Universidad de Valladolid

MÁSTER UNIVERSITARIO

Ingeniería Informática

TRABAJO FIN DE MÁSTER

La nube: Despliegue y Escalado de Servicios

Realizado por **PABLO DIEZ ÁLVAREZ**

❁❁❁

Universidad de Valladolid

19 de julio de 2021

Tutor: Benjamín Sahelices Fernández

Resumen

En el mundo actual en que vivimos cada vez se utilizan mayores servicios a través de internet y se extiende su uso a campos más amplios constantemente. Esto conlleva la aparición de nuevos términos con los que convivimos cada día, es el caso del término “nube”. Cada vez se hace más referencia a él y a los distintos servicios que ofrece, como “almacenamiento en la nube” o “computación en la nube”. Pero realmente, ¿qué conllevan estos términos?

El objetivo de este TFM es aportar un poco de luz al término general de “la nube” y mostrar cómo funcionan realmente estas por debajo; así como realizar un pequeño caso práctico que lo demuestre.

Índice de Contenidos

Índice de Tablas

Índice de Figuras

Capítulo 1. Introducción

1. Introducción:

Con el avance en la utilización masiva de Internet y la globalización en su uso simultáneo de forma concurrente por usuarios repartidos geográficamente por el planeta, el paradigma de funcionamiento de "la gran red" ha cambiado. La informática ha evolucionado desde un sistema "clásico" basado en Unidades Centrales y Terminales en los que los usuarios desarrollaban su trabajo, continuando por el paso a los Ordenadores Personales y la utilización de servidores dedicados para las cargas de trabajo y proveer recursos utilizados por multitud de usuarios. Tras estos movimientos desde un sistema centralizado a, posteriormente uno más descentralizado, la tendencia volvió a virar hacia la centralización de servicios en grandes centros de datos que almacenaban los servidores y, con la llegada de la virtualización de sistemas, esto supuso un mejor rendimiento ante el desuso en sistemas dedicados.

Pero, tras este paso de virtualización de los sistemas hardware, tenía que continuar por la virtualización de los sistemas software también. Esto se generó con los sistemas de contenerización de aplicaciones y servicios; lo que promovió el cambio de arquitectura de las aplicaciones desde una base monolítica a sistemas basados en microservicios trabajando juntos, lo que simplificaba el desarrollo y facilitaba la gestión tanto global como de fallos que pudieran darse para que no afectasen a otros servicios y, radicalmente, al funcionamiento de la aplicación completa.

Paralelamente a estos cambios de arquitectura software, fue apareciendo el famoso concepto de nube como un sistema etéreo en el que se desarrollan y proporcionan servicios a los usuarios. Pero las preguntas realmente serían, ¿Qué es una nube?, ¿Cómo funcionan?, ¿Cuál es la plataforma que "sustenta" el concepto de una nube? En este Trabajo de Fin de Máster se propone entonces, la consecución de los siguientes objetivos con motivo de alcanzar respuesta a dichas preguntas:

- Entender qué es la nube y cómo se llega a dicho concepto.
- Entender los distintos tipos existentes y sus diferencias principales
- Crear la metodología necesaria para desarrollar un ejemplo de plataforma que simule el comportamiento de la nube.
- Realizar el desarrollo de dicha plataforma y su despliegue para demostrar su funcionamiento
- Comprobar el correcto funcionamiento de la plataforma ante cargas de trabajo que pudieran darse en un entorno real de uso.
- Estudiar las conclusiones extraídas del proceso y valorar los posibles casos de trabajo futuro con el objetivo de ampliar dicha plataforma y su funcionabilidad.

Se desarrollará dicha plataforma y se irán explicando las tecnologías utilizadas, así como su elección y su interacción con el resto de las utilizadas para conformar la plataforma

Capítulo 2. Contexto

1. Cloud computing

1.1. Introducción

Se entiende como concepto de “nube” a los servidores que se accede a través de internet y el software y bases de datos que se ejecutan ellos para proveer un servicio. Estos servidores que conforman “la nube” se encuentran localizados en centros de datos repartidos por todo el mundo.

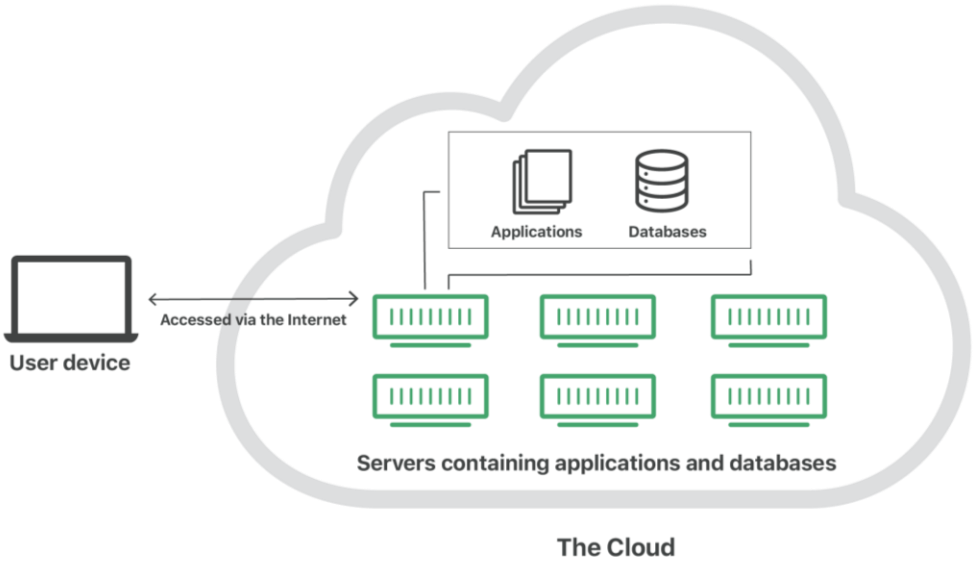


Ilustración 1: Esquema de "la nube" [1]

La computación en la nube posee una serie de ventajas para sus usuarios, como permitirles acceder **simultáneamente** a la información y aplicaciones que en ella se encuentran desde distintos puntos geográficos sin, por ello, haya diferencia en su forma de acceso. Además, esto supone un **ahorro** en los costes asociados de mantener una infraestructura propia tanto para usuarios finales como para las empresas, al permitir que no deban centrarse en el mantenimiento y actualización de esos sistemas. Lo que, a su vez, supone una **oportunidad de negocio** para los proveedores de “nube”, que pueden invertir en nuevos sistemas para mejorar la eficacia y eficiencia de sus sistemas y atraer nuevos usuarios a sus plataformas.

Pero el concepto de “nube” no sería posible sin la existencia de otro concepto básico en la informática, la **virtualización** pues esta es la base que desembocó en la nube tal como la conocemos debido a la evolución desde máquinas físicas a las máquinas virtuales y, a su vez, de los servidores y clúster físicos a los virtualizados. Veremos a continuación más en detalle estos conceptos.

1.2. Virtualización

Según [2], “La virtualización comenzó en los años 60 como un método de división lógica de los recursos proporcionados por los sistemas de computadores *mainframe* entre diferentes aplicaciones”. Podemos entender este concepto como el acto de crear una simulación virtual de un determinado recurso, incluidas plataformas de hardware, dispositivos de almacenamiento y recursos de redes de computación.

La virtualización puede dividirse en dos grandes ramas: la Virtualización de Hardware y la Virtualización Parcial (Paravirtualización).

1.2.1. Virtualización Hardware

Se entiende como virtualización hardware a la creación de una simulación completa que permita a los entornos software ejecutarse sin necesidad de ser modificados para ello. En este proceso, se crean las denominadas "máquinas virtuales"

En la virtualización hardware, el sistema anfitrión (*host*) utiliza un gestor de máquinas virtuales, llamados normalmente *hipervisores*, para crear la virtualización de la máquina huésped (*guest*) y su Sistema Operativo. Algunos de los hipervisores más conocidos y utilizados actualmente son:

- Oracle VirtualBox
- VMware vSphere
- Citrix XenServer
- Microsoft Hyper-V
- Red Hat Enterprise Virtualization

a) Tipos de Máquinas Virtuales

Aparte de por los hipervisores mencionados anteriormente, los distintos tipos de máquinas virtuales que pueden existir se pueden clasificar según su arquitectura virtualizada:

- Máquinas virtuales Windows

  La mayor parte de los Hipervisores soportan máquinas virtuales Windows, aunque este sistema operativo también posee integrado su propio hipervisor que gestiona el propio sistema anfitrión y los huéspedes a través de este.
- Máquinas virtuales Android

  El sistema operativo Android funciona en la arquitectura ARM, lo que le hace incompatible con la arquitectura pc x86, por ello no sirve un hipervisor para su virtualización. Se hace necesaria la utilización de otras herramientas que emulen dicha arquitectura mediante software.
- Máquinas virtuales Mac

  Apple prohíbe la utilización del sistema operativo macOS salvo en hardware de Apple, por lo que la emulación de máquinas virtuales con dicho Sistema Operativo se debe realizar sobre un sistema hardware Mac anfitrión.
- Máquinas virtuales iOS

  De forma similar al caso de las máquinas virtuales Mac, Apple no permite la ejecución del sistema operativo iOS fuera de dispositivos iOS. Aunque existe la emulación de dicho sistema mediante el software de desarrollo de software para el ecosistema Apple, *Xcode*.
- Máquinas virtuales Java/Python

  Las plataformas Java y Python permiten la creación de un entorno de ejecución que facilita la ejecución de código en distintas máquinas mediante la traducción del código de alto nivel a *Bytecode* y posteriormente a código máquina.

- Máquinas virtuales Linux

  Los sistemas operativos basados en Linux permiten su ejecución mediante máquinas virtuales sobre otros sistemas anfitriones. También incluyen su propio hipervisor (KVM) que permite la emulación de otros sistemas sobre un anfitrión Linux

Se puede encontrar más información acerca de estos tipos de máquinas virtuales en [3]

b) Virtualización de almacenamiento

La virtualización de almacenamiento permite la agrupación de los sistemas de almacenamiento físicos en clústeres que conformen un único almacenamiento virtual

Esto se puede llevar a cabo mediante el uso de tecnologías como SCSI, NFS, RAID, unRAID, ZFS

c) Virtualización de Red

La virtualización de redes consiste en la utilización de Software para desacoplar las redes virtuales de la capa Hardware de la red física.

Un ejemplo es los switches virtuales utilizados por los hipervisores en la gestión de la red entre los anfitriones-huéspedes

d) Virtualización de Entrada/Salida

La virtualización de dispositivos de entrada/salida permite hacer un uso compartido de estos entre múltiples máquinas virtuales y maquinas físicas. En ella, el hipervisor, se encarga de facilitar la comunicación entre dicho recurso presente en la maquina anfitriona y las distintas maquinas huéspedes.

1.2.2. Virtualización Parcial (Paravirtualización)

La virtualización parcial consiste en la simulación parcial de elementos aislados en su propio dominio. Normalmente estos elementos deben ser modificados de su forma original para permitir su funcionamiento en sistemas paravirtualizados.

a) Virtualización de escritorio

Consiste en la creación de una infraestructura virtual de escritorio sobre un mismo sistema, formada por varios escritorios de trabajo diferentes. Es el caso de los sistemas clásicos *IBM Mainframe* de los años 60, en que a un único sistema se conectaban terminales independientes.

b) Virtualización de aplicaciones

La virtualización de aplicaciones se puede considerar como un “empaquetado” de dicha aplicación y todos sus recursos necesarios para el correcto funcionamiento de ella.

Los máximos exponentes en este caso son los contenedores de aplicaciones aportados por tecnologías como Docker, KVM, Kubernetes, etc.…

1.2.3. Virtualización de Clústeres

Un clúster físico de computadoras es un conjunto de máquinas físicas conectadas por una red física, normalmente de alta velocidad y capacidad, que las permite trabajar conjuntamente como si de una única máquina se tratase.

Según la información obtenida de [4], el computo con clústeres de computadoras, surge de la convergencia de varias tendencias de avance relacionadas, como son: las variaciones en la disponibilidad de microprocesadores de alto rendimiento y las redes de alta velocidad; el desarrollo de nuevas tecnologías de software para computo distribuido de alto rendimiento y la creciente necesidad de potencia computacional par nuevas aplicaciones que la requieran. Esto hace que estos sistemas estén sujetos a grandes necesidades de escalabilidad y a sus restricciones asociadas, sean estas relacionadas con el abastecimiento, el encarecimiento de materiales o las propias restricciones de espacio físico que puedan conllevar. Además, existe la limitación en cuanto a que la plataforma sujeta, y viceversa, a las restricciones de las aplicaciones que en ella se ejecutan.

Si nos fijamos en la siguiente imagen, podemos hacernos una idea sencilla del problema si suponemos que cada uno de los clústeres presentes en el diagrama fuese un clúster físico individual. Ya que todos ellos deberían estar funcionando y accesibles en todo momento por el resto de ellos, aunque no fuesen necesarios en todo momento, lo que conllevaría unos recursos físicos activos sin uso real.

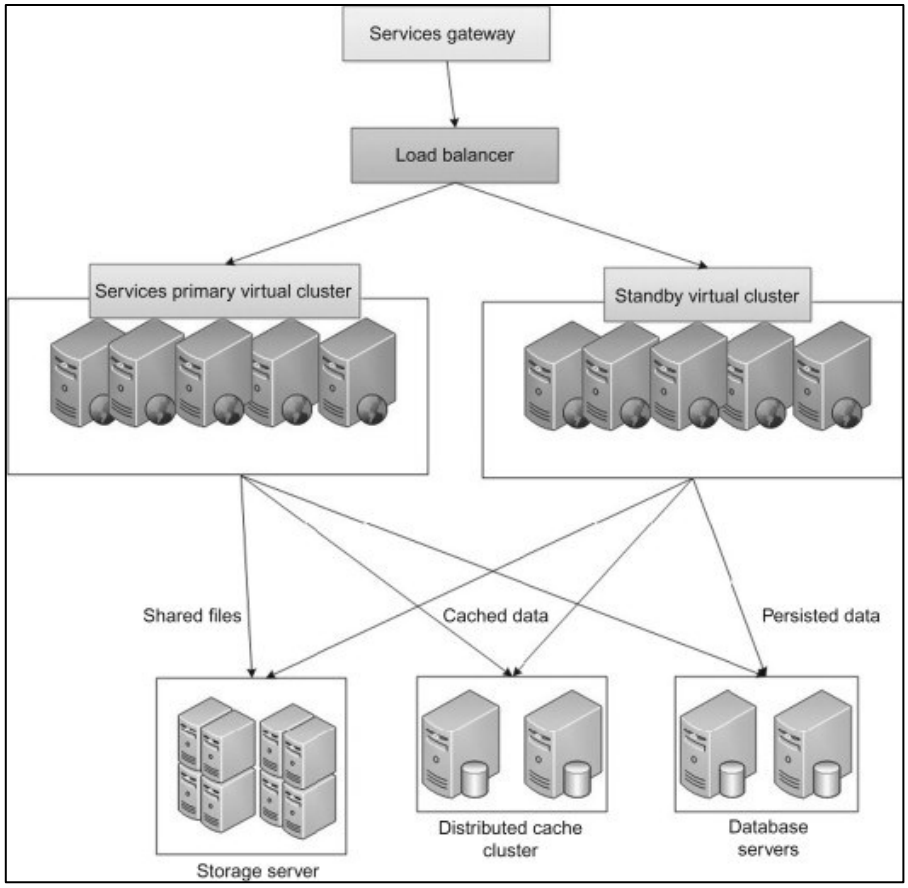


Ilustración 2: Estructura distribuida de clústeres [5]

La necesidad de versatilidad en cuanto a la ejecución de diferentes aplicaciones que conlleven diferentes necesidades sobre una misma infraestructura, junto a los problemas anteriormente descritos y otros extra como la eficiencia de las plataformas cuando se encuentran en una situación de infrautilización, hace que se genere un movimiento de transición desde clúster físicos a clúster individuales.

- Transición

  Esta transición a clúster virtuales se lleva a cabo paralela y relacionadamente al movimiento de virtualización de máquinas individuales que previamente se mencionó en cuanto a los sistemas de usuario. De tal forma que, a partir de grandes clústeres físicos de máquinas, se generan clústeres virtuales que cumplan con las características y necesidades de cada caso específico. Podemos ver en qué estado queda la estructura de estos clústeres virtuales y sus clústeres físicos asociados en la siguiente figura:

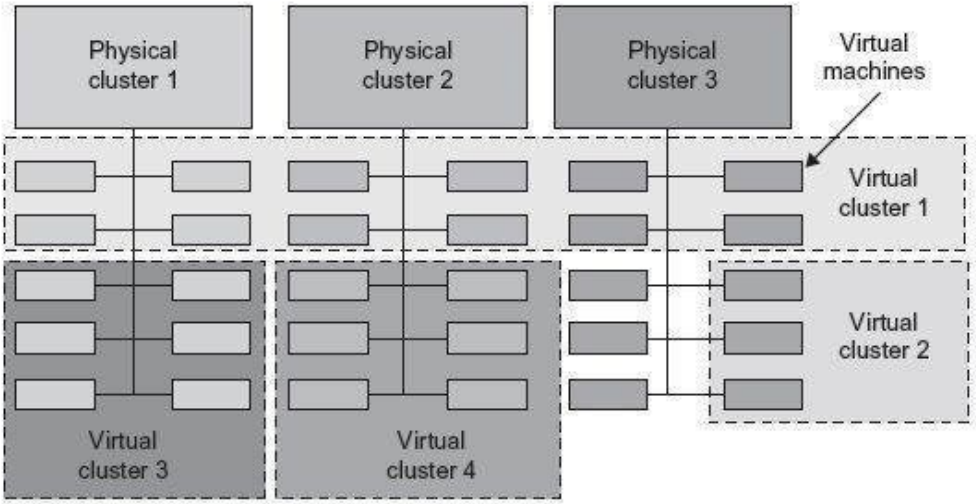


FIGURE 3.18

A cloud platform with four virtual clusters over three physical clusters shaded differently.

(Courtesy of Fan Zhang, Tsinghua University)

Ilustración 3: Estructura de clúster virtuales sobre clúster físicos [6]

Como puede verse en la figura anterior, se describe la estructuración de 4 clústeres virtuales sobre 3 clústeres físicos de servidores mediante la creación de máquinas virtuales en estos últimos conectadas entre ellas por redes de comunicación virtuales. Así se puede llevar a cabo la creación de clúster virtuales más pequeños que cumplan las necesidades específicas para la carga que en ellos se plantee aprovechando de una mejor forma los recursos.

Un ejemplo de esta "adecuación" específica a una determinada carga de trabajo se podría entender mediante la estructura definida en la siguiente imagen. En ella se describe un modelo de conjunto de máquinas virtuales pertenecientes a un clúster virtual y que acceden a determinados recursos. Estas máquinas pueden ser del tipo necesario para cada tarea según lo visto en el apartado **1.2.1 Tipos de Máquinas Virtuales**.

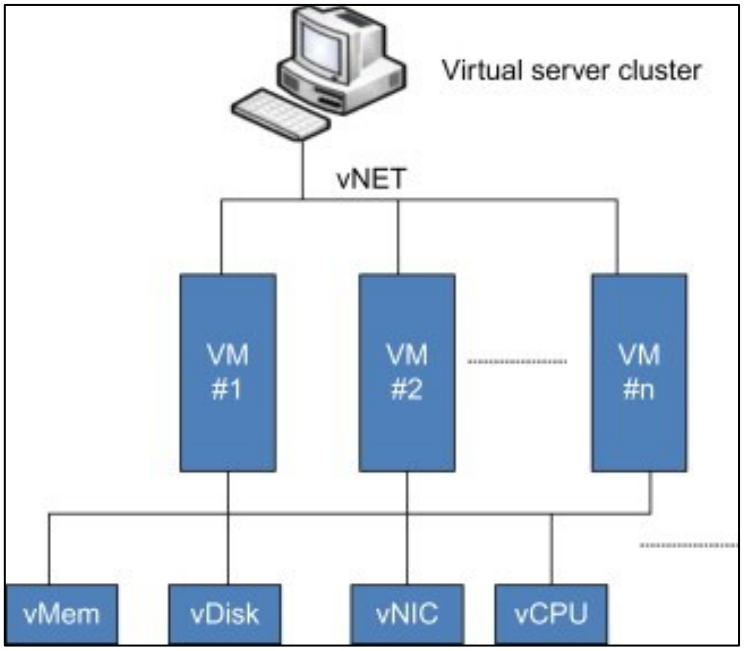


Ilustración 4: Máquinas virtuales en un clúster virtual [5]

Con la siguiente figura obtenemos una imagen completa del estado de las máquinas virtuales asociadas a cada aplicación determinada para la que están destinadas en los distintos hipervisores presentes en los clústeres virtuales.

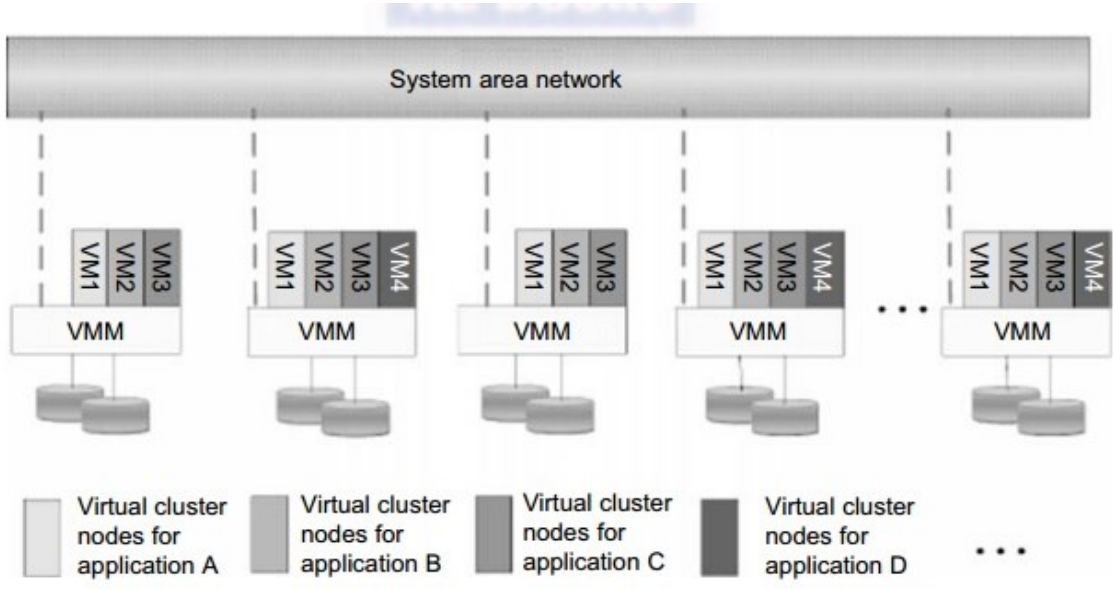


FIGURE 3.19

The concept of a virtual cluster based on application partitioning.

Ilustración 5: Máquinas virtuales en un clúster virtual [7]

1.3.La nube

Tras ver la evolución necesaria que dio lugar al concepto de "nube" y ver en qué consiste esta, en este apartado nos centraremos en los servicios que ofrecen estas y los distintos tipos que hay de ellas.

1.3.1. Qué ofrece

Los principales modelos de servicio que ofrece la computación en la nube son los presentes en la siguiente fotografía:

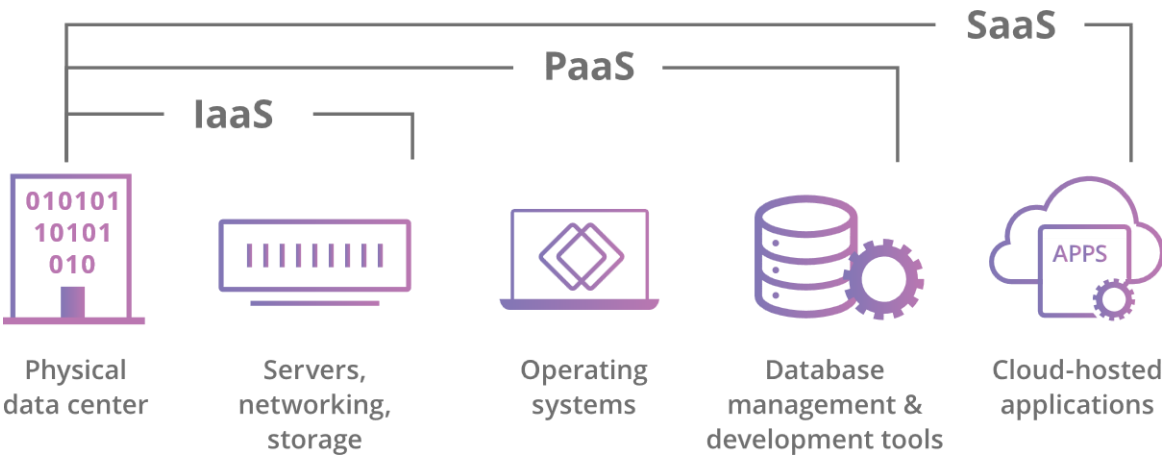


Ilustración 6: Servicios Cloud [1]

A continuación, veremos más en detalle en qué consisten estos servicios provistos por los sistemas de nube:

- *Infrastructure as a Service* (IaaS): este modelo consiste en el alquiler directo de los servidores y espacio de almacenamiento necesario para el desarrollo de las aplicaciones. Se ve la similitud en el hecho de alquilar una parcela para la construcción de un edificio, pero sin las herramientas necesarias, que debe ponerlas el interesado.
- *Plataform as a Service* (PaaS): este modelo se basa no en el alojamiento directo de las aplicaciones ni servicios, sino directamente en el hecho de proporcionar las herramientas necesarias para la creación de estas, sean desde desarrollo de código, infraestructura, bases de datos, etc... Puede encontrarse similitud con el alquiler de las herramientas y maquinaria necesaria para la construcción de una vivienda en sí.
- *Software as a Service* (SaaS): Este modelo aloja en los servidores de la nube lo servicios para que los usuarios accedan a ellos a través de internet. Desde el punto de vista de la utilización, podemos encontrar similitud con el alquiler de una vivienda, en el que el inquilino hace uso de ella sin ser su propietario.
- *Function as a Service* (Faas): Este nuevo modelo de servicio está en crecimiento actualmente y es el más nuevo en cuanto a concepto respecto a los anteriores. Consiste en la división de las aplicaciones en componentes más pequeños de tal forma que estos solo se utilizan cuando son necesarios individualmente, ahorrando así costes por no consumir cuando no son realmente necesarios. Si le buscamos similitud con el ejemplo de la vivienda, sería un hipotético caso en que realmente el alquiler fuese por partes, es decir, no se pagaría por estancias de la vivienda que no estuviesen siendo utilizadas en cada momento.

1.3.2. Tipos de nube

Las nubes se categorizan en función de su propiedad de la siguiente forma:

- Nube publica: nube no dedicada y ofrecida apara múltiples clientes.
- Nube privada: nube dedicada propia.
- Nube hibrida: combinación de nube pública y privada.
- Nube múltiple (multicloud): uso de múltiples nubes.

A continuación, se mencionarán más en detalle las características propias de cada tipo de nube junto con sus ventajas y desventajas.

a) Nube privada

La principal característica de las nubes privadas es el uso exclusivo por parte del cliente directo, sin compartir los recursos con otros.

Podría entenderse fácilmente su esquema de funcionamiento con el símil en que se alquilase o comprase una casa en su totalidad por un único usuario,

Las nubes privadas pueden estar presentes físicamente en dependencias propias de la organización que hace uso de ella o pueden estar siendo accedidas a través de internet en un servicio remoto.

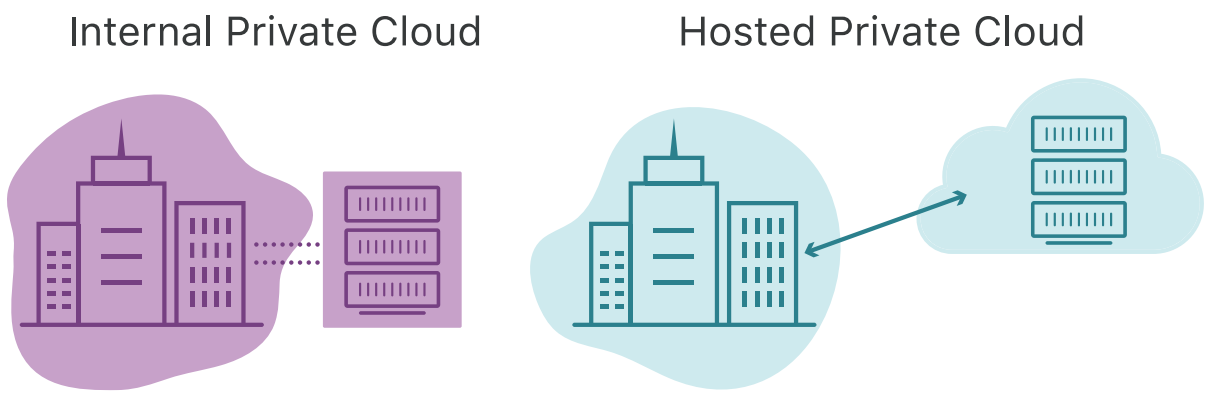


Ilustración 7: Nube privada interna vs remota [8]

Hay que diferenciar el concepto de nube privada interna del concepto de datacenter. Esta diferenciación radica en el uso de virtualización para alcanzar una mayor eficiencia y flexibilidad de la plataforma. Por ejemplo, una nube privada puede estar ubicada físicamente en un datacenter.

Entre las principales **ventajas** de las nubes privadas encontramos:

- Flexibilidad desde el punto de vista de adecuación a las necesidades propias del cliente directo.
- Seguridad, pues únicamente accederán a los recursos y servicios almacenados en ella los usuarios designados por el cliente.
- Escalabilidad posible en función de las necesidades y disponibilidades del propietario de los recursos

Aunque el hecho de contar con una nube privada posee también unas **desventajas** claras:

- Coste, pues el coste es mayor si el cliente que hace uso de esta no comparte los gastos con otros clientes.
- La Administración de la plataforma depende directamente del propio cliente que hace uso de ella
- Escalabilidad. Pese a ser una ventaja, también supone una desventaja la dependencia de la escalabilidad del cliente directamente, pues implica mayor gasto y complejidad en su gestión y depende de los recursos disponibles para llevarla a cabo.

b) Nube publica

Las principales características de las nubes públicas son su acceso a través de internet y su uso compartido con otros clientes que pueden ser ajenos a la organización propia.

El mejor símil para entender el modelo de funcionamiento de las nubes públicas es el de un alquiler de un apartamento compartido con otros inquilinos y todos ellos hacen uso de este a la vez.

Con la siguiente fotografía se puede entender fácilmente la diferencia del esquema de funcionamiento entre la nube pública y privada.

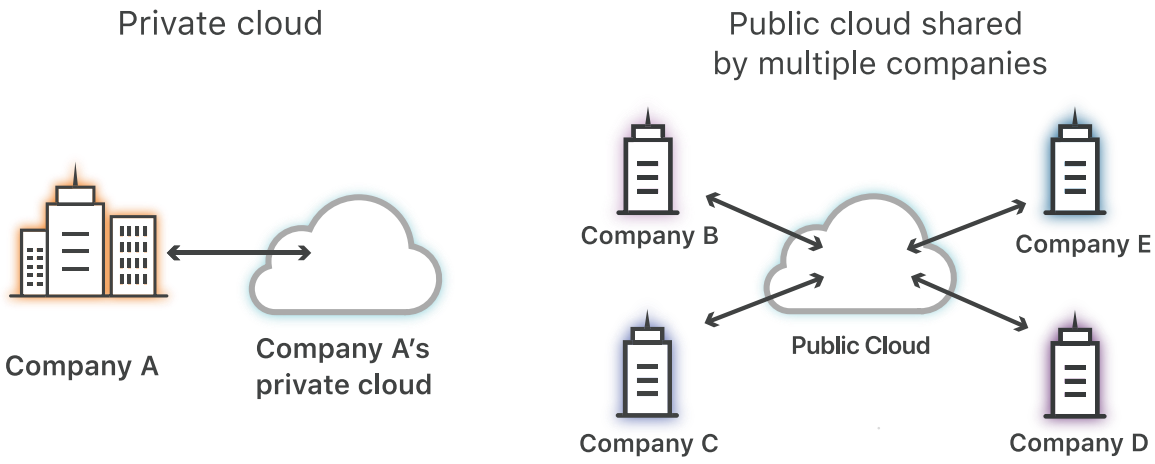


Ilustración 8: Nube pública vs. nube privada [9]

Entre las **ventajas** presentes en el uso de una nube pública podemos encontrar:

- Ahorro en costes
- Ahorro en complejidad de gestión
- Permite escalabilidad y mejora fiabilidad
- Seguridad incrementada

Casi todas las ventajas presentes vienen derivadas de la externalización de la mayor parte de la gestión básica de la plataforma por parte del proveedor de ella.

Así mismo, se aprecian unas claras **desventajas** en este modelo de nube:

- Riesgos de seguridad y conformidad de normativas: El hecho de compartir instancias y servicios con otros clientes ajenos a la organización propia, puede dar lugar a posibles riesgos de seguridad o de fuga de datos.
- Restricciones del proveedor: se depende de las tecnologías ofertadas y escogidas por el proveedor, lo que implica una posible necesidad de adaptación de los productos propios a la plataforma concreta.

c) Nube hibrida

El concepto de nube hibrida viene de la combinación de uso de nube publica con nube privada.

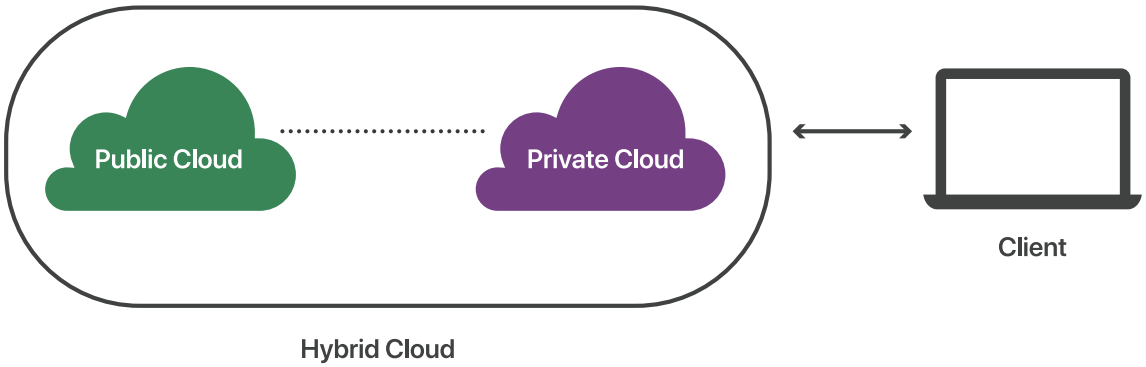


Ilustración 9: Nube hibrida [10]

Podemos encontrar similitud con las tecnologías presentes en los vehículos híbridos, lo que permite una combinación de diferentes tecnologías complementarias para alcanzar una mayor eficiencia y balance en el uso en función de diferentes factores.

El hecho de combinar tecnologías de nube pública y privada reporta una aplica gama de **ventajas**, pues agrupa muchas de ambas tecnologías y soluciona algunas de sus desventajas individuales, como principales **ventajas**, destacaremos las siguientes:

- Flexibilidad en cuanto a la capacidad de hacer uso de ambos tipos de nube simultáneamente en función de los requisitos de cada caso de uso.
- Variedad tecnológica respecto a la combinación de las tecnologías subyacentes propias o ajenas en distintos tipos de nubes y para diferentes necesidades que puedan aparecer.
- Redundancia y Disponibilidad, pues los datos y servicios pueden estar respaldados y repartidos sobre la infraestructura propia y la externa.
- Ahorro potencial en gastos. El hecho de poseer dos tipos de tecnologías distintas de nube permite la diferenciación en su uso en función de factores como el coste temporal de cada una de ellas.
- Seguridad respecto a el aislamiento de los datos en función de las necesidades.

Ahora bien, el hecho de combinar estas tecnologías genera también unas nuevas desventajas, como:

- Mayor superficie de ataque ante amenazas, pues la extensión de la plataforma conjunta es mayor.
- Integración compleja. El uso de estas dos tecnologías distintas de nube implica la necesidad de integrar su funcionamiento para permitir que dicha plataforma funcione correctamente pese a estar dividida en 2 partes.
- Complejidad en la seguridad. Ante la mayor superficie de ataque posible y la dificultad de integración de ambas partes de la plataforma, la gestión de la seguridad se complica.

d) Nube múltiple (multicloud)

El concepto de *multinube* consiste en la agrupación para el uso de distintos proveedores de nube pública.

La utilización de múltiples nubes publicas puede ser a causa de una búsqueda de redundancia y seguridad en los datos del sistema o puede utilizarse como división de los servicios en diferentes proveedores.

El esquema de funcionamiento de un sistema *multinube* se puede entender mediante la siguiente representación:

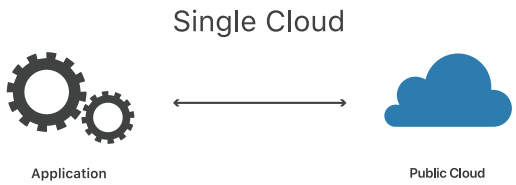


Ilustración 10: Entorno mono-nube [11]

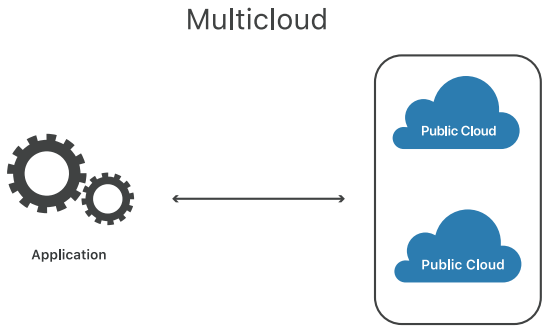


Ilustración 11: Entorno multinube [11]

El uso de un entorno *multinube* provee de una serie de ventajas asociadas a este esquema de combinación de proveedores, pudiendo destacarse las siguientes:

- Redundancia y fiabilidad: el uso de diferentes proveedores permite que se siga dando servicio en caso de fallo en alguna de las nubes siempre que otra esté disponible, además, unas pueden servir como respaldo de los datos de otras.

- Reducción de restricciones del proveedor en cuanto a la posibilidad de utilizar múltiples entornos nube diferentes y no estar "ligado" a uno concreto que pueda exigir unas condiciones más restrictivas o de funcionamiento que otros de ellos.
- Potencial ahorro de costes, permitiendo la elección de aquellos proveedores que mejor se adecuen a las necesidades para dar el servicio deseado y su cambio en función del coste de cada uno de ellos.

Eso sí, existen también unas claras desventajas en el uso de este sistema basado en múltiples proveedores de nube publica:

- Complejidad de administración por la necesidad de interacción entre diferentes proveedores con diferentes tecnologías y servicios.
- Incremento de latencia debido a la posible necesidad de comunicación entre servicios que se encuentran en diferentes proveedores de nube.
- Mayor superficie de ataque debido a la amplitud de la extensión de la plataforma a diferentes proveedores que tengan sistemas de seguridad diferentes.

e) Nube hibrida vs multinube

En este último apartado relativo a los tipos de nube, conviene diferenciar los modelos de nube hibrida y multinube debido a su gran similitud de funcionamiento en cuanto al esquema utilizado.

Los entornos multinube se basan en la utilización de múltiples proveedores de nube públicas, mientras que en los entornos de nube hibrida, se combinan nube privada y nube pública.

Sin embargo, pueden existir casos que se den a la vez ambos modelos, pues puede existir una estructura de plataforma que haga uso de nube privada y de múltiples proveedores de nube publica para su funcionamiento. Puede entenderse mejor con el siguiente esquema:

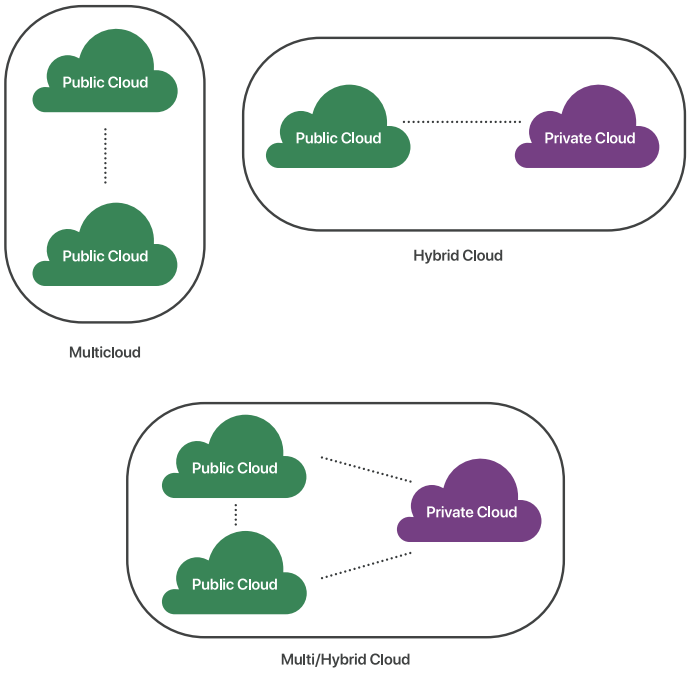


Ilustración 12:Nube híbrida múltiple [10]

2. Contexto Tecnológico

2.1.Introducción

En este apartado es mencionarán las tecnologías elegidas para el desarrollo del proyecto, se describirán y se aportarán las razones y ventajas de su elección para la consecución del proyecto

2.2.VirtualBox [12] [13]

Para el despliegue de la plataforma que se desarrollará, se ha elegido el software de virtualización *VirtualBox* debido a su simplicidad de uso y a que es posible utilizarlo de manera gratuita (suponiendo uso personal o de evaluación). Además, es el software utilizado en el Grado de Ingeniería Informática y el Máster en Ingeniería Informática de las Universidad de Valladolid ambos para las tareas de virtualización de máquinas y sistemas operativos, por lo que su conocimiento es elevado.

Ilustración 13: Logo VirtualBox [14]

VirtualBox permite la creación y configuración de máquinas virtuales "a la carta", es decir, que pueden definirse sistemas a virtualizar con las características hardware que se deseen en cada momento, desde diferentes unidades de disco emuladas (discos duros, SSD, CDs, DVDs, disquetes), hasta configuraciones internas respecto al procesador que va a ser utilizado en la emulación y tarjetas de red.

VirtualBox se instala sobre el sistema operativo de la máquina anfitriona como un programa estándar y se encuentra disponible en versiones para sistemas basados en Windows, Linux, Macintosh y Solaris [15]

Una de las ventajas a destacar de VirtualBox es que permite su conexión con distintas herramientas para la automatización del despliegue y configuración de las máquinas virtuales; en nuestro caso, *Vagrant*, que introduciremos más adelante.

2.3.Docker [16] [17] [18] [19]

Docker es un proyecto de código abierto desarrollado para permitir la automatización en el despliegue de aplicaciones en contenedores software sobre un sistema host. Una de las ventajas principales es que Docker permite aumentar la independencia con respecto a la plataforma hardware subyacente. Es por ello por lo que, al igual que *VirtualBox* fue elegida la tecnología responsable de la infraestructura *hardware* (virtualizada) del proyecto, Docker, ha sido elegida la base de la infraestructura *software* asociada con el despliegue de la propia plataforma que se va a desarrollar.

Ilustración 14: Logo Docker [20]

La base del desarrollo en Docker es la transición desde aplicaciones monolíticas a aplicaciones basadas en microservicios, lo que aumenta la compartimentación de las aplicaciones dotándolas de mayor seguridad y facilitando el desarrollo individualizado de estos servicios sin afectar al resto de ellos en gran medida. También facilita la gestión de errores y fallos al permitir que si falla un determinado servicio no por ello afecte al completo de la aplicación a la que pertenece.

Cuando se despliega cierta infraestructura en Docker, esta tiende a estar formada por varios contenedores y es en estos casos en que la gestión de estos se complica según aumenta el número de ellos, es por ello por lo que aparece la figura del *orquestador*. Los orquestadores son servicios especializados que se encargan de la gestión de los contenedores y su ciclo de vida:

- Gestión de la configuración y automatización de esta
- Automatización del despliegue de servicios basados en contenedores
- Balanceado de carga
- Auto escalado y reinicio de contenedores
- Monitorización del estado de los contenedores (“salud”)
- Servicios de intercambio de datos y redes de comunicación entre contenedores

Para el desarrollo de este proyecto se ha elegido el orquestador *Kubernetes* frente al propio de Docker, *Docker Swarm*. Esta elección está motivada debido a que, si bien el orquestador propio de Docker es el más sencillo y simple de manejar, no es tan potente y versátil como es *Kubernetes*, además de ser este último el más popular en el mercado profesional

2.4.Vagrant [21] [22]

Vagrant es una herramienta diseñada para facilitar la construcción y gestión de entornos de máquinas virtuales de una forma más sencilla enfocada en la automatización [23]. Vagrant funciona en línea de comandos y está disponible para los principales sistemas operativos: Windows, MacOS y GNU/Linux.

Ilustración 15: Logotipo Vagrant [24]

El funcionamiento de Vagrant se basa en un fichero *Vagrantfile* en el que vienen definidas las características propias y configuraciones necesarias a realizar para poder llevar a cabo el despliegue de un conjunto (o unidad) de máquinas virtuales. Este fichero *Vagrantfile* se trata de un fichero en texto plano, por lo que no es necesario de ningún software determinado para su creación o modificación; aunque sí que cuenta con una sintaxis de redacción propia que podría asemejarse a XML por su división en secciones. Esto, a su vez, permite acabar con el problema típico de "en mi máquina funcionaba" al independizar su funcionamiento de la máquina anfitriona en que fue desarrollado.

Por defecto, Vagrant utiliza VirtualBox como plataforma base para el despliegue de las máquinas virtuales, aunque también tiene compatibilidad con otras plataformas locales como *VMware, KVM, Hyper-V, Docker* e incluso con servicios en la nube como *AWS.*

La sencillez de Vagrant radica en la utilización única de un comando para realizar el despliegue una vez definido el fichero *Vagrantfile* con las características necesarias de las máquinas virtuales. Aparte de estas características "físicas" propias de cada máquina virtual, permite definir el sistema de aprovisionado a seguir para la configuración de estas, en este caso, se realizará mediante *Ansible*.

Es por todo esto y por el hecho de haber sido introducido durante el Máster en Ingeniería Informática que es el sistema elegido para realiza el despliegue de la plataforma de este proyecto.

2.5.Ansible [25] [26] [27]

Ansible es una plataforma de software libre utilizada para configurar y administrar sistemas, proceso comúnmente conocido como *aprovisionar* (del inglés *provision*). Permite gestionar de una forma sencilla el despliegue de servicios desde archivos de configuración de origen en los que son definidas las necesidades de configuración para el funcionamiento de ciertos servicios, así como prerrequisitos y dependencias entre ellos.

Ilustración 16: Logotipo Ansible [28]

Ansible permite configurar multitud de nodos conectándose a ellos mediante SSH y únicamente mediante Python, aplicar dichas configuraciones. Estas configuraciones se encuentran definidas en ficheros siguiendo la estructura YAML.

La ventaja de Ansible es que permite aplicar de una forma autónoma al usuario las configuraciones en multitud de máquinas e informa del estado de ejecución en que se encuentra en los distintos momentos de dicha ejecución. Ansible funciona mediante módulos para ejecutar diferentes ordenes simplificando la sintaxis de los archivos YAML.

El funcionamiento se basa en una máquina “controladora” que aplica ciertas configuraciones sobre otras máquinas “nodo” mediante conexión SSH. Y está disponible para sistemas basados en GNU/Linux, pero no Windows; por lo que se utilizará una distribución de Linux para el desarrollo del proyecto, Ubuntu 18.04 en concreto.

El motivo de elegir Ansible como orquestador de aprovisionado para el despliegue de la plataforma es que Vagrant permite definir el medio por el que realizar el aprovisionado de las máquinas a desplegar e incluye la posibilidad de utilizar Ansible como medio externo y este aporta sencillez por permitir definir por separado las configuraciones necesarias a llevar a cabo. Es por esto por lo que, sumado a la experiencia previa por parte del alumno, se opta por incluir Ansible en la cadena de despliegue del proyecto, encargándolo de configurar las máquinas virtuales creadas por Vagrant a través de VirtualBox y así que estas se encuentren correctamente configuradas para la parte final de la cadena: Kubernetes.

2.6.Kubernetes

Ilustración 17: Logotipo Kubernetes [29]

Kubernetes es una plataforma de código abierto diseñada para automatizar la implementación, escalado y administración de contenedores [30]. Así mismo, permite la administración de cargas de trabajo y servicios asociados a estos facilitando la automatización y configuración declarativa [31].

Kubernetes puede ser definido como:

- Plataforma de contenedores
- Plataforma de microservicios
- Plataforma portable de nube

Kubernetes ofrece un entorno de administración centrada en contenedores permitiendo la orquestación de la infraestructura de cómputo, redes y almacenamiento para liberar a los usuarios de su trabajo.

La arquitectura y funcionamiento de Kubernetes se basa en nodos, que contienen *pods* en los que se encuentran uno o más contenedores que realizan un trabajo o servicio.

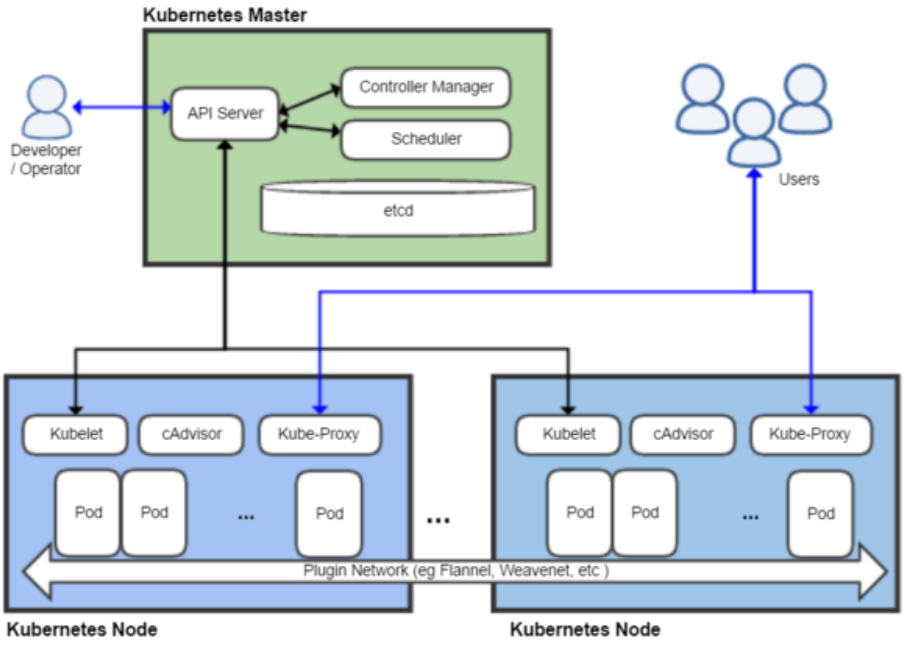


Ilustración 18: Arquitectura de Kubernetes [32]

- Los **nodos** se dividen en 2 tipos principales: el nodo *master* y los nodos *worker*
    - El nodo **máster** se encarga de coordinar el clúster y las actividades que se realizan en este
    - Los diferentes nodos **worker** son en los que se ejecutan las aplicaciones y contenedores.

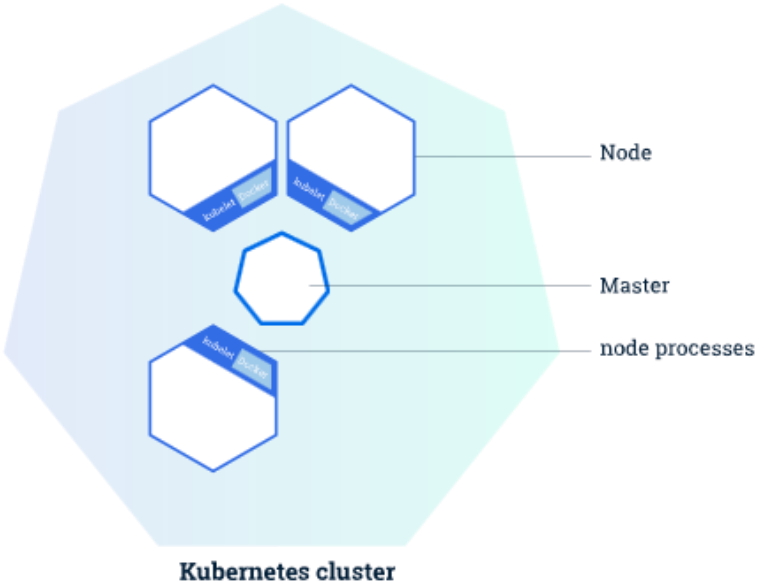


Ilustración 19: Clúster Kubernetes [32]

- Los *pods* contienen los contenedores (uno o más), su almacenamiento y red compartidos y una especificación de cómo ejecutar dichos contenedores.

En terminología de Docker, "*a Pod is similar to a group of Docker containers with shared namespaces and shared filesystem volumes.*" [33]

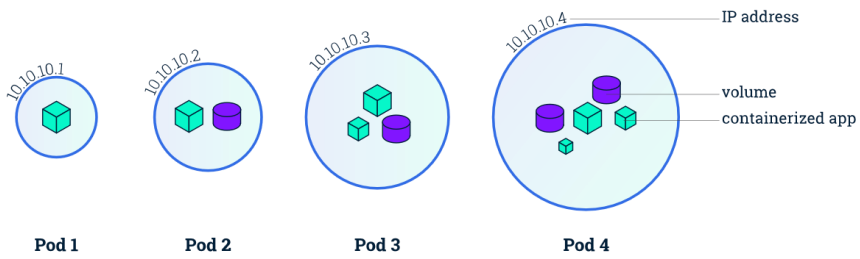


Ilustración 20: Diferentes configuraciones de pods [32]

2.6.1. Auto escalado

Kubernetes ofrece 3 herramientas de escalabilidad. Estas afectan a distintas capas de abstracción de la siguiente manera:

- Capa de infraestructura: *Cluster Autoscaler*. Se encarga de gestionar el escalado de nodos pertenecientes a un clúster cuando detecta que existen *pods* pendientes de ejecución debido a escasez de recursos. Así mismo se encarga de eliminar nodos sobrantes cuando su uso se reduce por debajo de unos márgenes definidos.
- Capa de Aplicación. En esta capa se encuentran:
    - *Horizontal pod autoscaler*, que escala el número de pods que se encuentran en ejecución (dentro de los márgenes definidos) en función de la utilización de estos.
    - *Vertical pod autoscaler*, que modifica las peticiones de recursos de los distintos pods en función de la utilización de estos por dichos pods

Capítulo 3. Plan de Desarrollo

1. Introducción

En este capítulo se llevará a cabo la definición del proceso seguido para la consecución del proyecto, en concreto, la elaboración del plan de desarrollo correspondiente y se explicarán las metodologías y los recursos utilizados, así como la planificación temporal correspondiente al desarrollo del proyecto.

2. Metodología

En este punto se describirá la metodología utilizada para el desarrollo del proyecto. La función base es la aplicación de un proceso de planificación y desarrollo que nos permita desarrollar de manera incremental la funcionalidad de la plataforma. Por esta razón, se decide utilizar el *Proceso Unificado* como referencia para el Plan de Desarrollo y, concretamente, su aplicación para el ámbito educativo: *UPEDU* (*Unified Process for EDUcation).*

2.1. Proceso Unificado

El Proceso Unificado (UP) proporciona un marco de proceso para el desarrollo de software extensible a organizaciones o al completo de proyectos específicos [34].

Las características en las que se basa la idea del Proceso Unificado son las siguientes:

- **Iterativo e incremental:** El UP está compuesto por cuatro fases (inicio, elaboración, construcción y transición). Estas fases, a su vez, se dividen en iteraciones, las cuales ofrecen como resultado cuando finalizan un "incremento" del producto que está siendo desarrollado que añade nuevas funcionalidades (o mejoras) a dicho producto.
- **Dirigido por casos de uso:** los casos de uso son utilizados para capturar los requisitos funcionales del producto.
- **Centrado en la arquitectura:** El UP asume que no existe un modelo único que cubra todos los aspectos del sistema, por lo que existen distintos modelos y vistas que definen la arquitectura del sistema.
- **Enfocado en los riesgos:** el UP requiere que desde una fase temprana del desarrollo del producto se dedique especial atención a la identificación de riesgos

Las fases en las que se divide el UP son:

- Fase de **Inicio**: esta es la primera fase del proyecto, en ella se planea el desarrollo, se definen los objetivos y se acota el alcance del proyecto.
- Fase de **Elaboración**: en esta segunda fase se centra el trabajo en el análisis y estudio de las tecnologías que van a ser empleadas.
- Fase de **Construcción**: en esta tercera fase se crea el producto siguiendo los objetivos definidos y en base a las tecnologías elegidas previamente.
- Fase de **Transición**: en esta última fase se hace la entrega del producto y se corrigen los problemas que aparezcan durante la implementación de este.

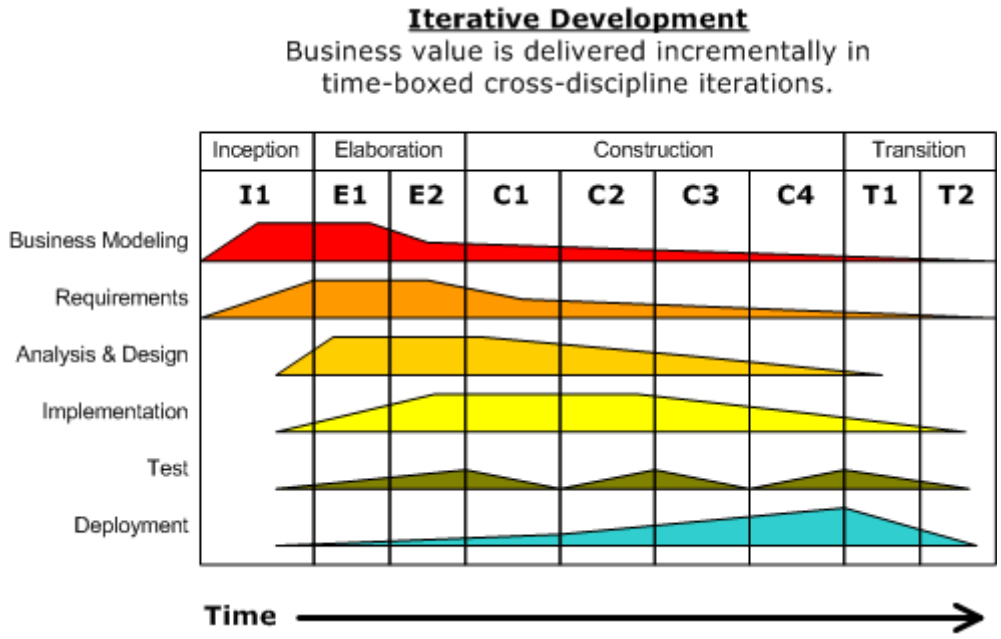


Ilustración 21: Fases del Proceso Unificado [35]

2.2. UPEDU

El Proceso Unificado para la Educación es una implementación basada en el UP destinada a enseñar a los estudiantes las técnicas necesarias para el buen hacer en cuanto a desarrollo de proyectos, por lo que sigue las mismas pautas que el UP, pero simplificando algunos aspectos y siendo menos estricta [36].

Fases

Las fases en que se divide el UPedu son las mismas que en el UP de forma general:

- **Inicial:** en esta fase se concretan los objetivos del ciclo de vida del proyecto. en esta fase se buscan los riesgos que pueden afectar al desarrollo del proyecto y los requisitos que el producto debe de cumplir. Los principales objetivos de esta fase son:
  - Establecer el alcance del software.
  - Discriminar los casos de uso según su importancia en el sistema.
  - Estimar el coste general y el cronograma de desarrollo del proyecto.
  - Estimar los potenciales riesgos que puedan darse.
  - Preparar el entorno de apoyo para el proyecto
- **Elaboración:** en esta fase se proporcionará una base estable para el diseño e implementación y la posterior fase de construcción. en esta fase se definirá la arquitectura a partir de los requisitos y los riesgos implicados.
  Los principales objetivos de esta fase son:
  - Garantizar la estabilidad de la arquitectura y requisitos definidos, así como que los riesgos se encuentren suficientemente mitigados para calcular el coste y cronograma de desarrollo del producto.
  - Abordar los riesgos arquitectónicos
  - Producir un prototipo evolutivo de los componentes.
  - Demostrar que la arquitectura soportará los requisitos del sistema.

- **Construcción:** en esta fase se aclararán los requisitos restantes y se completará la arquitectura básica definida.
  Los principales objetivos de esta fase son:
  - Minimizar los costes de desarrollo optimizando los recursos necesarios y utilizados
  - Lograr la calidad adecuada.
  - Lograr versiones útiles del producto de una forma rápida.
  - Completar el análisis, diseño, desarrollo y pruebas de todas las funcionalidades.
  - Llevar a cabo el desarrollo de manera iterativa e incremental del producto.
  - Decidir si el software y los usuarios están listos para realizar la implementación de la aplicación.
  - Lograr cierto grado de paralelismo entre las distintas tareas que se están desarrollando.

- **Transición:** el objetivo de esta fase es garantizar que el software esté listo para los usuarios finales. En esta fase se encuentran las pruebas del producto final, así como la realización de ajustes menores relacionados a lo comentarios proporcionados por los usuarios que prueban las funcionalidades desarrolladas.
  Los principales objetivos de esta fase son:
  - Probar el producto desarrollado para validar las expectativas del usuario.
  - Entrenar a los usuarios si fuera necesario para el correcto uso de la aplicación.
  - Realizar las actividades de ajuste necesarias en corrección de errores, mejora de rendimiento o usabilidad.

3. Recursos y herramientas para el desarrollo

En este apartado se enumerarán y describirán las principales herramientas utilizadas durante el desarrollo del proyecto, así como los recursos que han sido necesarios para este.

Todas las herramientas utilizadas son gratuitas, aunque con ciertas restricciones de *fair use.*

3.1. Visual Studio Code

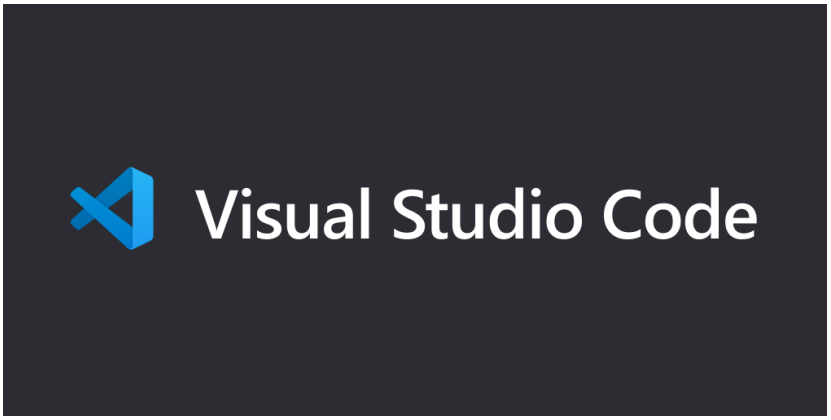


Ilustración 22: Logotipo Visual Studio Code [37]

Visual Studio Code es un editor de texto ligero que se ejecuta en el escritorio de diversos sistemas operativos. Posee soporte incrustado para JavaScript, *TypeScript* y *Node.js*. Además, permite la instalación de extensiones para añadir nueva funcionalidad relacionada con otros lenguajes y entornos de ejecución. Esto lo convierte en un editor de texto muy poderoso ya que, al añadir las extensiones relativas a las tecnologías que se van a usar en este proyecto, permite simplificar el desarrollo de la sintaxis y el funcionamiento en cuanto al control directo desde el propio editor.

3.2. GitHub

Ilustración 23: Logotipo GitHub [38]

Para el control de versiones (y almacenamiento del desarrollo del proyecto) se ha utilizado la herramienta Git, en concreto el repositorio GitHub. Orientado a mantener el desarrollo de un proyecto ordenado en función de las distintas versiones incrementales que se han ido desarrollando [39]

3.3. Documentaciones oficiales

Para la correcta utilización y funcionamiento de las tecnologías escogidas para la elaboración del proyecto, se utilizan las documentaciones oficiales asociadas a estas

3.4. Google Drive

Ilustración 24: Logotipo Google Drive [40]

Para poder realizar el proyecto de una forma "descentralizada" en cuanto a no ser necesario siempre de encontrarse en el mismo sitio, se ha utilizado el servicio de almacenamiento en la nube de Google Drive, que permite compartir y editar documentos en línea sin necesidad de instalar ningún tipo de software para ello [41]

3.5. Recursos Hardware

Los recursos Hardware directos utilizados ha sido un ordenador portátil con las siguientes características:

- Fabricante: Lenovo
- Modelo: Legion y520
- Procesador: Intel Core i7-7700HQ @ 2.80GHz
- Memoria RAM: 16 GB
- Disco Duro: 1TB
- Disco SSD: 250 GB
- Sistema Operativo: Windows 10 + Linux Mint 18.04

Así como un monitor externo conectado para extender el espacio de trabajo.

4. Planificación

4.1. Planificación temporal inicial

El Trabajo de Fin de Máster tiene asignados 6 créditos ECTS, lo que corresponde a una carga de trabajo mínima de 150 horas.

El comienzo de este proyecto se marca para el día 15 de Febrero de 2021 y, como fecha final, se estima el día 7 de Mayo de 2021.

El desarrollo del proyecto se realizará en este periodo de tiempo definido. Debido a la presencia del periodo no lectivo con motivo de la Semana Santa, la duración será de **11 semanas** reales de trabajo estimadas. Para ello, se fijarán sprints de desarrollo de 2 semanas cada uno, resultando en **5 sprints** y una última semana para correcciones y ajustes finales.

Se plantea una carga media de trabajo semanal de **15 horas**, dando como resultado un tiempo final de desarrollo de **165 horas** de trabajo finales.

Semana	Fecha	N.º de Sprint
0	8 Marzo – 14 Marzo	0
1	15 Marzo – 21 Marzo	1
2	22 Marzo – 28 Marzo	1
3	5 Abril – 11 Abril	2
4	12 Abril – 18 Abril	2
5	19 Abril – 25 Abril	3
6	26 Abril – 2 Mayo	3
7	3 Mayo – 9 Mayo	4
8	10 Mayo – 16 Mayo	4
9	17 Mayo – 23 Mayo	5
10	24 Mayo – 30 Mayo	5

Tabla 1: Previsión temporal inicial

4.1.1. Ajuste temporal final

Al concluir el tiempo de desarrollo inicial previsto, se vio necesario ampliar dicho cálculo debido a la influencia de factores externos que afectaron a su desarrollo. Lo que supuso el siguiente ajuste final de tiempo:

Semana	Fechas	N.º de Sprint
11	31 Mayo - 6 Junio	6
12	7 Junio - 13 Junio	6
13	14 Junio - 20 Junio	7
14	21 Junio - 27 Junio	7
15	28 Junio - 4 Julio	n/a

Tabla 2: Ajuste temporal final

Tras la ampliación del tiempo de desarrollo en **5 semanas**, se incrementó el tiempo de desarrollo en **75 horas** extra para poder desarrollar completamente el proyecto para su presentación.

4.2.Análisis de costes inicial

Para hacer el cálculo inicial del coste de desarrollo del proyecto debemos tener en cuenta el coste de personal y el costo derivado de la utilización de los recursos hardware:

- **Personal**

  El coste de personal para el desarrollo es el salario que se debería percibir por realizar dicho desarrollo, es decir, el salario para el desarrollador.

  Para hacer este cálculo tenemos en cuenta que, según la información proporcionada en el portal *Jobted* en [42], el sueldo base para un analista programador es de:

  $$\mathbf{29.600}\ €$$

  (Brutos anuales), por lo que haciendo el cálculo:

  $$29.600\ € \,/\, 12\ meses \,/\, 4\ semanas \,/\, 40\ horas/semana\ =\ \mathbf{15'42}\ €/\boldsymbol{hora}$$

  Por lo que obtenemos un salario aproximado de 15’42 €/h, que redondeamos a **15 €/h**. Sabiendo que el desarrollo está planificado en **165 horas**, calculamos el coste salarial de:

  $$165\ horas\ *\ 15\ €/h\ =\ \mathbf{2.475}\ €$$

- **Hardware**

  El hardware utilizado para el desarrollo ha sido un ordenador portátil con un coste de adquisición de 822 €. Aplicando el coeficiente de amortización lineal máximo, obtenemos:

$$822\ € * 0'25\% = 205'5\ € \ anuales$$
$$205'5€/año * 2'75\ meses\ de\ trabajo/12\ meses = \mathbf{47'10}\ €$$

- **Suministros**

  Debido a la dificultad real de calcular gastos como podrían ser electricidad, conexión a internet, agua, etc., se establece un cálculo aproximado de 25€ mensuales. Por lo que saldría un costo aproximado de:

$$25€ * 2'75\ meses = \mathbf{68'75}€$$

- **Factor multiplicativo**

  Aplicamos un factor multiplicativo para calcular el incremento del gasto previsto asociado a otros gastos relacionados, dando como resultado:

$$2.475€ + 47'1€ + 68'75€ = 2.590'85€$$
$$2590'85€ * 1'18\% = \mathbf{3.057'20}€$$

Por lo que, finalmente, el cálculo para el presupuesto sería de **3.057'20€**

4.2.1. Ajuste de costes final

Como consecuencia de la ampliación temporal realizada sobre la planificada, debemos calcular el incremento del gasto generado.

$$75\ horas * 15€/hora = 1125€$$

$$205'5€/año * 1'25\ meses\ extra/12\ meses = 21'35€$$

$$25€ * 1'25\ meses\ extra = 31'25€$$

Dando un total de:

$$1.125€ + 21'35€ + 31'25€ = \mathbf{1.177'6}€$$

Y, aplicando el factor multiplicativo:

$$1.177'6€ * 1'18\% = 1389'57€$$

Esto significa que el coste final de desarrollo del proyecto es de:

$$3.057'2€ + 1389'57€ = \mathbf{4.446'77}€$$

4.3.Análisis de Riesgos

Se considera un riesgo todo aquel acontecimiento presente o futuro que afecte directamente de forma negativa en el desarrollo de un proyecto.

La gestión de los riesgos es un aspecto muy importante que tener en cuenta durante el desarrollo de un proyecto debido a que, la aparición de estos puede condicionar el correcto avance y resultar del mismo.

Los responsables del proyecto han de ser capaces de anteponerse a los riesgos que puedan surgir. Para ello deberán ser identificados y evaluados teniendo en cuenta su posible impacto para poder desarrollar un plan de acción a seguir en caso de que ocurra cualquiera de ellos.

El **plan de gestión de riesgos** nos permitirá definir con antelación las acciones a realizar ante los riesgos que puedan surgir durante el desarrollo del proyecto.

El proceso de la gestión de riesgos incluye las siguientes etapas:

- Identificación de los riesgos: nos permite obtener la lista de riesgos potenciales
- Análisis de riesgos: con ello podemos realizar una priorización de los posibles riesgos
- Planificación de riesgos: nos permite realizar planes de mitigación y contingencia ante los riesgos.
- Evaluación de riesgos: nos permite valorar los riesgos que podrían llegar a darse.

Los riesgos pueden ser clasificados en función del **impacto** y su **probabilidad** de que ocurran, con ello podemos construir la llamada "Matriz de riesgos", que nos aporta una visión del carácter de importancia que tiene cada riesgo para nuestro proyecto de la siguiente forma:

Impacto / Probabilidad	**Bajo**	**Medio**	**Alto**
Baja	Bajo	Bajo	Medio
Media	Bajo	Medio	Alto
Alta	Medio	Alto	Alto

Tabla 3: Matriz de Riesgos

Se prestará especial atención a los riesgos de carácter ***Alto*** y el resto de los riesgos serán monitorizados para seguir su evolución.

R-01	Planificación temporal errónea
Descripción	Retraso en el desarrollo debido a una estimación incorrecta de la planificación temporal
Probabilidad	Alta
Impacto	Alto
Plan de mitigación	Tener en cuenta los posibles retrasos al asignar el tiempo para las tareas, dejando margen ante estos posibles imprevistos
Plan de contingencia	Aumentar el tiempo trabajo diario hasta alcanzar la planificación original y, en caso necesario, ampliar el tiempo final de desarrollo

Tabla 4: R-01 Planificación temporal errónea

R-02	Requisitos poco/mal definidos
Descripción	Los requisitos definidos no son lo suficientemente concretos, lo que puede ocasionar fallos en la definición del alcance
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Una vez definidos los requisitos, estos volverán a ser analizados para ver si se encuentran correctamente acotados.
Plan de contingencia	Realizar una reevaluación de los requisitos definidos previamente, redefinirlos si fuera necesario y ajustar la planificación ante dicho cambio.

Tabla 5: R-02 Rquisitos poco/mal definidos

R-03	Cambio en los requisitos
Descripción	Durante el desarrollo se produce un cambio (o añadido) en un requisito frente a cómo era inicialmente
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Realizar un análisis de requisitos inicial exhaustivo.
Plan de contingencia	Realizar un análisis directo de dicho requisito en búsqueda del alcance de influencia que pueda tener sobre el desarrollo y el resto de los requisitos

Tabla 6: R-03 Cambio en los requisitos

R-04	Alcance del proyecto mal definido
Descripción	El alcance está mal definido o no es viable
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Evaluar en cada Sprint la evolución del proyecto para detectar dicho fallo en la definición del alcance
Plan de contingencia	Realizar una reevaluación del alcance del proyecto y definir correctamente el alcance analizado.

Tabla 7: R-04 Alcance del Proyecto mal definido

R-05	Pérdida de datos
Descripción	Pérdida de documentación o código elaborado
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Utilizar control de versiones para mantener la evolución del código y su correcta conservación. Además, todos aquellos recursos documentales que se estén utilizando mantenerlos correctamente guardados mediante sistemas de replicación o en la nube.
Plan de contingencia	Evaluar el alcance de la pérdida de los datos, recuperar lo último guardado, analizar la pérdida real y replanificar el proyecto en relación con el nuevo tiempo necesario para rehacer dicho trabajo restante

Tabla 8: R-05 Pérdida de datos

R-06	Enfermedad
Descripción	Se produce una disminución en el progreso del proyecto debido a una enfermedad
Probabilidad	Media[1]
Impacto	Medio
Plan de mitigación	N/A
Plan de contingencia	Replanificar el proyecto en función del tiempo perdido por la enfermedad

Tabla 9: R-06 Enfermedad

R-07	Conocimientos insuficientes
Descripción	La falta de conocimientos específicos acerca de las tecnologías que van a ser usadas (herramientas, procedimientos, etc.…), junto con la falta de experiencia en la elaboración de proyectos, pueden provocar un mayor número de errores, retrasos e incremento en los costes
Probabilidad	Media
Impacto	Alto
Plan de mitigación	Realizar investigación previa de las tecnologías y herramientas que van a utilizarse en el desarrollo del proyecto
Plan de contingencia	Aumentar el tiempo destinado a formación

Tabla 10: R-07 Conocimientos insuficientes

[1] Debido a la pandemia por la enfermedad COVID-19, causada por el virus SARS-CoV-2, se incrementa la Probabilidad y el Impacto posible del riesgo por Enfermedad.

R-08	Problemas hardware/software
Descripción	Fallos en los recursos físicos o software, puede producir retrasos en el desarrollo en función del tiempo necesario para reparar dichos fallos
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	Realizar mantenimiento activo de equipos y copias de seguridad de todos los recursos necesarios para el desarrollo del proyecto
Plan de contingencia	Sustituir o reparar el elemento determinado, tener un equipo de reserva para realizar el proyecto

Tabla 11: R-08 Problemas Hardware

R-09	Fallo en los servicios de Almacenamiento (GitHub, Google Drive, …)
Descripción	Los servicios de almacenamiento tienen un fallo y se produce una incomunicación con ellos, impidiendo el acceso a los recursos almacenados en ellos
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	Llevar a cabo copias del desarrollo en otros servicios (local, remoto o nube) de los recursos necesarios para el avance del proyecto.
Plan de contingencia	Continuar el desarrollo utilizando los servicios de reserva, si fuese imposible se dedicará el tiempo a la revisión completa del trabajo ya realizado y el proyecto.

Tabla 12: R-09 Fallo en los servicios de Almacenamiento

4.3.1. Seguimiento de los riesgos

Las situaciones de riesgo que, finalmente, se materializaron fueron las siguientes:

- **R-01 Planificación temporal errónea**
  Este era el riesgo más probable que podía ocurrir y por ello se le puso como el primero de la lista. Al haber sido bien analizado se pudo corregir rápidamente el problema ampliando el tiempo de desarrollo del proyecto.

- **R-06 Enfermedad**
  El desarrollo del proyecto se vio afectado por motivos de enfermedad. Esto derivó finalmente en una necesidad de ampliación en el tiempo requerido para la finalización del desarrollo
- **R-07 Conocimientos insuficientes**
  Debido a la dificultad que tenía el desarrollo del proyecto relacionado con las tecnologías utilizadas, concretamente Kubernetes, en varios momentos se apreció que no se poseía el conocimiento necesario para continuar el desarrollo de ciertos elementos, por lo que se tuvo que pausar dicho desarrollo hasta adquirir dicho conocimiento para continuarlo.

Al ser riesgos que fueron previamente analizados y elaborados sus planes de mitigación y contingencia, la respuesta ante ellos fue directa, lo que evitó un impacto mayor en el desarrollo del proyecto.

5. Requisitos

El principal objetivo de este TFM es la definición de una metodología de trabajo que provea el estudio previo necesario para el desarrollo de una plataforma que permita desplegar de la forma más automática y autónoma posible un clúster Kubernetes [31] poblado de servicios alojado a su vez en un conjunto de nodos virtuales utilizando la tecnología VirtualBox [15]. Además, se buscará conseguir que dicha plataforma sea elástica tanto a nivel de escalado en los procesos como en los propios nodos en función de la demanda de recursos que haya en determinados momentos de mayor o menor demanda por los usuarios y los propios servicios alojados.

Para ello se definen los siguientes subobjetivos:

- Establecer una metodología que permita el despliegue de las **máquinas virtuales** necesarias para alojar el clúster Kubernetes. Para ello, se analizarán las tecnologías que permitan desplegar máquinas virtuales de forma **automática** en función de las necesidades definidas.

- Estudiar el funcionamiento de las herramientas que permiten el despliegue de una **plataforma Kubernetes**. Como se espera la presencia de más de un único nodo que conforme el clúster, esto será tenido en cuenta a la hora de la elección.

- Desplegar **servicios** sencillos que permitan mostrar el uso de la plataforma en una situación de uso real de está, escogiendo o desarrollando dichos servicios.

- Estudiar las tecnologías de **escalado** presentes para la plataforma que se va a desarrollar e implantarlas para demostrar su funcionamiento.

- **Automatizar** al máximo posible el **despliegue** y configuración de la plataforma al completo haciendo uso de las tecnologías que permitan su integración con las ya presentes para facilitar el ciclo de despliegue.

- Ejecutar **pruebas** sobre la plataforma para comprobar el correcto funcionamiento de esta ante distintas situaciones.

- Utilizar tecnologías que permitan que el despliegue de la plataforma no sea dependiente de la estructura física utilizada, alcanzando así **independencia** de la máquina base original en que fue desarrollada

Capítulo 4. Desarrollo y Pruebas

1. Un caso práctico

En este apartado aplicaremos lo visto con anterioridad a un pequeño caso práctico de muestra del potencial de las tecnologías mencionadas.

El objetivo será desplegar un entorno conformado por 3 máquinas virtuales que conformarán un clúster de trabajo Kubernetes y una serie de servicios para dotarlo de funcionalidad que nos permita comprobar su correcto funcionamiento y las funciones de auto escalado definidas.

Nota: *se recomienda leer este apartado previo a la ejecución de los comandos mencionados a no ser indicado lo contrario.*

1.1. Prerrequisitos

En este apartado se mencionarán los prerrequisitos necesarios para la ejecución del caso práctico que desarrollaremos. La instalación y configuración de los servicios requeridos puede realizarse de forma autónoma mediante los scripts provistos si así se desea.

Debido a las restricciones del software utilizado, será necesario que la máquina anfitrión esté basada en ***GNU-Linux***, para la prueba del correcto funcionamiento se ha utilizado ***Ubuntu 18.04***

Será necesario tener instalado y correctamente configurado en la máquina anfitrión los siguientes servicios:

- VirtualBox
- Vagrant
- Ansible

La instalación de estos servicios requeridos se puede realizar de forma autónoma mediante el uso del script *./config/setup.sh*, o, directamente, mediante el comando:

```
> ./config/setup.sh
```

Como servicio **opcional** extra, se propone la instalación de Kubernetes en la máquina anfitrión para facilitar los despliegues finales de servicios y no tener que conectarse manualmente a la máquina *mánager* del clúster Kubernetes final. Se podrá instalar de forma autónoma mediante el script bash shell: *./config/kube.sh*, el cual es posible ejecutar mediante el comando:

```
> ./config/kube.sh
```

2. Arquitectura base: "Hardware"

En este apartado veremos la definición y configuraciones que se llevarán a cabo para la puesta en marcha de las máquinas virtuales que crearán nuestra infraestructura de despliegue del clúster Kubernetes y los servicios deseados.

La configuración de este apartado será la utilizada por el servicio Vagrant para el despliegue y provisionado de las máquinas virtuales en VirtualBox y está presente en el archivo: *./kube_vagrant/Vagrantfile*

2.1. Recursos virtuales

En primer lugar, definiremos los recursos que serán asignados a cada máquina virtual por VirtualBox. En este caso, se decide que las máquinas poseerán:

- 2 Cores virtuales de CPU
- 1024 MB de memoria RAM

Esta configuración puede ser modificada cambiando el fragmento de código:

```
config.vm.provider "virtualbox" do |v|
        v.memory = 1024
        v.cpus = 2
    end
```

2.2. Configuración de máquinas virtuales

En segundo lugar, se definirá la configuración interna de las máquinas virtuales. En este momento comenzará ya la distinción entre la máquina *"máster"*, que será la encargada de gestionar el clúster Kubernetes; y las máquinas *"nodo"*, que serán las encargadas de conformar la fuerza de cómputo del clúster.

Como primera definición, tenemos la imagen del sistema operativo a utilizar en dichas máquinas. Se ha elegido una distribución *Ubuntu 18.04*, que será obtenida de los repositorios oficiales de forma automática por Vagrant:

```
IMAGE_NAME = "ubuntu/bionic64"
```

A continuación, se llevará a cabo la definición de la configuración de las propias máquinas de la siguiente forma:

- **Máquina "máster"**: esta máquina virtual es única y se encargará de crear y mantener el clúster Kubernetes y su funcionamiento. Se definirá la imagen de SO a utilizar, su dirección IP de red y su *hostname*. Así mismo, en el caso del máster del clúster, se le otorgará mayor asignación de memoria RAM dadas sus mayores necesidades.

```
config.vm.define "master" do |master|
        master.vm.box = IMAGE_NAME
        master.vm.network "private_network", ip: "192.168.50.10"
        master.vm.hostname = "master"
        master.vm.provider "virtualbox" do |v|
            v.name = "master"
```

```
            v.memory = 2048
            v.cpus = 2
        end
```

- **Máquinas "nodo"**: estas máquinas se encargarán de ejecutar los servicios desplegados en el clúster Kubernetes y pueden ser entre 1 y N. Para el caso base se crearán 2 de ellas y se configurarán en bucle para facilitar su asignación de nombre y dirección IP.

```
N = 2
...
(1..N).each do |i|
        config.vm.define "node-#{i}" do |node|
            node.vm.box = IMAGE_NAME
            node.vm.network "private_network", ip: "192.168.50.#{i + 10}"
            node.vm.hostname = "node-#{i}"
            ...
        end
    end
```

2.3. Provisionado

Se entiende como proceso de provisionado el conjunto de acciones realizadas para la adquisición y configuración de servicios y recursos necesarios en el desempeño de las tareas previstas para las máquinas virtuales.

En la definición de la configuración de ambos tipos de máquinas, también irán incluidas las directrices de provisionado de estas de la siguiente forma:

- Nodo máster

```
...
master.vm.provision "ansible" do |ansible|
            ansible.playbook = "kubernetes_setup/master-playbook.yml"
            ansible.extra_vars = {
                node_ip: "192.168.50.10",
            }
        End
...
```

- Nodos genéricos

```
...
node.vm.provision "ansible" do |ansible|
                ansible.playbook = "Kubernetes_setup/node-playbook.yml"
                ansible.extra_vars = {
                    node_ip: "192.168.50.#{i + 10}",
                }
            End
...
```

En estas definiciones se hace referencia a 2 playbooks diferentes en función de si se trata de una máquina máster o una nodo. Estos playbooks contienen la definición de configuraciones necesarias para llevar a cabo por Ansible en las diferentes máquinas y se encuentran presentes en el directorio ./kube_vagrant/kubernetes_setup/

El contenido de estos playbooks lo categorizaremos en función de su contenido en el siguiente apartado.

3. Arquitectura base: "Software"

En este apartado veremos la configuración que será llevada a cabo en cada una de las máquinas virtuales durante el proceso de provisionamiento de estas. Estas configuraciones están definidas en los playbooks correspondientes en función del tipo de máquina (máquina *máster* o máquinas *nodo*).

Estos *playbooks* se encuentran presentes en el directorio: *./kube_vagrant/kubernetes_setup/*. Y las configuraciones llevadas a cabo engloban:

- Instalación de Docker y sus componentes
- Deshabilitar el swap
- Instalar Kubernetes y sus componentes
- Inicializar y configurar el clúster Kubernetes (o unirse al clúster en el caso de los nodos)
- En último lugar se configura un manejador para confirmar el correcto funcionamiento de Docker y finalizar el proceso de provisionado.

3.1.Configuraciones comunes:

En este apartado se mencionarán las configuraciones que se llevarán a cabo en todas las máquinas ya sean estas nodos máster o nodos normales, son configuraciones comunes a ambos tipos. Podemos agruparlas de la siguiente forma:

3.1.1. Docker y sus componentes

A continuación, se muestran las configuraciones necesarias para instalar Docker junto con sus dependencias y requisitos: Instalación de paquetes, configuración de claves y paquetes propios de Docker

```
# Install Docker & components
  - name: Install packages that allow apt to be used over HTTPS
...
      packages:
      - apt-transport-https
      - ca-certificates
      - curl
      - gnupg-agent
      - software-properties-common

  - name: Add an apt signing key for Docker
    apt_key:
      url: https://download.docker.com/linux/ubuntu/gpg
      state: present

  - name: Add apt repository for stable version
    ...

  - name: Install docker and its dependencies
...
      packages:
      - docker-ce
      - docker-ce-cli
```

```
      - containerd.io
- name: Add vagrant user to docker group
    user:
      name: vagrant
      group: docker
```

3.1.2. Deshabilitar Swap

Kubernetes necesita que las máquinas en que está instalado para el despliegue de servicios, no tenga activado el espacio swap, por ello se desactiva y elimina de la lista de particiones a montar al inicio de las máquinas.

```
# Disable swap
  - name: Remove swapfile from /etc/fstab
    mount:
      name: "{{ item }}"
      fstype: swap
      state: absent
    with_items:
      - swap
      - none

  - name: Disable swap
    command: swapoff -a
    when: ansible_swaptotal_mb > 0
```

3.1.3. Instalación de Kubernetes y sus componentes

Para la instalación de Kubernetes y sus componentes es necesario llevar a cabo una serie de configuraciones entre las que se incluyen el añadir los repositorios de paquetes para poder descargar los paquetes de instalación de Kubernetes, la instalación de estos, la configuración de las IPs internas de los nodos y reiniciar el servicio de Kubernetes para que comience a ejecutarse correctamente:

```
# Install Kubernetes & components

# Install kubelet, kubeadm and kubectl
  - name: Add an apt signing key for Kubernetes
    apt_key:
      url: https://packages.cloud.google.com/apt/doc/apt-key.gpg
      ...

  - name: Adding apt repository for Kubernetes
    apt_repository:
      repo: deb https://apt.kubernetes.io/ kubernetes-xenial main
      ...

  - name: Install Kubernetes binaries
    apt:
      ...
```

```
      packages:
        - kubelet
        - kubeadm
        - kubectl

  - name: Configure node ip
    ...

  - name: Restart kubelet
    service:
      name: kubelet
      daemon_reload: yes
      state: restarted
```

3.2. Configuraciones del nodo máster

3.2.1. Creación del Clúster Kubernetes

La creación del clúster Kubernetes se lleva a cabo en el denominado como nodo máster. Para ello, son necesarias varias configuraciones: crear el clúster, otorgar permisos de acceso, crear una red interna para el clúster (se utilizará Calico), generar el comando para unir nuevos nodos al clúster y copiar dicho comando a la máquina anfitrión.

```
# Initialize the Kubernetes cluster
  - name: Initialize the Kubernetes cluster using kubeadm
    command: kubeadm init --apiserver-advertise-address="192.168.50.10" --
apiserver-cert-extra-sans="192.168.50.10"  --node-name k8s-master --pod-
network-cidr=192.168.0.0/16
```

3.2.2. Permisos de acceso

Al utilizar Vagrant para la gestión de las máquinas virtuales, el usuario por defecto del sistema es el usuario Vagrant. Por esta razón, hay que modificar los permisos de la configuración de Kubernetes para permitir su acceso a este a los comandos y configuración del clúster

```
# Setup the kube config file for the vagrant user to access Kubernetes cluster
  - name: Setup kubeconfig for vagrant user
    ...
     - mkdir -p /home/vagrant/.kube
     - cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config
     - chown vagrant:vagrant /home/vagrant/.kube/config
```

3.2.3. Red interna del cluster

Al desplegar posteriormente los servicios en el clúster Kubernetes que estamos creando, necesitaremos una red interna software que permita la comunicación entre ellos y con ellos desde el exterior de dicho clúster. Para ello haremos uso del servicio Calico, que provee una red de comunicación entre servicios y elementos en Kubernetes. El despliegue de Calico se hará directamente a través de Kubernetes como si de un servicio genérico se tratase:

```
# Setup the container networking provider and the network policy engine
  - name: Install calico pod network
    ...
    command: kubectl create -
f https://docs.projectcalico.org/manifests/calico.yaml
```

3.2.4. Unión de nodos al clúster

Tras las configuraciones realizadas, ya tenemos listo el clúster en Kubernetes con el nodo máster como único miembro de este. Para poder unir nuevos nodos al clúster necesitamos exportar el comando de unión de nodos, que contiene la clave de acceso al clúster que acabamos de crear y configurar. Este proceso se hace a través de Ansible, que copiará la salida del comando ejecutado en la máquina virtual máster en nuestra máquina local en el directorio ./kube_vagrant/kubernetes_setup/

```
# Generate kube join command
  - name: Generate join command
    command: kubeadm token create --print-join-command
    ...
  - name: Copy join command to local file
    ...
      dest: ./join-command
```

3.2.5. Autocompletado de comandos

En caso de que se quiera operar directamente sobre el clúster Kubernetes desde la máquina virtual máster, se configura el autocompletado de comandos para facilitar las tareas.

```
# Configure kubectl autocomplete
  - name: Configure kubectl autocomplete
    shell: echo "source <(kubectl completion bash)" >> ~/.bashrc
  - name: Copy cluster config to local host
    fetch:
      src: /home/vagrant/.kube/config
      dest: ~/.kube/
      flat: yes
```

3.2.6. Docker handler

Como ultima tarea, se crea un manejador que comprobará el estado del servicio Docker en la máquina en espera de que se encuentre correctamente funcionando para finalizar las configuraciones.

```
# Docker handler

# Handler for Docker daemon
  handlers:
    - name: docker status
      service: name=docker state=started
```

3.3.Configuración de los nodos

La configuración especifica propia de las máquinas virtuales "nodo" consiste únicamente en su unión al clúster Kubernetes que ha sido creado por la máquina virtual máster. Se realiza de la siguiente forma, utilizando el comando previamente guardado durante la creación del clúster en el nodo máster:

```
# Join to the Kubernetes cluster

# Join Kubernetes cluster
  - name: Copy the join command to server location
    copy: src=join-command dest=/tmp/join-command.sh mode=0777

  - name: Join the node to cluster
    command: sh /tmp/join-command.sh
```

3.3.1. Docker handler

De igual manera que en el nodo máster, se crea un manejador para comprobar el estado de ejecución de Docker.

```
# Docker handler

# Handler for Docker daemon
  handlers:
    - name: docker status
      service: name=docker state=started
```

4. Despliegue de la plataforma

En este apartado se verá la forma correcta para llevar a cabo el despliegue de la plataforma desarrollada. Para iniciar el despliegue, se debe ejecutar el siguiente comando en el directorio *./kube_vagrant/* , presente en los archivos relativos al desarrollo de este TFM, así como el resto de los comandos salvo indicación contraria:

```
> vagrant up
```

Durante el despliegue se irán mostrando en pantalla las distintas etapas por las que pasa el despliegue, indicando qué se realiza en cada una de ellas y el resultado de estas si es correcto o ha ocurrido algún error. Puede verse un ejemplo en la siguiente imagen:

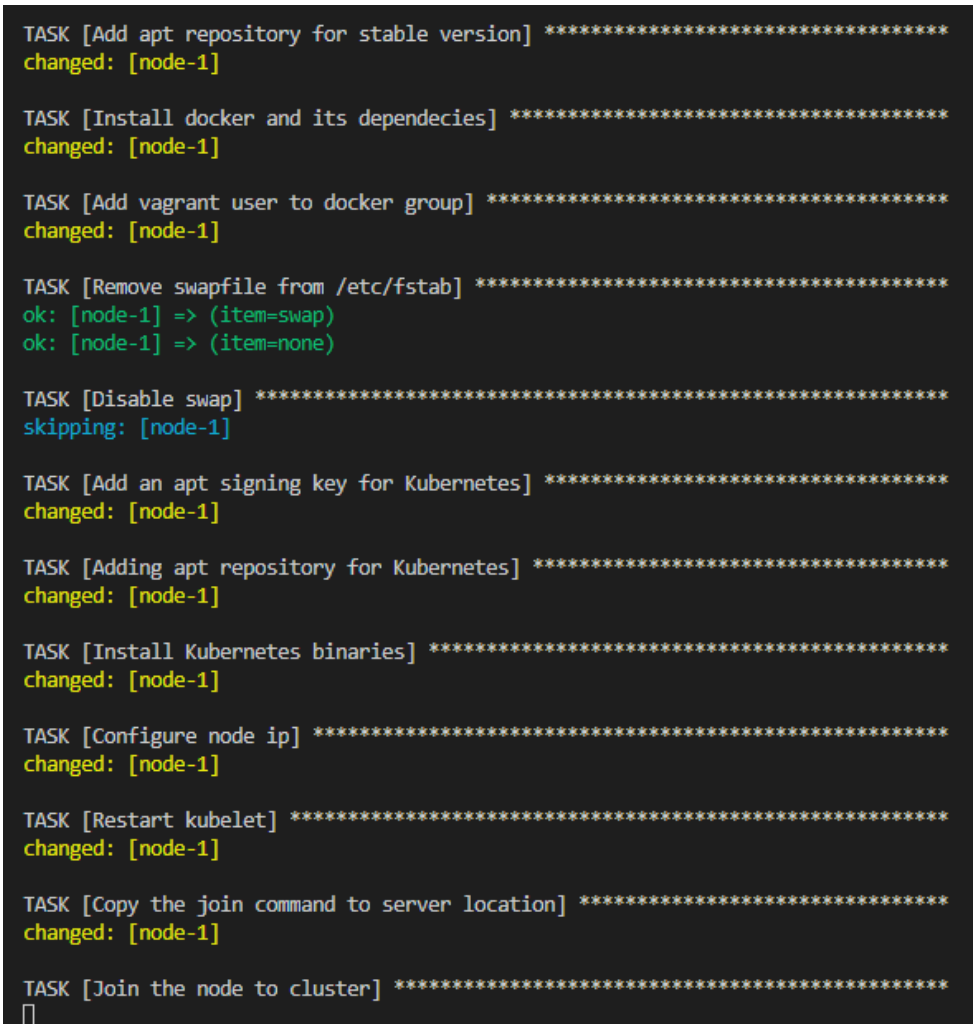


Ilustración 25: Ejemplo fases de despliegue

Tras la finalización del proceso de despliegue de la plataforma, se mostrará el resultado final resumen de las etapas. Podemos ver un ejemplo de dicho resultado final en el apartado de pruebas correspondiente a la prueba de despliegue correcto de la plataforma.

4.1. Añadir un nuevo nodo extra

Para añadir un nuevo nodo a la plataforma y que este se añada correctamente al clúster conformado en Kubernetes por los demás integrantes, se debe ejecutar el siguiente comando:

> *vagrant up /node-**X**/*

Donde **X** es el número de designación del nuevo nodo deseado, que debe ser uno no ya existente en el clúster (por defecto, el despliegue base de la plataforma crea el nodo *master*, *node-1* y *node-2*).

4.2. Destrucción de la plataforma

Para eliminar la plataforma completamente, se puede hacer ejecutando en el directorio ./kube_vagrant/ el comando:

> *vagrant destroy [-f]*

El opcional "*-f*" se añade en caso de no querer que se solicite confirmación individual para la eliminación de cada máquina virtual desplegada para la plataforma.

Durante este proceso de eliminación de los nodos virtuales, *Vagrant*, irá mostrando en pantalla las acciones realizadas y finalmente indicará que todos los nodos han sido eliminados.

4.2.1. Eliminar un único nodo

Se puede desear, en un momento determinado, eliminar un único nodo de la plataforma. Para ello, el comando a ejecutar es el siguiente:

> *vagrant destroy /node-**X**/*

Donde **X** es el número del nodo que se desea eliminar.

5. Despliegue de servicios

En este apartado se mostrará las tareas necesarias para realizar el despliegue de un servicio funcional sobre la plataforma para emular un caso de uso real de esta.

Para demostrar el funcionamiento en un caso "real" de un servicio a desplegar en la plataforma, se desplegará un servicio consistente en un servidor apache-php que realiza una operación de alta carga de CPU. Esto es así para luego poder comprobar el correcto funcionamiento del sistema de autoescalado de servicios en función de la demanda de recursos de estos.

Esta imagen Docker consistente en un servidor php-apache que realizará unos cálculos intensivos en CPU tal como se indica en la siguiente imagen:

```php
<?php
  $x = 0.0001;
  for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
  }
  echo "OK!";
?>
```

Ilustración 26: Carga de CPU en php

Esta imagen Docker se puede obtener de los repositorios oficiales de contenedores para Docker de Google en [43]

El despliegue del pod que contiene esta imagen Docker se hará mediante el archivo de configuración *./service/php-apache.yaml*

Este archivo se encuentra dividido en 2 partes:

- **Definición del despliegue del pod.** Aquí se describirán las características y requisitos que son necesarios por el pod para su correcta ejecución y del contenedor interno.

```yaml
...
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  ...
    spec:
      containers:
      - name: php-apache
        image: k8s.gcr.io/hpa-example
        ports:
        - containerPort: 80
```

```
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

Cabe destacar el requisito de 1 única réplica del pod ejecutándose y la imagen del contenedor Docker que contiene. Así como sus requisitos mínimo disponible y máximo de uso de CPU.

- **Definición del servicio.** Este servicio permitirá el acceso al pod para su uso externo

```
apiVersion: v1
kind: Service
...
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
  type: NodePort
```

En este caso, conviene destacar la necesidad de especificar el nombre del despliegue de pods al que va asociado el servicio y el puerto que este usará para permitir la comunicación

Para realizar el despliegue del servicio al completo, se ejecutará el siguiente comando desde el propio terminal local o conectándose al nodo máster mediante el comando:

> *kubectl apply -f ./service/php-apache.yaml*

O, si se ejecuta desde el propio nodo máster del clúster (al que se puede acceder mediante el comando: >*vagrant ssh master*) :

> *kubectl apply -f /Vagrant/service/php-apache.yaml*

5.1.Comprobación del correcto despliegue

Para comprobar que el despliegue del servicio y el pod ha sido correcto, se ejecutará el siguiente comando en el terminal local o en un terminal del nodo *master* del clúster virtual:

> *kubectl get pods -o wide*

El resultado del comando, en caso de que haya sido correcto el despliegue, mostrará algo similar a la siguiente imagen, pudiendo variar el nodo en que se encuentra ubicado, entre otros detalles menores:

```
vagrant@master:~$ kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE     IP              NODE     NOMINATED NODE   READINESS GATES
php-apache-d4cf67d68-5917z   1/1     Running   0          3m23s   192.168.247.1   node-2   <none>           <none>
```

Ilustración 27: Despliegue del servicio php-apache

6. Autoescalado de recursos

Uno de los objetivos era demostrar el funcionamiento del sistema de escalado en función de las necesidades de recursos que permite la plataforma a los servicios alojados en ella. Para poder comprobar este funcionamiento, utilizaremos el servicio que fue desplegado anteriormente, ya que, al hacer una petición a este, se ejecuta un cálculo que hace un alto uso de CPU.

El primer paso será indicarle al clúster Kubernetes que nuestro servicio debe estar disponible para realizar el escalado horizontal de los pods relacionados. Sin embargo, previamente es necesario que el servicio propio de Kubernetes de métricas esté desplegado y funcionando para monitorizar el clúster y los pods que se encuentren en él. Esto se realiza mediante el uso del siguiente comando:

```
> kubectl apply -f service metrics-server-deployment.yaml
```

Una vez desplegado el servidor de métricas, podemos definir el autoescalado de nuestro servicio *php-apache* ejecutando el siguiente comando:

```
> kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

Desglosando el comando:

- `kubectl autoscale` . Indica a Kubernetes que se trata de un comando para la creación de un autoescalado
- `deployment php-apache` indica el nombre del despliegue al que se relaciona el autoescalado.
- `--cpu-percent=50` indica el nivel máximo de uso medio de CPU que se desea entre todas las posibles réplicas del servicio
- `--min=1 --max=10` indica el número mínimo y máximo de réplicas deseadas en el clúster del despliegue objetivo

Puede comprobarse el estado del autoescalado definido mediante la ejecución del siguiente comando:

```
> kubectl get hpa
```

El resultado que se mostrará será algo similar a lo siguiente, donde cabe destacar principalmente el objetivo de uso medio de CPU y los números mínimo y máximo, así como actual, de réplicas del servicio en el clúster:

```
vagrant@master:/vagrant/kubernetes_setup$ kubectl get hpa
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          3m4s
```

Ilustración 28: Inicio del autoescalado

7. Pruebas

Tras los capítulos anteriores en que se ha definido la metodología y objetivos buscados para el desarrollo del proyecto consistente en la plataforma nube y, en el apartado anterior, haber mostrado el desarrollo final realizado y el procedimiento a seguir para replicar su correcto despliegue, en este apartado se diseñaran y realizaran una serie de pruebas relativas a demostrar el correcto funcionamiento de la plataforma en cuanto a la adecuación con los objetivos que se buscaba cumplir sobre ella y para demostrar su potencial.

Al ser uno de los objetivos el conseguir que el desarrollo y despliegue de la plataforma fuese independiente de la máquina anfitriona sobre la que se realizase, las pruebas se ejecutarán sobre dos anfitriones diferentes para comprobarlo. Es importante recordar que la plataforma debe desplegarse sobre un equipo con Sistema Operativo basado en Linux.

7.1. Máquinas anfitrionas

Para comprobar el correcto funcionamiento de la plataforma en distintas máquinas anfitrionas, se realizará en 2 equipos diferentes propiedad del alumno con las siguientes características:

- Equipo (1), sobremesa:
  - CPU: Intel Core i7-2600k
  - RAM: 16 GB 1333 MHz
  - Sistema Operativo: Ubuntu 18.04
- Equipo (2), portátil:
  - CPU: Intel Core i7-7700HQ
  - RAM: 16 GB 2400 MHz
  - Sistema Operativo: Xubuntu 18.04

7.2. Despliegue

Para evaluar que el despliegue se hace de forma correcta en ambas máquinas anfitrionas se utilizará como prueba las salidas por consola concernientes al provisionador Ansible en cada nodo virtual, que no debe contener ningún error. Como dato referencial se calculará el tiempo que tarda dicho despliegue.

La estructura de resultados se mostrará tras la ejecución del comando base para el despliegue de la plataforma "`> vagrant up`"

- **Equipo 1:**

```
PLAY RECAP *********************************************************************
master                     : ok=20   changed=17   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

PLAY RECAP *********************************************************************
node-1                     : ok=15   changed=12   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

PLAY RECAP *********************************************************************
node-2                     : ok=15   changed=12   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

```
real    7m42,081s
user    0m59,768s
sys     0m15,072s
```

Ilustración 29: Resultados despliegue Equipo 1

Observamos que el despliegue se ha realizado de forma correcta en el primer equipo y ha tardado cerca de 8 minutos en completar el proceso.

- **Equipo 2:**

```
PLAY RECAP *********************************************************************
master                     : ok=20   changed=17   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

PLAY RECAP *********************************************************************
node-1                     : ok=15   changed=12   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

PLAY RECAP *********************************************************************
node-2                     : ok=15   changed=12   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

```
real    15m36,958s
user    1m57,989s
sys     0m32,163s
```

Ilustración 30: Resultados despliegue Equipo 2

Podemos observar que la plataforma se ha desplegado correctamente en el segundo equipo anfitrión y ha tardado, aproximadamente, 15 minutos.

7.3. Escalado Horizontal

Para demostrar el funcionamiento del sistema de escalado automático que hemos definido previamente, deberemos hacer múltiples peticiones concurrentes para aumentar la carga del servicio y observar cómo se crean nuevas réplicas en el clúster para dar servicio a dichas peticiones.

Debemos generar la carga sobre este servicio para comprobar su escalado en busca de alcanzar el objetivo de uso medio de CPU de las réplicas. El método que se utilizará para realizar esta prueba consiste en "inundar" de peticiones http el servicio para que cada una deba ser procesada y se incremente masivamente el uso de CPU.

La forma de realizar estas peticiones será mediante la ejecución de una imagen *busybox* en Docker que ejecute un bucle continuo de peticiones web sobre nuestro servicio. El comando para realizar dicha ejecución del pod correspondiente es el siguiente:

```
> kubectl run -i --tty load-generator --rm --image=busybox --restart=Never --
/bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

Donde se le indicará a Kubernetes:

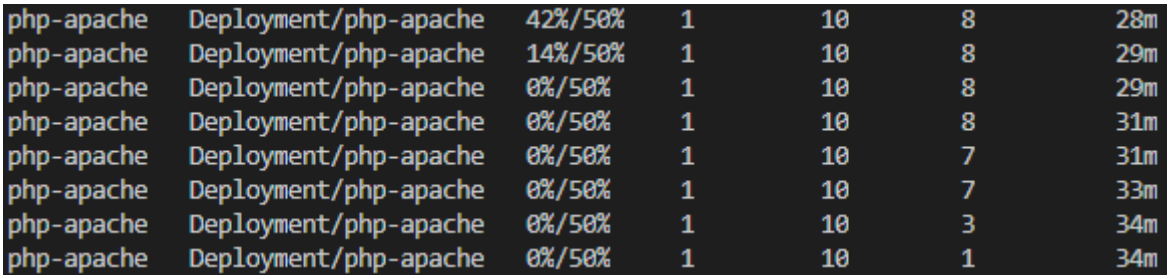
- `kubectl run -i --tty load-generator –rm`: Que ejecute un pod con el nombre de *"load-generator"* en un terminal interactivo y lo elimine cuando se salga de dicho terminal
- `--image=busybox --restart=Never`: la imagen que contendrá dicho pod será una imagen busybox y no la reiniciará en caso de fallo
- `-- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"`: Ejecutará un commando en dicho terminal que consistirá en un bucle infinito de peticiones *http*

Tras ejecutar el comando, podemos volver a solicitar el estado del autoescalado con el comando `> kubectl get hpa -w` y nos mostrará algo similar a lo siguiente, en donde podremos observar que la carga media está incrementándose y, con el tiempo, esta carga media bajará según se incremente el número de réplicas en funcionamiento:

```
vagrant@master:/vagrant/kubernetes_setup$ kubectl get hpa -w
NAME         REFERENCE                     TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          8m19s
php-apache   Deployment/php-apache   48%/50%   1         10        1          10m
php-apache   Deployment/php-apache   246%/50%  1         10        1          10m
php-apache   Deployment/php-apache   202%/50%  1         10        4          10m
php-apache   Deployment/php-apache   96%/50%   1         10        5          10m
php-apache   Deployment/php-apache   65%/50%   1         10        5          11m
php-apache   Deployment/php-apache   91%/50%   1         10        5          11m
php-apache   Deployment/php-apache   90%/50%   1         10        8          11m
php-apache   Deployment/php-apache   55%/50%   1         10        8          11m
php-apache   Deployment/php-apache   41%/50%   1         10        8          12m
php-apache   Deployment/php-apache   41%/50%   1         10        8          12m
```

Ilustración 31: Autoescalado en funcionamiento

Si cerramos el terminal que está realizando el bucle de peticiones al servicio, podemos ver que el escalado se invierte debido a la infrautilización de recursos, permitiendo así ahorrar estos y volviendo al mínimo que habíamos definido:



php-apache	Deployment/php-apache	42%/50%	1	10	8	28m
php-apache	Deployment/php-apache	14%/50%	1	10	8	29m
php-apache	Deployment/php-apache	0%/50%	1	10	8	29m
php-apache	Deployment/php-apache	0%/50%	1	10	8	31m
php-apache	Deployment/php-apache	0%/50%	1	10	7	31m
php-apache	Deployment/php-apache	0%/50%	1	10	7	33m
php-apache	Deployment/php-apache	0%/50%	1	10	3	34m
php-apache	Deployment/php-apache	0%/50%	1	10	1	34m

Ilustración 32: Autoescalado hacia abajo de recursos

Capítulo 5. Conclusiones

1. Introducción

En este capítulo se presentan las conclusiones obtenidas tras la finalización del desarrollo del proyecto y las posibles líneas de trabajo futuro a seguir para la plataforma

2. Conclusiones

Tras finalizar el desarrollo de este proyecto enmarcado dentro del Trabajo de Fin de Máster del Máster en Ingeniería Informática, se puede ver cómo ha servido el estudio de dicho grado en la formación necesaria para el desarrollo de proyectos complejos siguiendo un plan previamente establecido y la investigación en nuevas tecnologías emergentes para su uso presente y capacidad futura.

De este proceso de desarrollo se extrae la importancia de realizar un estudio y planificación previos sobre los objetivos a desarrollar y alcanzar para que, ante la aparición de problemas o inconvenientes durante dicho desarrollo, se pueda alcanzar una solución más sencilla y rápidamente que en caso de no existir dicho plan. No definir correctamente el alcance del proyecto era un riesgo muy importante debido a que la tecnología central utilizada, *Kubernetes*, es una tecnología bastante compleja y aun no ampliamente utilizada y conocida en el mundo tecnológico relacionado, por lo que la documentación, aunque creciente, es escasa fuera de las fuentes oficiales y, al ser una tecnología en constante desarrollo, dichas documentaciones son desactualizadas con relativa facilidad ante los grandes cambios que está sufriendo dicha tecnología. El desarrollo de una plataforma completamente funcional y su despliegue en "producción", como se suele decir, es una tarea cuasi imposible de realizar en el tiempo concerniente a este TFM, 150 horas, por lo que el objetivo final de este proyecto era demostrar un caso de uso básico para entender su funcionamiento de cara a una posible ampliación de dicha plataforma si fuera necesario o para que sirva de ejemplo a quienes quieran introducirse en el uso de dicha tecnología en concreto o, de forma general, en el uso de técnicas de despliegue automático y plataformas automatizadas en la gestión de recursos y servicios.

El desarrollo de esta plataforma ha servido para entender la importancia de que exista una documentación asociada fuerte y concisa acerca de las tecnologías que se van a usar y, en este caso concreto, relacionada con el trabajo directo realizado

Respecto a la planificación conviene destacar la "facilidad" de realizarla a priori de un desarrollo en el caso de que todo se dé correctamente y como debiera según esta. La dificultad radica realmente en el momento real de seguirla y que aparezcan grandes imprevistos como los relativos a la pandemia asociados al virus COVID-19, más información en [44].

3. Trabajo Futuro

En esta sección se tratarán la líneas de desarrollo posible futuro acerca de ampliar las funcionalidades y alcance del proyecto de la plataforma.

- **Ahorro y huella energética:** como primera línea, me gustaría destacar uno de los principales beneficios que tiene este tipo de tecnologías utilizadas, de forma indirecta al uso relacionado propiamente con la plataforma; y es el hecho de que los sistemas de autoescalado como el presente en *Kubernetes*, y utilizado como uno de los casos de uso a desarrollar en este proyecto, permiten que en momentos de menor necesidad de uso de recursos, los servicios, sean reducidos al mínimo, permitiendo así reducir a su vez el uso energético de los sistemas físicos asociados a la plataforma y, por ello, la reducción el impacto y huella ecológica relacionada de forma directa con dicho sistema. Cabe destacar el artículo [45], que menciona las posibilidades que abre el uso de una tecnología como *Kubernetes* de cara a la reducción del carbono generado en su uso y del uso de la capacidad de no ser una tecnología dependiente de la máquina física en que se encuentra para su despliegue en diferentes regiones en que los recursos energéticos sean diferentes en cuanto a coste y medio de producción asociado. Así como también se mencionan los problemas que podría enfrentarse en los diferentes casos de aplicar este principio al despliegue.

- **Ampliación de la plataforma:** La plataforma podría ampliarse con el desarrollo de nuevos servicios de despliegue automatizados para simular mejor el uso real que podría darse en ella y, relacionado con esto:

- **Migración a proveedores nube:** se puede estudiar la migración de la plataforma a servicios proveedores de recursos en la nube como podrían ser AWS [46], Azure [47], GCP [48], entre otros. Lo que permitiría la integración con el resto de los servicios que estos ofrecen como, por ejemplo, podría activarse el módulo de *Kubernetes* que permite el escalado directamente de los nodos virtuales de forma tanto horizontal como vertical en ellos.

Referencias

[1] Cloudflare, «¿Qué es la nube?,» [En línea]. Available: https://www.cloudflare.com/es-es/learning/cloud/what-is-the-cloud/. [Último acceso: 19 05 2021].

[2] Wikipedia, «Virtualization,» [En línea]. Available: https://en.wikipedia.org/wiki/Virtualization. [Último acceso: 19 05 2021].

[3] IBM Cloud, «Virtual Machines (VMs),» [En línea]. Available: https://www.ibm.com/cloud/learn/virtual-machines#toc-types-of-v-L8PmXYJx. [Último acceso: 2020 04 20].

[4] Wikipedia, «Clúster de Computadoras,» [En línea]. Available: https://es.wikipedia.org/wiki/Cl%C3%BAster_de_computadoras. [Último acceso: 26 05 2021].

[5] ScienceDirect, «Virtual Cluster,» Elsevier, [En línea]. Available: https://www.sciencedirect.com/topics/computer-science/virtual-cluster. [Último acceso: 26 Mayo 2021].

[6] Search Networking, «Distributed and Cloud Computing: Keep virtual clusters manageable,» TechTarget, Noviembre 2011. [En línea]. Available: https://searchnetworking.techtarget.com/feature/Distributed-and-Cloud-Computing-Keep-virtual-clusters-manageable. [Último acceso: 26 Mayo 2021].

[7] Brainkart.com, «Virtual Clusters and Resource Management,» [En línea]. Available: https://www.brainkart.com/article/Virtual-Clusters-and-Resource-Management_11343/. [Último acceso: 26 Mayo 2021].

[8] Cloudflare, «What Is a Private Cloud? | Private Cloud vs. Public Cloud,» [En línea]. Available: https://www.cloudflare.com/learning/cloud/what-is-a-private-cloud/. [Último acceso: 04 Mayo 2021].

[9] Cloudflare, «What Is a Public Cloud? | Public vs. Private Cloud,» [En línea]. Available: https://www.cloudflare.com/learning/cloud/what-is-a-public-cloud/. [Último acceso: 04 Mayo 2021].

[10] Cloudflare, «What Is Hybrid Cloud? | Hybrid Cloud Definition,» [En línea]. Available: https://www.cloudflare.com/learning/cloud/what-is-hybrid-cloud/. [Último acceso: 04 05 2020].

[11] Cloudflare, «What Is Multicloud? | Multicloud Definition,» [En línea]. Available: https://www.cloudflare.com/learning/cloud/what-is-multicloud/. [Último acceso: 04 Mayo 2021].

[12] Oracle, «Documentation - Oracle VM VirtualBox,» [En línea]. Available: https://www.virtualbox.org/wiki/Documentation. [Último acceso: 07 01 2021].

[13] Wikipedia, «VirtualBox,» [En línea]. Available: https://en.wikipedia.org/wiki/VirtualBox. [Último acceso: 07 01 2021].

[14] Oracle Corporation, «Virtualbox Logo,» 30 Septiembre 2010. [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:Virtualbox_logo.png. [Último acceso: 16 Febrero 2021].

[15] Oracle, «Oracle VM VirtualBox,» Oracle, [En línea]. Available: https://www.virtualbox.org/. [Último acceso: 07 01 2021].

[16] Docker Company, «Docker,» [En línea]. Available: https://www.docker.com/. [Último acceso: 2021 Enero 29].

[17] T. Rodríguez, «De Docker a Kubernetes: entendiendo qué son los contenedores y por qué es una de las mayores revoluciones de la industria del desarrollo,» Xataka, 10 Septiembre 2019. [En línea]. Available: https://www.xataka.com/otros/docker-a-kubernetes-entendiendo-que-contenedores-que-mayores-revoluciones-industria-desarrollo. [Último acceso: 2021 Enero 29].

[18] Red Hat, «What is Docker?,» [En línea]. Available: https://www.redhat.com/en/topics/containers/what-is-docker. [Último acceso: 2021 01 29].

[19] campusMVP, «Las 10 herramientas más importantes para orquestación de contenedores Docker,» 7 Agosto 2018. [En línea]. Available: https://www.campusmvp.es/recursos/post/las-10-herramientas-mas-importantes-para-orquestacion-de-contenedores-docker.aspx. [Último acceso: 29 Enero 2021].

[20] Docker, «Docker Logos,» Docker, [En línea]. Available: https://www.docker.com/company/newsroom/media-resources. [Último acceso: 14 Febrero 2021].

[21] Wikipedia, «Vagrant (Software),» [En línea]. Available: https://en.wikipedia.org/wiki/Vagrant_(software). [Último acceso: 06 Enero 2020].

[22] M. Pintor, «Vagrant, la herramienta para crear entornos de desarrollo reproducibles,» Conasa ITWorks, 04 Febrero 2015. [En línea]. Available: https://www.conasa.es/blog/vagrant-la-herramienta-para-crear-entornos-de-desarrollo-reproducibles/. [Último acceso: 06 Enero 2021].

[23] HashiCorp, «Introduction to Vagrant,» [En línea]. Available: https://www.vagrantup.com/intro. [Último acceso: 06 Enero 2021].

[24] Vagrant, «Vagrant Logo,» Hashicorp, [En línea]. Available: https://www.vagrantup.com/img/logo-hashicorp.svg. [Último acceso: 16 Febrero 2021].

[25] Wikipedia, «Ansible (Software),» [En línea]. Available: https://en.wikipedia.org/wiki/Ansible_(software). [Último acceso: 05 Abril 2021].

[26] Red Hat Ansible, «Ansible,» [En línea]. Available: https://www.ansible.com/. [Último acceso: 05 Abril 2021].

[27] Ansible, «Ansible Documentation,» Red Hat, [En línea]. Available: https://docs.ansible.com/ansible/latest/index.html. [Último acceso: 05 Abril 2021].

[28] gilbarbara, «Ansible Logo,» 15 Julio 2015. [En línea]. Available: https://github.com/gilbarbara/logos/blob/master/logos/ansible.svg. [Último acceso: 14 Febrero 2021].

[29] Wikipedia, «Kubernetes Logo,» Wikimedia, 7 Octubre 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:Kubernetes_logo.svg. [Último acceso: 14 Febrero 2021].

[30] Kubernetes, «Kubernetes,» [En línea]. Available: https://kubernetes.io/es/. [Último acceso: 15 Enero 2021].

[31] Kubernetes, «¿Qué es Kubernetes?,» 16 Junio 2020. [En línea]. Available: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/. [Último acceso: 17 Enero 2021].

[32] Cloud-DI Team - Departamento de Informática, «Kubernetes. Un orquestador de contenedores que debes poner en tu vida,» Universidad de Almería, [En línea]. Available: https://ualmtorres.github.io/SeminarioKubernetes/. [Último acceso: 17 09 2020].

[33] Kubernetes, «Pods - What is a Pod,» 10 9 2020. [En línea]. Available: https://kubernetes.io/docs/concepts/workloads/pods/#what-is-a-pod. [Último acceso: 17 9 2020].

[34] BAwiki, "THE UNIFIED SOFTWARE DEVELOPMENT PROCESS," [Online]. Available: http://www.bawiki.com/wiki/Unified-Process.html. [Accessed 26 Noviembre 2019].

[35] Wikipedia, «Unified Process,» [En línea]. Available: https://en.wikipedia.org/wiki/Unified_Process. [Último acceso: 27 Noviembre 2019].

[36] P. N. Robillard, P. Kruchten y P. d'Astous, «YOOPEEDOO (UPEDU): A Process for Teaching Software Process,» 19-21 Febrero 2001.

[37] Microsoft, «Visual Studio Code,» Visual Studio, [En línea]. Available: https://code.visualstudio.com/. [Último acceso: 28 Abril 2021].

[38] Github, «GitHub Logos and Usage,» [En línea]. Available: https://github.com/logos. [Último acceso: 20 Abril 2021].

[39] Wikipedia, «GitHub,» [En línea]. Available: https://en.wikipedia.org/wiki/GitHub. [Último acceso: 20 Abril 2021].

[40] Google, «Google Drive Logo,» [En línea]. Available: https://www.google.com/drive/static/images/drive/logo-drive.png. [Último acceso: 20 Abril 2021].

[41] Google, «Google Drive,» [En línea]. Available: https://www.google.com/intl/es_ALL/drive/. [Último acceso: 20 Abril 2021].

[42] Jobted, «Sueldo del Analista Programador en España,» [En línea]. Available: https://www.jobted.es/salario/analista-programador#:~:text=Sueldo%20del%20Analista%20Programador%20en%20Espa%C3%B1a&text=El%20salario%20medio%20de%20un,salario%20medio%20anual%20en%20Espa%C3%B1a.. [Último acceso: 28 Abril 2021].

[43] Google Cloud, «Container Registry,» Google, [En línea]. Available: k8s.gcr.io/hpa-example. [Último acceso: 23 06 2021].

[44] WHO, «Brote de enfermedad por coronavirus (COVID-19),» [En línea]. Available: https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019. [Último acceso: 2021 Julio 15].

[45] B. Johnson, «Carbon-Aware Kubernetes,» Microsoft, 05 Octubre 2020. [En línea]. Available: https://devblogs.microsoft.com/sustainable-software/carbon-aware-kubernetes/. [Último acceso: 2021 Julio 17].

[46] AWS, «Amazon Web Services,» Amazon, [En línea]. Available: https://aws.amazon.com/. [Último acceso: 01 Julio 2021].

[47] Microsoft Azure, «Microsoft Azure,» Microsoft, [En línea]. Available: https://azure.microsoft.com/es-es/. [Último acceso: 01 Julio 2021].

[48] Google Cloud, «Google Cloud Plataform,» Google, [En línea]. Available: https://cloud.google.com/. [Último acceso: 01 Julio 2021].

Anexo I: Contenido del Trabajo

En el desarrollo de este TFM se incluyen los siguientes recursos:

- Memoria con el contenido del plan de proyecto (este documento)
- Archivo comprimido con el desarrollo de la plataforma propuesta con las siguientes subcarpetas:
    - *Config:* con las configuraciones previas necesarias para instalación de programas requeridos en la máquina anfitriona
    - *Kube_vagrant* con el contenido de los archivos necesarios para el despliegue de la plataforma
        - *Kubernetes_setup*
        - *Service*
        - *Vagrantfile*

Todos estos recursos se encuentran presentes en el repositorio alojado en el servicio GitHub en: https://github.com/Odd10/TFM_Cloud