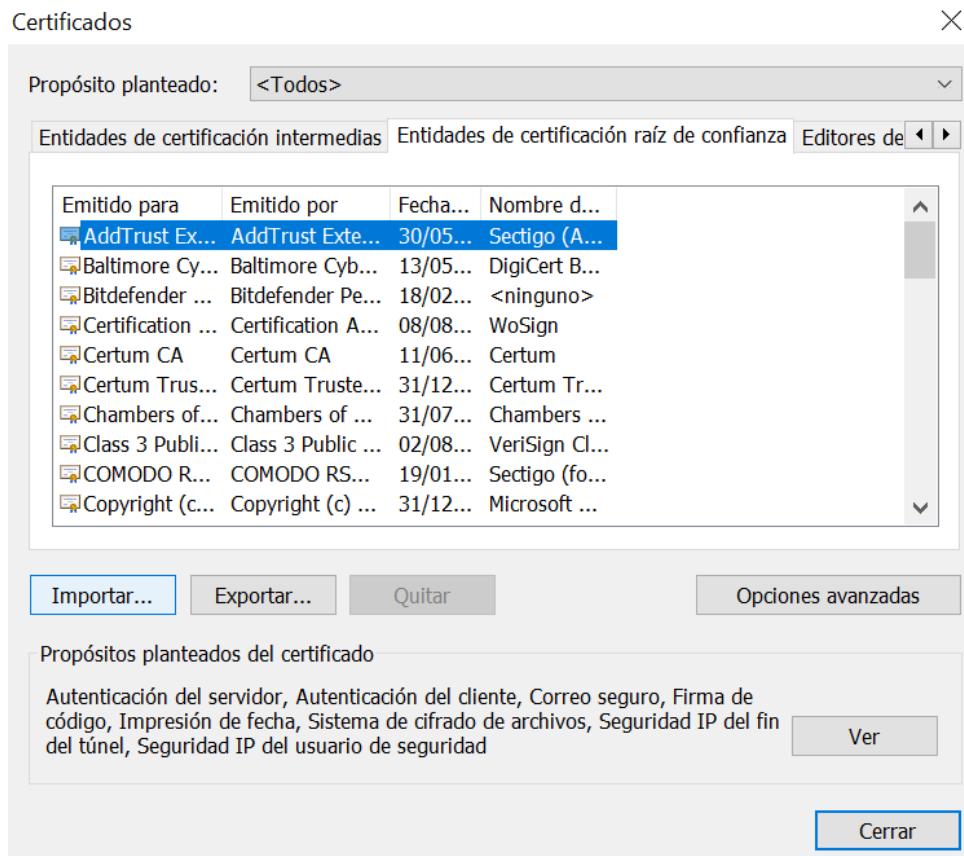
En el menú de configuración busqué el apartado de privacidad y seguridad. En dicho apartado, la última opción nos permite gestionar los certificados en nuestro navegador.



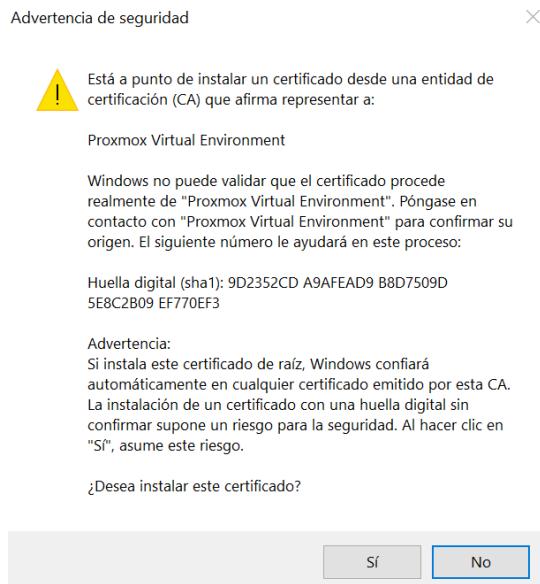
Al pulsar sobre el gestionamos los certificados locales. La función principal de un certificado es la de autenticar la identidad del propietario del certificado ante los demás. Un certificado contiene la clave pública del propietario, mientras que el propietario conserva la clave privada. La clave pública se puede usar para cifrar los mensajes enviados al propietario del certificado. Solo el propietario tiene acceso a la clave privada, por lo que únicamente él puede descifrar esos mensajes.

Al ser en Windows tiene varias sub pestañas ya que gestiona cada certificado según su tipo. Hace la siguiente organización:

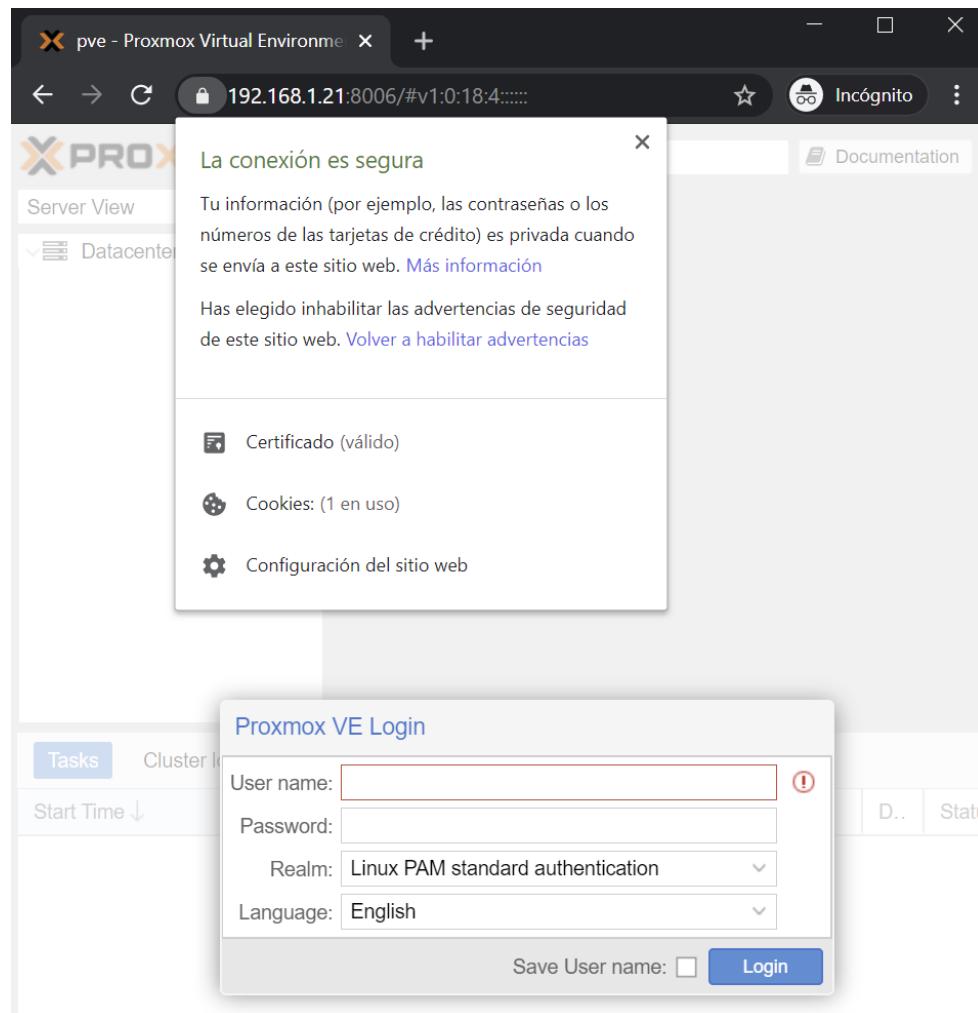
- **Personal:** Los certificados asociados a este usuario.
- **Otras personas:** Los certificados no asociados a este usuario directamente.
- **Entidades de certificación intermedias:** Certificados del controlador de dominio o de inicio de sesión de tarjetas inteligentes.
- **Entidades de certificación raíz:** Los certificados sin firma o autofirmados que identifican la autoridad de certificación raíz (CA).
- **Editores de confianza:** Los certificados de los programas que instalas en el equipo.
- **Fabricantes que no son de confianza:** Los certificados revocados por las entidades.



En este caso, el más acertado sería entidades de certificación raíz. En la su propia pestaña pulso sobre Importar y seleccionó el certificado descargado previamente en el Escritorio. Aparece un mensaje de aviso para confirmar la acción. La huella digital y el nombre del emisor es correcta así que pulso Sí.



Ahora cierro y vuelvo abrir el navegador para que la nueva instancia del Chrome utilice el certificado recién añadido. Abro una pestaña y vuelvo a escribir HTTPS, la IP del servidor y el puerto ocho mil seis. El certificado ya aparece como valido. Si sufriera un ataque hacia el certificado SSL el navegador nos avisaría como al inicio.



En Linux, en el mismo navegador, en la misma localización, tan solo tenemos que pulsar importar sin necesidad de elegir el tipo de entidad

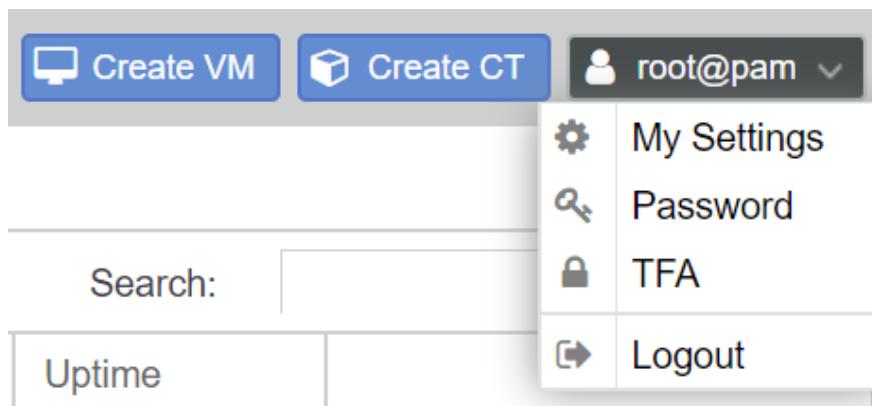


Para aumentar la seguridad voy a implementar autenticación en dos pasos (A2F). Es un método para confirmar el acceso utilizando la contraseña y un segundo factor distinto. En este caso, el segundo factor que voy a utilizar va a ser un número de seis dígitos generado aleatoriamente cada treinta segundos.

Lo primero es iniciar sesión con el usuario administrador root y la contraseña puesta en el momento de la instalación del sistema operativo al panel web.

The screenshot shows the Proxmox VE Login interface. It has fields for 'User name' (root), 'Password' (redacted), 'Realm' (Linux PAM standard authentication), and 'Language' (English). There is a 'Save User name' checkbox and a 'Login' button.

Ahora en la esquina derecha se puede ver el usuario. Al pulsar sobre él tendremos las opciones de configuración de la cuenta, en este caso de la cuenta de administrador. Y pulso sobre TFA, acrónimo de *Two-Factor Authentication*.

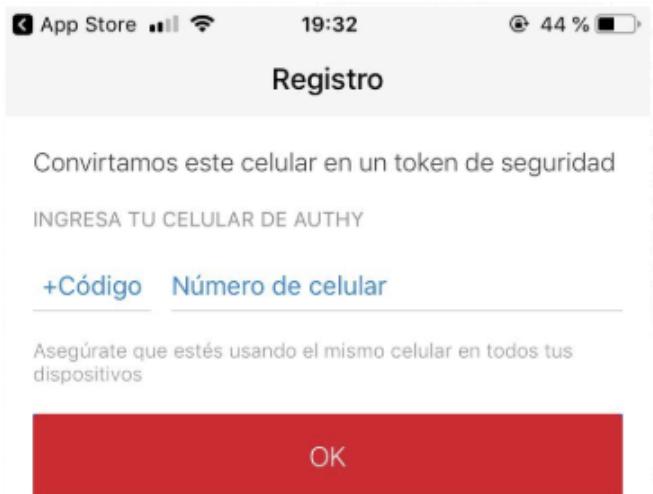


En la ventana que se abre se puede elegir entre dos opciones. La primer es *Time-based One-time Password* (TOTP), es una secuencia de números válida solo durante un breve periodo de tiempo. La segunda es *Universal 2nd Factor* (U2F), este tipo de autentificación en dos pasos requiere un dispositivo USB o NFC específicamente diseñado con el algoritmo FIDO. El precio de estos dispositivos ronda al rededor de 25 euros hasta los 70 euros según las prestaciones.

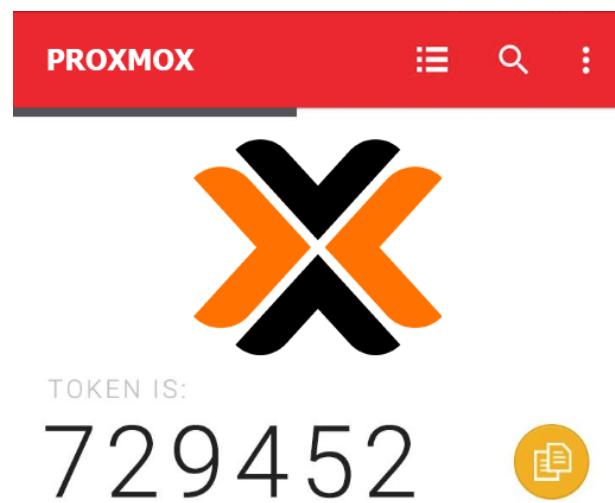
He elegido la primera opción ya que no requiere un dispositivo especializado y puedo realizarlo desde mi teléfono móvil. Para ello hay varias opciones pero las más fiables son Google Authenticator y Authy. Me decanté por Authy ya que tiene guardado en la nube, así si pierdo o me roban el dispositivo móvil no pierdo el acceso a todas las cuentas.

Así que lo primero es descargar la aplicación de Authy en el dispositivo móvil.
Desde la Play Store: <https://play.google.com/store/apps/details?id=com.authy.authy>.
Desde la App Store: <https://apps.apple.com/us/app/authy/id494168017>
En PC: <https://authy.com/download/>

Una vez descargado en cualquier dispositivo, crearé una cuenta con mi número de teléfono. Este número de teléfono creará un token único que a su vez tendrá una contraseña y como añadido la huella dactilar del propio teléfono.



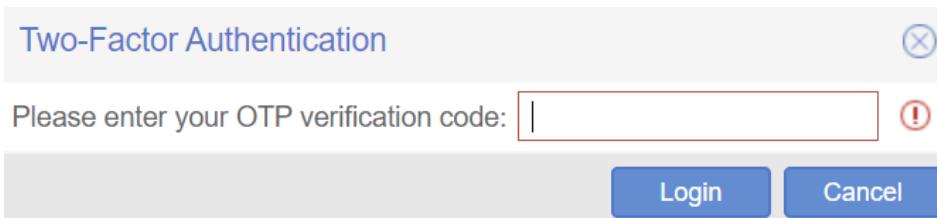
Ahora vuelvo al panel web y selecciono la primera opción: TOTP. Con la aplicación del teléfono, pulsamos agregar cuenta en la esquina derecha y escaneamos el código QR del panel web.



Al agregar la cuenta y aparece el código de seis dígitos numéricos cada treinta segundos. El último paso es ya agregar ese código en el panel web, donde pone *verification code*. Ahora cada vez que inicie sesión tendrá que utilizar el teléfono para ver el código y ponerlo antes de que pasen los treinta segundos.



Ahora cada vez que inicie sesión correctamente además pedirá el código de verificación.



Ya he finalizado la primera capa de seguridad orientada a la verificación de del certificado SSL y la segunda capa orientada al inicio de sesión. Las siguientes capas van a ser activar los repositorios gratuitos para tener las últimas actualizaciones, añadir un antivirus y un servicio para prohibir la entrada a atacantes que han intentado entrar por fuerza bruta en el servidor

Por defecto Proxmox viene con los repositorios de pago, así que a la hora de hacer una actualización desde la consola nos aparecerá un error de IP no registrada. Para poder actualizar debo quitar los repositorios *enterprise* y poner los *No-Subscription*.

Para hacer esto tengo que modificar el archivo de configuración de Debian que se encuentra en el directorio sources.list.d.

Para ello abro con un editor de texto el archivo y procedo a cambiar los repositorios.

```
nano /etc/apt/sources.list.d/pve-enterprise.list
```

Delante del repositorio pongo el símbolo de almohadilla en vez de borrarlo por si en un futuro actualizara a versión de pago y fuese necesario. Por lo que quedaría así.

```
GNU nano 3.2          /etc/apt/sources.list.d/pve-enterprise.list

# deb https://enterprise.proxmox.com/debian/pve buster pve-enterprise
```

Ahora añado el repositorio gratuito al archivo general. El repositorio gratuito tiene esta estructura: *deb http://download.proxmox.com/debian/pve buster pve-no-subscription*

```
nano /etc/apt/sources.list
```

Por lo que quedaría así.

```
GNU nano 3.2          /etc/apt/sources.list

deb http://ftp.es.debian.org/debian buster main contrib
deb http://ftp.es.debian.org/debian buster-updates main contrib
# security updates
deb http://security.debian.org buster/updates main contrib
deb http://download.proxmox.com/debian/pve buster pve-no-subscription
```

Ahora cada vez que actualice no saldrá ningún error y podré tener los últimos parches de seguridad tanto de Debian como de Proxmox y el resto de paquetes.

La siguiente capa es un antivirus. Los antivirus son programas cuyo objetivo es detectar y eliminar virus informáticos. He elegido utilizar ClamAV. ClamAV es un motor antivirus de código abierto para detectar troyanos, virus, malware y otras amenazas maliciosas. Es mantenido por Cisco desde 2004. Voy a instalar la versión sin interfaz gráfica (GUI) ya que al ser un servidor no tiene. Para instalarlo ejecuto el comando:

```
apt-get install clamav clamav-daemon -y
```

Y activo los dos procesos principales de este antivirus para que se ejecuten de forma autónoma en cada inicio del sistema.

```
/etc/init.d/clamav-daemon start
```

```
/etc/init.d/clamav-freshclam start
```

La última capa de seguridad que voy a añadir es un servicio para prohibir la entrada a atacantes que han intentado entrar por fuerza bruta. Se llama Fail2Ban. El funcionamiento de este servicio es analizar los archivos de registro y prohibir las direcciones IP que muestran signos maliciosos. Para prohibir el acceso de las direcciones IP actualizar las reglas del firewall durante un período de tiempo específico determinado en su configuración.

Para instalarlo ejecuto el siguiente comando:

```
apt-get install fail2ban -y
```

Ahora voy a copiar el archivo de configuración de Fail2Ban (jail.conf) y crear uno nuevo llamado jail.local que será el que modifique para adaptar las reglas a lo que necesito:

```
cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

```
root@luisdieguez:~# apt-get install fail2ban -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python3-pyinotify python3-systemd whois
Suggested packages:
  monit python-pyinotify-doc
The following NEW packages will be installed:
  fail2ban python3-pyinotify python3-systemd whois
0 upgraded, 4 newly installed, 0 to remove and 41 not upgraded.
Need to get 527 kB of archives.
After this operation, 2,560 kB of additional disk space will be used.
Get:1 http://ftp.es.debian.org/debian buster/main amd64 fail2ban all 0.10.2-2.1 [385 kB]
Get:2 http://ftp.es.debian.org/debian buster/main amd64 python3-pyinotify all 0.9.6-1 [26.9 kB]
Get:3 http://ftp.es.debian.org/debian buster/main amd64 python3-systemd amd64 234-2+b1 [37.2 kB]
Get:4 http://ftp.es.debian.org/debian buster/main amd64 whois amd64 5.4.3 [77.8 kB]
Fetched 527 kB in 0s (1,784 kB/s)
Selecting previously unselected package fail2ban.
(Reading database ... 43636 files and directories currently installed.)
Preparing to unpack .../fail2ban_0.10.2-2.1_all.deb ...
```

Y con el editor nano voy a modificar la configuración.

- *ignoreip*: Después del igual (=) podemos añadir las IPs que aunque tenga muchos intentos fallidos, no quiero que sean baneadas. La ip que muestra por defecto es la del servidor, la dirección local. Se pueden añadir IPs individuales o rangos de IP.
- *bantime*: Es el tiempo que estará baneada una IP que a fallado al entrar en el servidor. Se mide en segundos, por defecto trae 600 segundos que son 10 minutos.
- *maxretry*: Es el máximo de intentos que una IP puede intentar acceder antes de ser baneada. Por defecto son 5 pero se puede aumentar o disminuir.
- *findtime*: Este contador se pone a cero si no se realizan detecciones en los segundos especificados.
- *ignorecommand*: Permite ejecutar un comando externo que tomará como argumento la IP. Se suele utilizar para hacer más comprobaciones con scripts propios.

```
GNU nano 3.2                                     /etc/fail2ban/jail.local

# will not ban a host which matches an address in this list. Several addresses
# can be defined using space (and/or comma) separator.
ignoreip = 127.0.0.1/8 ::1

# External command that will take an tagged arguments to ignore, e.g. <ip>,
# and return true if the IP is to be ignored. False otherwise.
#
# ignorecommand = /path/to/command <ip>
ignorecommand =

# "bantime" is the number of seconds that a host is banned.
bantime  = 10m

# A host is banned if it has generated "maxretry" during the last "findtime"
# seconds.
findtime  = 10m

# "maxretry" is the number of failures before a host get banned.
maxretry = 5
```

4.2 INSTALACIÓN DE LA VPN

El acceso al servidor y al panel de control se tiene que realizar desde un dispositivo en la misma red. No es posible administrar el servidor desde el exterior aunque si tenga salida a Internet. Eso puede ser un problema ya que en cualquier momento puede ocurrir un error o fallo y sea necesario acceder al panel.

Para solucionar esto voy a crear una *Virtual Private Network* (VPN) o red privada virtual en español. Una VPN además de solucionar ese problema, añade otra capa más de seguridad, protección y privacidad en la conexión. Con esta tecnología me aseguro de la confidencialidad e integridad de la información transmitida al crear una red privada virtual.

Para ello tenía varias opciones, podría crear un contenedor LXC dentro de Proxmox e instalar allí la VPN. Pero al final opté por utilizar un dispositivo externo del servidor, ya que si en algún momento se cae el servidor, no tendré acceso a nada dentro de la red.

El dispositivo elegido es una Raspberry Pi. Es un ordenador de placa reducida (SBC). Todos sus componentes ya están previamente ensamblados. Es una placa de bajo consumo, alrededor de quince euros en electricidad al año ya que consume cinco voltios DC. Y de bajo costo, el precio del último modelo ronda los cincuenta euros y tiene las siguientes características:

- RAM: 4 GB DDR3
- Procesador: ARM-Cortex-A72
- USB 3.0
- Gigabit Ethernet
- WiFi Dual Band 2.4 GHz y 5.0 GHz
- GPIO
- Soporte 4K, doble salida micro-HDMI

Para crear la VPN he elegido el *software* llamado OpenVPN. Es una herramienta OpenSource de grado militar que ofrece conectividad punto-a-punto con validación jerárquica de usuarios y host conectados remotamente. Además es una herramienta multiplataforma por lo que podré conectar desde cualquier tipo de dispositivo. Permite elegir entre el tipo de protocolo de conexión TCP o UDP. Siendo TCP la más lenta ya que comprueba cada paquete uno a uno y los verifica. Y UDP la más rápida ya que no para la conexión si un paquete se pierde por lo tanto suele ser la recomendada para conexiones VPN. Pero como no voy a transmitir gran cantidad de datos he elegido TCP ya que cuenta con un sistema de control de congestión y de flujo del tráfico.

Ahora para la creación de la VPN voy a conectar la Raspberry Pi a la red con un cable ethernet y añadiré un teclado, ratón y monitor para poder gestionarla momentáneamente, ya que cuando termine, al igual que el servidor no necesitará periféricos de ningún tipo.

El primer paso es descargar el sistema operativo Raspbian. Es un sistema operativo específicamente optimizado para Raspberry Pi, basado en Debian. Lo descargo de su página oficial: <https://www.raspberrypi.org/downloads/raspbian/>

Una vez descargado procedo a grabar la imagen (ISO) descargada en un tarjeta micro SD. El procedimiento es el mismo que cuando grabe la imagen ISO de Proxmox. Utilizando el programa Etcher y selecciono la tarjeta SD. El tamaño de la tarjeta debe ser de mínimo ocho GB.

Una vez grabada la imagen en la tarjeta SD la introduzco en la Raspberry Pi. En el primer inicio introduciré los datos básicos. Lo primero será la localización, el país, el lenguaje y la zona horaria. En la segunda pestaña es para la red WiFi, en este caso pulso sobre *Skip* para Saltarla ya que solo va a tener conexión por red cableada ethernet. La última pestaña es para actualizar todo el sistema, una vez actualizado pulso sobre *Reboot* para reiniciar la Raspberry Pi.

Ahora la Raspberry Pi ya tiene un sistema operativo. Voy a instalar y configurar OpenVPN para poder tener una red privada virtual. Para ello voy a utilizar un script de instalación. Ejecuto en la terminal el siguiente comando para descargarlo:

```
wget https://raw.githubusercontent.com/Nyr/openvpn-install/master/openvpn-install.sh
```

Y le asigno permisos de Leer+Ejecutar+Escribir para todos los usuarios y carpetas.

```
chmod 777 openvpn-install.sh
```

Una vez el script tiene todos los permisos necesarios para ejecutarse procedo a recopilar información sobre mi propia red para posteriormente poder ejecutar el script e introducir esos datos. Voy a necesitar la IP Privada (interna) y la IP Pública (externa). Para ello voy a ejecutar dos comandos. El primero es el comando *ip*, nuevo estándar y sustituto de *ipconfig*. El segundo es el comando *dig*, con el cual haré una petición TXT DNS hacia los resolvidores de Google, específicamente diseñados para devolver la IP pública de quien les haga una petición.

```
ip route
```

```
dig TXT +short o-o.myaddr.l.google.com @ns1.google.com
```

Me devuelven mi IP privada el comando *ip* y mi IP pública el comando *dig*.

```
pi@raspberrypi:~ $ ip route
default via 192.168.1.1 dev eth0 proto dhcp src 192.168.1.50 metric 202
192.168.1.0/24 dev eth0 proto dhcp scope link src 192.168.1.50 metric 202
```

```
pi@raspberrypi:~ $ dig TXT +short o-o.myaddr.l.google.com @ns1.google.com
"92.186."
```

La IP privada es: 192.168.1.50

La IP pública la he tapado ya que es la asignada por el ISP y no es recomendado compartirla de cara a posibles ataques a toda mi red. Pero comienza por 92.186.

Ahora ya con todos los datos necesarios de la red voy a proceder a ejecutar el script.

```
sudo bash openvpn-install.sh
```

El menú consta de seis pasos. El primero es la IP privada, automáticamente ya la detecta pero podría ser el caso que fallará o no la detectará correctamente. En el caso de estar bajo una NAT no sería necesario introducirla.

El segundo paso es la IP Pública. Introduzco la IP pública de mi red que extraje anteriormente con el comando *dig*.

El tercer paso es el protocolo que con el que voy a realizar las comunicaciones. Da a elegir entre TCP y UDP. Como mencioné anteriormente seleccionó TCP.

El cuarto paso es el puerto por el que voy a conectarme a la VPN. Por defecto es mil ciento noventa y cuatro. Al no tener ninguna otra aplicación utilizando ese puerto y por lo tanto está libre, lo dejo por defecto.

El quinto es el resovedor DNS. Lo recomendable es Google o 1.1.1.1. Siendo Google una opción segura pero los servidores 1.1.1.1 son de la empresa CloudFlare y tienen mejor tiempo de respuesta y mayor privacidad. Ya que a las 24 horas borran todos los registros a diferencia de Google que guarda esa información. Por lo tanto me decanto por los resovedores de CloudFlare.

El sexto y último paso es el nombre del cliente. En este caso lo voy a llamar proxmox para poder identificarlo de forma sencilla.

```
Welcome to this OpenVPN road warrior installer!

I need to ask you a few questions before starting setup.
You can use the default options and just press enter if you are ok with them.

This server is behind NAT. What is the public IPv4 address or hostname?
Public IPv4 address / hostname [92.186. [REDACTED]]: 92.186. [REDACTED]

Which protocol do you want for OpenVPN connections?
  1) UDP (recommended)
  2) TCP
Protocol [1]: 2

What port do you want OpenVPN listening to?
Port [1194]: 1194

Which DNS do you want to use with the VPN?
  1) Current system resolvers
  2) 1.1.1.1
  3) Google
  4) OpenDNS
  5) Verisign
DNS [1]: 2

Finally, tell me a name for the client certificate.
Client name [client]: proxmox

Okay, that was all I needed. We are ready to set up your OpenVPN server now.
Press any key to continue... [REDACTED]
```

Una vez configurado procedo a presionar cualquier tecla para que complete el proceso de instalación con los parámetros introducidos. Al finalizar aparece el siguiente mensaje.

```
Finished!

Your client configuration is available at: /root/proxmox.ovpn
If you want to add more clients, just run this script again!
```

Ahora para copiar el archivo del cliente hacia el escritorio utilizo el siguiente comando:

```
sudo cp /root/proxmox.ovpn ~/Desktop
```

Y una vez en el Escritorio utilizo Google Drive para guardarla. Ahora tengo que abrir los puertos del router (*port forwarding*). El *port forwarding* es un método para que un ordenador en una red, sea accesible otros ordenadores en Internet. Se usa comúnmente en juegos, configuración de cámaras de seguridad, voz sobre ip y descarga de archivos.

Para hacer esto vamos a la dirección IP de la puerta de enlace o del router. En mi caso y la más usual es: 192.168.1.1

Accedo con el usuario y contraseña dado por el ISP que suele estar detrás del propio router.

En la sección de configuración avanzada, en la sección de DHCP, lo primero que voy a hacer es reservar la dirección IP estática tanto para el servidor de Proxmox como para el servidor de OpenVPN en la Raspberry Pi.

dirección IP estática			
nombre	dirección IP	dirección MAC	
Unknown Device ▾	192.168.1.22	84:C7:EA:1F:1C:A7	<button>añadir</button>
Unknown Device	192.168.1.21	00:25:90:F5:2B:FB	<button>borrar</button>
raspberrypi	192.168.1.50	DC:A6:32:27:4A:F7	<button>borrar</button>

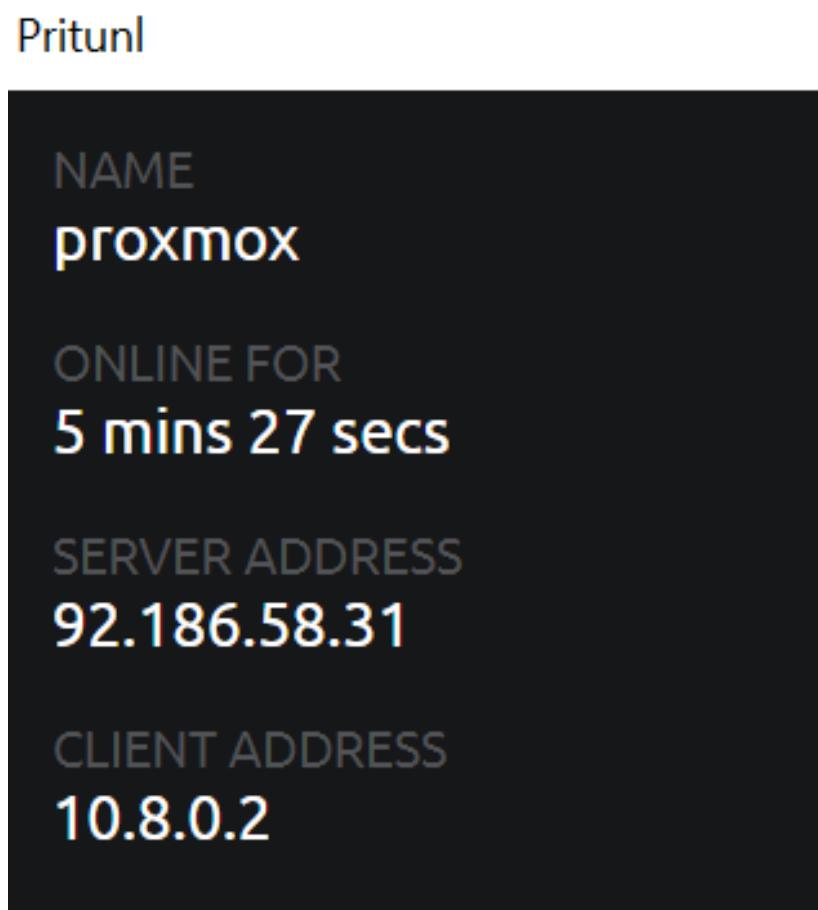
En la sección de configuración avanzada, en la sección de NAT/PAT/CGNAT, procedo a añadir el *port forwarding*. De nombre le pongo VPN-PROXMOX, el puerto interno será el mil ciento noventa y cuatro, el puerto externo será el mil ciento noventa y cuatro, el protocolo TCP y la IP del servidor OpenVPN en la Raspberry Pi que es el 192.168.1.50

<input checked="" type="checkbox"/>	VPN-PROXMOX	1194	1194	TCP	192.168.1.50	<input type="checkbox"/>	delete
-------------------------------------	-------------	------	------	-----	--------------	--------------------------	---------------

Ahora ya puedo conectarme desde cualquier dispositivo con el cliente que guarde en Google Drive.

Para conectarme tanto desde el sistema operativo Window, Linux o MacOS voy a utilizar Pritunl: <https://client.pritunl.com>

Su uso es muy sencillo, después de instalarlo y ejecutarlo no hay que configurar nada. Solo pulso sobre *Import Profile* y seleccionó el archivo descargado de Google Drive proxmox.ovpn. Luego en el menú de la esquina superior derecha pulso sobre conectar.



4.3

INSTALACIÓN DE RANCHER

Hasta ahora ya tengo un servidor virtualizado con el hipervisor de tipo uno o nativo Proxmox. Y una VPN para poder acceder a él desde fuera de la red.

Rancher es el producto principal de la empresa Rancher Labs. Esta empresa tiene sede en siete capitales a lo largo de todo el mundo. Y es una empresa puntera en el desarrollo orientado a la nube.

Rancher es un orquestador de orquestadores. Lo que significa que voy a poder desplegar toda la arquitectura de diferentes orquestadores como Kubernetes, Cattle, Mesos o Docker Swarm. Utilizando los servidores de Google, Azure, Digitalocean, AWS y otras nubes públicas, utilizar mis propios servidores en privado y controlarlos desde un mismo panel. Esto es muy útil a la hora de realizar nubes híbridas que tiene varios proveedores distintos y permite una gran flexibilidad al poder realizar grandes cambios en un mismo lugar.

También permite dividir por entornos. Una empresa divide sus trabajadores por tipo y especialización. Con Rancher puedo tener un entorno específico para cada tipo de aplicación en desarrollo. Además puedo gestionar dentro de cada aplicación sus etapas:

1. Desarrollo (Development)
2. Pruebas (Testing)
3. Imitación de entorno real (Staging)
4. Entorno real (Production)

Permite la gestión de usuarios. La asignación de roles y permisos para limitar las acciones de cada desarrollador. Y la creación de grupos para reunir todos los usuarios con características similares o del mismo equipo.

También tiene un catálogo de aplicaciones definidas previamente para lanzarlas desde el panel. Esta basado en Helm, un gestor de paquetes para aplicaciones de Docker.

Para instalar Ramcher procedo creando una máquina virtual en el servidor local con Proxmox. Para crear máquinas virtuales en Proxmox lo primero es subir la imagen ISO en la que va a estar basada la máquina.

Para subir imágenes pulsamos sobre el disco y en el botón *Upload* seleccionó la ISO. Voy a utilizar *Ubuntu Server* ya que tiene un bajo peso y un gran rendimiento. La descargo de la página oficial: <https://ubuntu.com/download/server>

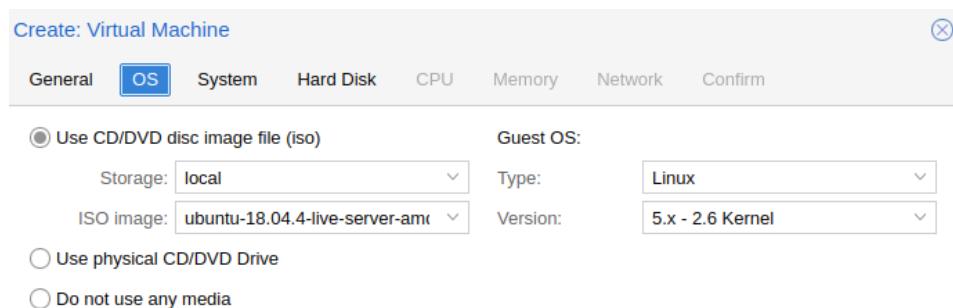
The screenshot shows two parts of the Proxmox Web Interface. The top part is a modal dialog titled 'Upload' with a content type dropdown set to 'ISO image'. Below it is a file selection input field and a 'Select File...' button. At the bottom are 'Abort' and 'Upload' buttons. The bottom part shows the 'Storage' screen for 'local' on node 'luisdieguez'. It has tabs for 'Summary', 'Content' (which is selected), 'Permissions', and 'Restore', 'Remove', 'Templates', 'Upload', and 'Show Configuration' buttons. Under 'Content', there is a table with one item: 'ISO image (1 Item)' containing the file 'ubuntu-18.04.4-live-server-amd64.iso'.

Ahora pulso en la esquina derecha superior sobre *Create VM (Virtual Machine)*. Aparece el menú de creación. En la primera pestaña, General, tengo para elegir los siguientes parámetros:

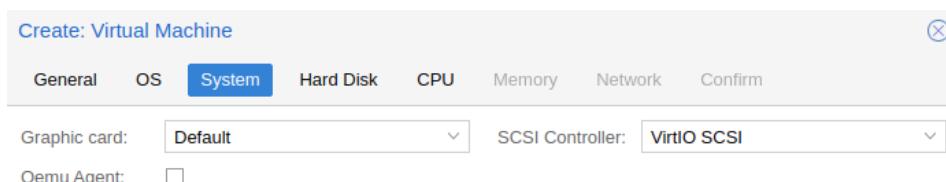
- *Node*: El nodo donde se va a levantar la máquina virtual o contenedor.
- *VM ID*: Es un número único que permite identificar cada máquina virtual o contenedor.
- *Name*: El nombre de la máquina virtual o contenedor.
- *Resource pool*: Las *pools* son agrupaciones de máquinas virtuales bajo una misma etiqueta.

The screenshot shows the 'Create: Virtual Machine' dialog with the 'General' tab selected. Other tabs include 'OS', 'System', 'Hard Disk', 'CPU', 'Memory', 'Network', and 'Confirm'. The 'General' tab fields are: 'Node' set to 'luisdieguez', 'VM ID' set to '102', and 'Name' set to 'rancheros'. The 'Resource Pool' dropdown is empty.

La siguiente pestaña, OS, es sobre el sistema operativo. Tan solo seleccionó de la lista de imágenes subidas a *Ubuntu Server*.



La siguiente pestaña, *System*, la puedo dejar por defecto ya que no voy a necesitar tarjeta gráfica.



La siguiente pestaña, *Hard Disk*, es relativa al método de almacenamiento de la máquina virtual. Contiene los siguientes parametros:

- **Bus/device:** Un bus es la ruta de datos en la placa base del ordenador que conecta al procesador con, en este caso, el disco duro. Puedo elegir entre distintos tipos de virtualización del bus.
 - IDE - Escritura lenta.
 - SCSI - Escritura más rápida.
 - SATA - Escritura más rápida que SCSI y IDE pero mayor dificultad de virtualización.
 - VIRTIO - Escritura más rápida (más que SCSI e IDE) pero solo con controladores adicionales. Linux tiene por defecto esos controladores y en Windows se pueden instalar como *drivers*.
- **Storage:** El disco físico donde va se va a virtualizar.
- **Disk size (GiB):** El tamaño del disco virtual.
- **Cache:** Es el tipo de cache que va a utilizar el disco virtual. Hay distintos tipos:
 - none - Sin cache. Equilibra el rendimiento y la seguridad (mejores escrituras).
 - write-through - Los datos se escriben en la caché y en el disco al mismo tiempo. Equilibra el rendimiento y la seguridad (mejores lecturas).
 - write-back: Los datos se escriben en el disco en segundo plano, pero no espera a la confirmación de finalización. Mayor velocidad pero puede perder datos en caso de corte de energía dependiendo del hardware.
 - write-back (unsafe): No realiza la confirmación en ningún momento.
 - direct-sync: Similar a write-through. Como añadido se realiza una sincronización fsync para cada escritura. Fsync sincroniza el estado completo en memoria de un fichero con el del disco
- **Discard:** Es una característica de Linux. Utiliza el *thin provisioning* para simular que el disco tiene más espacio que el real.

Create: Virtual Machine

General	OS	System	Hard Disk	CPU	Memory	Network	Confirm
Bus/Device:	VirtIO Block	0	Cache:	Direct sync			
Storage:	local-lvm		Discard:	<input type="checkbox"/>			
Disk size (GiB):	32						
Format:	Raw disk image (raw)						

La siguiente pestaña es CPU. Por lo tanto puedo elegir cuantos procesadores (*sockets*) y cuantos núcleos (*cores*) puede utilizar la máquina virtual como máximo.

Create: Virtual Machine

General	OS	System	Hard Disk	CPU	Memory	Network	Confirm
Sockets:	2	Type:	Default (kvm64)				
Cores:	2	Total cores:	4				

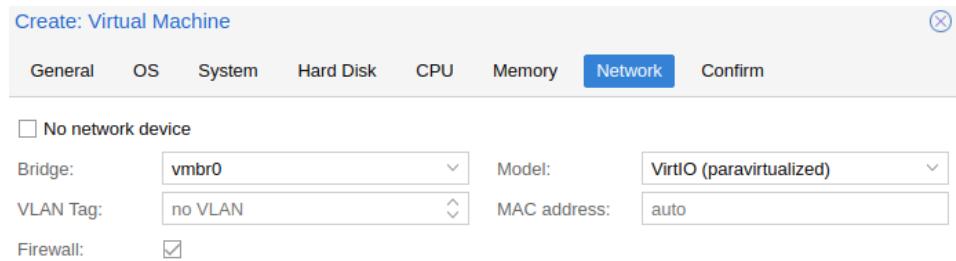
La siguiente pestaña, *Memory*, es relativa a la cantidad de memoria RAM que la máquina virtual puede utilizar como máximo.

Create: Virtual Machine

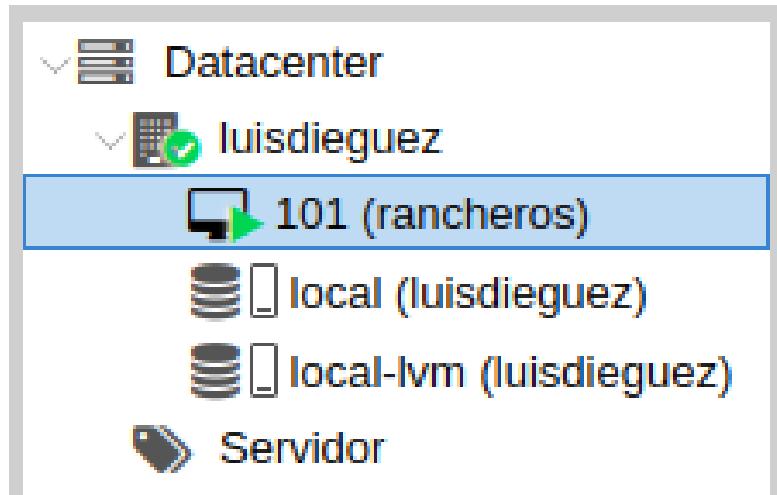
General	OS	System	Hard Disk	CPU	Memory	Network	Confirm
Memory (MiB):	4072						

La siguiente pestaña, *Network*, es relativa a la configuración de red para la máquina virtual. Puedo especificar que no haya tarjeta de red con *No network device*. Contiene los siguientes parámetros:

- *Bridge*: Los *bridges* son switchs físicos implementados con software. Todas las máquinas virtuales pueden compartir un solo *bridge*, o se puede crear múltiples bridges para separar dominios de red.
- *VLAN Tag*: La *VLAN tag* es parte de la configuración de la red externa, por medio de un switch que soporte VLAN, de nivel 1. Una red se puede dividir en diferentes VLAN. Y para utilizar cada VLAN necesitas asignar un identificador, una VLAN Tag a la red del dispositivo.
- *Firewall*: Permite activar un Firewall externo a nivel de Proxmox en la máquina virtual.
- *Model*: Permite elegir el modelo de tarjeta de red a utilizar en la máquina virtual.
 - Intel E1000: Es el controlador básico de Intel. Tiene conectividad Gigabit. Suele ser la mejor opción ya que es la que más compatibilidad tiene, tanto para Windows como Linux.
 - VirtIO (paravirtualized): Son los controladores propios de Linux. Tienen mayor velocidad de transferencia en sistemas Linux.
 - Realtek RTL8139: Este adaptador de red suele ser usado para temas de pentesting.
 - VMware vmxnet3: Permite emular la red de VMware. Suele ser usado cuando se virtualiza una máquina con VMware para aumentar la compatibilidad y características.
- *MAC address*: Define la dirección MAC de la tarjeta de red.

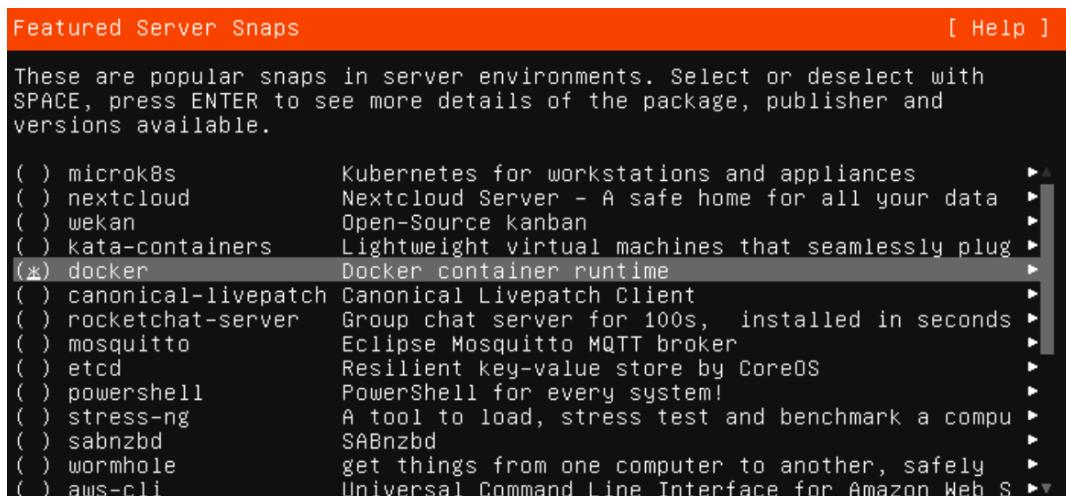


La última pestaña es la confirmación. Una vez creada ya puedo ver como se esta iniciando, cuando se inicie ya puedo interactuar con ella.



Una vez creada y corriendo puedo acceder a la consola. Ahora procedo a configurar el sistema operativo Ubuntu. Necesito instalar Docker para que pueda correr Rancher.

Es una instalación de Ubuntu típica. Seleccióno el idioma Español, la distribución de teclado Español, actualizar en el momento, la red, no instalo SSH y sí instalo Docker de la lista de snaps. Snaps es un sistema de gestión de paquetes e implementación de software creado por Canonical, la empresa detrás de Ubuntu.



Una vez seleccionado, procederá la instalación de Ubuntu.

Ahora procedo a instalar Rancher. Aunque más que instalar es correr un contenedor Docker con la imagen de Rancher. Con Docker puedo usar los contenedores como máquinas virtuales extremadamente livianas y modulares. La tecnología Docker no solo aporta la capacidad de ejecutar contenedores; también facilita el proceso de creación y diseño de contenedores, de envío de imágenes y de creación de versiones de imágenes.

Para crear el contenedor Docker con Rancher ejecuto el siguiente comando dentro de la máquina virtual con Ubuntu. Lo que estoy haciendo con este comando es llamar al operador (*docker*) para que ejecute (*run*) un contenedor para que se ejecute en modo segundo plano (*-d*) y su política de reinicio (*--restart*) sea en caso de fallo siempre reiniciar el contenedor a menos que lo pare manualmente (*unless-stopped*). Que abra el puerto (*-p*) 80 del contenedor con el 80 de la máquina virtual. Que abra el puerto (*-p*) 443 del contenedor con el 443 de la máquina virtual. Y por último que descargue la última versión de la imagen del Docker hub del usuario rancher llamada rancher.

```
docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher:latest
```

```
[rancher@rancher ~]$ docker run -d --restart=unless-stopped -p 80:80 -p 443:443
rancher/rancher
Unable to find image 'rancher/rancher:latest' locally
latest: Pulling from rancher/rancher
5bed26d33875: Downloading 1.649MB/26.69MB
f11b29a9c730: Download complete
930bda195c84: Download complete
78bf9a5ad49e: Waiting
```

Una vez ejecutado el comando comenzará a descargar la imagen de Rancher y cuando terminé automáticamente comenzará a correr el contenedor. A los pocos minutos ya puedo acceder al panel de Rancher desde cualquier dispositivo con la IP de la máquina virtual de Ubuntu.

Una vez accedo al panel por primera vez va a pedir dos pasos de configuración básica. El primer es para confirmar los términos y condiciones de Rancher y establecer la contraseña del usuario administrador. La segunda es para confirmar la IP del nodo, en este caso de la máquina virtual.

Welcome to Rancher

The first order of business is to set a strong password for the default **admin** user.

Use a new randomly generated password:
 Set a specific password to use:

New Password
.....

Confirm Password
.....

Allow collection of anonymous statistics [Learn More](#)
 I agree to the [Terms and Conditions](#) for using Rancher.

Continue

Rancher Server URL

What URL should be used for this Rancher installation? All the nodes in your clusters will need to be able to reach this.

URL

`https:// 192.168.1.120`



Are you sure all the hosts you will create will be able to reach `192.168.1.120`?
It looks like a private IP or local network.

Save URL

INSTALACIÓN DE KUBERNETES

Kubernetes es un sistema para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores. Se le llama orquestador de contenedores ya que desde él puedo gestionar todos los aspectos de los contenedores.

Se suele usar para operaciones de alto nivel. Las más comunes son:

- Despliegue de contenedores para aplicaciones (*deployment*)
- Almacenamiento no volátil de estas (*persistent storage*)
- Monitoreo de los contenedores (*health*)
- Gestión de todos los recursos
- Escalado de las aplicaciones
- Alta disponibilidad (HA)
- Redes y carga balanceada
- Corrección de las versiones de las aplicaciones en contenedores (*roll out / roll over / roll back*)

Puede estar formado por tantos ordenadores como queramos añadir. Se divide en:

- Master: El controlador de las actividades
- Nodo: Donde se corren los contenedores

Aunque el nodo maestro (master) tenga todo lo necesario para ejecutar sus propios contenedores, no es recomendado usarlo en producción. Lo mejor es que solo se encarguen de la gestión de los nodos trabajadores (workers). Esta formado por:

- Kube API server: Utilizada en los nodos trabajadores y clientes para comunicarse de forma segura.
- Kube scheduler: Decide la gestión de las tareas, como por ejemplo, en que nodo se ejecuta cada contenedor.
- Kube controller: Ejecuta controladores. Cada controlador es un *deamon* (proceso en segundo plano) que esta revisando que todo se cumpla. Por ejemplo que haya siempre un numero de replicas.
- Etcd: Base de datos de claves. Importante tener copias de seguridad.

En el nodo worker es donde están y se ejecutan los contenedores y pods. Esta formado por:

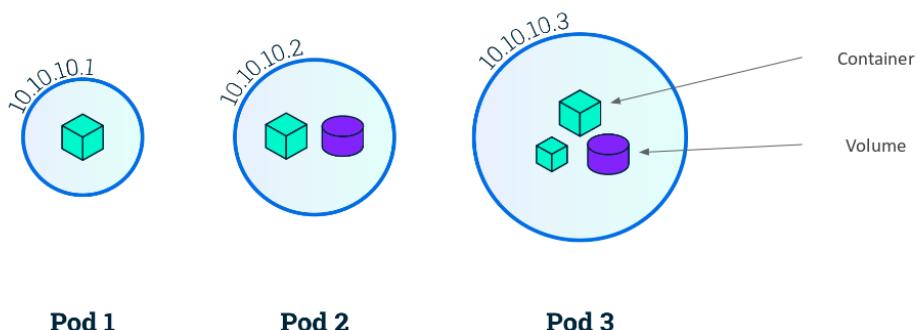
- Kubelet: Es el agent, el programa que se encarga de que todos los contenedores y pods estén en correcto funcionamiento.
- Docker --> Pods --> API
- Kube proxy: Define la red general. Se encarga de gestionar las IPs virtuales. Es el encargado de exponer los puertos de cada pod también.

Todo en Kubernetes es un Objeto. A la hora de crear un objeto la forma más habitual es utilizar un archivo YAML y especificar los atributos del objeto.

Un *pod* es un grupo de contenedores. Estos contenedores comparten los volúmenes del sistema y también los nombres de dominio. Para hacer una metáfora sería como una capa opaca. Interactúas directamente con el *pod* y él se encarga de los contenedores que los trata como instancias, en ningún momento ves los contenedores.

Los *pods* son efímeros. Se crean y se destruyen rápidamente. Por eso hay que pensar en el almacenamiento fijo no volátil y en su replicación (*ReplicaSet* ó *Deployment*). Son efímeros ya que si un *pod* se cae por cualquier razón, Kubernetes simplemente va a crear otro nuevo.

En Kubernetes todo es un Objeto y un *Pod* es el objeto más pequeño que se puede crear.



El *ReplicaSet* se encarga de las replicas de los pods. Es el utilizado para que no haya caídas. En el momento que detecta que falta uno, crea otro al instante. Por eso cuando se dice que en Kubernetes no hay caídas, el encargado de ello es el objeto *ReplicaSet*.

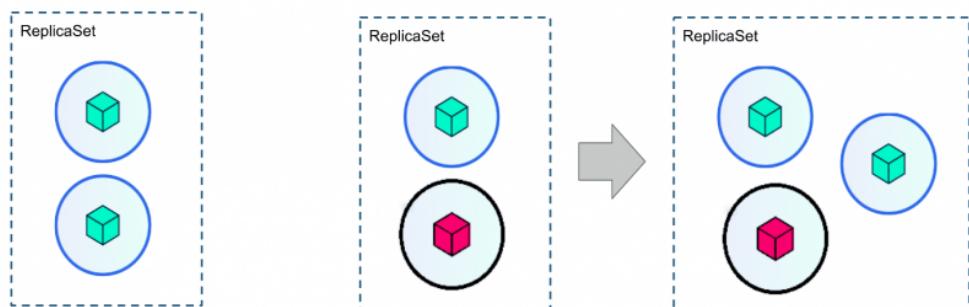
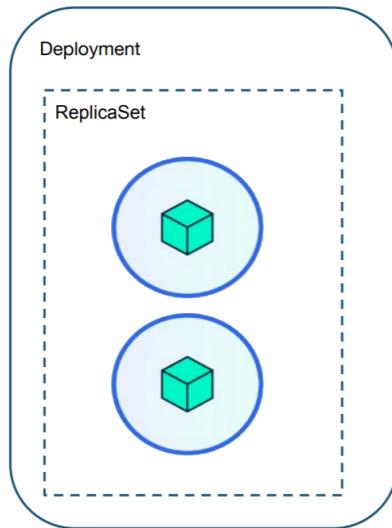


Imagen 1: *ReplicaSet* con dos *Pods* "sanos" (activos).

Imagen 2: El *ReplicaSet* detecta la caída de uno

Imagen 3: Automáticamente crea uno nuevo "sanoso" (activo).

El *Deployment* es el siguiente nivel, por encima del *ReplicaSet* y por encima de los *Pods*. Con él puedo controlar todos los apartados del *ReplicaSet* y *Pods*. Además, puedo escalar los *pods* por tipos, limpieza y volver a antiguas de versiones (RollBack).



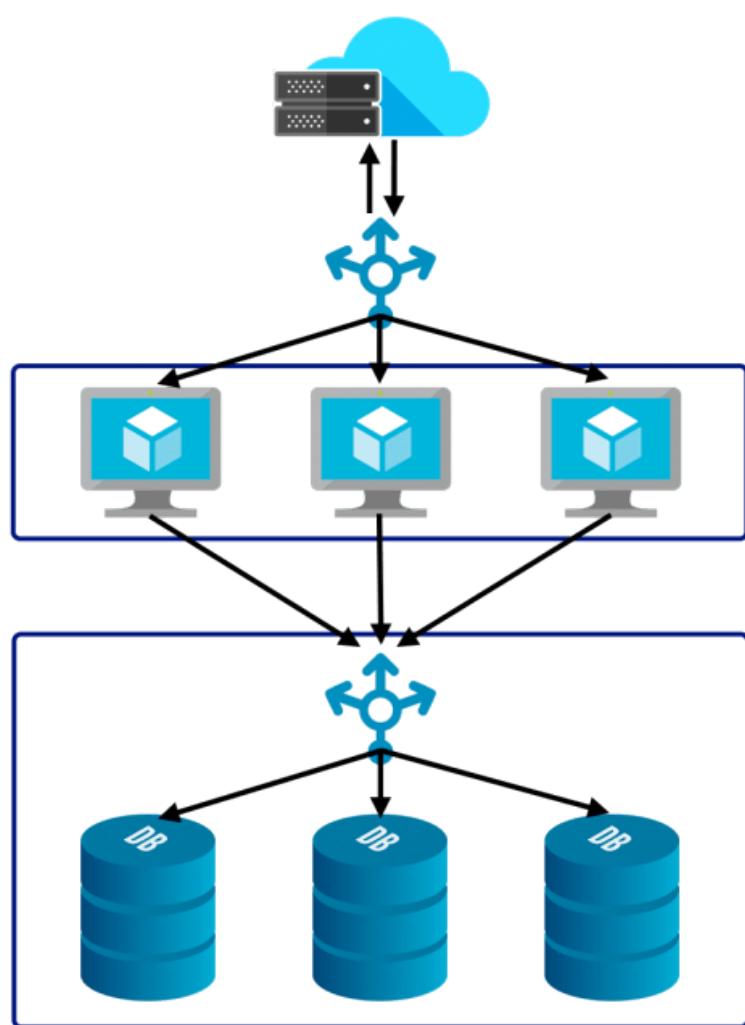
El espacio de nombres o *NameSpace* lo que te permite es aislar objetos. Solo los objetos dentro del mismo espacio de nombres van a verse entre sí.

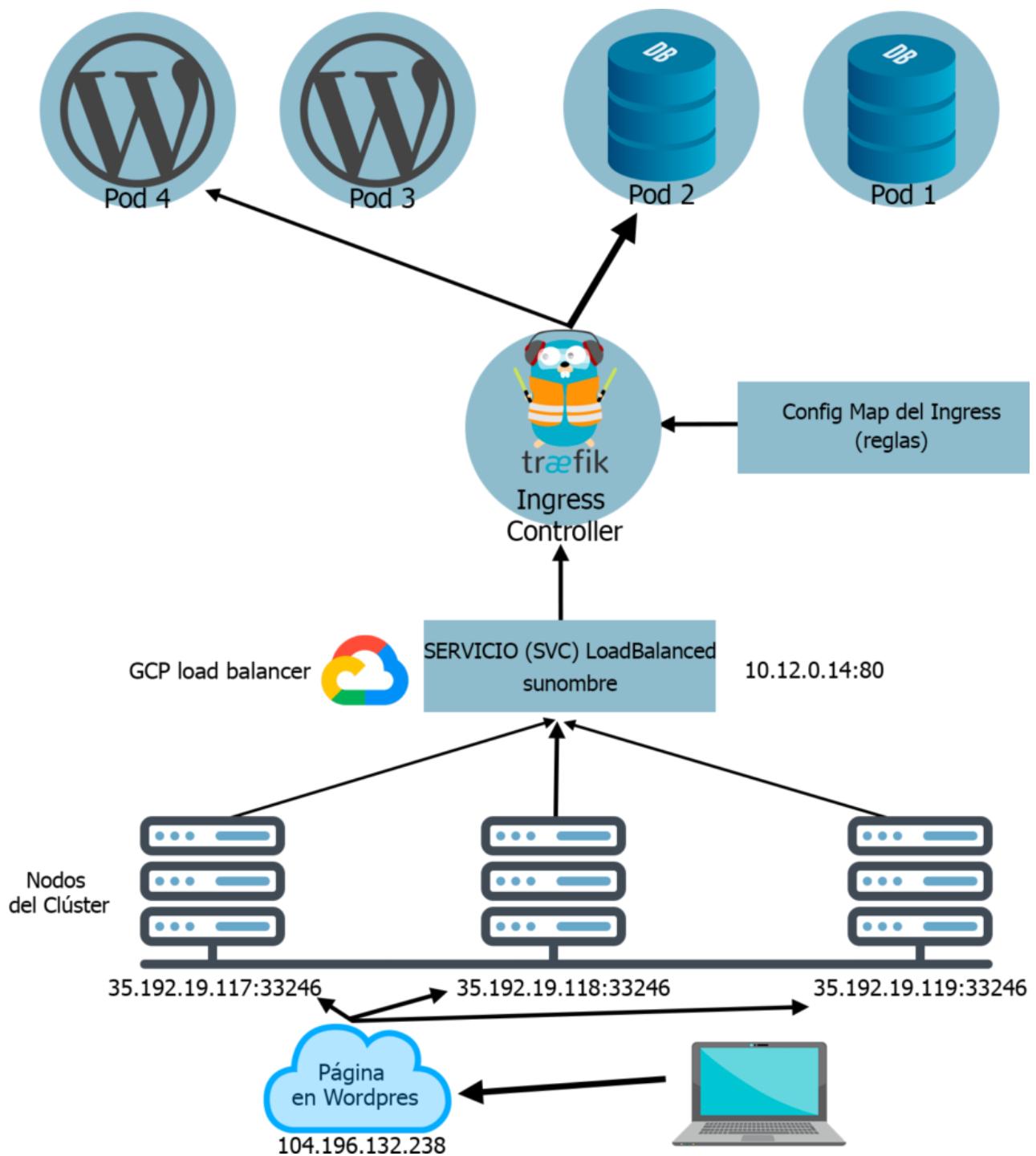
Un *Service* se utiliza para realizar comunicaciones entre *Pods*. Es una capa de abstracción que nos permite enrutar mediante las etiquetas (*labels*) cualquier puerto. Utiliza *kube-proxy* con *IPtables*. Hay diferentes tipos:

- ClusterIP: Por defecto. Permite la comunicación dentro del Pod.
- NodePort: Asigna un puerto a cada nodo.
- LoadBalancer: Solo en distribuidores Cloud.

Un *Ingress* es un objeto que va a permitir controlar muchos aspectos de la red en el cluster de Kubernetes. Puedo definirlo en cada servicio (tanto la comunicación interna como externa con salida a Internet). Permite hacer un balanceo de carga (load balancing), definir rutas HTTP y HTTPS, añadir SSL y definir nombres para cada servicio. Añadir un *Ingress* va además a ahorrar mucho dinero ya que no tendrá que darle un IP Pública a cada servicio, puedo definir una ruta hacia cada uno de ellos y que todos salgan desde la IP Pública del controlador.

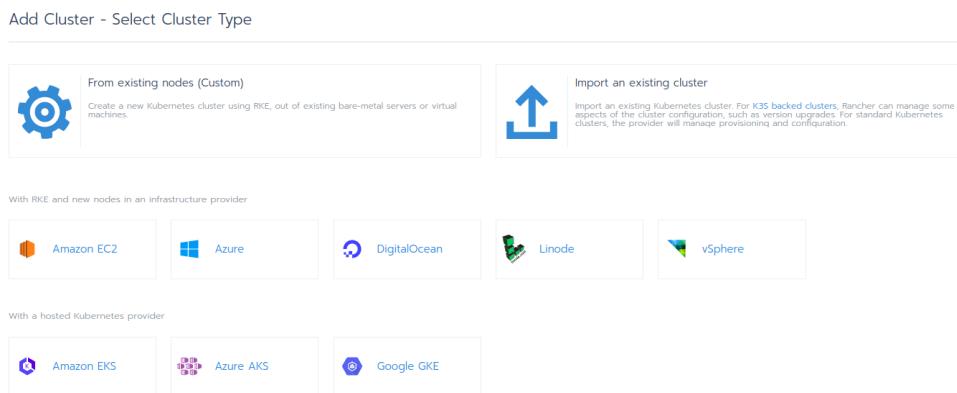
Al crear un *Ingress* tengo dos pasos. El primero es levantar el *Deployment* del *Ingress Controller* y sus archivos de configuración. El segundo es el crear el YAML del servicio tipo *LoadBalancer* o *NodePort* y especificar el *Ingress*.





Ahora ya puedo ver y acceder al panel de Rancher desde la IP de la máquina con Ubuntu Server.

Desde el panel de Rancher puedo crear clusters de Kuberentes al momento. Tiene las principales empresas Cloud como AWS, Azure, Google Cloud. También tiene opción para crear los nodos del cluster de Kubernetes en las máquinas de proveedores de servicios como Linode o DigitalOcean. Y también permite crear los nodos del cluster en máquinas locales propias (*custom*).



En este caso va a ser una aplicación real y como explique al inicio comprar grandes servidores no es sostenible al inicio. Por lo tanto voy a utilizar un proveedor Cloud. Mi elección a sido Google Cloud ya que además de ser el creador de Kubernetes, tiene una prueba gratuita de doce meses o trescientos dolares, lo que primero se acabe.

Para crear un cluster de Kuberentes en Rancher utilizándolo los servidores de Google Cloud tengo que crear una cuenta de gestor para Rancher para que pueda utilizar los recursos de mi cuenta. Lo primero es crear la cuenta de Google Cloud. Tan solo tengo que añadir una tarjeta de crédito a mi cuenta de Google normal: <https://cloud.google.com/free>. Una vez tenga la cuenta tengo que crear la cuenta de gestión (servicio) en administración de identidades y accesos (IAM).

Al pulsar sobre crear cuenta de servicio aparecen tres pestañas. La primera pestaña son los detalles de la cuenta de servicio. En la que introduzco el nombre de la cuenta de servicio, el ID y una breve descripción para identificarla.

Detalles de la cuenta de servicio

Nombre de cuenta de servicio	rancher
Nombre visible de esta cuenta de servicio	
ID de cuenta de serv...	rancher @durable-matter-230618.iam.gserviceaccount.com X C
Descripción de la cuenta de servicio	Cuenta de gestión de servicios para Rancher
Describe lo que hará esta cuenta de servicio	
CREAR CANCELAR	

La siguiente pestaña son los permisos que va tener la cuenta de servicio. Para que Rancher pueda crear e interactuar con el cluster de Kuberentes y los recursos necesita una serie de permisos:

- Lector de Compute Engine (*Compute Viewer*): Acceso de solo lectura para obtener y mostrar los recursos de Compute Engine sin poder leer los datos almacenados en ellos.
- Lector de proyecto (*Project Viewer*): Permiso para realizar acciones de solo lectura que no modifiquen ningún estado.
- Administrador de Kubernetes Engine (*Kubernetes Engine Admin*): Brinda acceso para administrar los clústeres de contenedor en su totalidad y los objetos de la API de Kubernetes.
- Usuario de cuenta de servicio (*Service Account User*): Permisos para ejecutar operaciones como cuenta de servicio.

Permisos de la cuenta de servicio (opcional)

Concede acceso a My Project a esta cuenta de servicio para que pueda realizar acciones concretas con los recursos del proyecto. [Más información](#)

Rol	Condición	
Lector de Compute	Añadir condición	
Acceso de solo lectura para obtener y mostrar información sobre todos los recursos de Compute Engine, incluidos discos, instancias y cortafuegos. Permite obtener y mostrar información sobre discos, imágenes y capturas, pero no permite la lectura de los datos almacenados en estos.		
Lector	Añadir condición	
Acceso de lectura a todos los recursos.		
Administrador de Kuberne...	Añadir condición	
Administración total de los clústeres de Kubernetes y de sus objetos de la API de Kubernetes.		
Usuario de cuenta de serv...	Añadir condición	
Permiso para ejecutar operaciones como cuenta de servicio.		

[+ AÑADIR OTRO ROL](#)

[CONTINUAR](#) [CANCELAR](#)

En la siguiente y última pestaña es donde descargaré el archivo JSON con la clave de acceso.

Conceder a los usuarios acceso a esta cuenta de servicio (opcional)

Grant access to users or groups that need to perform actions as this service account.
[Learn more](#)

Rol de usuarios de la cuenta de servicio	
Concede a los usuarios permisos para desplegar tareas y VMs con esta cuenta de servicio	
Rol de administradores de la cuenta de servicio	
Concede a los usuarios permiso para administrar esta cuenta de servicio	

Crear clave (opcional)

Descarga el archivo que contiene la clave privada. Guárdalo en un lugar seguro, ya que no podrás recuperar la clave si se pierde. Si no tienes claro si necesitas una clave, sáltate este paso de momento.

Claves

ID de clave
dfd359ce1b3ed1cb516172ab707317757d49825c



[+ CREAR CLAVE](#)

[LISTO](#) [CANCELAR](#)

Ya he finalizado la creación de la cuenta. Ahora con la clave de la cuenta de servicio en formato JSON vuelvo al panel de Rancher. Y en la pestaña de Clusters pulso sobre añadir un nuevo cluster (*Add a cluster*). Ahí seleccionó sobre Google GKE. Procedo a configuralo, escribo el nombre del cluster (*Cluster Name*). En *Member Roles* puedo seleccionar otros usuarios y los permisos que puedan tener sobre el cluster. En *Labels and Annotations* puedo añadir metadatos para las aplicaciones y para identificar el cluster.

En Service Account procedo a añadir el archivo JSON anteriormente creado en la cuenta de Google Cloud. Para ello pulso sobre *Read from a file*. Una vez seleccionado el archivo se desbloquean dos nuevas pestañas.

La primera nueva pestaña desbloqueada se llama *Kubernetes Options*. En ella puedo elegir:

- *Location Type*: La región o zona de los servidores de Google Cloud.
- *Kubernetes Version*: La versión de Kubernetes.
- *Alpha Features*: Activar o desactivar las nuevas novedades de Kubernetes.
- *Stackdriver Logging*: Permite almacenar, buscar, analizar, supervisar y utilizar alertas sobre eventos y datos de registros desde Google Cloud Platform.
- *Stackdriver Monitoring*: Permite ver con mayor claridad el rendimiento, el tiempo de funcionamiento y el estado general de las aplicaciones en la nube. Este sistema recopila métricas, eventos y metadatos de Google Cloud Platform.
- *Legacy Authorization*: Control de acceso basado en rol (RBAC) dentro de los recursos de Kuberentes.
- *Kubernetes Dashboard*: Es una interfaz de usuario basada en web de uso general para clústeres de Kubernetes
- *Htt Load Balancing*: El *Load Balancer* de capa 4 (o *Load Balancing* externo) reenvía el tráfico a *Nodeports*. Permite reenviar el tráfico HTTP y TCP.
- *Horizontal Pod Autoscaling*: Es una función de Kubernetes que permite configurar el clúster para escalar automáticamente los servicios que está ejecutando.
- *Maintenance Window*: Franja de tiempo en el cual se pueden realizar actualizaciones u otro tipo de mantenimiento.
- *Network*: Selecciona la red PVC del Cloud.
- *Node Subnet*: Permite seleccionar el rango de la sub red donde van a estar los contenedores.
- *IP Aliases*: Proporciona la capacidad de señalar alias de servicio en lugar de apuntar directamente a los servicios.

The screenshot shows the 'Kubernetes Options' configuration page. It includes fields for Location Type (Zonal selected), Region (europe-west3), Kubernetes Version (1.15.11-gke.3), Alpha Features (Enabled selected), Stackdriver Logging (Enabled selected), Stackdriver Monitoring (Enabled selected), Maintenance Window (3:00AM), Network (default), Node Subnet (Auto Create Subnetwork), and IP Aliases (Enabled selected).

En la segunda pestaña, *Node Options*, puedo seleccionar los aspectos técnicos de los nodos de Kubernetes dentro de Google Cloud.

- *Node Count*: El número de nodos iniciales del cluster de Kubernetes.
- *Machine Type*: El tipo de máquina de Google Cloud.
- *Image Type*: La imagen del sistema para la máquina del Google Cloud.
- *Root disk type*: El tipo del disco duro base para cada máquina. HDD o SSD.
- *Root Disk Size*: El tamaño del disco duro base para cada máquina.
- *Local SSD disks*: Añadir discos SSD a cada máquina.
- *Preemptible nodes (beta)*: Las máquinas interrumpibles son una característica de los proveedores Cloud. Son instancias de cómputo asequibles pero de corta duración. Duran aproximadamente un día. Al día se reinician por lo que tienen un pequeño corte en el servicio. Pero su precio es un 80% menor que una instancia normal.
- *Auto Upgrade*: Se va a actualizar automáticamente la versión de Kubernetes.
- *Auto Repair*: Si Kubernetes encuentra algún fallo en un nodo va a intentar solucionarlo o borrar ese nodo y crear uno nuevo. Realiza controles periódicos al estado de cada nodo en el clúster
- *Node Pool Autoscaling*: Si la carga de trabajo en un nodo es muy superior permite crear uno nuevo, esto es muy útil a la hora de una aplicación ya que no se va a quedar sin recursos.

Node Options		
Customize the nodes that will be created		
Node Count	n1-standard-2 (2 vCPUs, 7.5 GB RAM)	
Image Type	Container-Optimized OS	
Root Disk Size	100 GB	
Local SSD disks	0 GB	
Preemptible nodes (beta)	<input checked="" type="radio"/> Enabled <input type="radio"/> Disabled	
Auto Upgrade	<input checked="" type="radio"/> Enabled <input type="radio"/> Disabled	
Auto Repair	<input checked="" type="radio"/> Enabled <input type="radio"/> Disabled	
Node Pool Autoscaling		
Node Pool Autoscaling	Minimum Node Count: 1	Maximum Node Count: 5

Al pulsar sobre *Create* ya empezará a crearse el cluster. Pasará por distintos estados. Primero se ejecutará el *Provisioning* qué es cuando Rancher esta creando toda la arquitectura del cluster de Kuberentes.

Clusters						Add Cluster
State	Cluster Name	Provider	Nodes	CPU	RAM	
Provisioning	asir-proyecto	Google GKE	0	n/a	n/a	

Provisioning cluster c-kw49h...

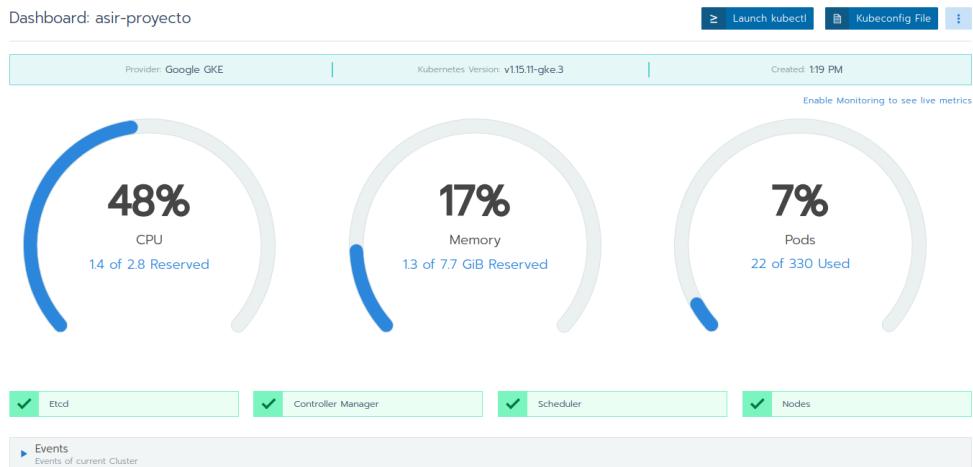
Después pasará a *Updating*, qué es cuando se instala Kubernetes y se actualiza a la versión puesta anteriormente.

The screenshot shows a table titled 'Clusters' with columns: State, Cluster Name, Provider, Nodes, CPU, and RAM. A search bar and an 'Add Cluster' button are at the top right. One row is highlighted in yellow, indicating the cluster is in the 'Updating' state. The cluster name is 'asir-proyecto'. The provider is Google GKE v1.15.11-gke.3, with 3 nodes, 15/2.8 Cores (52%), and 15/7.7 GB (20%) RAM.

Por último pasará a *Active* y ya podre interactuar con el cluster.

The screenshot shows the same 'Clusters' table. The 'Updating' row has been replaced by a new row where the 'State' column is now green and labeled 'Active'. The cluster name is still 'asir-proyecto'. The provider information remains the same: Google GKE v1.15.11-gke.3, 3 nodes, 15/2.8 Cores (52%), and 15/7.7 GB (20%) RAM.

Al pulsar sobre el nombre podré ver las estadísticas de consumo del cluste y otros avisos.



4.5

CREACIÓN DE DOCKER

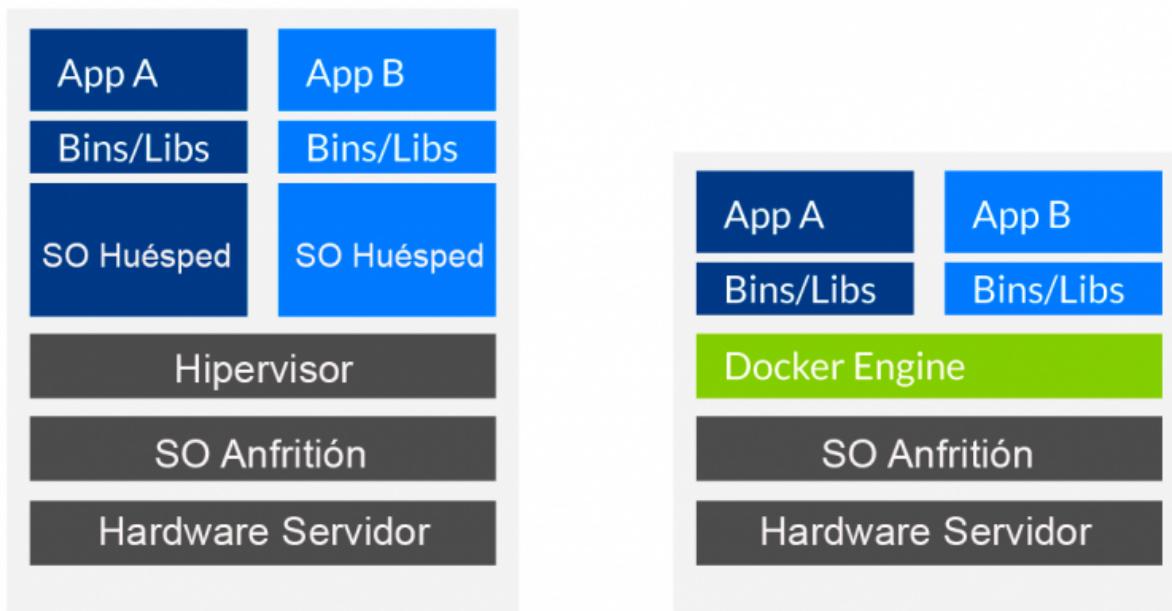
Docker es una tecnología creada en 2013 se ha convertido en algo indispensable a día de hoy para un administrador de sistemas. También para casi cualquier ámbito relacionado con la informática, especialistas IT o especialista TIC.

Es de código abierto (Open Source). Y se basa en la virtualización ligera. Esto es cuando no necesitas un sistema operativo completo. Solo una aplicación en concreto. Como por ejemplo MySQL o Apache. Y no necesitas instalar Ubuntu o Windows para poder utilizarla.

Una máquina virtual tradicional tiene que replicar todo un sistema operativo para utilizar una aplicación. Con Docker solo necesitas las librerías necesarias para esa aplicación.

Una máquina virtual tradicional en la que quiera instalar Apache, primero necesitaría instalar todo Debian y posteriormente las librerías propias de Apache. El total da más de 2GB. Y eso con la versión Lite. Un Docker de Apache ocupa menos de 100 MB ya que solo instala las librerías. Y es un Apache totalmente funcional y completo.

Máquinas Virtuales Vs Docker



Docker necesita varios elementos para funcionar correctamente:

- Docker Engine
 - Docker Daemon
 - Docker Client
 - API Rest
- Docker Image
- Dockerfile
- Volumes
- Docker Compose

Docker Engine es el motor de Docker. Son las herramientas que ejecutan, administran y compilan los contenedores y las imágenes. Se ejecuta de forma nativa en Linux. Esta formado a su vez del Deamon que se ejecuta en segundo plano y del Cliente que se comunica con el Deamon para ejecutar los comandos. Ademas de la API para interactuar de forma remota.

Docker Images son plantillas que se utilizan para construir los contenedores. Esta compuesto de varias capas. Las instrucciones se escriben en el Dockerfile. Suelen comenzar con una imagen base. Luego están las capas de herramientas, librerías del sistema y dependencias. Por último esta la capa de escritura (capa contenedor), en ella se almacenan los archivos y permite la personalización del contenedor. Así los contenedores pueden compartir la misma imagen base y tener su propia capa de escritura.

Dockerfile, como mencionaba, son una serie de instrucciones que van a crear una Imagen Docker. Las principales son:

- FROM: Elige la imagen base. Por ejemplo: FROM ubuntu:19.10
- LABEL: Metadatos en general. Por ejemplo la versión, el correo del desarrollador o la descripción de la imagen: LABEL maintainer="Desarrollador@ejemplo.es"
- RUN: Ejecuta un comando. Por ejemplo: RUN apt-get update
- EXPOSE: Asigna puertos. Por ejemplo: EXPOSE 80/udp
- ENV: Variable de Entorno. Por ejemplo: ENV NumeroManzanas=5

Volumenes. Son los datos de un contenedor. Se almacenan independientes del propio contenedor así que aunque se borre dicho contenedor los datos permanecen, y hay que borrarlos /editarlos manualmente. También si no se asigna ningún volumen entonces el contenedor es volátil, lo que significa que en el momento que se pare el contenedor los datos se pierden. Los volúmenes se pueden compartir entre varios contenedores. Son fáciles de hacer backup y migrar. Se pueden utilizar por linea de comandos (CLI) o por la API. Compatibles tanto en Contenedores Linux como en Windows. Compatible con Cloud.

Docker Compose utiliza ficheros YML para definir la configuración de varios contenedores al mismo tiempo.

Los beneficios de Docker se pueden ver desde distintos puntos según el tipo de informático que lo utilice.

Un administrador de sistemas tiene que garantizar que la actualización o nueva implementación no dañe nada del sistema actual. Para ello crea una copia (snapshot) del sistema actual y la utiliza. Hacer esto en Máquinas Virtuales (VM) es una tarea, sencilla, pero gasta muchos recursos. En cambio, con Docker se puede hacer con muy pocos recursos.

Un tester tiene que probar que todo funciona y no hay errores críticos y a ser posible, tampoco advertencias. Aquí Docker ayuda por ejemplo creando varios espacios de pruebas independientes pero con las mismas características. Y pudiendo borrarlos o crear otros de forma inmediata.

Un desarrollador necesita una serie de herramientas para que su código funcione. Por ejemplo, no es lo mismo Ubuntu 16 que Ubuntu 19. Tampoco es lo mismo Python 2 que Python 3. De versión a versión los cambios suelen ser notables. Pero con Docker puede tener la versión específica o todas, cada una en su propio contenedor Docker.

Esta tecnología tiene actualmente millones de desarrolladores utilizándola. Siendo las empresas más notables y pioneras Adobe, PayPal, AT&M, Tinder y Netflix.

Voy a comenzar definiendo los servicios y seguido escribiendo el Dockerfile de cada uno.

El primer servicio es un contenedor gestor. Tiene como base los binarios de Ubuntu y las librerías de:

- gcloud: La interfaz de línea de comandos gcloud es una herramienta que proporciona la CLI principal a Google Cloud Platform.
- Kubectl: Es una interfaz de línea de comandos para ejecutar comandos sobre despliegues clusterizados de Kubernetes. Esta interfaz es la manera estándar de comunicación con el clúster ya que permite realizar todo tipo de operaciones sobre el mismo.
- Servidor SSH: SSH o Secure Shell, es un protocolo de administración remota que le permite a los usuarios controlar y modificar sus servidores remotos a través de Internet a través de un mecanismo de autenticación.

Al acceder a este contenedor desde SSH puedo gestionar el cluster sin necesidad de instalar las herramientas en el dispositivo desde el que acceda, por lo tanto, puedo acceder desde cualquier tipo de dispositivo y utilizar las herramientas sin necesidad de que tengan compatibilidad para el dispositivo.

Peso total entre 170 y 185 MB.

El segundo servicio es un bot para redes sociales, específicamente orientado a Instagram. Un bot es un programa informático que permite automatizar tareas, como likes o follows. Utiliza el software de Instabot, escrito en python. Es OpenSource y está todo el código en Github. Permite realizar las siguientes acciones:

- Follow
 - Follow from hashtag
 - Follow followers
 - Follow following
 - Follow by likes on media
- Like
 - Like from hashtag(s)
 - Like followers
 - Like following
 - Like last media likers
 - Like our timeline
- Comment
 - Comment from hashtag
 - Comment spesific user media
 - Comment userlist
 - Comment our timeline
- Unfollow
 - Unfollow non followers
 - Unfollow everyone

El tercer servicio es similar al anterior. Es un bot para redes sociales solo que esta vez es para Twitter. Tambien esta basado en un proyecto OpenSource y el código fuente está en Github. Permite las siguientes acciones:

- Follow back users that follow you.
- Follow the followers of another user.
- Follow users based on a keyword.
- Follow users who retweeted a tweet.
- Unfollow users that don't follow you back.
- Unfollow all users.
- Like tweets based on a keyword.
- Unlike all tweets.
- Send a DM to users that follow you.
- Get follower and following count.

El cuarto servicio es orientado a juegos. En este servicio elegí el juego llamado Minecraft. Minecraft es un videojuego de construcción, de tipo mundo abierto. Este servicio lo que te permite es tener tu propio servidor de Minecraft. Esto te permite jugar en modo multijugador en un mapa personalizado.

El quinto servicio es orientado a almacenamiento de archivos. En este servicio utilzo Nextcloud. Nextcloud es un software que nos permite sincronizar cualquier tipo de archivo, carpetas, calendarios y contactos entre múltiples dispositivos. Es similar a Google Drive con la gran diferencia que los archivos están alojados de forma privada y además es Open Source.

El sexto y último servicio es orientado a almacenamiento web. En este servicio

4.5.1 SERVICIO GESTOR GCLOUD

El código está en el repositorio de Github. Es el siguiente código:

```
FROM ubuntu:18.04

# gcloud
RUN apt-get update && apt-get install lsb-release curl gnupg2 -y && apt-get clean all
RUN export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)" && \
    echo "deb http://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" | tee -a /etc/apt/sources.list.d/google-cloud-sdk.list && \
    curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && \
    apt-get update -y && apt-get install google-cloud-sdk -y

# Kubectl
RUN apt-get update && apt-get install -y apt-transport-https
RUN curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
RUN echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | tee -a /etc/apt/sources.list.d/kubernetes.list
RUN apt-get update && apt-get install kubectl -y

# Servidor SSH
RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:THEPASSWORDYOUCREATED' | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/sshd

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Utilizo los binarios de Ubuntu, la versión 18.04, posteriormente actualizo los paquetes y hago una instalación de las herramientas necesarias en cada división. Una vez se ejecute el contenedor lo primero que se ejecutará será el servidor SSH y el contenedor tendrá todas las herramientas de gestión ya instaladas.

- FROM: Defino la imagen del contenedor. En este caso utilizo el sistema operativo Ubuntu, específicamente la versión 18.04.

```
# gcloud
```

- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - apt-get update: Actualiza el índice de los repositorios.
 - apt-get install: Instala un paquete.
 - lsb-release: Es una herramienta propia de Linux que permite mostrar datos sobre el sistema.
 - curl: Es una herramienta que permite la transferencia de archivos.
 - gnupg2: Es una herramienta para la comunicación segura y el almacenamiento de datos. Se puede utilizar para cifrar datos y crear firmas digitales.
 - apt-get clean all: Limpia la caché del comando apt.
 - export CLOUD_SDK_REPO="cloud-sdk-\$(lsb_release -c -s)": Crea y guarda una variable del sistema llamada CLOUD_SDK_REPO con el valor cloud-sdk- seguido de la versión del sistema operativo. Posteriormente se utilizará en la instalación.

- echo "deb http://packages.cloud.google.com/apt \$CLOUD_SDK_REPO main" | tee -a /etc/apt/sources.list.d/google-cloud-sdk.list: El comando tee lee una entrada estándar y la escribe en la salida estándar. El comando echo imprime texto en pantalla, la salida estándar. Por lo tanto este comando escribe el repositorio de paquetes de GCLOUD utilizando la variable anteriormente creada para saber el sistema operativo y versión y lo añade sin sobreescibir en el archivo google-cloud-sdk.list
- curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -: Descarga la firma electrónica de los repositorios de Google y posteriormente la instala en el sistema.
- apt-get update -y && apt-get install google-cloud-sdk: Vuelvo a actualizar los indices con el nuevo repositorio y su firma e instalo el paquete ahora disponible de GCLOUD.

Kubectl

- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - apt-get update: Actualiza el índice de los repositorios.
 - apt-get install: Instala un paquete.
 - apt-transport-https: Es una serie de utilidades que habilitan el soporte https en aptitude para actualizar y instalar paquetes con una capa de seguridad SSL.
 - curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -: Descarga la firma electrónica de los repositorios de Google, la instala en el sistema y la guarda.
 - echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | tee -a /etc/apt/sources.list.d/kubernetes.list: El comando tee lee una entrada estándar y la escribe en la salida estándar. El comando echo imprime texto en pantalla, la salida estándar. Por lo tanto este comando escribe el repositorio de paquetes de Kubectl y lo añade sin sobreescibir en el archivo kubernetes.list.
 - apt-get update && apt-get install kubectl -y: Vuelvo a actualizar los indices con el nuevo repositorio y su firma e instalo el paquete ahora disponible de kubectl.

Servidor SSH

- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - apt-get update: Actualiza el índice de los repositorios.
 - apt-get install: Instala un paquete.
 - openssh-server: Es un conjunto de herramientas para el control remoto y la transferencia de datos entre ordenadores en red.
 - mkdir /var/run/sshd: Creo el directorio vacío donde va a ejecutarse el symlink al ejecutar SSH.
 - echo 'root:THEPASSWORDYOUCREATED' | chpasswd: Cambio la contraseña del usuario root. Y como no tenia, la establezco de forma temporal.
 - sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config: Permite al usuario root poder iniciar sesión con SSH.

```
# SSH login fix. Otherwise user is kicked off after login
```

- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/sshd: Arregla un bug conocido. Sino, el usuario se le vuelve a cerrar sesión. Pam_loginuid establece el atributo de proceso *loginuid* para la sesión que se autenticó.
- ENV: Variables de entorno.
 - NOTVISIBLE "in users profile": Ejemplo de una variable no visible cuando un usuario inicie sesión pero sí en el sistema general.
- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - echo "export VISIBLE=now" >> /etc/profile: Ejemplo de como poder utilizar variables dentro de cada perfil de cada usuario.
- EXPOSE: Indico el puerto a escuchar del contenedor. Este es el por defecto de los servidores SSH.
- CMD: Establece el comando que se ejecutará al ejecutar la imagen.
 - ["/usr/sbin/sshd", "-D"]: Inicia el *deamon* de SSH en segundo plano al inicio.

4.5.2

SERVICIO BOT INSTAGRAM

El código está en el repositorio de Github. Es el siguiente código:

```
FROM debian

RUN apt-get update -y && apt-get dist-upgrade -y
RUN apt-get install python-setuptools git python-pip python3-pip -y
RUN pip install -U instabot
RUN git clone https://github.com/instagrambot/instabot.git

WORKDIR /instabot
CMD sleep infinity
```

Utilizo los binarios de Debian, la última versión automáticamente, posteriormente actualizo los paquetes y hago una instalación de las herramientas necesarias. Una vez se ejecute el contenedor no se ejecutará nada, quedará a la espera en el directorio instabot para que el usuario pueda configurar a su gusto el bot.

- FROM: Defino la imagen del contenedor. En este caso utilizo el sistema operativo Debian, la última versión.
- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - apt-get update -y: Actualiza el índice de los repositorios.
 - apt-get dist-upgrade -y: Actualiza el sistema operativo.
 - apt-get install: Instala un paquete.
 - python-setuptools: Es una biblioteca con procesos de desarrollo de paquetes diseñada para facilitar el empaquetamiento de proyectos de Python al mejorar la biblioteca estándar de Python distutils.
 - git: Es un software de control de versiones. Me va a permitir clonar y configurar el repositorio del bot.
 - python-pip: Es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python.
 - python3-pip: Misma herramienta anterior pero con compatibilidad para la versión 3 de Python.
 - pip install -U instabot: Utilizo Pip, un gestor de paquetes de Python, para instalar el bot.
 - git clone https://github.com/instagrambot/instabot.git: Utilizo Git para clonar los archivos del repositorio oficial del bot.

- WORKDIR: Establece el directorio de trabajo para cualquier instrucción RUN, CMD, ENTRYPOINT, COPY y ADD que la siga en el Dockerfile./Twitter-Follow-and-Unfollow-Bot: Especifico el directorio del bot anteriormente instalado.
 - /instabot: Especifico el directorio del bot anteriormente instalado.
- CMD: Establece el comando que se ejecutará al ejecutar la imagen.
 - sleep infinity: Este comando permite dejar en suspensión el contenedor hasta que el usuario acceda o realice alguna acción.

4.5.3

SERVICIO BOT

TWITTER

El código está en el repositorio de Github. Es el siguiente código:

```
FROM python:3
RUN git clone https://github.com/yousefissa/Twitter-Follow-and-Unfollow-Bot.git
RUN pip install tweepy

WORKDIR /Twitter-Follow-and-Unfollow-Bot
CMD sleep infinity
```

Utilizo directamente los binarios de Python, sin necesidad de sistema operativo base, hago directamente la instalación de las herramientas necesarias. Una vez se ejecute el contenedor no se ejecutará nada, quedará a la espera en el directorio Twitter-Follow-and-Unfollow-Bot para que el usuario pueda configurar a su gusto el bot.

- FROM: Defino la imagen del contenedor. En este caso es directamente los binarios de Python.
- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - git clone https://github.com/yousefissa/Twitter-Follow-and-Unfollow-Bot.git: Utilizo Git para clonar los archivos del repositorio oficial del bot.
 - pip install tweepy: Utilizo Pip, un gestor de paquetes de Python, para instalar el bot.
- WORKDIR: Establece el directorio de trabajo para cualquier instrucción RUN, CMD, ENTRYPOINT, COPY y ADD que la siga en el Dockerfile.
 - /Twitter-Follow-and-Unfollow-Bot: Específico el directorio del bot anteriormente instalado.
- CMD: Establece el comando que se ejecutará al ejecutar la imagen.
 - sleep infinity: Este comando permite dejar en suspensión el contenedor hasta que el usuario acceda o realice alguna acción.

4.5.4 SERVICIO JUEGO

MINECRAFT

El código está en el repositorio de Github. Es el siguiente código:

```
FROM ubuntu:16.04

RUN apt-get update && apt-get install -y git default-jdk wget
RUN mkdir minecraft && \
    wget "https://hub.spigotmc.org//jenkins/job/BuildTools/lastSuccessfulBuild/artifact/target/BuildTools.jar" -O minecraft/BuildTools.jar && \
    git config --global core.autocrlf input && \
    java -jar minecraft/BuildTools.jar --rev latest
RUN apt-get purge -y --autoremove git wget
RUN echo "eula=true" > eula.txt && mkdir plugins

CMD java -Xms512m -Xmx1024m -jar spigot*.jar nogui
EXPOSE 25565
```

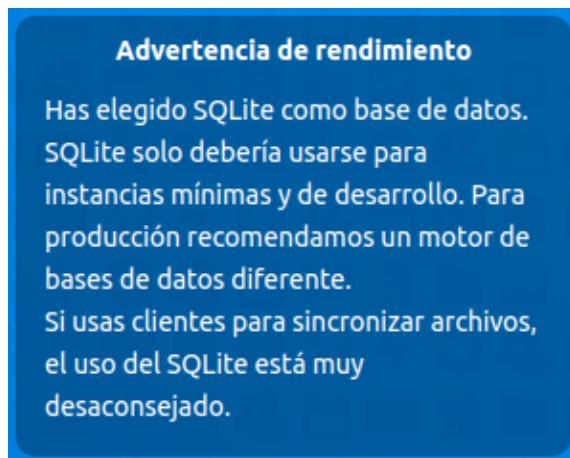
Utilizo los binarios de Ubuntu, la versión 16.04, posteriormente actualizo los paquetes y hago una instalación de las herramientas necesarias. Luego descargo y configuro el contenedor con Java. Una vez se ejecute el contenedor lo primero que se ejecutará será el servidor del juego y desde el primer momento cualquier usuario podrá jugar.

- FROM: Defino la imagen del contenedor. En este caso es Ubuntu, la versión 16.04.
- RUN: Ejecuta comandos en una capa a la hora de crear el contenedor, si el comando da error no continua la creación.
 - apt-get update: Actualiza el índice de los repositorios.
 - apt-get install: Instala un paquete.
 - git: Es un software de control de versiones. Me va a permitir clonar y configurar el repositorio con las herramientas del servidor.
 - default-jdk: Realiza la instalación estándar de Java compatible con el sistema.
 - wget: Herramienta que permite la descarga de contenidos desde servidores web.
 - mkdir minecraft: Crea una carpeta con el nombre de minecraft.
 - wget"https://hub.spigotmc.org//jenkins/job/BuildTools/lastSuccessfulBuild/artifact/targe t/BuildTools.jar" -O minecraft/BuildTools.jar: Descarga la última versión de las herramientas del servidor de Minecraft y lo guarda en la carpeta anteriormente creada minecraft bajo el nombre de BuildTools.jar. Un archivo JAR es un tipo de archivo que permite ejecutar aplicaciones y herramientas escritas en el lenguaje Java.
 - git config --global core.autocrlf input: Configura Git para garantizar que los finales de línea en los archivos sean correctos para Linux.
 - java -jar minecraft/BuildTools.jar --rev latest: Ejecuta el archivo anteriormente descargado, descarga con git la última versión de la aplicación disponible y la instala.
- apt-get purge -y --autoremove git wget: Una vez termina la instalación no necesito más las herramientas Git ni Wget, así que las desinstalo completamente del sistema.
- echo "eula=true" > eula.txt: Confirmo el contrato de Minecraft, es necesario para poder utilizar el software.
- mkdir plugins: Creo una carpeta de plugins para futura utilización.

- CMD: Establece el comando que se ejecutará al ejecutar la imagen.
 - `java -Xms512m -Xmx1024m -jar spigot-*jar nogui`: Ejecuta la aplicación Java, sin GUI, en segundo plano y utiliza siempre la última versión del programa. Xmx especifica la asignación máxima de memoria para la aplicación. Xms especifica la memoria inicial y mínima.
- EXPOSE: Indico el puerto a escuchar del contenedor. Este es el por defecto de los servidores de Minecraft.

4.5.5 SERVICIO ARCHIVOS NEXTCLOUD

Este servicio esta formado por dos contenedores diferenciados. Por un lado está Nextcloud, el sistema para almacenar los archivos. Puede funcionar con una base de datos llamada SQLite incluida con el propio Nextcloud, pero tal y como dice en la instalación, no es lo recomendable en entornos de producción.



Por lo tanto necesito desplegar otro contenedor con un motor de base de datos más robusto. He elegido MySQL por ser el estándar en sistemas de bases de datos relacionales.

El código está en el repositorio de Github. Es el siguiente código:

```
FROM mysql:5

ENV MYSQL_ROOT_PASSWORD="t[W=a;G1xprNxIoP7Yv+" \
    MYSQL_DATABASE="bd-nextcloud" \
    MYSQL_USER="usuario-nextcloud" \
    MYSQL_PASSWORD="7qYFwDb+ptC[bqnBA0[ ]"

EXPOSE 3306
```

- FROM: Defino la imagen del contenedor. En este caso es MySQL, la versión 5. Esta versión es la última de las *General Availability (GA) Releases*, la diferencia con las nuevas versiones es que es más estable.
- ENV: Variables de entorno.
 - MYSQL_ROOT_PASSWORD: Es la contraseña del usuario administrador root de todo el sistema de base de datos.
 - MYSQL_DATABASE: Especifico el nombre de la base de datos a crear.
 - MYSQL_USER: Especifico el nombre del usuario asociado a la anterior base de datos creada.
 - MYSQL_PASSWORD: Especifico la contraseña del anterior usuario.
- EXPOSE: Indico el puerto a escuchar del contenedor. Este es el por defecto de MySQL.

El siguiente contenedor es el propio de Nextcloud, el sistema de almacenamiento de archivos. El código está en el repositorio de Github. Es el siguiente código:

```
FROM nextcloud

ENV NEXTCLOUD_ADMIN_USER="administrador" \
    NEXTCLOUD_ADMIN_PASSWORD="s;0MC1}c|Vj4@rAgRFs8"
ENV MYSQL_DATABASE="bd-nextcloud" \
    MYSQL_USER="usuario-nextcloud" \
    MYSQL_PASSWORD="7qYFwDb+ptC[bqnBA0[]" \
    MYSQL_HOST="172.16.0.3"

EXPOSE 80
EXPOSE 443
```

- FROM: Defino la imagen del contenedor. En este caso es la última versión de Nextcloud que utiliza apache como servidor HTTP.
- ENV: Variables de entorno.
 - NEXTCLOUD_ADMIN_USER: Es el nombre del usuario administrador de Nextcloud.
 - NEXTCLOUD_ADMIN_PASSWORD: Especifico la contraseña del usuario administrador anteriormente definido.
 - MYSQL_DATABASE: Especifico el nombre de la base de datos del contenedor MySQL.
 - MYSQL_USER: Especifico el nombre del usuario asociado a la base del contenedor MySQL.
 - MYSQL_PASSWORD: Especifico la contraseña del anterior usuario del contenedor MySQL.
 - MYSQL_HOST: Es la IP o ruta del contenedor MySQL.
- EXPOSE: Indico el puerto a escuchar del contenedor. Especifico tanto el puerto 80 para HTTP como el 443 para HTTPS.

4.6 AUTOMATIZAR

GITHUB ACTIONS

GitHub permite a los desarrolladores almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre dicho código. Una Versión de Control ayuda a los desarrolladores a llevar un registro y administrar cualquier cambio en el código del proyecto. A medida que crece este proyecto, la versión de control se vuelve esencial. Todos los cambios son registrados y pueden ser revertidos si es necesario.

Git es un sistema de control específico de versión de fuente abierta creada por Linus Torvalds en el 2005. Es un sistema de control de versión distribuida, lo que quiere decir que tanto el código como su historial se encuentran disponibles cada ordenador de cada desarrollador, lo cual permite un fácil acceso a las bifurcaciones y fusiones.

GitHub es una compañía propiedad de Microsoft que ofrece un servicio de hosting de repositorios almacenados en la nube. Hace que sea más sencillo tanto para desarrolladores individuales como para equipos utilizar Git como la versión de control.

GitHub Actions es una característica de GitHub que permite crear flujos de trabajo de ciclo de vida (SDLC) de software personalizados directamente cada repositorio GitHub. El ciclo de desarrollo de sistemas (SDLC) es el proceso de creación o modificación de los sistemas, modelos y metodologías que los desarrolladores utilizan para crear software. Cada SDLC es diferente pero tiene seis fases base:

- Plan: Recolección de los requerimientos mínimos del programa.
- Análisis: Entender los problemas y buscar soluciones.
- Diseño: Diseño de la interfaz, los planos y recursos.
- Programación: Escribir el código y tener configurados los servidores.
- Pruebas: Utilizar diferentes tipos de pruebas y niveles para probar el código.
- Mantenimiento: Despues de levantar el servicio, realizar monitoreo, evaluación e ir añadiendo características con el tiempo.

Con GitHub Actions puedo crear acciones de integración continua (CI, por sus siglas en inglés) de extremo a extremo y de desarrollo continuo (CD, por sus siglas en inglés). Con la integración continua (CI), cada vez que se añade código se construye la aplicación y se realizan pruebas automáticas. El desarrollo continuo (CD) es suma a la integración continua (CI) y luego el despliegue de la aplicación.

Para realizarlo tan solo tengo que ir a Github. El primer paso es crear el repositorio donde va a estar almacenado todo el código tanto de los servicios como de la página.

Tanto el nombre del repositorio del proyecto como la descripción van a ser en inglés ya que es el idioma más entendible de forma internacional.

El nombre del repositorio es: open-source-hosting

Y la descripción es: Full platform. FrontEnd: HTML5/CSS3/JS. BackEnd: NodeJS (main page) & PHP (for the Wordpress blog). Services: Google Cloud, Kubernetes & Docker.

El repositorio es público así cualquier persona puede verlo y realizar modificaciones en el código. Luego está inicializado con un README. Al editar ese archivo podrá mostrar una definición del proyecto y unas prácticas de uso.

El .gitignore es un archivo que especifica las terminaciones o nombres de los archivos que no quiero que se suban al repositorio. Y por último la licencia es GNU GPLv3, esta licencia permite realizar modificaciones en el proyecto pero no permite utilizarlo a terceros de forma comercial.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner Repository name *

 RedxLus  / **open-source-hosting** 

Great repository names are short and memorable. Need inspiration? How about [silver-fiesta](#)?

Description (optional)

Full platform. FrontEnd: HTML5/CSS3/JS. BackEnd: NodeJS (main page) & PHP (for the Wordpress blog). S

 **Public**
Anyone can see this repository. You choose who can commit.

 **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: Node  Add a license: GNU General Publi...  

Create repository

Ahora ya solo queda subir el código de los anteriores servicios. Por ello creo una clonación local vacía en mi ordenador con el comando:

```
git clone https://github.com/RedxLus/open-source-hosting.git
```

Una vez tenga el repositorio en local ya puedo organizar la estructura del proyecto. Voy a crear una carpeta principal llamada HOSTING y luego subdividirla con carpetas individuales del género de cada servicio en inglés para mantener un mismo idioma en todo el repositorio. Por ejemplo: BOTS, GAMES o STORAGE. Dentro de estas carpetas irá otra carpeta con el nombre en minúscula del servicio. Por lo tanto los archivos relativos al bot de Instagram siguen la siguiente ruta:

open-source-hosting:

HOSTING

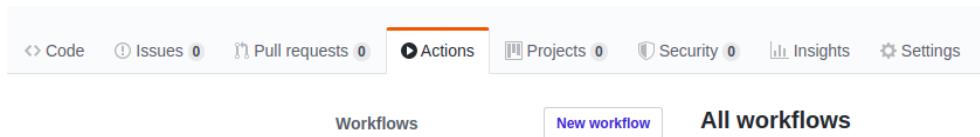
```
|-- BOTS  
|   |-- Instagram  
|   |-- Twitter  
|-- GAMES  
|   |-- Minecraft  
...
```

Una vez subidos todos los archivos a la carpeta local y realizado la estructura para subir los archivos utilizando Git al repositorio de GitHub utilice el siguiente comando:

```
git add . && git commit -a -m "New upload: dockerfiles and structure" && git push
```

Con *git add* lo que hago es añadir todos los cambios en los archivos de mi repositorio local al "Staging Area" de Git en donde queda a espera de la revisión con el comando *git commit* al que le agrego un mensaje. Con *git push* lo subo al repositorio de GitHub.

Una vez ya está todo el código en el repositorio voy a la pestaña en GitHub llamada *Actions*. Y ahí pulso sobre *New workflow*.



Selecciono sobre *Docker Image* ya que lo quiero automatizar la creación de las imágenes a partir de los dockerfiles previamente subidos.



Una vez pulsado sobre *Set up this workflow* se va a crear un directorio oculto, ya que comienza con el símbolo punto, llamado .github y otro subdirectorio llamado workflows que es donde van a estar los archivos de configuración individuales para cada GitHub Actions.

Ahora procedo a crear cada archivo individual siguiendo el mismo procedimiento.

4.6.1 AÑADIR SECRETOS EN GITHUB

Los secretos son variables de entorno que se cifran y solo se exponen a acciones seleccionadas. Cualquier persona con acceso de colaborador al repositorio puede usar estos secretos en un flujo de trabajo.

Los secretos no se pasan a los flujos de trabajo de terceros si hacen un fork del repositorio.

Para crear un secreto voy a la pestaña de configuración dentro del proyecto. Y seleccionó Secretos. Ahí puedo añadir o borrar secretos.

The screenshot shows the GitHub repository settings interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. The Settings tab is active. On the left, a sidebar menu lists Options, Manage access, Branches, Webhooks, Notifications, Integrations, Deploy keys, **Secrets** (which is selected and highlighted with an orange border), and Actions. The main content area is titled "Secrets". It contains a note explaining that secrets are environment variables encrypted and exposed to selected actions, available to collaborators. It also states that secrets are not passed to workflows triggered by pull requests from forks. Below this, there are two existing secrets: "PASS_DOCKER" and "USER_DOCKER", each with a "Remove" button. A "Add a new secret" section follows, with fields for "Name" (containing "YOUR_SECRET_NAME") and "Value" (a large text input field). A green "Add secret" button is at the bottom of this section.

4.6.2 ACTIONS GESTOR GCLOUD

El código está en el repositorio de Github. Es el siguiente código:

```
17 lines (17 sloc) | 692 Bytes
Raw Blame History ⚙️ 🗑️
1 name: Google Cloud - Docker Image CI
2 on:
3   schedule:
4     - cron: "0 0 1 * *"
5   jobs:
6     build:
7       runs-on: ubuntu-latest
8       steps:
9         - uses: actions/checkout@v2
10        - name: Build the Docker image of HOSTING --> ACCESS --> GCloud
11        run: |
12          docker build --file HOSTING/ACCESS/dockerfile --tag gcloud .
13          docker tag gcloud:latest ${secrets.USER_DOCKER}/gcloud-sdk-google-cloud:${date +%Y-%B}
14        - name: Docker login
15        run: docker login --username='${secrets.USER_DOCKER}' --password=${secrets.PASS_DOCKER}
16        - name: Docker push image of HOSTING --> ACCESS --> GCloud
17        run: docker push ${secrets.USER_DOCKER}/gcloud-sdk-google-cloud:${date +%Y-%B}
```

Crea cada inicio de mes la imagen del gestor de Gcloud del dockerfile y la sube al repositorio de imágenes Docker Hub con mis credenciales encriptadas.

- name: Es el nombre del *workflow*. No influye en el resultado pero es útil para ordenarlos.
- on: Indico a continuación como va a iniciar y activar el *workflow*. A continuación se puede indicar *push* o *schedule*. Cuando se indica *push* se va a activar cuando se produzcan cambios en cualquier rama del repositorio y filtrar por tags. Cuando se indica *schedule* utiliza la sintaxis cron POSIX para especificar un rango de tiempo.
 - schedule: Utilizo la sintaxis cron POSIX para especificar un rango de tiempo.
 - - cron: "0 0 1 * *": Esta expresión es para que se ejecute a las 00:00 del primer día de cada mes. Todos los meses de todos los años.
- jobs: Indico a continuación las características del servidor que va a crear la imagen y cada una de las fases de la construcción.
 - build: Permite hacer varias imágenes distintas en un mismo *workflow*.
 - runs-on: El sistema operativo del servidor que va a crear la imagen. En este caso he elegido la última versión de Ubuntu ya que está orientada a servidores Linux.
 - steps: Los pasos de cada una de las fases de la construcción:
 - - uses: actions/checkout@v2: GitHub actions permite utilizar plugins. Este es una acciones estándar para realizar una copia del código del repositorio. Ahora la máquina con Ubuntu tiene una copia completa del repositorio y puedo interactuar con ella.
 - - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker build --file HOSTING/ACCESS/dockerfile --tag gcloud . : Con este comando propio de docker creo la imagen del contenedor basada en el dockerfile de Gcloud.

- docker tag gcloud:latest \${{ secrets.USER_DOCKER }}/gcloud-sdk-google:\$date +%Y-%B: Con este comando etiqueto la imagen anteriormente creada y le pongo el nombre de mi usuario de Docker Hub seguido del nombre de la imagen del repositorio de Docker Hub. Por último le añado la versión que para identificarla va a utilizar el comando de Linux date y va a ser el año de creación seguido del mes.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker login --username='\${{ secrets.USER_DOCKER }}' --password=\${{ secrets.PASS_DOCKER }}: Inicia sesión en Docker Hub con mis credenciales cifradas.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker push \${{ secrets.USER_DOCKER }}/gcloud-sdk-google-cloud:\$date +%Y-%B: Sube la imagen con la etiqueta y la versión al repositorio de Docker Hub con mis credenciales cifradas.

4.6.3

ACTIONS BOT INSTAGRAM

El código está en el repositorio de Github. Es el siguiente código:

```
17 lines (17 sloc) | 698 Bytes
Raw Blame History
1 name: Instagram Bot - Docker Image CI
2 on:
3   schedule:
4     - cron: '0 0 1 * *'
5   jobs:
6     build:
7       runs-on: ubuntu-latest
8       steps:
9         - uses: actions/checkout@v2
10        - name: Build the Docker image of HOSTING --> BOTS --> Instagram
11        run: |
12          docker build --file HOSTING/BOTS/Instagram/dockerfile --tag instagram-bot .
13          docker tag instagram-bot:latest ${ secrets.USER_DOCKER }/instagram-bot:${(date +%Y-%B)}
14        - name: Docker login
15        run: docker login --username='${ secrets.USER_DOCKER }' --password=${ secrets.PASS_DOCKER }
16        - name: Docker push image of HOSTING --> BOTS --> Instagram
17        run: docker push ${ secrets.USER_DOCKER }/instagram-bot:${(date +%Y-%B)}
```

Crea cada inicio de mes la imagen del bot de Instagram del dockerfile y la sube al repositorio de imágenes Docker Hub con mis credenciales encriptadas.

- name: Es el nombre del *workflow*. No influye en el resultado pero es útil para ordenarlos.
- on: Indico a continuación como va a iniciar y activar el *workflow*. A continuación se puede indicar *push* o *schedule*. Cuando se indica *push* se va a activar cuando se produzcan cambios en cualquier rama del repositorio y filtrar por *tags*. Cuando se indica *schedule* utiliza la sintaxis cron POSIX para especificar un rango de tiempo.
 - *schedule*: Utilizo la sintaxis cron POSIX para especificar un rango de tiempo.
 - - cron: "0 0 1 * *": Esta expresión es para que se ejecute a las 00:00 del primer día de cada mes. Todos los meses de todos los años.
- jobs: Indico a continuación las características del servidor que va a crear la imagen y cada una de las fases de la construcción.
 - *build*: Permite hacer varias imágenes distintas en un mismo *workflow*.
 - *runs-on*: El sistema operativo del servidor que va a crear la imagen. En este caso he elegido la última versión de Ubuntu ya que está orientada a servidores Linux.
 - *steps*: Los pasos de cada una de las fases de la construcción:
 - - *uses*: *actions/checkout@v2*: GitHub actions permite utilizar plugins. Este es una acciones estándar para realizar una copia del código del repositorio. Ahora la máquina con Ubuntu tiene una copia completa del repositorio y puedo interactuar con ella.
 - - *name*: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - *run*: Permite realizar comandos dentro la máquina.
 - *docker build --file HOSTING/BOTS/Instagram/dockerfile --tag instagram-bot .*: Con este comando propio de docker creo la imagen del contenedor basada en el dockerfile del bot de Instagram.

- docker tag instagram-bot:latest \${ secrets.USER_DOCKER }/instagram-bot:\$(date +%Y-%B): Con este comando etiqueto la imagen anteriormente creada y le pongo el nombre de mi usuario de Docker Hub seguido del nombre de la imagen del repositorio de Docker Hub. Por último le añado la versión que para identificarla va a utilizar el comando de Linux date y va a ser el año de creación seguido del mes.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker login --username='\${ secrets.USER_DOCKER }' --password=\${ secrets.PASS_DOCKER }: Inicia sesión en Docker Hub con mis credenciales cifradas.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker push \${ secrets.USER_DOCKER }/instagram-bot:\$(date +%Y-%B): Sube la imagen con la etiqueta y la versión al repositorio de Docker Hub con mis credenciales cifradas.

4.6.4 ACTIONS BOT TWITTER

El código está en el repositorio de Github. Es el siguiente código:

```
17 lines (17 sloc) | 681 Bytes
Raw Blame History ⚙️ 🗑️
1 name: Twitter Bot - Docker Image CI
2 on:
3   schedule:
4     - cron: '0 0 1 * *'
5   jobs:
6     build:
7       runs-on: ubuntu-latest
8       steps:
9         - uses: actions/checkout@v2
10        - name: Build the Docker image of HOSTING --> BOTS --> Twitter
11        run: |
12          docker build --file HOSTING/BOTS/Twitter/dockerfile --tag twitter-bot .
13          docker tag twitter-bot:latest ${{ secrets.USER_DOCKER }}/twitter-bot:${{ date +%Y-%B }}
14        - name: Docker login
15        run: docker login --username='${{ secrets.USER_DOCKER }}' --password=${{ secrets.PASS_DOCKER }}
16        - name: Docker push image of HOSTING --> BOTS --> Twitter
17        run: docker push ${{ secrets.USER_DOCKER }}/twitter-bot:${{ date +%Y-%B }}
```

Crea cada inicio de mes la imagen del bot de Twitter del dockerfile y la sube al repositorio de imágenes Docker Hub con mis credenciales encriptadas.

- name: Es el nombre del *workflow*. No influye en el resultado pero es útil para ordenarlos.
- on: Indico a continuación como va a iniciar y activar el *workflow*. A continuación se puede indicar *push* o *schedule*. Cuando se indica *push* se va a activar cuando se produzcan cambios en cualquier rama del repositorio y filtrar por *tags*. Cuando se indica *schedule* utiliza la sintaxis cron POSIX para especificar un rango de tiempo.
 - schedule: Utilizo la sintaxis cron POSIX para especificar un rango de tiempo.
 - - cron: "0 0 1 * *": Esta expresión es para que se ejecute a las 00:00 del primer día de cada mes. Todos los meses de todos los años.
- jobs: Indico a continuación las características del servidor que va a crear la imagen y cada una de las fases de la construcción.
 - build: Permite hacer varias imágenes distintas en un mismo *workflow*.
 - runs-on: El sistema operativo del servidor que va a crear la imagen. En este caso he elegido la última versión de Ubuntu ya que está orientada a servidores Linux.
 - steps: Los pasos de cada una de las fases de la construcción:
 - - uses: actions/checkout@v2: GitHub actions permite utilizar plugins. Este es una acciones estándar para realizar una copia del código del repositorio. Ahora la máquina con Ubuntu tiene una copia completa del repositorio y puedo interactuar con ella.
 - - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker build --file HOSTING/BOTS/Twitter/dockerfile --tag twitter-bot . : Con este comando propio de docker creo la imagen del contenedor basada en el dockerfile del bot de Twitter.

- docker tag twitter-bot:latest \${ secrets.USER_DOCKER }/twitter-bot:\$(date +%Y-%B): Con este comando etiqueto la imagen anteriormente creada y le pongo el nombre de mi usuario de Docker Hub seguido del nombre de la imagen del repositorio de Docker Hub. Por último le añado la versión que para identificarla va a utilizar el comando de Linux date y va a ser el año de creación seguido del mes.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker login --username='\${ secrets.USER_DOCKER }' --password=\${ secrets.PASS_DOCKER }: Inicia sesión en Docker Hub con mis credenciales cifradas.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker push \${ secrets.USER_DOCKER }/twitter-bot:\$(date +%Y-%B): Sube la imagen con la etiqueta y la versión al repositorio de Docker Hub con mis credenciales cifradas.

4.6.5 ACTIONS JUEGO

MINECRAFT

El código está en el repositorio de Github. Es el siguiente código:

```
17 lines (17 sloc) | 702 Bytes
Raw Blame History ⚙️ 🗑️
1 name: Minecraft Server - Docker Image CI
2 on:
3   schedule:
4     - cron: '0 0 1 * *'
5   jobs:
6     build:
7       runs-on: ubuntu-latest
8       steps:
9         - uses: actions/checkout@v2
10        - name: Build the Docker image of HOSTING --> GAMES --> Minecraft
11        run: |
12          docker build --file HOSTING/GAMES/Minecraft/dockerfile --tag minecraft .
13          docker tag minecraft:latest ${{ secrets.USER_DOCKER }}/minecraft-server:${{ date +%Y-%B }}
14        - name: Docker login
15        run: docker login --username='${{ secrets.USER_DOCKER }}' --password=${{ secrets.PASS_DOCKER }}
16        - name: Docker push image of HOSTING --> GAMES --> Minecraft
17        run: docker push ${{ secrets.USER_DOCKER }}/minecraft-server:${{ date +%Y-%B }}
```

Crea cada inicio de mes la imagen del servidor de juego Minecraft del dockerfile y la sube al repositorio de imágenes Docker Hub con mis credenciales encriptadas.

- name: Es el nombre del *workflow*. No influye en el resultado pero es útil para ordenarlos.
- on: Indico a continuación como va a iniciar y activar el *workflow*. A continuación se puede indicar *push* o *schedule*. Cuando se indica *push* se va a activar cuando se produzcan cambios en cualquier rama del repositorio y filtrar por *tags*. Cuando se indica *schedule* utiliza la sintaxis cron POSIX para especificar un rango de tiempo.
 - schedule: Utilizo la sintaxis cron POSIX para especificar un rango de tiempo.
 - - cron: "0 0 1 * *": Esta expresión es para que se ejecute a las 00:00 del primer día de cada mes. Todos los meses de todos los años.
- jobs: Indico a continuación las características del servidor que va a crear la imagen y cada una de las fases de la construcción.
 - build: Permite hacer varias imágenes distintas en un mismo *workflow*.
 - runs-on: El sistema operativo del servidor que va a crear la imagen. En este caso he elegido la última versión de Ubuntu ya que está orientada a servidores Linux.
 - steps: Los pasos de cada una de las fases de la construcción:
 - - uses: actions/checkout@v2: GitHub actions permite utilizar plugins. Este es una acciones estándar para realizar una copia del código del repositorio. Ahora la máquina con Ubuntu tiene una copia completa del repositorio y puedo interactuar con ella.
 - - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker build --file HOSTING/GAMES/Minecraft/dockerfile --tag minecraft . : Con este comando propio de docker creo la imagen del contenedor basada en el dockerfile del servidor de juego Minecraft.

- docker tag minecraft:latest \${ secrets.USER_DOCKER }}/minecraft-server:\${(date +%Y-%B)}: Con este comando etiqueto la imagen anteriormente creada y le pongo el nombre de mi usuario de Docker Hub seguido del nombre de la imagen del repositorio de Docker Hub. Por último le añado la versión que para identificarla va a utilizar el comando de Linux date y va a ser el año de creación seguido del mes.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker login --username='\${ secrets.USER_DOCKER }' --password=\${ secrets.PASS_DOCKER }): Inicia sesión en Docker Hub con mis credenciales cifradas.
- - name: Es el nombre del paso. Se muestra luego en el *workflow* y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker push \${ secrets.USER_DOCKER }}/minecraft-server:\${(date +%Y-%B)}: Sube la imagen con la etiqueta y la versión al repositorio de Docker Hub con mis credenciales cifradas.

4.6.6 ACTIONS

NEXTCLOUD Y MYSQL

El código está en el repositorio de Github. Es el siguiente código:

```
24 lines (24 sloc) | 1.09 KB
Raw Blame History
1 name: Nextcloud & MySQL - Docker Image CI
2 on:
3   push:
4     branches:
5       - master
6   jobs:
7     build:
8       runs-on: ubuntu-latest
9       steps:
10      - uses: actions/checkout@v2
11      - name: Build the Docker image of HOSTING --> STORAGE --> Nextcloud --> mysql
12        run: |
13          docker build --file HOSTING/STORAGE/Nextcloud/mysql/dockerfile --tag mysql .
14          docker tag mysql:latest ${{ secrets.USER_DOCKER }}/mysql-img:${{date +%Y-%B}}
15      - name: Build the Docker image of HOSTING --> STORAGE --> Nextcloud
16        run: |
17          docker build --file HOSTING/STORAGE/Nextcloud/dockerfile --tag nextcloud .
18          docker tag nextcloud:latest ${{ secrets.USER_DOCKER }}/nextcloud-img:${{date +%Y-%B}}
19      - name: Docker login
20        run: docker login -username='${{ secrets.USER_DOCKER }}' --password=${{ secrets.PASS_DOCKER }}
21      - name: Docker push image of HOSTING --> ACCESS --> Nextcloud --> mysql
22        run: docker push ${{ secrets.USER_DOCKER }}/mysql-img:${{date +%Y-%B}}
23      - name: Docker push image of HOSTING --> ACCESS --> Nextcloud
24        run: docker push ${{ secrets.USER_DOCKER }}/nextcloud-img:${{date +%Y-%B}}
```

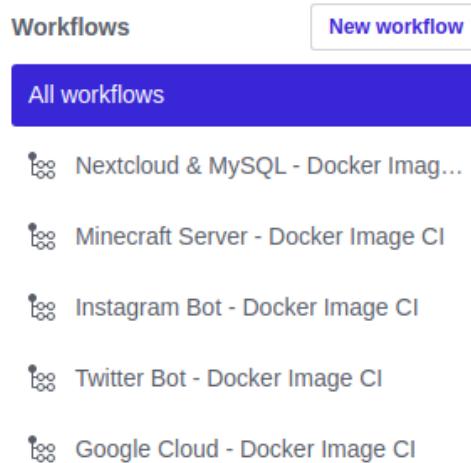
Crea cada inicio de mes tanto la imagen de la base de datos MySQL como del sistema de almacenamiento Nextcloud desde cada dockerfile y la sube a cada repositorio en Docker Hub con mis credenciales encriptadas.

- name: Es el nombre del *workflow*. No influye en el resultado pero es útil para ordenarlos.
- on: Indico a continuación como va a iniciar y activar el *workflow*. A continuación se puede indicar *push* o *schedule*. Cuando se indica *push* se va a activar cuando se produzcan cambios en cualquier rama del repositorio y filtrar por *tags*. Cuando se indica *schedule* utiliza la sintaxis cron POSIX para especificar un rango de tiempo.
 - schedule: Utilizo la sintaxis cron POSIX para especificar un rango de tiempo.
 - - cron: "0 0 1 * *": Esta expresión es para que se ejecute a las 00:00 del primer día de cada mes. Todos los meses de todos los años.
- jobs: Indico a continuación las características del servidor que va a crear la imagen y cada una de las fases de la construcción.
 - build: Permite hacer varias imágenes distintas en un mismo *workflow*.
 - runs-on: El sistema operativo del servidor que va a crear la imagen. En este caso he elegido la última versión de Ubuntu ya que está orientada a servidores Linux.
 - steps: Los pasos de cada una de las fases de la construcción:
 - - uses: actions/checkout@v2: GitHub actions permite utilizar plugins. Este es una acciones estándar para realizar una copia del código del repositorio. Ahora la máquina con Ubuntu tiene una copia completa del repositorio y puedo interactuar con ella.

- name: Es el nombre del paso. Se muestra luego en el workflow y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker build --file HOSTING/STORAGE/Nextcloud/mysql/dockerfile --tag mysql . : Con este comando propio de docker creo la imagen del contenedor basada en el dockerfile del sistema gestor de base de datos MySQL.
 - docker tag mysql:latest \${{ secrets.USER_DOCKER }}/mysql-img:\$(date +%Y-%B): Con este comando etiqueto la imagen anteriormente creada y le pongo el nombre de mi usuario de Docker Hub seguido del nombre de la imagen del repositorio de Docker Hub. Por último le añado la versión que para identificarla va a utilizar el comando de Linux date y va a ser el año de creación seguido del mes.
- name: Es el nombre del paso. Se muestra luego en el workflow y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker build --file HOSTING/STORAGE/Nextcloud/dockerfile --tag nextcloud . : Con este comando propio de docker creo la imagen del contenedor basada en el dockerfile del sistema gestor de archivos Nextcloud.
 - docker tag nextcloud:latest \${{ secrets.USER_DOCKER }}/nextcloud-img:\$(date +%Y-%B): Con este comando etiqueto la imagen anteriormente creada y le pongo el nombre de mi usuario de Docker Hub seguido del nombre de la imagen del repositorio de Docker Hub. Por último le añado la versión que para identificarla va a utilizar el comando de Linux date y va a ser el año de creación seguido del mes.
- name: Es el nombre del paso. Se muestra luego en el workflow y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker login --username='\${{ secrets.USER_DOCKER }}' --password=\${{ secrets.PASS_DOCKER }}: Inicia sesión en Docker Hub con mis credenciales cifradas.
- name: Es el nombre del paso. Se muestra luego en el workflow y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker push \${{ secrets.USER_DOCKER }}/mysql-img:\$(date +%Y-%B): Sube la imagen con la etiqueta y la versión al repositorio de Docker Hub con mis credenciales cifradas.
- name: Es el nombre del paso. Se muestra luego en el workflow y permite identificar donde están en un futuro los errores a la hora de la creación.
 - run: Permite realizar comandos dentro la máquina.
 - docker push \${{ secrets.USER_DOCKER }}/nextcloud-img:\$(date +%Y-%B): Sube la imagen con la etiqueta y la versión al repositorio de Docker Hub con mis credenciales cifradas.

4.6.7 RESUMEN GITHUB ACTIONS

Ahora en la pestaña de Actions puedo ver todos los *workflows* creados.



Y en la parte derecha puedo ver el resultado de cada uno. Si todo ha salido correcto podré ver un tick verde y en caso de que haya habido algún error se marcará con una equis roja y me mandará un email a mi correo electrónico avisándome.

The image contains two separate log entries from GitHub Actions:

- Update dockerImageUploadMinecraft.yml**: Status is green (✓), branch is master, ran last month, took 53s. It was triggered by a commit from the Instagram Bot - Docker Image CI repository.
- Update dockerImageUploadGoogleCloud.yml**: Status is red (✗), branch is master, ran 10 days ago, took 45s. It was triggered by a commit from the Google Cloud - Docker Image CI repository.

En ambos casos puedo pulsar sobre ellos y ver los registros del servidor y también puedo identificar el paso en caso de que haya habido un fallo.

This screenshot shows the detailed log for the failed 'Update dockerImageUploadGoogleCloud.yml' action. The log title is 'Google Cloud - Docker Image CI / build' and it failed 10 days ago in 31s. The log details the steps taken:

- Set up job (green checkmark)
- Run actions/checkout@v2 (green checkmark)
- Build the Docker Image of HOSTING --> ACCESS --> GCloud (red X)

The log output shows the command history and the error message:

```
508 done.
509 Removing intermediate container c063b8d17999
510 => e7da2eb08810
511 Step 3/16 : RUN export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)" && echo "deb http://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && apt-get
512     >> Running in 8b819495e71
513 deb http://packages.cloud.google.com/apt cloud-sdk-eoan main
514 % Total    % Received % Xferd  Average Speed   Time   Time  Current
515          Dload  Upload Total Spent   Left  Speed
516
517      0    0    0    0    0    0    0    0 --::---- ::----::-- ::----::--  0
518 100  653 100  653    0    0 36277    0 --::---- ::----::-- ::----::-- 36277
519 Warning: apt-key output should not be parsed (stdout is not a terminal)
520 OK
521 Ign:1 http://packages.cloud.google.com/apt cloud-sdk-eoan InRelease
522 Err:2 http://packages.cloud.google.com/apt cloud-sdk-eoan Release
523     404 Not Found [IP: 172.217.13.238 80]
524 Hit:3 http://security.ubuntu.com/ubuntu eoan-security InRelease
525 Hit:4 http://archive.ubuntu.com/ubuntu eoan InRelease
526 Hit:5 http://archive.ubuntu.com/ubuntu eoan-updates InRelease
527 Hit:6 http://archive.ubuntu.com/ubuntu eoan-backports InRelease
528 Reading package lists...
529 E: The repository 'http://packages.cloud.google.com/apt cloud-sdk-eoan Release' does not have a Release file.
530 The command '/bin/sh -c export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)" && echo "deb http://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && apt-get
531     >> returned a non-zero code: 100
532 #[error]Process completed with exit code 100.
```

At the bottom, there is a link to 'Docker login'.

4.7 LEVANTANDO EN KUBERNETES

Kubernetes proporciona muchos controles que pueden simplificar la creación, lanzamiento y mantenimiento de una aplicación. Ya explicados los objetos básicos de Kuberentes hay que realizar la arquitectura y posteriormente levantarla e implementarla.

El proyecto va a tener un Ingress con el software Traefik. Traefik es el proxy inverso de código abierto y un loadbalancer para aplicaciones basadas en HTTP y TCP. Este Ingress va a ser el encargado de redirigir el tráfico y ahorrar IP Públicas, ya que solo necesita una, en vez de tener una por cada servicio.

Luego cada servicio va a tener una capa de abstracción de la red dentro del cluster llamada clusterIP. Esta capa va a permitir de forma dinámica interactuar con cada objeto según las etiquetas que tenga adjuntadas. Luego el Ingress se va a conectar a esta capa y va a permitir redirigir el tráfico hacia el objeto.

Cada objeto va a utilizar su imagen previamente creada.

Los datos de configuración y acceso van a estar almacenados en Secretos dentro del propio cluster de Kubernetes. Los secretos son objetos seguros que almacenan datos sensibles, como contraseñas, tokens de OAuth y llaves SSH. Almacenar datos sensibles en secretos es más seguro que en el texto simple ya que puedes encriptarlos.

Cada servicio va a estar encapsulado en un contenedor dentro de un Pod individual, que a su vez va a estar gestionado por un ReplicaSet, que a su vez va a estar controlado por un Deployment.

Aquellos que necesiten guardar datos dentro del contenedor tendrán un volumen persistente con disco persistente adjuntado a cada contenedor individual.

4.7.1

KUBERNETES

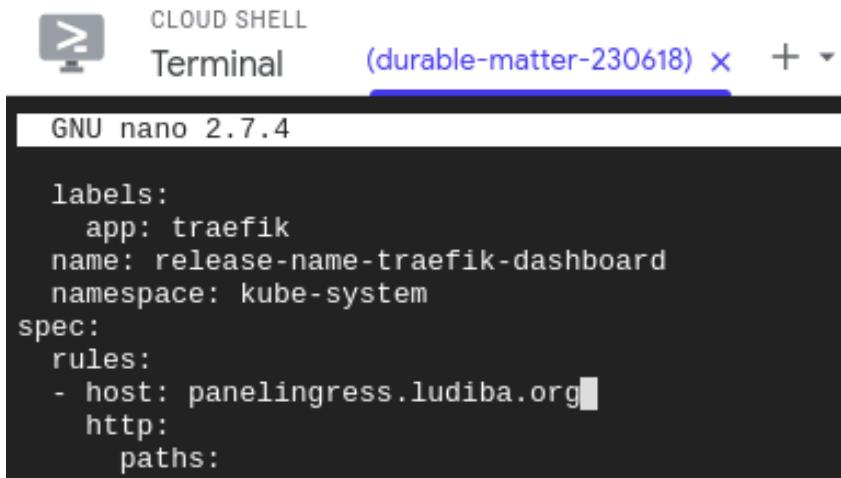
INGRESS TRAEFIK

Como mencione anteriormente voy a utilizar el software Traefik como Ingress dentro del cluster de kubernetes. Para ello accedo al cluster desde la consola de Google Cloud y ejecuto lo siguiente.

Primero clono el repositorio con los archivos de Traefik:

```
git clone https://github.com/RedxLus/traefik-simple-kubernetes.git
```

Cambio mi dominio en el archivo de configuración `traefik-ingress-dashboard.yaml` utilizando el editor nano.



The screenshot shows a Google Cloud Shell interface. At the top, it says "CLOUD SHELL" and "Terminal (durable-matter-230618)". Below that is a terminal window titled "GNU nano 2.7.4". The content of the file is as follows:

```
labels:
  app: traefik
  name: release-name-traefik-dashboard
  namespace: kube-system
spec:
  rules:
    - host: panelingress.ludiba.org
      http:
        paths:
```

Una vez cambiado por mi dominio y subdominio, ya puedo aplicar los cambios y ejecutar todos los archivos de configuración para tener Traefik funcionando en el cluster. Ejecuto el comando:

```
kubectl apply -f traefik-simple-kubernetes/V1.7/
```

Por lo tanto se crearan todos los recursos.

```
ludibagroup@cloudshell:~ (durable-matter-230618)$ kubectl apply -f traefik-simple-kubernetes/V1.7/
deployment.extensions/release-name-traefik created
ingress.extensions/release-name-traefik-dashboard created
clusterrole.rbac.authorization.k8s.io/release-name-traefik created
clusterrolebinding.rbac.authorization.k8s.io/release-name-traefik created
serviceaccount/release-name-traefik created
secret/release-name-traefik-default-cert created
service/release-name-traefik-dashboard created
service/release-name-traefik created
configmap/release-name-traefik created
```

KUBERNETES

GESTOR GCLOUD

El código está en el repositorio de Github. Es el siguiente código:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: gcloud
  labels:
    app: gcloud
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: gcloud
        tipo: deploy
        version: "0.0.0"
    spec:
      containers:
        - name: gcloud
          imagePullPolicy: Always
          image: redxius/gcloud-sdk-google-cloud:latest-with-kubectl-and-ssh
          ports:
            - containerPort: 22
              name: ssh
...
kind: Service
apiVersion: v1
metadata:
  name: loadbalancer-gcloud
spec:
  type: LoadBalancer
  selector:
    app: gcloud
    tipo: deploy
    version: "0.0.0"
  ports:
    - protocol: TCP
      port: 22
      targetPort: 22
      name: ssh
```

En el tercer apartado creo la aplicación (Deployment).

- apiVersion: extensions/v1beta1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión avanzada para aplicaciones 1.
- kind: Deployment - Declaro el tipo de objeto. En este caso es una aplicación (Deployment).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: gcloud - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - app: gcloud
- spec: - Son las características técnicas del objeto, en este caso a aplicación (Deployment).
 - replicas: 1: - Indica el número de instancias del objeto. Se pueden indicar varias entonces sería un servidor a prueba de tolerancia.
 - template: - A continuación indica las características de los Pods que va a controlar el Deployment.
 - metadata: - Son anotaciones, etiquetas y el nombre del objeto, en este caso cada Pod dentro del Deployment.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - spec: - Son las características técnicas del objeto, en este caso cada Pod dentro del Deployment.
 - containers: - A continuación detallo la información relativa a cada contenedor individual dentro de cada Pod y a su vez dentro de cada Deployment. Tiene formato Array así que se pueden tener varios contenedores por Pod.
 - - name: gcloud - Es el nombre que va a tener dentro del Pod.
 - image: redxlus/gcloud-sdk-google-cloud:latest-with-kubectl-and-ssh - Es la imagen de Docker que va a utilizar el contenedor. Es la imagen creada automáticamente en GitHub con las herramientas de gestión.
 - imagePullPolicy: Always - Indico que cada vez que se creé el contenedor vuelva a actualizar la imagen para tener la última versión.
 - ports: - Son los puertos expuestos dentro del clúster de Kubernetes para que luego los servicios puedan ser accesibles. Tiene formato Array así que se puede exponer varios puertos.
 - - containerPort: 22 - Es el puerto expuesto del contenedor hacia el clúster de Kubernetes.
 - name: ssh - Es un nombre único e irrepetible para identificar el puerto del contenedor.

4.7.3

KUBERNETES SERVIDOR MINECRAFT

El código está en el repositorio de Github. Es el siguiente código, cada objeto esta separado por tres símbolos menos seguidos (---).

```
---
apiVersion: v1
kind: Service
metadata:
  name: minecraft-service
spec:
  ports:
    - name: minecraft
      targetPort: 25565
      port: 25565
  selector:
    app: minecraft
    tipo: deploy
    version: "0.0.0"

---
apiVersion: v1
kind: Secret
metadata:
  name: serverpp
type: Opaque
data:
  server.properties: I01pbmVjcmFmdCBzZXJ2ZXIgcHJvcGVydGlcwojU2F0IFNlcCAxNCax0ToxNzo0NiBVVEMgMjAx0QpzcGF3bi1wcm90ZWNoaW9uPTE2CmlheC10aWNrLXRpbWU9NjAwMDAKcXvLcn

---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: minecraft
  labels:
    app: minecraft
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: minecraft
        tipo: deploy
        version: "0.0.0"
    spec:
      containers:
        - name: spigot
          image: redxlus/minecraft-server
          ports:
            - containerPort: 25565
              name: mine
              volumeMounts:
                - name: properties
                  mountPath: /server.properties
                  subPath: server.properties
                - name: world
                  mountPath: /world
          volumes:
            - name: properties
              secret:
                secretName: serverpp
            - name: world
              persistentVolumeClaim:
                claimName: world-minecraft

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: world-minecraft
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi

---
kind: Service
apiVersion: v1
metadata:
  name: loadbalancer-minecraft
spec:
  type: LoadBalancer
  selector:
    app: minecraft
    tipo: deploy
    version: "0.0.0"
  ports:
    - protocol: TCP
      port: 25565
      targetPort: 25565
      name: minecraft
```

En el primer apartado creo el servicio (Service).

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: Service - Declaro el tipo de objeto. En este caso es un servicio (Service).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: minecraft-service - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- spec: - Son las características técnicas del objeto.
 - ports: - Al ser un servicio tengo que especificar los puertos a utilizar. Indica a que puertos va a reenviar las solicitudes, tanto dentro de los Pods como externamente. Puedo especificar varios ya que se utiliza el método de distinción de un Array.
 - name: - Es el nombre del puerto. Debe ser único e irrepetible.
 - port: 25565 - Es el puerto expuesto dentro del clúster. El servicio es visible en este puerto y enviará las solicitudes realizadas a los pods. En este caso es el por defecto de los servidores de Minecraft.
 - targetPort: 25565 - Es el puerto expuesto dentro de los Pods. Es el puerto en el que al pod se le envía la solicitud. En este caso es el por defecto de los servidores de Minecraft.
 - selector: - A continuación enumero las etiquetas que debe tener el un Pod para poder utilizar este servicio. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.

En el segundo apartado creo el secreto (Secret).

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: Secret - Declaro el tipo de objeto. En este caso es un secreto (Secret).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: serverpp - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible. El nombre en este caso significa *server properties* y es un archivo de configuración del servidor de Minecraft.
- type: Opaque - Dentro de los secretos en Kubernetes puedo especificar varios tipos diferentes según el grado de cifrado. Si no tiene cifrado entonces es un configmap y tampoco tiene tipo.
- data: - A continuación indico la información con formato base64 con estilo llave valor.

En el tercer apartado creo la aplicación (Deployment).

- apiVersion: extensions/v1beta1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión avanzada para aplicaciones 1.
- kind: Deployment - Declaro el tipo de objeto. En este caso es una aplicación (Deployment).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: minecraft - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - app: minecraft
- spec: - Son las características técnicas del objeto, en este caso a aplicación (Deployment).
 - replicas: 1: - Indica el número de instancias del objeto. Se pueden indicar varias entonces sería un servidor a prueba de tolerancia.
 - template: - A continuación indica las características de los Pods que va a controlar el Deployment.
 - metadata: - Son anotaciones, etiquetas y el nombre del objeto, en este caso cada Pod dentro del Deployment.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - spec: - Son las características técnicas del objeto, en este caso cada Pod dentro del Deployment.
 - containers: - A continuación detallo la información relativa a cada contenedor individual dentro de cada Pod y a su vez dentro de cada Deployment. Tiene formato Array así que se pueden tener varios contenedores por Pod.
 - - name: spigot - Es el nombre que va a tener dentro del Pod.
 - image: redxlus/minecraft-server - Es la imagen de Docker que va a utilizar el contenedor. Es la imagen creada automáticamente en GitHub del servidor de minecraft.
 - ports: - Son los puertos expuestos dentro del clúster de Kubernetes para que luego los servicios puedan ser accesibles. Tiene formato Array así que se puede exponer varios puertos.
 - - containerPort: 25565 - Es el puerto expuesto del contenedor hacia el clúster de Kubernetes.
 - - name: mine - Es un nombre único e irrepetible para identificar el puerto del contenedor.
 - volumeMounts: - Los contenedores por defecto no tienen almacenamiento ya que son volátiles, se crean y se destruyen de forma rápida. Para solucionar esto se pueden adjuntar volúmenes en partes concretas del contenedor. Tiene formato Array así que se pueden añadir varios volúmenes.

- - name: properties - Es un nombre único e irrepetible para identificar el volumen del contenedor.
- mountPath: /server.properties - Indica en que directorio dentro del contenedor van a almacenarse los datos en el volumen.
- subPath: server.properties - Indica el nombre específico del archivo que va a ser almacenado dentro del volumen. El cual va a venir del secreto anteriormente creado.
- volumes: - Indica la procedencia de los volúmenes utilizados dentro de los contenedores. Tiene formato Array así que se pueden añadir varios volúmenes de distintos tipos y procedencias.
 - - name: properties - Es un nombre único e irrepetible para identificar el volumen del contenedor. Es obligatorio que coincida con el nombre del volumeMounts a utilizar.
 - secret: - Indica la procedencia del volumen. Va a proceder del secreto anteriormente creado.
 - secretName: serverpp - Es el nombre del secreto anteriormente creado.
 - - name: world - Es un nombre único e irrepetible para identificar el volumen del contenedor. Es obligatorio que coincida con el nombre del volumeMounts a utilizar.
 - persistentVolumeClaim: Indica que voy a utilizar un proveedor de servicios, en este caso el mismo Google Cloud, para tener almacenamiento persistente en los datos.
 - claimName: world-minecraft - Indica el nombre del volumen de almacenamiento persistente. Es obligatorio que coincida con el nombre del PersistentVolumeClaim que a continuación voy a crear.

En el cuarto apartado creo *PersistentVolumeClaim*.

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: PersistentVolumeClaim - Declaro el tipo de objeto. En este caso es un *PersistentVolumeClaim*. Es una abstracción de solicitud de almacenamiento persistente que posteriormente es satisfecho por el Cloud con un Volumen.
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: world-minecraft - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- spec: - Son las características técnicas del objeto, en este caso de cada volumen dado por el *PersistentVolumeClaim*
 - accessModes: - Indica el tipo de acceso e interacción que pueden tener los Pods con el Volumen.
 - - ReadWriteOnce - Indica que el volumen puede ser montado como lectura-escritura por un solo objeto al mismo tiempo. Así se previene la escritura múltiple.
 - resources: - Indica la cantidad de almacenamiento del volumen.
 - requests: - Realiza la petición para satisfacer la cantidad de almacenamiento en el volumen.
 - storage: 2Gi - Especifica exactamente cuánto almacenamiento necesita el volumen para ser utilizado por un contenedor.

En el quinto apartado creo Balanceador de Red (Load Balancer).

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: Service - Declaro el tipo de objeto. En este caso es un Service ya que es una abstracción de la red.
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: loadbalancer-minecraft - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- spec: - Son las características técnicas del objeto.
 - type: LoadBalancer - Indica que específicamente es un servicio de balanceador de carga.
 - selector: - A continuación enumero las etiquetas que debe tener el un Pod para poder utilizar este servicio. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
- ports: - Al ser un servicio tengo que especificar los puertos a utilizar. Indica a que puertos va a reenviar las solicitudes, tanto dentro de los Pods como externamente. Puedo especificar varios ya que se utiliza el método de distinción de un Array.
 - - protocol: TCP - Indico el tipo de protocolo a utilizar en ese puerto. Puede ser TCP o UDP.
 - port: 25565 - Es el puerto expuesto al exterior de Internet. Por lo tanto el servidor va a ser accessible desde la IP:puerto
 - targetPort: 25565 - Es el puerto expuesto dentro de los Pods. Es el puerto en el que al pod se le envía la solicitud.
 - name: - Es el nombre del puerto. Debe ser único e irrepetible.

4.7.4 KUBERNETES BOT

INSTAGRAM

El código está en el repositorio de Github. Es el siguiente código, cada objeto esta separado por tres símbolos menos seguidos (- - -).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ig-bot
  labels:
    app: ig-bot
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: ig-bot
        tipo: deploy
        version: "0.0.0"
    spec:
      containers:
        - name: ig-bot
          image: redxlus/instagram-bot
---
```

En el tercer apartado creo la aplicación (Deployment).

- apiVersion: extensions/v1beta1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión avanzada para aplicaciones 1.
- kind: Deployment - Declaro el tipo de objeto. En este caso es una aplicación (Deployment).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: ig-bot - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - app: ig-bot
- spec: - Son las características técnicas del objeto, en este caso a aplicación (Deployment).
 - replicas: 1: - Indica el número de instancias del objeto. Se pueden indicar varias entonces sería un servidor a prueba de tolerancia.
 - template: - A continuación indica las características de los Pods que va a controlar el Deployment.
 - metadata: - Son anotaciones, etiquetas y el nombre del objeto, en este caso cada Pod dentro del Deployment.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - spec: - Son las características técnicas del objeto, en este caso cada Pod dentro del Deployment.
 - containers: - A continuación detallo la información relativa a cada contenedor individual dentro de cada Pod y a su vez dentro de cada Deployment. Tiene formato Array así que se pueden tener varios contenedores por Pod.
 - - name: ig-bot - Es el nombre que va a tener dentro del Pod.
 - image: redxlus/instagram-bot - Es la imagen de Docker que va a utilizar el contenedor. Es la imagen creada automáticamente en GitHub con las herramientas de gestión.

4.7.5 KUBERNETES BOT TWITTER

El código está en el repositorio de Github. Es el siguiente código, cada objeto esta separado por tres símbolos (---).

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: twitter-bot
  labels:
    app: tw
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: tw
        tipo: deploy
        version: "0.0.0"
    spec:
      containers:
        - name: twitter-bot
          image: redxlus/twitter-bot
          volumeMounts:
            - name: configuration
              mountPath: /Twitter-Follow-and-Unfollow-Bot/config.json
              subPath: config.json
      volumes:
        - name: configuration
          secret:
            secretName: tw-config
...
apiVersion: v1
kind: Secret
metadata:
  name: tw-config
type: Opaque
data:
  config.json: ewogICAgImFldGgiOiB7CiAgICAgICAgInNjcmVlbl9uYw1lIjogIlIpVVigU0NSRUVOIE5BTUUgd2l0aG91dCB0aGUgQCBzaWduIiwKICAgICAgICAiQ090U1VRVJfS0VZIjogIlIpVVig...
...
```

En el tercer apartado creo la aplicación (Deployment).

- apiVersion: extensions/v1beta1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión avanzada para aplicaciones 1.
- kind: Deployment - Declaro el tipo de objeto. En este caso es una aplicación (Deployment).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: twitter-bot - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - app: tw
- spec: - Son las características técnicas del objeto, en este caso a aplicación (Deployment).
 - replicas: 1: - Indica el número de instancias del objeto. Se pueden indicar varias entonces sería un servidor a prueba de tolerancia.
 - template: - A continuación indica las características de los Pods que va a controlar el Deployment.
 - metadata: - Son anotaciones, etiquetas y el nombre del objeto, en este caso cada Pod dentro del Deployment.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - spec: - Son las características técnicas del objeto, en este caso cada Pod dentro del Deployment.
 - containers: - A continuación detallo la información relativa a cada contenedor individual dentro de cada Pod y a su vez dentro de cada Deployment. Tiene formato Array así que se pueden tener varios contenedores por Pod.
 - - name: twitter-bot - Es el nombre que va a tener dentro del Pod.
 - image: redxlus/twitter-bot - Es la imagen de Docker que va a utilizar el contenedor. Es la imagen creada automáticamente en GitHub con las herramientas de gestión.
 - volumeMounts: - Los contenedores por defecto no tienen almacenamiento ya que son volátiles, se crean y se destruyen de forma rápida. Para solucionar esto se pueden adjuntar volúmenes en partes concretas del contenedor. Tiene formato Array así que se pueden añadir varios volúmenes.
 - - name: configuration - Es un nombre único e irrepetible para identificar el volumen del contenedor.
 - mountPath: /Twitter-Follow-and-Unfollow-Bot/config.json - Indica en que directorio dentro del contenedor van a almacenarse los datos en el volumen.
 - subPath: config.json - Indica el nombre específico del archivo que va a ser almacenado dentro del volumen. El cual va a venir del secreto creado a continuación.

- volumes: - Indica la procedencia de los volúmenes utilizados dentro de los contenedores. Tiene formato Array así que se pueden añadir varios volúmenes de distintos tipos y procedencias.
 - - name: configuration - Es un nombre único e irrepetible para identificar el volumen del contenedor. Es obligatorio que coincida con el nombre del volumeMounts a utilizar.
 - secret: - Indica la procedencia del volumen. Va a proceder del secreto anteriormente creado.
 - secretName: tw-config - Es el nombre del secreto anteriormente a continuación.

En el segundo apartado creo el secreto (Secret).

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: Secret - Declaro el tipo de objeto. En este caso es un secreto (Secret).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: tw-config - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible. En este caso es un archivo de configuración del bot de Twitter.
- type: Opaque - Dentro de los secretos en Kubernetes puedo especificar varios tipos diferentes según el grado de cifrado. Si no tiene cifrado entonces es un configmap y tampoco tiene tipo.
- data: - A continuación indico la información con formato base64 con estilo llave valor.

4.7.6 KUBERNETES

NEXCLOUD Y MYSQL

El código está en el repositorio de Github. Es el siguiente código, cada objeto esta separado por tres símbolos menos seguidos (---).

```
---  
apiVersion: v1  
kind: Secret  
metadata:  
  name: config-nextcloud  
type: Opaque  
data:  
  # User admin nextcloud  
  nextcloud-username: YWRtaW5pc3RyYWRvcg==  
  nextcloud-password: czswTUMxfWN8Vm00QHJBZ1JGczg=  
  # Database (MYSQL)  
  mysql-database: Ym0tbmV4dGNsB3Vk  
  mysql-user: dXNlYXJpbyluZkh0Y2xvdWQ=  
  mysql-password: N3FZRndEYitwdENbYnFuQkEwW10=  
  mysql-host: bxLzcWwtc2VydmljZTozMzA2  
  # Trust domain in Nextcloud  
  nextcloud-domain: bmV4dC5dwRpYmEub3Jn  
  # Directory to store the data  
  nextcloud-dir-data: L3Zhci93d3cvaHRtbC9kYXRh  
  
apiVersion: v1  
kind: Secret  
metadata:  
  name: config-mysql  
type: Opaque  
data:  
  mysql-root-password: dFtXPWE7RzF4cHJ0eElvUDdZdis=  
  mysql-name-database: Ym0tbmV4dGNsB3Vk  
  mysql-user: dXNlYXJpbyluZkh0Y2xvdWQ=  
  mysql-user-password: N3FZRndEYitwdENbYnFuQkEwW10=  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: nextcloud  
  labels:  
    app: nextcloud  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: nextcloud  
  template:  
    metadata:  
      labels:  
        app: nextcloud  
    spec:  
      containers:  
        - name: nextcloud  
          image: redxlus/nextcloud-img  
          ports:  
            - name: https  
              containerPort: 443  
              protocol: TCP  
            - name: http  
              containerPort: 80  
              protocol: TCP  
          env:  
            - name: NEXTCLOUD_ADMIN_USER  
              valueFrom:  
                secretKeyRef:  
                  name: config-nextcloud  
                  key: nextcloud-username
```

```

- name: NEXTCLOUD_ADMIN_PASSWORD
  valueFrom:
    secretKeyRef:
      name: config-nextcloud
      key: nextcloud-password
- name: MYSQL_DATABASE
  valueFrom:
    secretKeyRef:
      name: config-nextcloud
      key: mysql-database
- name: MYSQL_USER
  valueFrom:
    secretKeyRef:
      name: config-nextcloud
      key: mysql-user
- name: MYSQL_PASSWORD
  valueFrom:
    secretKeyRef:
      name: config-nextcloud
      key: mysql-password
- name: MYSQL_HOST
  valueFrom:
    secretKeyRef:
      name: config-nextcloud
      key: mysql-host
- name: NEXTCLOUD_TRUSTED_DOMAINS
  valueFrom:
    secretKeyRef:
      name: config-nextcloud
      key: nextcloud-domain
- name: NEXTCLOUD_DATA_DIR
  valueFrom:
    secretKeyRef:
      name: config-nextcloud
      key: nextcloud-dir-data
volumeMounts:
- name: pvc
  mountPath: /var/www/html
  subPath: html
volumes:
- name: pvc
  persistentVolumeClaim:
    claimName: pvc-nextcloud
# Will mount configuration files as www-data (id: 33) for nextcloud
securityContext:
  fsGroup: 33
initContainers:
- name: init-check
  image: busybox
  command: ["sh", "-c", "echo 'until nc mysql-service 3306' > script.sh && \
            echo 'do' >> script.sh && \
            echo 'sleep 3' >> script.sh && \
            echo 'done' >> script.sh && \
            sh script.sh"]
...
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: redxlus/mysql-img
          ports:
            - containerPort: 3306
          volumeMounts:
            - name: pvc
              mountPath: /var/lib/mysql
              subPath: mysql
          volumes:
            - name: pvc
              persistentVolumeClaim:
                claimName: pvc-mysql
...
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  ports:
    - name: mysql
      targetPort: 3306
      port: 3306
  selector:
    app: mysql

```

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: nextcloud-service  
spec:  
  ports:  
    - name: https  
      targetPort: 443  
      port: 443  
    - name: http  
      targetPort: 80  
      port: 80  
      selector:  
        app: nextcloud  
---  
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: nextcloud-ingress  
  annotations:  
    kubernetes.io/ingress.class: traefik  
spec:  
  rules:  
    - host: next.ludiba.org  
      http:  
        paths:  
          - path: /  
            backend:  
              serviceName: nextcloud-service  
              servicePort: http  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: pvc-nextcloud  
spec:  
  accessModes:  
    - ReadWriteOnce  
  resources:  
  requests:  
    storage: 100Gi  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: pvc-mysql  
spec:  
  accessModes:  
    - ReadWriteOnce  
  resources:  
  requests:  
    storage: 50Gi
```

En el primer y segundo apartados creo los secreto (Secret) para configurar ambos servicios.

Primero Nextcloud:

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: Secret - Declaro el tipo de objeto. En este caso es un secreto (Secret).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: config-nextcloud - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- type: Opaque - Dentro de los secretos en Kubernetes puedo especificar varios tipos diferentes según el grado de cifrado. Si no tiene cifrado entonces es un configmap y tampoco tiene tipo.
- data: - A continuación indico la información con formato base64 con estilo llave valor. Declaro la contraseña de administrador, el nombre de usuario del administrador. Luego la base de datos a utilizar, el nombre de la base de datos, el usuario y su contraseña ademas de la ruta donde está localizada la base de datos. A continuación declaro el nombre del dominio donde va a estar Nextcloud y por último el directorio donde se van a almacenar los datos.

Primero Nextcloud:

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: Secret - Declaro el tipo de objeto. En este caso es un secreto (Secret).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: config-mysql - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- type: Opaque - Dentro de los secretos en Kubernetes puedo especificar varios tipos diferentes según el grado de cifrado. Si no tiene cifrado entonces es un configmap y tampoco tiene tipo.
- data: - A continuación indico la información con formato base64 con estilo llave valor. Declaro la contraseña del usuario root, el nombre de la base de datos, el nombre del usuario nuevo y su contraseña.

En el tercer apartado creo la aplicación (Deployment) del servicio de Nextcloud.

- apiVersion: apps/v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión avanzada para aplicaciones 1.
- kind: Deployment - Declaro el tipo de objeto. En este caso es una aplicación (Deployment).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: nextcloud - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - app: nextcloud
- spec: - Son las características técnicas del objeto, en este caso a aplicación (Deployment).
 - replicas: 1: - Indica el número de instancias del objeto. Se pueden indicar varias entonces sería un servidor a prueba de tolerancia.
 - strategy: - Al levantar una aplicación compleja se pueden elegir entre varios tipos de estrategias para levantarla.
 - type: Recreate - Terminar la versión anterior y lanzar la nueva. Cada vez que se actualice primero termina con todos los contenedores y luego crea con la nueva imagen todos al mismo tiempo.
 - template: - A continuación indica las características de los Pods que va a controlar el Deployment.
 - metadata: - Son anotaciones, etiquetas y el nombre del objeto, en este caso cada Pod dentro del Deployment.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - spec: - Son las características técnicas del objeto, en este caso cada Pod dentro del Deployment.
 - containers: - A continuación detallo la información relativa a cada contenedor individual dentro de cada Pod y a su vez dentro de cada Deployment. Tiene formato Array así que se pueden tener varios contenedores por Pod.
 - - name: nextcloud - Es el nombre que va a tener dentro del Pod.
 - image: redxlus/nextcloud-img - Es la imagen de Docker que va a utilizar el contenedor. Utilizo la imagen creada anteriormente.
 - ports: - Al ser un servicio tengo que especificar los puertos a utilizar. Indica a qué puertos va a reenviar las solicitudes, tanto dentro de los Pods como externamente. Puedo especificar varios ya que se utiliza el método de distinción de un Array.
 - name: https - Es el nombre del puerto. Debe ser único e irrepetible.
 - protocol: TCP - Indico el protocolo a utilizar en dicho puerto.
 - containerPort: 443 - Es el puerto expuesto del contenedor hacia el clúster de Kubernetes.

- name: http - Es el nombre del puerto. Debe ser único e irrepetible.
- protocol: TCP - Indico el protocolo a utilizar en dicho puerto.
- containerPort: 80 - Es el puerto expuesto del contenedor hacia el clúster de Kubernetes.
- env: - Declaro las variables del sistema disponibles dentro del contenedor. En este caso provienen del Secret anteriormente creado por lo tanto declaro como nombre (name) el nombre del secreto y como llave (key) el nombre del dato a utilizar.
- volumeMounts: - Los contenedores por defecto no tienen almacenamiento ya que son volátiles, se crean y se destruyen de forma rápida. Para solucionar esto se pueden adjuntar volúmenes en partes concretas del contenedor. Tiene formato Array así que se pueden añadir varios volúmenes.
 - - name: pvc - Es un nombre único e irrepetible para identificar el volumen del contenedor.
 - mountPath: /var/www/html - Indica en qué directorio dentro del contenedor van a almacenarse los datos en el volumen.
 - subPath: html - Indica el directorio específico que va a ser almacenado dentro del volumen y no afecta al resto de directorios.
- volumes: - Indica la procedencia de los volúmenes utilizados dentro de los contenedores. Tiene formato Array así que se pueden añadir varios volúmenes de distintos tipos y procedencias.
- securityContext: - Define la configuración de privilegios y control de acceso para un Pod o Contenedor. La configuración del contexto de seguridad incluye, pero no se limita.
 - fsGroup: 33 - Asigna un mismo grupo a todos los procesos.
 - initContainers: - Crea un pequeño contenedor inicial que realiza comprobaciones antes de iniciar el contenedor de la aplicación. En este caso utiliza la imagen de busybox, un sistema operativo de tan solo 700 KB. En él voy a correr un script creado en bash para comprobar si el contenedor de MySQL está activo y recibiendo o de lo contrario hay que esperar a que lo esté. Esto es para evitar errores de conexión entre contenedores. El código en bash es el siguiente:
 - until nc mysql-service 3306
 - do
 - sleep 3
 - done

En el cuarto apartado creo la aplicación (Deployment) del servicio de MYSQL.

- apiVersion: apps/v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión avanzada para aplicaciones 1.
- kind: Deployment - Declaro el tipo de objeto. En este caso es una aplicación (Deployment).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: mysql - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - app: mysql
- spec: - Son las características técnicas del objeto, en este caso a aplicación (Deployment).
 - replicas: 1: - Indica el número de instancias del objeto. Se pueden indicar varias entonces sería un servidor a prueba de tolerancia.
 - template: - A continuación indica las características de los Pods que va a controlar el Deployment.
 - metadata: - Son anotaciones, etiquetas y el nombre del objeto, en este caso cada Pod dentro del Deployment.
 - labels: - Etiquetas que sirven para comunicarse entre objetos dentro del clúster de forma ordenada. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.
 - spec: - Son las características técnicas del objeto, en este caso cada Pod dentro del Deployment.
 - containers: - A continuación detallo la información relativa a cada contenedor individual dentro de cada Pod y a su vez dentro de cada Deployment. Tiene formato Array así que se pueden tener varios contenedores por Pod.
 - - name: mysql - Es el nombre que va a tener dentro del Pod.
 - image: redxlus/mysql-img - Es la imagen de Docker que va a utilizar el contenedor. Utilizo la imagen creada anteriormente.
 - ports: - Al ser un servicio tengo que especificar los puertos a utilizar. Indica a que puertos va a reenviar las solicitudes, tanto dentro de los Pods como externamente. Puedo especificar varios ya que se utiliza el método de distinción de un Array.
 - containerPort: 3306 - Es el puerto expuesto del contenedor hacia el clúster de Kubernetes.
 - volumeMounts: - Los contenedores por defecto no tienen almacenamiento ya que son volátiles, se crean y se destruyen de forma rápida. Para solucionar esto se pueden adjuntar volúmenes en partes concretas del contenedor. Tiene formato Array así que se pueden añadir varios volúmenes.
 - volumes: - Indica la procedencia de los volúmenes utilizados dentro de los contenedores. Tiene formato Array así que se pueden añadir varios volúmenes de distintos tipos y procedencias.

En el quinto apartado creo los servicio (Service) tanto de MYSQL como de Nextcloud.

Primero de MySQL:

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: Service - Declaro el tipo de objeto. En este caso es un servicio (Service).
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: mysql-service - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- spec: - Son las características técnicas del objeto.
 - ports: - Al ser un servicio tengo que especificar los puertos a utilizar. Indica a que puertos va a reenviar las solicitudes, tanto dentro de los Pods como externamente. Puedo especificar varios ya que se utiliza el método de distinción de un Array.
 - name: mysql - Es el nombre del puerto. Debe ser único e irrepetible.
 - port: 3306 - Es el puerto expuesto dentro del clúster. El servicio es visible en este puerto y enviará las solicitudes realizadas a los pods. En este caso es el por defecto de los servidores de MYSQL.
 - targetPort: 3306 - Es el puerto expuesto dentro de los Pods. Es el puerto en el que al pod se le envía la solicitud. En este caso es el por defecto de los servidores de MYSQL.
 - selector: - A continuación enumero las etiquetas que debe tener el un Pod para poder utilizar este servicio. Las etiquetas tienen el formato llave valor. Puede ser cualquier palabra llave y cualquier palabra valor.

Seguido de Nextcloud, el cual tiene la misma estructura pero en este caso los puertos a abrir son los de HTTP y HTTPS: 80 y 443

En el sexto apartado creo el Ingres para Nextcloud.

- apiVersion: extensions/v1beta1 - Dentro de kubernetes utilizo su Interfaz de extensiones con la versión beta 1.
- kind: Ingress - Declaro el tipo de objeto. En este caso es un Ingress.
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
- name: nextcloud-ingress - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- annotations: - Declaro el tipo de ingress a utilizar, en este caso declaro el previamente desplegado Traefik.
- spec: - Son las características técnicas del objeto.
 - host: next.ludiba.org - Declaro la URL de la web en la que se va a mostrar el servicio. Posteriormente declaro la ruta, en este caso es el servicio de nextcloud el cual ya tiene expuestos los puertos 443 y 80.

En el séptimo apartado creo los *PersistentVolumeClaim* para Nextcloud y MySQL.

Para Nextcloud:

- apiVersion: v1 - Dentro de kubernetes utilizo su Interfaz de programación de aplicaciones (API) con la versión clásica e inicial 1.
- kind: PersistentVolumeClaim - Declaro el tipo de objeto. En este caso es un *PersistentVolumeClaim*. Es una abstracción de solicitud de almacenamiento persistente que posteriormente es satisfecho por el Cloud con un Volumen.
- metadata: - Son anotaciones, etiquetas y el nombre del objeto.
 - name: pvc-nextcloud - Es el nombre que va a tener dentro del clúster de Kubernetes. Debe ser único e irrepetible.
- spec: - Son las características técnicas del objeto, en este caso de cada volumen dado por el *PersistentVolumeClaim*
 - accessModes: - Indica el tipo de acceso e interacción que pueden tener los Pods con el Volumen.
 - - ReadWriteOnce - Indica que el volumen puede ser montado como lectura-escritura por un solo objeto al mismo tiempo. Así se previene la escritura múltiple.
 - resources: - Indica la cantidad de almacenamiento del volumen.
 - requests: - Realiza la petición para satisfacer la cantidad de almacenamiento en el volumen.
 - storage: 100Gi - Especifica exactamente cuánto almacenamiento necesita el volumen para ser utilizado por un contenedor.

Para la base de datos de MySQL es igual al anterior pero cambiando el nombre por pvc-mysql y el tamaño de almacenamiento por 50Gi.

4.8

COMPRAR DOMINIO

Una vez todos los servicios están listos necesito un dominio para poder mostrarlos y también enlazarlos. Al ser una organización voy a utilizar un dominio ORG. La empresa donde he comprado el dominio es Namecheap ya que tiene una serie de añadidos sin costes muy interesantes. Por ejemplo permite gestión avanzada DNS y tiene de forma gratuita WhoisGuard. WhoisGuard es un servicio de protección de privacidad para cuando realizan una búsqueda Whois en coada dominio y sub dominio. Protege de posibles correos no deseados e incluso robos de identidad.

Para comprar el domino tan solo hay que acceder a la página web: <https://namecheap.com> Y seleccionó el dominio.



Una vez comprado ya estará en el panel de usuario para editar los detalles.



4.8.1 UTILIZAR

CDN Y ANTI-DDOS

Cloudflare tiene una extensa red global de centros de datos que almacenan en caché el contenido estático más cerca de los usuarios y ofrecen contenido dinámico privado más rápido y confiable.

Permite activar HTTP/3. Es la última generación del protocolo de transferencia web. En lugar de utilizar TCP como la capa de transporte, HTTP / 3 utiliza QUIC, un nuevo protocolo de transporte de Internet que está encriptado de manera predeterminada y ayuda a acelerar la entrega del tráfico. Esto permite conexiones más rápidas, más confiables y más seguras.

Para activar esta capa de seguridad y protección lo primero es registrarse en su página web: <https://dash.cloudflare.com/sign-up>

A continuación agregamos el dominio recién comprado. Para ello pulso sobre el botón:

+ Add a Site

En la siguiente página pide introducir el nombre del dominio. En este caso ludiba.org

Accelerate and protect your site with Cloudflare

Enter your site (example.com):

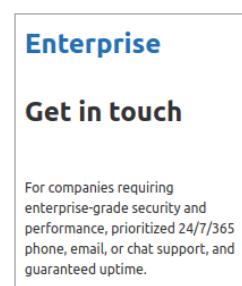
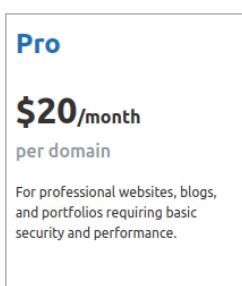
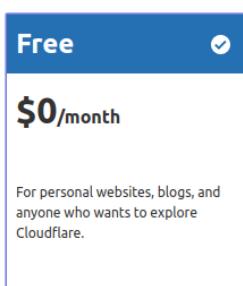
ludiba.org

Add site

Ahora muestra los diferentes tipos de planes y sus precios. Los planes PRO y BUSINESS son para aquellos que tienen gran cantidad de tráfico o quieren atención al cliente personalizada. Ya que no es el caso, puedo elegir el plan FREE.

← Back

Select a plan



[Learn more about our plans](#)

Confirm plan

Por último tengo que cambiar los DNS del dominio por los de Cloudflare. Apunto los DNS de Cloudflare que aparece como siguiente paso pulsando sobre *Click to copy*.

[← Back](#)

Change your nameservers

i Pointing to Cloudflare's nameservers is critical for activating your site successfully. Otherwise, Cloudflare is unable to manage your DNS and optimize your site.

1. Log in to your registrar account

Determine your registrar via [WHOIS](#).

Remove these nameservers:

a.iana-servers.net
b.iana-servers.net

2. Replace with Cloudflare's nameservers:

Nameserver 1

[lilyana.ns.cloudflare.com](#)

[Click to copy](#)

Nameserver 2

[nero.ns.cloudflare.com](#)

[Click to copy](#)

Abro en otra pestaña el panel de administración del dominio de NameCheap y pulso sobre el botón *Manage* al lado de mi dominio. Ahora en la pestaña de Nameservers pulso sobre *Custom DNS*. Y allí ya puedo cambiar los DNS por los de Cloudflare.

NAMESERVERS

Custom DNS

lilyana.ns.cloudflare.com

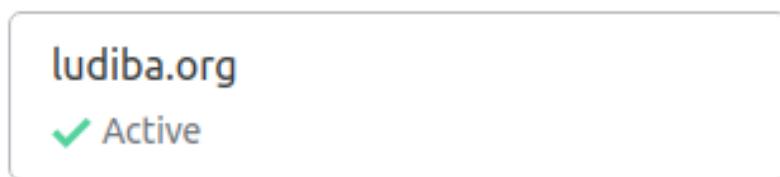
nero.ns.cloudflare.com

[Click to copy](#)

[Click to copy](#)

+ ADD NAMESERVER

Ahora en el panel de Cloudflare ya nos aparecerá como activo y tendremos acceso a todos los beneficios como CDN, Anti-DOSS, Firewall, certificado SSL gratuito, HTTP/3, 0-RTT.



4.8.2 ALMACENAMIENTO PÁGINA WEB

Banahosting tiene una seguridad y supervisión las veinticuatro horas al día, los 7 días de la semana. Tienen varios planes de almacenamiento web, el más adecuado para comenzar es un almacenamiento compartido SSD. Este alojamiento compartido se basa en que dentro de un servidor con grandes capacidades, propiedad de Banahosting, tienen varias instancias dentro del sistema operativo RedHat Enterprise Linux.

Para alquilar una de esas instancias voy a su página web: <https://www.banahosting.com>

Y en la pestaña de Web Hosting selecciono el plan más adecuado. En este caso es el BANA-STARTER DELUXE.



Una vez seleccionado y pagado ya puedo acceder desde mi cuenta al CPanel. Ahora para añadir el dominio recién comprado y con los DNS de Cloudflare voy a la pestaña de dominios. Y pulso sobre el botón CREATE NEW DOMAIN.



Ahora escribo el nombre de mi dominio y pulso sobre ENVIAR. Una vez vuelva a cargar, ya saldrá el mensaje de que todo a salido correctamente.

Create a New Domain

Dominio ?

Enter the domain that you would like to create:

Document Root (File System Location)

Share document root (/home/pyoxglmi/public_html)

Éxito: You have successfully created the new "ludiba.org" domain with the document root of "/home/pyoxglmi/public_html".

ENVIAR **SUBMIT AND CREATE ANOTHER**

4.9

PÁGINAS WEB

Ahora con almacenamiento procedo a crear las páginas web. La primera página web va a ser un subdominio de la página principal: blog.ludiba.org

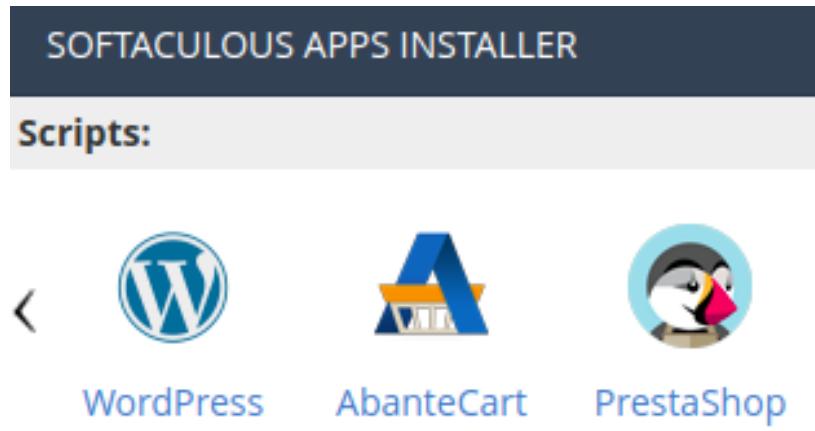
En este subdominio voy a instalar Wordpress utilizando SOFTACULOUS, una biblioteca de scripts con la que se pueden instalar aplicaciones comerciales y de código abierto con lo que automatiza la instalación. Esta incluido en Banahosting e integrado dentro de CPanel.

Voy a proceder a crear el diseño o tema de Wordpress desde cero sin utilizar plugins de creación, directamente en el lenguaje de programación PHP.

La página principal va a estar creada a partir de NodeJS. Para ello voy a utilizar Javascript para crear el servidor HTTP que va a responder las respuestas. También va a estar en Banahosting ya que el CPanel tiene integrado para desplegar el código de aplicaciones escritas en NodeJS.

4.9.1 SUBDOMINIO CON WORDPRESS

Para instalar Wordpress voy al CPanel de Banahosting. Nada más entrar puedo ver la pestaña de SOFTACULOUS APPS INSTALLER. Y el primer script es el de Wordpress.



Al pulsar sobre el ícono de Wordpress nos lleva a la página de instalación en la que hay un descripción corta, las características más importantes, unas capturas de pantalla de la aplicación, una demo en la que se puede probar como será, las puntuaciones de los usuarios, los comentarios de los usuarios, la opción de importar una instalación ya existente a otro dominio y el botón de instalar. En este caso quiero instalarla así que pulso sobre el respectivo botón. Este me lleva al formulario de instalación con el cual voy a poder configurar los aspectos generales de la aplicación Wordpress:

- Versión a instalar: 5.4.1
- URL donde instalar y protocolo: HTTPS://blog.ludiba.org
- Nombre del Sitio: Blog de LUDIBA GROUP
- Descripción del Sitio: Publicaremos las últimas novedades sobre LUDIBA GROUP.
- Habilitar Multisite (WPMU): No
- Usuario Administrador: administrador
- Contraseña Administrador: generada automáticamente
- Email Administrador: luisdieguezbarrio@gmail.com
- Selecciona Lenguaje: Spanish
- Limit Login Attempts (Loginizer): Sí
- Classic Editor: Sí
- wpCentral - Manage Multiple WordPress: No
- Base de datos: ludibaorg_wp
- Backup Location: Local Folder

Y al pulsar INSTALL se creará la base de datos MYSQL, se descargarán los archivos de Wordpress, se instalará dentro de la carpeta pública del sub dominio y se configurará automáticamente.

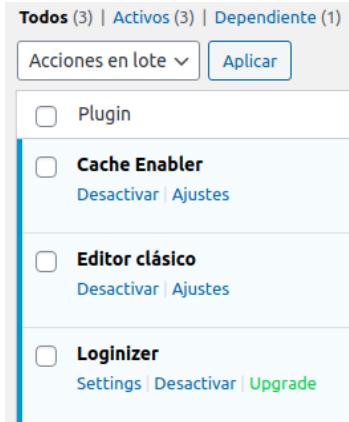
4.9.2

PLUGINS WORDPRESS

Una vez instalado Wordpress puedo acceder con las credenciales a la página de administración de la web en: blog.ludiba.org/wp-admin

Una vez aquí debo instalar una serie de plugins para mejorar el funcionamiento general de la web. He seleccionado los siguientes:

- Cache Enabler: Es un plugin desarrollado por KeyCDN, una empresa de CDN premium. Lo que hace este plugin es crear archivos HTML estáticos y los almacena en los discos del servidor. El servicio web entregará el archivo HTML estático, lo que evita procesos extra que tienen un consumo intensivo de recursos (núcleo de wordpress, plugins y base de datos).
- Editor clásico: Instalado automáticamente al seleccionarlo en el anterior paso de SOFTACULOUS. Este plugin desarrollado por Wordpress hace más sencillo editar las entradas del blog.
- Loginizer: También instalado automáticamente al seleccionarlo en el anterior paso de SOFTACULOUS. Este plugin desarrollado por Raj Kothari evita ataques de fuerza bruta bloqueando el inicio de sesión para la IP después de alcanzar el máximo de intentos permitidos. Tiene lista negra y lista blanca para las direcciones IP.



CÓDIGO WORDPRESS

Una vez instalado los plugins procedo a crear el código en PHP. Lo primero es crear la carpeta contenedora y las sub carpetas llamadas CSS, JS y IMG. Para crear un tema de Wordpress necesita tener varios archivos obligatorios.

- footer.php - Es donde irá el código de la parte inferior de la página en la que generalmente se incluyen links de navegación, enlaces de interés, copyright o las redes sociales.
- functions.php - Es la biblioteca donde voy a introducir las funciones de PHP para luego utilizarlas en dentro del tema de Wordpress.
- header.php - Es donde irá el código de los metadatos de la página, como son el título, la descripción, las palabras clave, el autor, los scripts JS y los estilos CSS.
- index.php - Es el archivo que se va a cargar al entrar en la página, dentro de él hay que referencias a los otros archivos para tener la página completa.
- nav.php - Es donde irá la primera parte del código después de la etiqueta head, generalmente se incluye una barra de navegación con los enlaces hacia las páginas la web.
- screenshot.png - Este archivo es una captura de pantalla para mostrar como quedaría el tema una vez instalado y que aparece en la pestaña de temas en Wordpress.
- single.php - Es el archivo que se va a cargar al cuerpo de la página, dentro de él llamaré a las entradas de Wordpress, al autor de la entrada, la fecha de creación y otros datos relativos a ella.
- style.css - Es el archivo principal del tema, en donde voy a declarar un comentario con los datos relativos al tema como el nombre del tema, el autor, la URL del autor, la descripción, la versión, la licencia y la URL de la licencia.

Por lo tanto el primer archivo footer.php contiene el siguiente código:

```
<!--Footer-->
<footer class="bg-white">
    <div class="container mx-auto px-8">
        <div class="w-full flex flex-col md:flex-row py-6">
            <div class="flex-1 mb-6">
                <a class="text-blue-600 no-underline hover:no-underline font-bold text-2xl lg:text-4xl" href="/">
                    LUDIBA
                </a><BR>
                <a class="text-blue-400 no-underline hover:no-underline font-bold text-2xl lg:text-3xl" href="/">
                    GROUP
                </a>
            </div>
            <div class="flex-1">
                <p class="uppercase text-gray-500 md:mb-6">Links interés</p>
                <ul class="list-reset mb-6">
                    <li class="mt-2 inline-block mr-2 md:block md:mr-0">
                        <a href="/faq" class="no-underline hover:underline text-gray-800 hover:text-orange-500">Preguntas (FAQ)</a>
                    </li>
                    <li class="mt-2 inline-block mr-2 md:block md:mr-0">
                        <a href="/contacto" class="no-underline hover:underline text-gray-800 hover:text-orange-500">Contacto</a>
                    </li>
                </ul>
            </div>
            <div class="flex-1">
                <p class="uppercase text-gray-500 md:mb-6">Legal</p>
                <ul class="list-reset mb-6">
                    <li class="mt-2 inline-block mr-2 md:block md:mr-0">
                        <a href="/terminos-y-condiciones" class="no-underline hover:underline text-gray-800 hover:text-orange-500">Términos y condiciones</a>
                    </li>
                    <li class="mt-2 inline-block mr-2 md:block md:mr-0">
                        <a href="/politica-privacidad" class="no-underline hover:underline text-gray-800 hover:text-orange-500">Política de privacidad</a>
                    </li>
                </ul>
            </div>
            <div class="flex-1">
                <p class="uppercase text-gray-500 md:mb-6">Redes sociales</p>
                <ul class="list-reset mb-6">
                    <li class="mt-2 inline-block mr-2 md:block md:mr-0">
                        <a href="https://blog.ludiba.org" class="no-underline hover:underline text-gray-800 hover:text-orange-500">Blog</a>
                    </li>
                    <li class="mt-2 inline-block mr-2 md:block md:mr-0">
                        <a href="https://www.linkedin.com/in/luis-diequez-barrio/" class="no-underline hover:underline text-gray-800 hover:text-orange-500">Linkedin</a>
                    </li>
                </ul>
            </div>
        </div>
    </div>

</footer>
<script>
    var scrollpos = window.scrollY;
    var header = document.getElementById("header");
    var navcontent = document.getElementById("nav-content");
    var navaction = document.getElementById("navAction");
    var brandname = document.getElementById("brandname");
    var toToggle = document.querySelectorAll(".toggleColour");

    document.addEventListener('scroll', function () {
        /*Apply classes for slide in bar*/
        scrollpos = window.scrollY;

        if (scrollpos > 10) {
            header.classList.add("bg-white");
            navaction.classList.remove("bg-white");
            navaction.classList.add("gradient");
            navaction.classList.remove("text-gray-800");
            navaction.classList.add("text-white");
            //Use to switch toggleColour colours
            for (var i = 0; i < toToggle.length; i++) {
                toToggle[i].classList.add("text-gray-800");
                toToggle[i].classList.remove("text-white");
            }
            header.classList.add("shadow");
            navcontent.classList.remove("bg-gray-100");
            navcontent.classList.add("bg-white");
        } else {
            header.classList.remove("bg-white");
            navaction.classList.remove("gradient");
            navaction.classList.add("bg-white");
            navaction.classList.remove("text-white");
            navaction.classList.add("text-gray-800");
        }
    });
</script>
```

```

    //Use to switch toggleColour colours
    for (var i = 0; i < toToggle.length; i++) {
        toToggle[i].classList.add("text-white");
        toToggle[i].classList.remove("text-gray-800");
    }

    header.classList.remove("shadow");
    navcontent.classList.remove("bg-white");
    navcontent.classList.add("bg-gray-100");
}

});

</script>
<script>
    /*Toggle dropdown list*/
    /*https://gist.github.com/slavapas/593e8e50cf4cc16ac972afcbad4f70c8*/
    var navMenuDiv = document.getElementById("nav-content");
    var navMenu = document.getElementById("nav-toggle");

    document.onclick = check;

    function check(e) {
        var target = (e && e.target) || (event && event.srcElement);

        //Nav Menu
        if (!checkParent(target, navMenuDiv)) {
            // click NOT on the menu
            if (checkParent(target, navMenu)) {
                // click on the link
                if (navMenuDiv.classList.contains("hidden")) {
                    navMenuDiv.classList.remove("hidden");
                } else {
                    navMenuDiv.classList.add("hidden");
                }
            } else {
                // click both outside link and outside menu, hide menu
                navMenuDiv.classList.add("hidden");
            }
        }
    }

    function checkParent(t, elm) {
        while (t.parentNode) {
            if (t == elm) {
                return true;
            }
            t = t.parentNode;
        }
        return false;
    }
</script>

</body>
</html>

```

Comienzo con la etiqueta footer que como su propio nombre indica, es para la sección del footer. Dentro de ésta voy a crear una estructura de divs para crear secciones dentro de la etiqueta del footer. El primero es un contenedor que va a ocupar el máximo espacio dentro del footer. El segundo es un contenedor que se va a adaptar si cambia el tamaño de la pantalla.

Dentro del segundo van a ir las divisiones, en este caso he creado cuatro divisiones, la primera es para el logo en texto plano con dos colores y una separación mayor. El segundo es el de los links de interés y contiene dos enlaces hacia preguntas frecuentes y hacia la página de contacto. El tercero es del apartado legal y contiene dos enlaces hacia términos y condiciones y hacia la política de privacidad. El cuarto y último es de las redes sociales, con dos enlaces hacia este mismo blog y hacia mi Linkedin.

Una vez terminado cierro la etiqueta del footer.

Continuo con una etiqueta script para incluir código Javascript ya que los scripts deben ser incluidos en la parte inferior y no en el head para no bloquear el árbol de acciones dentro de una página web.

Este primer script sirve para ir cuando un usuario haga scroll con el ratón cambie la transparencia del header hacia un color, en este caso blanco.

El siguiente script va a permitir mostrar el menú vertical en la cabecera.

Una vez terminado con los scripts ya puedo cerrar el body y todo el html.

Por lo tanto el segundo archivo functions.php contiene el siguiente código:

```
<?php
//Tamaño de imagen
add_image_size( 'destacada', 1254, 300, true );
add_theme_support( 'post-thumbnails' );
add_theme_support( 'custom-logo', array(
    //ALTO
    'height'      => 50,
    //ANCHO
    'width'       => 250,
    //PERMITIR FLEXIBILIDAD EN EL TAMAÑO
    'flex-height' => true,
    'flex-width'  => true,
    //
    'header-text' => array( 'site-title', 'site-description' ),
) );
?>
```

En este código ya de PHP procedo a añadir el tamaño de imagen de la primera entrada, luego añado las imágenes de miniatura y el logo . Por último añado el título y la descripción del sitio.

Por lo tanto el tercer archivo header.php contiene el siguiente código:

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>LUDIBA GROUP - OpenSource Hosting</title>
    <meta name="description" content="Blog de LUDIBA GROUP en donde se pueden leer las últimas novedades">
    <meta name="keywords" content="ludiba,blog informatica,servicios,organizacion">
    <meta name="author" content="Luis Dieguez Barrio">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="<?php echo get_template_directory_uri();?>/css/estilos-min.css">
    <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:400,700" rel="stylesheet">
    <link rel="stylesheet" href="<?php echo get_template_directory_uri();?>/css/gradiente.css">
    <link rel="stylesheet" href="<?php echo get_template_directory_uri();?>/css/menu-drop.css">
    <link rel="stylesheet" href="<?php echo get_template_directory_uri();?>/css/recuadro.css">
    <link rel="stylesheet" href="<?php echo get_template_directory_uri();?>/css/boton.css">
</head>
```

En este archivo contiene el inicio del HTML, comienza con la declaración que es un archivo HTML. Continua abriendo la etiqueta html y declara que el idioma es español, seguido de abrir la etiqueta head con las metaetiquetas:

- Content-Type
- viewport
- X-UA-Compatible
- title
- description
- keywords
- author

A continuación ya declaro todos los archivos CSS:

- font-awesome
- estilos generales
- gradiente fondo
- menú
- recuadro de cada post
- botón del post

Por lo tanto el cuarto archivo index.php contiene el siguiente código:

```
<?php get_header(); ?>
<?php include 'nav.php'; ?>

<!--Principal-->


<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
    <div class="container px-3 mx-auto flex flex-wrap flex-col md:flex-row items-center">
        <div class="recuadro">
            <?php the_post_thumbnail('destacada',array( 'class' => 'img-fluid rounded recuadro-imagen' )); ?>
            <div class="recuadro-body">
                <h2 class="recuadro-titulo"><?php the_title(); ?></h2>
                <p class="card-text"><?php the_excerpt(); ?></p>
                <a href="<?php the_permalink(); ?>" class="boton-nuevo">Leer más &rarr;</a>
            </div>
            <div class="recuadro-texto-abajo">
                <?php echo get_the_date(); ?> - <?php the_author(); ?>
            </div>
        </div>
    </div>
    <?php endwhile; else : ?>
    | | <p>Lo siento, no hemos encontrado ningún post.</p>
    <?php endif; ?>
</div>

</div>
<div class="relative -mt-12 lg:-mt-24">
    <svg viewBox="0 0 1428 174" version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
        <g stroke="none" stroke-width="1" fill="none" fill-rule="evenodd">
            <g transform="translate(-2.00000, 44.00000)" fill="#FFFFFF" fill-rule="nonzero">
                <path d="M0,0 C90.7283404,0.927527913 147.912752,27.187927 291.910178,59.9119003 C387.908462,81.7278826 543.605069,89.334785 759,82.7326078 C469.336065,156.2 <path d="M100,104.708498 C277.413333,72.2345949 426.147877,52.5246657 546.203633,45.5787101 C666.259389,38.6327546 810.524845,41.7979068 979,55.0741668 C931.5 <path d="M1046,51.6521276 C1130.83045,29.328812 1279.08318,17.607883 1439,40.1656806 L1439,120 C1271.17211,77.9435312 1140.17211,55.1609071 1046,51.6521276 Z" />
            <g transform="translate(-4.00000, 76.00000)" fill="#FFFFFF" fill-rule="nonzero">
                <path d="M0.457,34.035 C57.086,53.198 98.208,65.809 123.822,71.865 C181.454,85.495 234.295,90.29 272.033,93.459 C311.355,96.759 396.635,95.801 461.025,91.663 </path>
            </g>
        </g>
    </svg>
</div>
<?php get_footer(); ?>


```

En este archivo es donde se hace referencia a todos los archivos y funciones necesarias para mostrar la página completa.

Comienza con la función de Wordpress get_header(). Con esto se incluye todo el código de header.php

Continua con la función de PHP include que tiene el mismo uso que la anterior, añadir código de un archivo externo, en este caso permite declarar el archivo y es nav.php.

Después vuelvo a utilizar la estructura de divs para declarar el contenedor de cada entrada. Y utilizo una expresión condicional IF para comprobar si hay o no posts publicados. En caso afirmativo se ejecuta un código que crea cada contenedor individual y pide su miniatura destacada, en otro recuadro el título, en la parte inferior una pequeña parte del texto de la entrada, crea el botón hacia el enlace de la entrada y por último el autor y la fecha de creación. En caso negativo, muestra el texto: Lo siento, no hemos encontrado ningún post.

A continuación declaró el fondo en formato SVG con un div para hacer un efecto al acabar la página.

Por último utilizo la función de Wordpress get_footer(). Con esto se incluye todo el código de footer.php

Por lo tanto el quinto archivo nav.php contiene el siguiente código:

En este archivo es donde comienza el cuerpo de la página y está la barra de navegación superior.

Por ello comienza con la etiqueta body en la que incluyo los estilos para todo el cuerpo de la página.

Continuo con la etiqueta nav para indicar que comienza la barra de navegación. Vuelvo a utilizar la estructura de divs para organizar el documento. Comienzo con el div de la esquina superior izquierda en el cual indico la ruta del logo dentro de la carpeta img.

En el siguiente div indico que si cambia a un tamaño pequeño el menú se oculte y se muestre un ícono de tres líneas que al pulsarlo mostrará el menú. Esto es para que el tema sea *responsive* y pueda ser visualizado en dispositivos móviles o pantallas pequeñas.

En el siguiente div muestro la barra de navegación normal. Tiene cinco apartados:

- Almacenamiento
- Bots
- Juegos
- Vídeo

Permite añadir más y también permite dejar inactivo uno, por ejemplo en la pestaña de juegos está inactivo otro juego y pone próximamente así que al pulsar no lleva a ninguna página.

Acaba con un hr para separar la barra de navegación y cierra la etiqueta nav.

Por lo tanto el sexto archivo single.php contiene el siguiente código:

```
<?php get_header(); ?>
<div class="pt-24">
<?php include 'nav.php';?>
<div class="container px-3 mx-auto flex flex-wrap flex-col md:flex-row items-center">
    <div class="recuadro" style="margin-bottom: 20px; margin-top: 20px;">
        <!-- Page Content -->
        <div class="container">
            <div class="row">
                <!-- Post Content Column -->
                <div class="col-lg-8">
                    <?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
                        <!-- Title -->
                        <h1 class="recuadro-titulo"><?php the_title(); ?></h1>
                        <!-- Author -->
                        <p class="lead">
                            Por
                            <?php the_author(); ?>
                        </p>
                        <hr>
                        <!-- Date/Time -->
                        <p>Publicado <?php the_date(); ?> </p>
                        <hr>
                        <!-- Preview Image -->
                        <?php the_post_thumbnail('destacada',array( 'class' => 'img-fluid rounded recuadro-imagen' )); ?>
                        <hr>
                    <!-- Post Content -->
                    <br>
                    <br>
                    <div class="textos">
                        <?php the_content(); ?>
                    </div>
                    <hr>
                    <?php endwhile; else : ?>
                    | | <p>Lo siento, no hemos encontrado ningún post.</p>
                    <?php endif; ?>
                </div>
            </div>
            <!-- /.row -->
        </div>
        <!-- /.container -->
        | | </div>
        </div>
        </div>
    <?php get_footer(); ?>
```

En este archivo es la plantilla de cada entrada individual. Lo que se muestra cuando en la página se pulsa sobre el botón Leer más.

Al ser una página completa necesita realizar llamadas. Por ello necesito llamar lo primero al código del head con Wordpress get_header().

A continuación utilizo la estructura de divs para crear la estructura de la página.

Luego incluyo el código de la barra de navegación con include 'nav.php'.

Luego continuo utilizando divs para crear la estructura y creo el recuadro de la entrada individual con sus divisiones. Incluyo un recuadro para el título y hago una llamada en PHP para recibir dicha información de Wordpress. Luego hago lo mismo para el autor de la entrada, la fecha de publicación, la imagen destacada y el texto completo de la entrada. En caso de que ocurra algún error muestra un: Lo siento, no hemos encontrado ningún post.

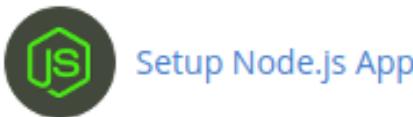
Por último cierro los divs y hago una llamada para terminar con el footer.

4.9.4

DOMINIO

PRINCIPAL NODEJS

Para instalar la aplicación de NodeJS voy al CPanel de Banahosting. En la parte inferior puedo ver la pestaña SOFTWARE. Una vez allí puedo ver la opción de *Setup NodeJS App*.



[Setup Node.js App](#)

Al pulsar sobre este botón me llevará a la página de gestión de aplicaciones NodeJS. Allí hay un botón CREATE APPLICATION con el cual podré desplegar el código de la aplicación. Al pulsarlo pedirá llenar una serie de campos para configurar la aplicación.

WEB APPLICATIONS [CREATE APPLICATION](#) CANCEL CREATE

Node.js version	12.9.0
Application mode	Development
Adds value for NODE_ENV variable	
Application root	codigoLudibaOrg
It is a physical address to your application on a server that corresponds with its URI. Upload your application files here.	
Application URL	ludiba.org
It is an HTTP/HTTPS link to your application	
Application startup file	/home/pyoxgimi/codigoLudibaOrg/app.js
Passenger log file	/home/pyoxgimi/

Una vez pulsado sobre CREATE se instalará NodeJS. Para que la aplicación sea totalmente funcional es necesario instalar los plugins por lo que pulso sobre RUN NPM INSTALL que auto detecta el archivo de configuración package.json.



En unos instantes comenzará la instalación.



Y si todo ha sido correcto ya podre verla en el menú principal.

App URI	App Root Directory	Mode	Status	Actions
ludiba.org/	/home/pyoxgimi/codigoLudibaOrg	development	● started (v12.9.0)	

CÓDIGO NODEJS

El código de la aplicación de NodeJS tiene varias partes. La primera e inicial es el servidor HTTP que escribí en el archivo app.js, este archivo es el primero en iniciarse y declara los plugins, las rutas a seguir y los aspectos generales de la aplicación.

Después en cada ruta voy a renderizar un archivo, al cual le puedo pasar variables desde la misma ruta. Para ello voy a utilizar EJS (Embedded JavaScript templates).

Luego dentro de cada archivo EJS voy a dividirlo en partes para hacer una organización y que cuando cambie, por ejemplo la barra de navegación, solo tenga que cambiarla en un archivo y se cambie a lo largo de toda la aplicación.

Para comenzar con este proyecto creo un archivo app.js y en el mismo directorio ejecuto npm init. Ese comando va a pedir una serie de valores para la configuración general:

- package name - El nombre de la aplicación.
- version - En caso de ser actualizada se irá aumentando el número de la versión.
- description - Una descripción sobre la aplicación.
- entry point - El archivo del servidor HTTP, en este caso app.js
- test command - Se pueden añadir tests y comandos para ejecutar al inicio de la aplicación para probar que todo funcione correctamente.
- git repository - El repositorio oficial donde se va almacenar el código.
- keywords - Varias palabras clave para identificar la aplicación.
- author - El nombre del autor del código.
- license - El tipo de licencia sujeta al código.

Lo primero que tengo que hacer es instalar los siguientes paquetes para que tanto el servidor HTTP como posteriormente la página funcione correctamente:

- axios - Es un cliente HTTP basado en promesas. Sirve para añadir esta funcionalidad de Javascript.
- body-parser - Es un middleware que sirve para manejar la solicitud de HTTP POST en Express. Analiza los datos JSON, búfer, cadena y URL codificados enviados utilizando la solicitud HTTP POST
- ejs - Es un lenguaje de plantillas que permite generar marcado HTML con JavaScript. Permite organizar las cosas.
- express - Es un framework mínimo y flexible para aplicaciones web y móviles.
- node-fetch - Este plugin permite añadir window.fetch. API Fetch proporciona una interfaz para realizar peticiones recursos.
- nodemon - Este plugin es solo para desarrollo, no va a ser instalado en producción. Permite ejecutar en segundo plano la aplicación mientras realizo cambios en el código.

Para instalar los plugins ejecuto *npm install* y el nombre de cada plugin. Luego en el archivo package.json ya podré ver todos los plugins como dependencias, nodemon como dependencia de solo desarrollo no producción y las características generales escritas al ejecutar el *npm init* al inicio.

```
{  
  "name": "ludiba-group",  
  "version": "1.0.0",  
  "description": "",  
  "main": "app.js",  
  "scripts": {  
    "test": "echo \\\"No test specified\\\" && exit 1"  
  },  
  "repository": {  
    "type": "git",  
    "url": "https://github.com/RedxLus/open-source-hosting.git"  
  },  
  "author": "Luis Dieguez Barrio",  
  "license": "ISC",  
  "dependencies": {  
    "axios": "^0.19.2",  
    "body-parser": "^1.18.3",  
    "ejs": "2.6.1",  
    "express": "4.16.3",  
    "node-fetch": "2.6.0"  
  },  
  "devDependencies": {  
    "nodemon": "^2.0.4"  
  }  
}
```

Una vez tenga la instalación de todas las dependencias y aspectos generales puedo comenzar con el código de la aplicación. Lo primero es crear el servidor HTTP en el archivo app.js. Es el siguiente código.

```
//jshint esversion:6
const express = require("express");
const bodyParser = require("body-parser");
const ejs = require("ejs");
const axios = require('axios');
const fetch = require('node-fetch');

const port = process.env.PORT || 3000;
const app = express();
app.use(bodyParser.urlencoded({extended: true}));
app.use(express.static(__dirname + '/public'));
app.set('view engine', 'ejs');

app.get("/", function (req,res) {
  res.render("inicio", {
    primero: ""
  });
});

app.get("/juegos/minecraft", function (req,res) {
  res.render("minecraft", {
  });
});

app.get("/terminos-y-condiciones", function (req,res) {
  res.render("footerEnlaces/terminos", {
  });
});

app.get("/politica-privacidad", function (req,res) {
  res.render("footerEnlaces/privacidad", {
  });
});

app.get("/faq", function (req,res) {
  res.render("footerEnlaces/faq", {
  });
});

app.get("/contacto", function (req,res) {
  res.render("footerEnlaces/contacto", {});
});

app.use((req, res, next) => {
  res.status(404).render("error404", {});
});

app.listen(port, function() {
  console.log("El servidor esta escuchando en el puerto: " + port);
});
```

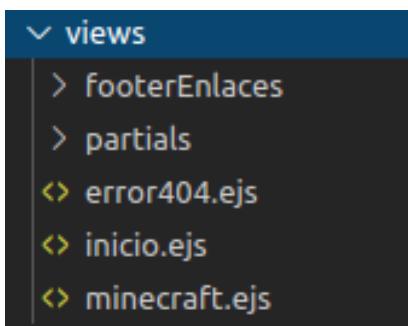
Lo primero que hago es declarar la versión ECMAScript a utilizar. Para ello utilizo un comentario y utilizo la versión esversion:6.

Lo siguiente es hacer referencia a las dependencias dentro del código como constantes para poder utilizarlas posteriormente. También declaro el puerto a escuchar y configuro Express para que utilice esas dependencias. Declaró una carpeta llamada public en donde irá todo lo que no sea javascript, como las imágenes a utilizar o el código CSS.

Por último declaro las rutas del servidor HTTP a escuchar.

- La primera es home (/) y por ello renderizará el archivo inicio.ejs cada vez que alguien entre a esa ruta.
- La segunda es una ruta doble (/juegos/minecraft) y renderizará el archivo minecraft.ejs cada vez que alguien entre a esa ruta completa.
- La tercera son los términos y condiciones (/terminos-y-condiciones) y renderizará el archivo terminos.ejs dentro de la carpeta de templates del footer cada vez que alguien entre a esa ruta.
- La cuarta es la política de privacidad (/politica-privacidad) y renderizará el archivo privacidad.ejs dentro de la carpeta de templates del footer cada vez que alguien entre a esa ruta.
- La quinta es la página de contacto (/contacto) y renderizará el archivo contacto.ejs dentro de la carpeta de templates del footer cada vez que alguien entre a esa ruta.
- La siguiente ruta es la página de error, cada vez que haya un error del tipo 404 o no encontrado, renderizará la página error404.ejs.
- La última página sirve como test para que al ejecutar la aplicación pueda ver que funciona correctamente y en qué puerto está corriendo.

Una vez declaradas las rutas, procedo a crear una carpeta llamada views para guardar todos los archivos EJS que van a ser renderizados al hacer cada petición.



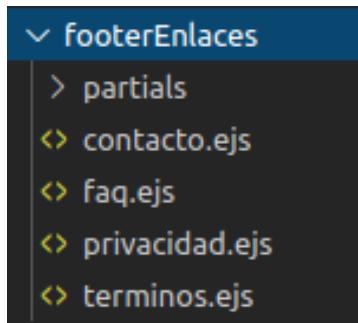
Dentro de dicha carpeta creo otras dos carpetas, una van a ser las partes específicas que se van a repetir a lo largo de toda la aplicación:

- footer.ejs
- heead.ejs
- headNav.ejs
- scriptAPI.ejs

El código en general es bastante similar al del blog pero adaptado a NodeJS para tener un mismo aspecto a lo largo de toda la aplicación.

Luego dentro de footerEnlaces están los archivos EJS respectivos a las páginas de los enlaces del footer con su respectiva carpeta de partes específicas ligeramente diferentes de las otras.

- head.ejs
- headNav.ejs



La página de error404.ejs contiene el siguiente código:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    body,
    html {
      padding: 0;
      margin: 0;
      font-family: 'Quicksand', sans-serif;
      font-weight: 400;
      overflow: hidden;
      background: linear-gradient(90deg, #rgb(55, 216, 201) 0%, #rgba(148,187,233,1) 100%);
    }

    .writing {
      width: 320px;
      height: 200px;
      background-color: #3f3f3f;
      border: 1px solid #bbbbbb;
      border-radius: 6px 6px 4px 4px;
      position: relative;
    }

    .writing .topbar {
      position: absolute;
      width: 100%;
      height: 12px;
      background-color: #f1f1f1;
      border-top-left-radius: 4px;
      border-top-right-radius: 4px;
    }

    @media screen and (max-width: 370px) {
      .container {
        -webkit-transform: scale(.6);
        transform: scale(.6);
      }
    }
  </style>
</head>

<body>
  <div class="container">
    <div class="error">
      <h1>404</h1>
      <p>Parece que algo ha salido mal. Compruebe que la URL a la que accedió es correcta o <a class="lnk" href="/">vuelva al inicio</a>.</p>
    </div>
    <div class="stack-container">
      <div class="card-container">
        <div class="perspec" style="--spreaddist: 125px; --scaledist: .75; --vertdist: -25px;">
          <div class="card">
            <div class="writing">
              <div class="topbar">
                <div class="red"></div>
                <div class="yellow"></div>
                <div class="green"></div>
              </div>
              <div class="code">
                <ul>
                </ul>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
    <div class="card-container">
      <div class="perspec" style="--spreaddist: 100px; --scaledist: .8; --vertdist: -20px;">
        <div class="card">
          <div class="writing">
            <div class="topbar">
              <div class="red"></div>
              <div class="yellow"></div>
              <div class="green"></div>
            </div>
            <div class="code">
              <ul>
              </ul>
            </div>
          </div>
        </div>
      </div>
    </div>
    <div class="card-container">
      <div class="perspec" style="--spreaddist:75px; --scaledist: .85; --vertdist: -15px;">
        <div class="card">
          <div class="writing">
            <div class="topbar">
              <div class="red"></div>
              <div class="yellow"></div>
              <div class="green"></div>
            </div>
            <div class="code">
              <ul>
              </ul>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
```

```

<div class="card-container">
  <div class="perspec" style="--spreaddist: 50px; --scaledist: .9; --vertdist: -10px;">
    <div class="card">
      <div class="writing">
        <div class="topbar">
          <div class="red"></div>
          <div class="yellow"></div>
          <div class="green"></div>
        </div>
        <div class="code">
          <ul>
            </ul>
        </div>
      </div>
    </div>
  </div>
</div>
<div class="card-container">
  <div class="perspec" style="--spreaddist: 25px; --scaledist: .95; --vertdist: -5px;">
    <div class="card">
      <div class="writing">
        <div class="topbar">
          <div class="red"></div>
          <div class="yellow"></div>
          <div class="green"></div>
        </div>
        <div class="code">
          <ul>
            </ul>
        </div>
      </div>
    </div>
  </div>
</div>
<div class="card-container">
  <div class="perspec" style="--spreaddist: 0px; --scaledist: 1; --vertdist: 0px;">
    <div class="card">
      <div class="writing">
        <div class="topbar">
          <div class="red"></div>
          <div class="yellow"></div>
          <div class="green"></div>
        </div>
        <div class="code">
          <ul>
            </ul>
        </div>
      </div>
    </div>
  </div>
</div>
</div>
<script>
  const stackContainer = document.querySelector('.stack-container');
  const cardNodes = document.querySelectorAll('.card-container');
  const perspecNodes = document.querySelectorAll('.perspec');
  const perspec = document.querySelector('.perspec');
  const card = document.querySelector('.card');

  let counter = stackContainer.children.length;

  //function to generate random number
  function randomIntFromInterval(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
  }

  //after tilt animation, fire the explode animation
  card.addEventListener('animationend', function () {
    perspecNodes.forEach(function (elem, index) {
      elem.classList.add('explode');
    });
  });

  perspec.addEventListener('animationend', function (e) {
    if (e.animationName === 'explode') {
      cardNodes.forEach(function (elem, index) {

        //add hover animation class
        elem.classList.add('popup');

        //add event listener to throw card on click
        elem.addEventListener('click', function () {
          let updown = [800, -800];
          let randomY = updown[Math.floor(Math.random() * updown.length)];
          let randomX = Math.floor(Math.random() * 1000) - 1000;
          elem.style.transform =
            `translate(${randomX}px, ${randomY}px) rotate(-540deg)`
          elem.style.transition = "transform 1s ease, opacity 2s";
          elem.style.opacity = "0";
          counter--;
          if (counter === 0) {
            stackContainer.style.width = "0";
            stackContainer.style.height = "0";
          }
        });
      });
    }
  });

  //generate random number of lines of code between 4 and 10 and add to each card
  let numLines = randomIntFromInterval(5, 10);

  //loop through the lines and add them to the DOM
  for (let index = 0; index < numLines; index++) {
    let lineLength = randomIntFromInterval(25, 97);
    var node = document.createElement("li");
    node.classList.add('node-' + index);
    elem.querySelector('.code ul').appendChild(node).setAttribute('style',
      '--linelength: ' + lineLength + '%;');
  }
</script>

```

```
//draw lines of code 1 by 1
if (index == 0) {
    elem.querySelector('.code ul .node-' + index).classList.add('writeLine');
} else {
    elem.querySelector('.code ul .node-' + (index - 1)).addEventListener(
        'animationend',
        function (e) {
            elem.querySelector('.code ul .node-' + index).classList.add(
                'writeLine');
        });
}
});
</script>
</body>
</html>
```

Como se puede ver aunque sea EJS es HTML.

Comienzo con las metaetiquetas, seguidas del código CSS de está página y la adaptabilidad para distintas pantallas.

Luego en el body utilizo la estructura de divs para crear varias cajas contenedoras y poder introducir varias imágenes que luego con Javascript voy a animar para hacer un efecto de movimiento.

Por último añado el script que va a crear el movimiento. Una vez pulses sobre la imagen si el numero es par elimina la imagen hacia arriba y si es impar hacia abajo. Una vez no queden imágenes el texto del error 404 se centra en la pantalla.

La página de inicio.ejs contiene el siguiente código:

```
<% include("partials/head"); -%>
<% include("partials/headNav"); -%>

<!--Principal-->


<!--Columna izquierda--&gt;
<div class="flex flex-col w-full md:w-2/5 justify-center items-start text-center md:text-left">


uppercase tracking-loose w-full">Completa tu plataforma todo en 1



leading-normal text-2xl mb-8">¿Quieres almacenar archivos, hosting de webs, jugar, utilizar bots para redes sociales o una subir tus propios videos?



<%= primero %>



Utiliza el menú superior para ver las distintas propuestas


<!--Columna derecha--&gt;
&lt;style&gt;
    .imagineSlider {display:none;}
&lt;/style&gt;
<div class="w-full md:w-3/5 py-6 text-center">





```

En esta página ya utilizo etiquetas propias de EJS para hacer llamadas a otros archivos. Comienzo llamando al archivo head.ejs y headNav.ejs dentro de la carpeta partials.

Luego creo varios contenedores con divs para el texto, el botón y las imágenes que van a estar animadas en carrusel con Javascript.

Y el último div es para crear un efecto curvo al finalizar el documento al igual que en el blog de wordpress.

Después esta el script para mover las imágenes en modo carrusel cada 2 segundos.

Y para acabar llamo al archivo footer.ejs dentro de la carpeta partials.

La página de minecraft.ejs contiene el siguiente código:

```
<%- include("partials/head"); -%>
<%- include("partials/headNav"); -%>

<!--Hero-->
<link href='https://fonts.googleapis.com/css?family=Lato:400,300,700,100' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="/css/tabla.css">
<style>
.error {
    background: red;
    display: block;
    text-decoration: none;
    padding: 20px;
    color: white;
    position: relative;
    overflow: hidden;
    transition: all 0.3s ease-in-out;
}
</style>

<div class="pt-24">
    <div class="container px-3 mx-auto flex flex-wrap flex-col md:flex-row items-center">
        <div class="price-table-wrapper">
            <div class="pricing-table">
                <h2 class="pricing-table__header">- Servidor en <b>España</b> - 
                <h3 class="pricing-table__price">Detalles del servidor: </h3>
                <a class="pricing-table__button" href="#" id="btnprimero">
                    <span id="primero">cargando ...</span>
                </a>
                <ul class="pricing-table__list">
                    <li><b>Mensaje:</b><span id="mensaje1">cargando ...</span></li>
                    <li><b>Jugadores:</b><span id="jugadores1">cargando ...</span></li>
                    <li><b>IP:</b><span id="ip1">cargando ...</span></li>
                    <li><b>Puerto:</b><span id="puerto1">cargando ...</span></li>
                </ul>
            </div>
            <div class="pricing-table">
                <h2 class="pricing-table__header">- Servidor en <b>Argentina</b> - 
                <h3 class="pricing-table__price">Detalles del servidor: </h3>
                <a class="pricing-table__button" href="#" id="btnsegundo">
                    <span id="segundo">cargando ...</span>
                </a>
                <ul class="pricing-table__list">
                    <li><b>Mensaje:</b><span id="mensaje2">cargando ...</span></li>
                    <li><b>Jugadores:</b><span id="jugadores2">cargando ...</span></li>
                    <li><b>IP:</b><span id="ip2">cargando ...</span></li>
                    <li><b>Puerto:</b><span id="puerto2">cargando ...</span></li>
                </ul>
            </div>
            <div class="pricing-table">
                <h2 class="pricing-table__header">- Servidor en <b>EEUU</b> - 
                <h3 class="pricing-table__price">Detalles del servidor: </h3>
                <a class="pricing-table__button" href="#" id="btntercero">
                    <span id="tercero">cargando ...</span>
                </a>
                <ul class="pricing-table__list">
                    <li><b>Mensaje:</b><span id="mensaje3">cargando ...</span></li>
                    <li><b>Jugadores:</b><span id="jugadores3">cargando ...</span></li>
                    <li><b>IP:</b><span id="ip3">cargando ...</span></li>
                    <li><b>Puerto:</b><span id="puerto3">cargando ...</span></li>
                </ul>
            </div>
        </div> </div>
        <div class="relative mt-12 lg:-mt-24">
            <svg viewBox="0 0 1428 174" version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
                <g stroke="none" stroke-width="1" fill="none" fill-rule="evenodd">
                    <g transform="translate(-2.00000, 44.00000)" fill="#FFFFFF" fill-rule="nonzero">
                        <path d="M0,0 C90.7283404,0.927527913 147.912752,27.187927 291.910178,59.9119003 C387.908462,81.7278826 543.605069,89.334785 759,82.7326078 C469.336065,156.2 51.6521276 C130.83045,29.328812 1279.08318,17.607883 1439,40.1656806 L1439,120 C1271.17211,77.9435312 1140.17211,55.1609071 1046,51.6521276 Z" />
                    <g transform="translate(-4.00000, 76.00000)" fill="#FFFFFF" fill-rule="nonzero">
                        <path d="M0.457,34.035 C57.086,53.198 98.208,65.809 123.822,71.865 C181.454,85.495 234.295,90.29 272.033,93.459 C311.355,96.759 396.635,95.801 461.025,91.665 Z" />
                    </g>
                </g>
            </svg>
        </div>
        <%- include("partials/scriptAPI2"); -%>
        <%- include("partials/footer"); -%>
    </div>
```

Utilizo las etiquetas propias de EJS para hacer llamadas a otros archivos. Comienzo llamando al archivo head.ejs y headNav.ejs dentro de la carpeta partials.

Luego creo varios contenedores con divs para las columnas tres columnas donde se van a mostrar los servidores de este juego.

Y el último div es para crear un efecto curvo al finalizar el documento al igual que en el blog de wordpress.

Después esta el script que va a interactuar con cada servidor de este juego y cambiar los valores de las anteriores columnas. Este script está en la carpeta partials y se llama scriptAPI.ejs.

Y para acabar llamo al archivo footer.ejs dentro de la carpeta partials.

5.

PRESUPUESTO

El presupuesto para esta aplicación completa se basa en la factura de los costes por el alquiler de los servidores de Google Cloud.

SKU	Producto	ID de SKU	Utilización	↓ Coste
● Network Load Balancing: Forwarding Rule Minimum Service Charge in Americas	Compute Engine	B16D-040F-F105	1.069,37 hour	24,51 €
● N1 Predefined Instance Core running in Netherlands	Compute Engine	62A6-21EE-5C6A	517,22 hour	16,47 €
● Storage PD Capacity in Netherlands	Compute Engine	AE8C-46C3-4994	330,02 gibibyte month	13,57 €
● Preemptible N1 Predefined Instance Core running in Americas	Compute Engine	D498-1ECA-87C1	1.950,39 hour	11,91 €
● N1 Predefined Instance Ram running in Netherlands	Compute Engine	7919-4540-EDB2	1.939,56 gibibyte hour	8,28 €
● Storage PD Capacity	Compute Engine	D973-5D65-BAB2	306,94 gibibyte month	8,07 €
● Network Load Balancing: Forwarding Rule Minimum Service Charge in Netherlands	Compute Engine	BD72-394D-4654	258,43 hour	6,50 €
● Preemptible N1 Predefined Instance Ram running in Americas	Compute Engine	5451-0A15-0123	7.314,04 gibibyte hour	5,99 €
● External IP Charge on a Preemptible VM	Compute Engine	4AF8-7C1F-39C4	1.951,88 hour	3,58 €
● External IP Charge on a Standard VM	Compute Engine	C054-7F72-A02E	517,56 hour	1,89 €
● Network Inter Zone Egress	Compute Engine	DE9E-AFBC-A15A	9,53 gibibyte	0,09 €
● Network Load Balancing: Data Processing Charge in Americas	Compute Engine	EECF-0EFC-74CA	3,06 gibibyte	0,02 €
● N1 Predefined Instance Core running in Americas	Compute Engine	2E27-4F75-95CD	0,44 hour	0,01 €
● Licensing Fee for Windows Server version 1809 Datacenter Edition (CPU cost)	Compute Engine	6AE7-6CBB-DE7E	0,34 hour	0,01 €
● Network Load Balancing: Data Processing Charge in Netherlands	Compute Engine	5ADA-0BDE-006C	1,32 gibibyte	0,01 €

El coste aproximado de los servidores, cluster y la red sería de: 100,90 euros por dos meses de uso continuado.

Luego el servidor al ser comprado por piezas puedo desglosar su coste:

Placa madre (motherboard): Supermicro X8DT3	109,46
Procesador (CPU): Intel Xeon X5680 (x2 Sockets)	140,58
Memoria (RAM): 16GB DDR3 ECC REG	278,50
Disco Duro (SSD): KINGSTON SA400S3 240GB	33,94
Fuente de alimentación (PSU): EVGA 750 G3 Gold	99,95
Ventiladores: ARTIC Alpine 11 PLUS (x2)	62,01
Gabinete: NEO R810 4U	74,80

Lo que hace un total de 799,24 euros por el servidor local como único pago para comprar las piezas.

El coste de electricidad lo puedo calcular en base del consumo máximo de la fuente de alimentación, en este caso 750 vatios. Y el número de días y horas, en este caso 24 y 7.

Consumo del dispositivo en vatios*

750

Esta cantidad de vatios consume el dispositivo. Puede encontrar consejos para realizar las medidas en el texto de abajo.

¿Con qué frecuencia se utiliza el dispositivo?*

a diario

¿Cuánto tiempo utiliza el dispositivo normalmente?*

24 horas

Coste de la electricidad en € por kilovatio hora*

0,31

(Precio/kWh). Ya se ha preseleccionado un valor medio, más información en el texto de abajo.

Resultado

El consumo anual del dispositivo es **6570,00 kWh**. Por tanto, el coste en electricidad del dispositivo al año es **2036,70**

6.

CONCLUSIONES

En conclusión, este proyecto FTC demuestra que se puede construir una empresa de servicios utilizando solamente herramientas de código abierto. Y con una rentabilidad tanto alta como de recuperación rápida, ya que al segundo día de publicar la página web ya me llegaron correos de contacto para contratar los servicios ofertados y otros que aun no estaban disponibles.

Además, demuestro que se pueden utilizar lenguajes totalmente diferentes en un mismo proyecto, todo ello pensando para que está específicamente diseñado cada lenguaje y adaptándome a ello. Por ejemplo podía haber utilizado solo Javascript pero entonces crear el blog hubiera sido mucho más complicado. Y por el otro lado podía haber utilizado solo PHP pero entonces no hubiera podido utilizar los packages y hubiera tardado mucho más tiempo en desarrollarlo.

También demuestro como la fecha de creación no es realmente importante, ya que utilice lenguajes de programación como Javascript o PHP cuyo inicio es de aproximadamente 1995 junto con herramientas como Docker, Kubernetes, Proxmox y Rancher que han sido creadas en los últimos años.

Demuestro como crear una nube híbrida en vez de tener todos los servidores centralizados. Por ello utilice para los servicios el Google Cloud y para la gestión de ellos un servidor local con Proxmox.

De misma manera, demuestro y aplico las medidas de seguridad, muchas veces olvidadas, pero que son de primera necesidad en caso de ataque. Tanto la seguridad de los servidores Cloud utilizando Rancher como gestor, como las medidas de seguridad del servidor local con Promox ya que dentro está el gestor de Rancher y también al ponerlo todo bajo una CDN evitando dichos ataques.

Por ello en este proyecto, además de hacer una empresa altamente rentable, he incluido premeditadamente todas las asignaturas de este ciclo. Bases de datos, su gestión y administración para Wordpress y el servicio de Nextcloud. Planificación y administración de redes para gestionar los servidores de peticiones HTTP, la red CDN y la VPN. Implantación de sistemas operativos para instalar Proxmox, instalar Ubuntu server y gestionar ambos. Fundamentos de hardware para realizar la instalación por piezas del servidor local. Lenguajes de marcas e Implantación web para realizar ambas aplicaciones, tanto la página principal como el blog. Seguridad y alta disponibilidad para los servidores, crear y gestionar los clusters y aplicar medidas de seguridad.

7.

BIBLIOGRAFÍA

Jaramillo, F. (2018) DE APLICACIONES MONOLÍTICAS A MICROSERVICIOS. Aplyca. <https://www.aplyca.com/es/blog/aplicaciones-monoliticas-o-microservicios>

Fowler, M. (2014) MICROSERVICES, A DEFINITION OF THIS NEW ARCHITECTURAL TERM. <https://martinfowler.com/articles/microservices.html>

Alviz, J. (2016) PRINCIPIOS DEL DESARROLLO ÁGIL. Renacen. <https://www.renacen.com/blog/principios-del-desarrollo-agil-metodologias-agiles/>

Maurer, M.(2020) PROXMOX VE INSTALLATION. Proxmox. <https://pve.proxmox.com/wiki/Installation>

Rojo, M. (2005) EULA: COMPROMISOS Y DERECHOS FRENTE A LOS PROGRAMAS INFORMÁTICOS. Eroski. <https://www.consumer.es/tecnologia/software/eula-compromisos-y-derechos-frente-a-los-programas-informaticos.html>

Dieguez, L. (2018) COMO VIRTUALIZAR UN SERVIDOR Y CREAR UN CLUSTER. <https://luisiblogdeinformatica.com/como-virtualizar-un-servidor-y-crear-un-cluster/>

Lottner, M. (2017) TRABAJAR CON CERTIFICADOS. Microsoft. <https://docs.microsoft.com/es-es/dotnet/framework/wcf/feature-details/working-with-certificates>

Fernández, D. (2019) AGREGAR UN CERTIFICADO INTERMEDIO A LAS ENTIDADES DE CERTIFICACIÓN INTERMEDIAS. Vmware. <https://docs.vmware.com/es/VMware-Horizon-7/7.0/com.vmware.horizon-view.installation.doc/GUID-8D783A3E-2E0D-4A2B-9752-79F70C6D57EF.html>

Maurer, M. (2020) PROXMOX VE ADMINISTRATION GUIDE. Proxmox. <https://pve.proxmox.com/pve-docs/pve-admin-guide.html>

Esler, J. (2020) OPEN SOURCE STANDARD SCANNING SOFTWARE. Cisco. <https://www.clamav.net/>

Christopher, E. (2020) TEACH, LEARN, AND MAKE. Raspberry pi foundation. <https://www.raspberrypi.org/>

Dieguez, L. (2018) INSTALAR OPENVPN EN 5 MINUTOS. <https://luisiblogdeinformatica.com/instalar-openvpn-en-5-minutos/>

Choudury, S. (2020) RUN KUBERNETES EVERYWHERE. WHY RANCHER?. Rancher labs. <https://rancher.com/why-rancher/>

Choudury, S. (2020) RUN KUBERNETES EVERYWHERE. FEATURES. Rancher labs. <https://rancher.com/docs/rancher/v2.x/en/overview/>

Romero, G. (2020) CREA Y HABILITA CUENTAS DE SERVICIO PARA INSTANCIAS. Google cloud.

<https://cloud.google.com/compute/docs/access/create-enable-service-accounts-for-instances>

Romero, G. (2020) REGIONES, ZONAS Y CLÚSTERES. Google cloud.

<https://cloud.google.com/compute/docs/regions-zones>

Wanstrath, C. (2020) AUTOMATE YOUR WORKFLOW FROM IDEA TO PRODUCTION. Github. <https://github.com/features/actions>

Ludovic, F. (2020) THE CLOUD NATIVE EDGE ROUTER. Containo.

<https://containo.us/traefik/>

Lancey, V. (2019) DEPRECATED APIS REMOVED. Kubernetes.

<https://kubernetes.io/blog/2019/07/18/api-deprecations-in-1-16/>