uc3m | Universidad Carlos III de Madrid

Grado en ingeniería telemática

curso 2017-2018

Trabajo Fin de Grado

Orquestación de contenedores con Kubernetes

Javier Nogués García

Tutor:

Carlos Jesús Bernardos

Director:

Pablo Serrano Yañez-Mingot

15 de octubre de 2018, Leganés

Resumen

Actualmente se exige mucho a los sistemas de prestación de servicios en la nube, por lo que para optimizar los recursos y atender a la demanda actual de los usuarios se ha llegado a disponer de virtualización ligera con contenedores, que virtualizan en una máquina física otra máquina entera con solo con lo necesario, pudiendo generarse así múltiples servidores. Además, los usuarios exigen que los servicios estén disponibles en todo momento y que los estos cada vez sean mejores. Lo que conlleva actualizaciones periódicas y que los servicios estén disponibles independientemente de la cantidad de gente que quiera acceder a la vez, es decir, es necesario gestionar la carga de peticiones de los servicios repartiéndolas entre varios servidores del mismo servicio, en sistemas descentralizados.

Por lo que el objetivo del proyecto se centra en el estudio de Kubernetes como tecnología orquestadora de contenedores, que nos ofrece: balance de carga para gestionar las peticiones, escalado de las aplicaciones para atender a más demanda, actualización de las aplicaciones con control de versiones y sin interrumpir el servicio, y auto-reparación de los elementos del sistema para que el sistema siempre funcione. Para el análisis de las características de Kubernetes que cubren las necesidades que motivan el proyecto se ha creado un escenario de Kubernetes en nube pública, creando un clúster a través de la plataforma de Google (GCP), y se han desplegado dos servidores, web y de hora, configurándolos para probar Kubernetes.

La conclusión ha sido satisfactoria, dando como resultado que Kubernetes cubre las necesidades de prestación de servicios que motivan este proyecto, con una pequeña excepción ya que se ha detectado interrupción del servicio al actualizar aplicaciones, teniendo por defecto configurada una pequeña demora para esta acción.

Abstract

The evolution of the necessities in the provision of cloud storage services in the world has triggered the search of new technologies which can adapt to these demands.

In the past, providing several services of the same kind required the use of programs in charge of managing the requests about the service provider, in order to function properly, which can cause complicated configurations. Furthermore, among services of different types, some problems can originate due to the use of the resources of the physical machine on which they are hosted, which forces to use different machines for each service provider. Obviously, this is not efficient, since a machine which does not use all its resources available is employed for a single service, which implies a waste of resources.

This is how virtual machines arise, which allow a physical machine to virtualize complete environments to create servers inside it, but this requires an entire environment, that is to say, an operating system for each virtual machine and its resulting reserve of resources which, despite helping to alleviate the problem, does not solve it. That is the reason why light virtualization mechanisms emerge, which are used to create optimized environments, providing the environment only with what it needs.

All this necessity of optimizing the cloud service provision comes from the increasing demand for services by users. for this reason, servers which are able to increase their resources at times of peak demand and also able to be available at any time are needed; un inaccessible service cannot be permitted, even during the updating of service providers. All this promotes the study of decentralized technologies, with the

purpose of preventing the system failure in case a machine fails the system does not continues working, which added to technologies of self-repair of the services, makes the server system reliable. In addition, the high demand for services may cause that a single server is not enough, giving rise to the need to have two servers providing the same service, so the load balancing technology of these requests is an important tool and, therefore, to cover in this project.

Currently, there are lightweight virtualization technologies based on containers and methods to manage them. A container can be defined as a program packaged in its basic environment and optimized to be as light as possible. Through the need to have many containers and take advantage of the resources of the networks, the aim is to create decentralized environments that execute in a simple way a group of containers belonging to the same service and that are distributed in several machines that work together, being an environment of orchestrated containers. This solves the problem of single point of failure that exists in centralized systems and provides the capacity to dispose of the resources of several machines that are connected in a network.

For the study of orchestration technology, Kubernetes has been chosen as the orchestrator. Kubernetes is a technology created by Google, and since 2015 is an open source code technology, allowing developers to access the code to improve it or simply understand it in order to create plugins that fit perfectly to Kubernetes. The expansion of Kubernetes due to its ability to orchestrate containers, together with the release of the code has generated the existence of communities of application developers who collaborate to improve this technology.

This circumstance has generated the need to certify these people and ensure that they are able to create efficient applications. In this way companies or whoever that needs applications, can be sure to hire the right person. This is not only limited to people, as there has also been a need to certify applications so that customers are confident that they will run on any platform that offers Kubernetes as a service. It is such an advantageous technology for providing services that it has also led several companies to focus on offering Kubernetes as a service in itself (KaaS), which has also led to the creation of certifications for service providers that guarantee customers that when contracting Kubernetes services, they will function perfectly. These certificates

are issued by the Cloud Native Computing Foundation (CNCF), which is a foundation that organizes open source to promote the cloud as a place for application implementation.

For the development of the Project, the available container types that can run in Kubernetes were investigated first, focusing in two types, Linux Containers (LXC) and Docker Containers. Linux Containers uses the kernel of the Linux operating system on which containers are run, and therefore entails the need for all containers to be functional with the same kernel. Docker allows abstraction of the operating system of the machine on which the containers are executed, as long as the Docker Engine can be installed. Presently it is available in Linux, Windows y macOS, which is the reason why the use of this type of containers has been chosen for the project.

For the creation and execution of Docker containers, first the image has to be created, which is a packaging of the application that is wanted to be run in the container. For the creation of its execution environment a base image is used, which can be an optimized operating system in order to be lightweight and on which necessary tools for the program are installed, what would be the equivalent of installing an operating system on a computer and the programs to be used by a user, which in the case of the container is the program. Once the image has been created, it is uploaded to an online repository called Docker Hub for its subsequent distribution. The containers that will be executed by Kubernetes are downloaded from the Docker Hub.

Kubernetes orchestrates the containers in a group of machines of the network, these machines together form a cluster that work together using specific drivers and components so that Kubernetes works as it should. In the cluster the machines are denominated as nodes, there are two kinds. The master nodes (Master) that dispose master components for the operation of Kubernetes, and the rest of the nodes are the “slave nodes” and have components to communicate with the other nodes and the master.

The containers are executed in the cluster by pods, which are containers groupings that are executed in a common environment, such as the one belonging to the same application that is responsible for providing services. The pods are the

minimum unit of execution in the Kubernetes cluster, so no containers will be found running outside of a pod. If a single container is needed to be executed in order to have an application working, this will be done in a pod dedicated to this container exclusively.

Kubernetes has deployment systems to automate the execution of pods that are desired in the cluster. For which definition files are used where desired containers and execution conditions are configured, like necessary memory and others. Therefore, just have to execute the deployment system and Kubernetes manages the creation of the containers for the desired application. So, in the case that more servers are needed due to demand excess, just an indication to Kubernetes to scale the deployment and it through its controllers performs this action automatically, generating replicas of the pods.

An application designed to provide services needs to be accessible in some way by users, in order to request these services since when deploying an application only runs in the cluster. For this reason, the service should be configured requesting Kubernetes to expose a deployed application.

There are three exposure types for the services: LoadBalancer, ClusterIP y NodePort. When exposing an application with LoadBalancer, Kubernetes balances the load between the replicas of the application. NodePort enables a port that, together with the node address, makes possible to request a service. And ClusterIP, only enables an address so that, from the cluster, service requests can be made internally.

For the Kubernetes study two applications have been used, the first is a web server and the second a time server, since in this way it is clearly demonstrated that Kubernetes fulfils all the project motivations. The web server used is the web server Nginx, which is an application that is available in public repositories of Docker Hub and will be executed in the cluster. In relation to the time server, it is an implemented server on MQTT technology, which is a communication protocol oriented to work on the "internet of things"(IoT).

In order to prove that load balancing exists, a web server has been launched with three replicas, in three different nodes and it has been configured to show different webs, thus when making the request to the web server it can be seen which server responds since one or another web is received.

To prove that the resources can be managed, the cluster has been configured to put resource limits on pods that do not have a specified limit, indicating memory maximums that can be used and pods limits that can be executed simultaneously in the cluster.

To examine the services self-recovery, a pod has been eliminated and it has been verified how the controllers of the cluster have immediately recovered the service, generating another pod to replace the eliminated one.

For the study of the decentralized server creation, it started by the hour server and then two containers for the components where built. The first container is the broker, it informs the time to all of its subscribers who subscribe to receive the time, the second container of the service is the time publisher, which informs the time to the broker. After creating this server in a Kubernetes cluster, the publisher versions have been updated to observe how the update affects the provision of the service. Was possible to verify that there is a temporary loss of service. Also, was possible to recognize that the loss of service is for a very short period of time, being the update controlled and being able to return to the previous version in case of serious problems with the new version, due to the existence of the version control.

Finally, it can be concluded that Kubernetes provide the necessary solutions to the problems that currently exist in order to provide services in the cloud and that motivate this project. However, in the case of the applications updates, the service is interrupted for a short period of time that is controlled, while the expected is to update the application without service interruptions.

Dedicatoria

Dedico el Trabajo de Fin de Grado a mi pareja y mi familia por toda su paciencia conmigo durante la realización del documento y su gran apoyo en los momentos difíciles de la carrera.

Índice

Índice de figuras

Índice de tablas

Índice de ficheros

Capítulo 1

Introducción

En el capítulo se trata primero la motivación del proyecto, la cual ha hecho que el tema de orquestación de contenedores con Kubernetes haya sido el elegido. Se habla de la evolución que ha sufrido la prestación de servicios hasta la situación actual que motiva el proyecto.

También se tratan los objetivos para estudiar y conocer esta tecnología, partiendo de un estudio previo para su posterior desarrollo y finalizar con un análisis de los resultados y sus conclusiones.

En un último apartado se resume la estructura del documento, donde se detallan los apartados de este, y su contenido, explicando cada capítulo del documento brevemente para dar a conocer en que parte se encuentran los temas a abordar durante la realización del proyecto.

1.1 Motivación del proyecto

Para entender cómo hemos llegado a necesitar tecnologías de orquestación de contenedores, tenemos que ver de dónde venimos, es decir, el pasado de la tecnología que vamos a estudiar.

Al principio, ya hace muchos años, para prestar un servicio en una red, necesitábamos alojarlo en un ordenador físico. Y para varios servicios del mismo tipo, hacía falta algún programa intermediario, que se encargara de identificar el servicio que tenía que servir nuestra máquina.

Además, existe el problema de que un servidor necesite un programa que utilice una versión diferente de herramientas del sistema de la máquina usada, que las que necesita otro programa de la misma máquina, lo que podríamos decir que es un sistema de servidores incompatibles. Esto hace necesario el uso de dos máquinas físicas diferentes, cada una para alojar a uno de los servidores, es decir, obliga que cada máquina física sea un servidor dedicado a un servicio específico.

Esta forma de prestar servicios evolucionó con la aparición de las máquinas virtuales (VMs), que permiten crear entornos completos, ya que virtualizan máquinas físicas, donde alojar nuestros servicios. Así configurándolas para que tengan su propia dirección IP, es posible hacer funcionar varios servidores del mismo tipo de servicio, lo que da solución al problema de ejecutar servicios incompatibles en la misma máquina física, ya que tienen direcciones diferentes para enviarles solicitudes de servicios.

La solución que aporta las VMs, trae además un nuevo problema, y es que para cada VM tenemos un entorno completo, es decir, un sistema operativo (Windows, Linux, macOS, …) por cada VM, lo que supone un gran gasto de recursos de la máquina física donde se alojan, ya que, por ejemplo, para virtualizar cuatro máquinas, tenemos que tener funcionando cuatro sistemas operativos. Además, es un problema que se agrava, ya que hay que tener en cuenta que un servidor no siempre está funcionando a la su capacidad máxima, es decir, estamos consumiendo recursos que además no se usan durante bastante tiempo.

Para resolver este problema apareció la tecnología de contenedores, que ofrece virtualización ligera, ya que, se genera un entorno ligero (el mínimo necesario) junto a la aplicación que se encarga de prestar servicios. Así los servidores siguen ejecutándose en entornos separados, pero estos están optimizados para el programa o servidor que se ejecute en ellos, lo que genera un gran ahorro de recursos de la máquina física donde se encuentren.

Estos sistemas anteriormente mencionados son sistemas centralizados y tienen como gran problema a tratar la existencia de un único punto de fallo, es decir, si falla perdemos el servicio ofrecido por el servidor. Así podemos detectar que cada servidor sirve en una sola máquina, ya que, aunque sea una máquina virtual, esta debe alojarse en una máquina física y si esta falla, perdemos el servicio. Para resolver esta situación es interesante poder alojar un servidor que pueda prestar un servicio a través de varias máquinas que permitan crear un sistema descentralizado.

Todo esto motiva el estudio de una tecnología capaz de coordinar varias máquinas que prestan un servicio, gestionando contenedores, ya que es el sistema más eficiente en cuanto al tratamiento de los recursos de la máquina física donde se ejecuta. Así nace la motivación de estudiar un sistema de orquestación de contenedores para que varias máquinas puedan ofrecer un servicio en la nube de forma coordinada.

Así se tiene que hay varios contenedores ejecutándose en varias máquinas, que pueden ser tanto físicas como virtuales, y prestando un servicio. Por esto hay que tener un control del número de máquinas de nuestro sistema, control de la organización de los servicios que queremos ejecutar a través de servidores instalados en los contenedores orquestados. Además, hoy en día, los usuarios están acostumbrados a que las aplicaciones se actualicen para mejorar rendimientos o adaptarse mejor a sus intereses como consumidores, algo que también afecta a las aplicaciones creadas para prestar servicios. Así los desarrolladores en este sistema de contenedores distribuidos necesitan enfrentarse a actualizaciones periódicas de aplicaciones repartidas por varias máquinas, sin que la aplicación a actualizar deje de estar activa, para que los usuarios no noten este tipo de acciones.

Otro factor que nos encontramos en la sociedad es la necesidad de cubrir mucha demanda de servicios, ya que el avance tecnológico hace que, según que servicios, existan muchos usuarios. Esto genera la necesidad de disponer de varios servidores de un mismo servicio y de tecnología que se encargue de decidir cual de estos servidores atiende y gestiona las peticiones de un cliente en particular.

Todos estos temas, motivan el estudio de Kubernetes como tecnología de orquestación de contenedores, ya que nos ofrece mecanismos de control sobre un grupo de máquinas para orquestar, actualizar sin interrupción, y prestar servicios con contenedores repartidos por todo el grupo de máquinas, pudiendo tratarse incluso de un mismo servicio replicado para atender a altas demandas.

Así esto motiva tanto el estudio, como el desarrollo de un escenario donde poder comprobar si Kubernetes cumple con las exigencias de prestación de servicios que demanda la sociedad actual.

1.2 Objetivos del proyecto

El objetivo del proyecto se centra en el estudio de las ventajas que ofrece Kubernetes, al orquestar contenedores para el cumplimiento de las motivaciones del desarrollo del proyecto. Así se pretende realizar un escenario que nos permita analizar sus características principales a la hora de ofrecer servicios descentralizados, así como de la propia gestión del escenario.

Para este objetivo primero hay que aprender una nueva tecnología, desconocida durante la carrera, y familiarizarnos con ella. Para lo que hay que:

- Conocer la tecnología de contenedores.
- Conocer los diferentes sistemas de uso de contenedores (LXC y Docker).
- Identificar las ventajas de empaquetar una aplicación en contenedores.
- Conocer el sistema de orquestación de Kubernetes.
- Identificas las ventajas que nos ofrecen las principales características del uso del orquestador de Kubernetes, que cubran la motivación del proyecto.

Después de conocer el funcionamiento de la tecnología, el objetivo es encontrar las posibles formas de implantar los sistemas, para el estudio de las ventajas. Para lo que hay que:

- Valorar los escenarios en los que se pueda ejecutar Kubernetes.
- Aprender a usar la línea de comandos (CLI) para la configuración necesaria.
- Aprender a desplegar un entorno de orquestación de contenedores en el escenario escogido.

Con la tecnología entendida e instalada, es decir, ya con la infraestructura en funcionamiento, se abordará cada característica a estudiar de Kubernetes y se comprobará si el funcionamiento es el esperado. Así hay que:

- Estudiar las características de orquestación en el escenario escogido, que cubran la motivación del proyecto.

- Estudiar la descentralización en Kubernetes, como parte de la motivación del proyecto.
- Realizar la conclusión de lo analizado.

Tras el análisis se procede a estudiar el marco regulador de Kubernetes como orquestador de contenedores y su importancia económico-social.

Como último objetivo se realiza una conclusión de los resultados y de lo aprendido en el desarrollo del proyecto.

1.3 Estructura del documento

En el este documento hay varios capítulos para abordar la correcta comprensión del objetivo a resolver, precedidos por un glosario, resumen e índices.

Así nos encontraremos la estructura queda de la siguiente manera:

- **Resumen**: breve resumen del proyecto
- **Abstract**: apartado del documento donde, en inglés, el proyecto.
- **Índices**: muestra la estructura del documento, profundizando en tres niveles.
- **Capítulo 1 – Introducción:** habla de la motivación por la que se desarrolla, los objetivos que hay que afrontar para la solución del problema y, en el apartado actual, se explica brevemente la estructura de este.
- **Capítulo 2 – Estado del arte:** se explican los conocimientos de las tecnologías necesarias para poder afrontar el objetivo del proyecto y tecnologías alternativas, tanto de contenedores como de orquestadores de contenedores.
- **Capítulo 3 – Estudio y análisis de Kubernetes**: en este capítulo se aborda el estudio de los posibles escenarios sobre los que analizar Kubernetes, también la puesta en marcha de un escenario escogido, y el análisis de las características que nos ofrece Kubernetes y que las que motivan la realización del proyecto. Por último, se realiza un análisis de los resultados a modo de conclusión y se comprueba si lo estudiado en el capítulo 2 se cumple. Así en este capítulo se determinará si la motivación del proyecto se cumple satisfactoriamente.

- **Capítulo 4 – Entorno socio-económico y marco regulatorio**: capítulo con dos apartados, el primero referente al entorno socio-económico, donde se trata como Kubernetes actúa actualmente en el entorno laboral y social. El segundo apartado hace referencia a los certificados aconsejables de Kubernetes, las leyes que afectan a las herramientas usadas y la propiedad intelectual de la tecnología de Kubernetes.

- **Capítulo 5 – Planificación y presupuesto**: se desarrollan tanto los plazos necesarios para la ejecución de este proyecto y el presupuesto para el escenario elegido.

- **Capítulo 5 – Conclusiones**: capítulo con lo aprendido del estudio del proyecto y las experiencias del análisis. También se tratará si se han cumplido los objetivos y con las motivaciones del proyecto.

- **Bibliografía**: apartado con las referencias de las fuentes usadas, para la realización del proyecto.

- **Anexos**: en este capítulo se muestran anexos de información que pueden resultar útil.

Capítulo 2

Estado del arte

En este capítulo se describen los aspectos teóricos necesarios para abordar de manera correcta los objetivos del proyecto.

Para lo cual se definen los contenedores, la tecnología de creación de contenedores como docker, y orquestación de contenedores en Kubernetes. Así como los componentes y aspectos relacionados con Kubernetes.

Además, se estudian brevemente las tecnologías alternativas, tanto de contendedores, como de orquestación.

2.1 Virtualización

Como se ha visto en el apartado de la motivación del proyecto, la virtualización surge de la necesidad de crear entornos software que recreen otro entorno como el real, con el objetivo de solo necesitar un entorno hardware capaz de soportar varios virtualizados. En este apartado se estudian diferentes formas de virtualizar.

2.1.1 Hipervisor

Otra forma de virtualizar es usando hipervisores, un entorno sobre el que se pueden ejecutar maquinas virtuales. Hay dos clases de hipervisores [1]:

- **Tipo 1**: tanto el sistema operativo (OS) principal de la máquina física, como los sistemas virtuales se ejecutan sobre el hipervisor de forma aislada, es decir, independiente. Así el hipervisor proporciona acceso a los controladores y recursos.

- **Tipo 2:** El hipervisor se ejecuta sobre el sistema operativo, de esta forma los sistemas virtuales se ejecutan sobre el hipervisor. Así el hipervisor proporciona acceso de los sistemas virtuales al OS de la máquina física, y el aislamiento solo es entre los sistemas virtuales, no entre cada sistema virtual y el OS. Actualmente los programas de virtualización de máquinas se basan en hipervisores de este tipo.

2.1.2 Maquina Virtual

Para virtualizar entornos completos de OS se pueden usan maquinas virtuales (VMs), ya que así se virtualiza un ordenador completo, incluso configurando la red en modo “bridge” la máquina virtual se ve en la red como un ordenador más.

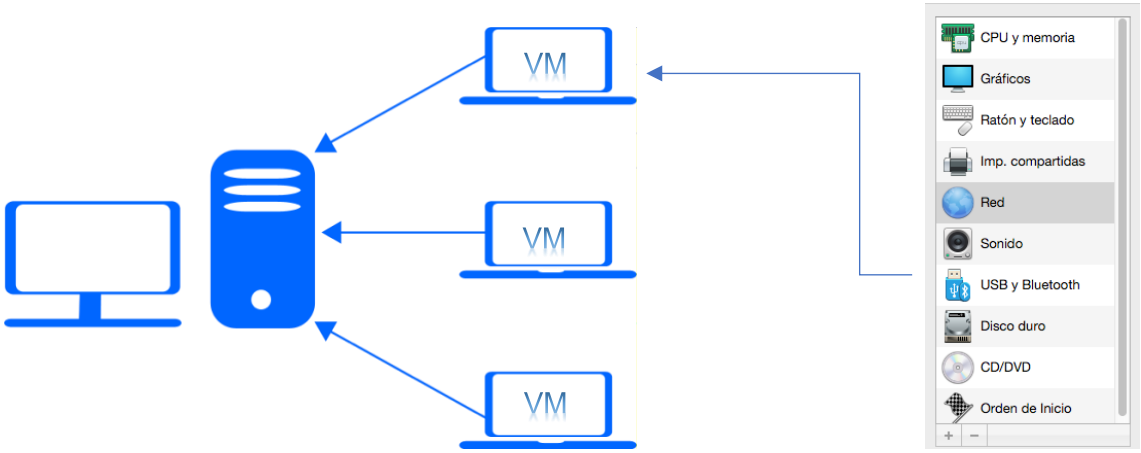


Figura 1. Esquema de máquinas virtuales en máquina física

Como se ve en la figura anterior se virtualizan máquinas en una máquina física, pudiéndose configurar todos los elementos hardware para virtualizar un ordenador de nuestra red. Donde como gran ventaja se tiene la posibilidad de ejecutar en una máquina física, varios OS. Pero esto conlleva, como gran desventaja, el consumo de los recursos, de forma no óptima, al disponer de instalaciones completas de OS por cada VM.

2.1.3 Virtualización ligera

Es un tipo de virtualización que se encarga de crear el entorno para el sistema virtual, usando un entorno lo más ligero posible compartiendo lo común del este y que no hace falta aislar en cada sistema virtual. Por ejemplo, para virtualizar máquinas Linux sobre un OS con el mismo kernel, no es necesario cargar todo el OS, ya se comparte.

Con esta idea surgen los contenedores, que en el siguiente apartado se estudian en más profundidad, y que llegan a ser tan ligeros que en algunos sitios llega a decirse que no es un sistema virtualizado, siendo solo un empaquetamiento de herramientas.

2.2 Contenedores

La tecnología de contenedores nos permite ejecutar aplicaciones de manera que no es necesario arrancar una VM con todo un OS, ya que utiliza las herramientas necesarias para la aplicación, siendo más eficiente y ligero que una VM.

Así se puede definir que un contenedor es un paquete que contiene una aplicación y las herramientas necesarias para que se ejecute. Por lo tanto, un contenedor está dotado de su propio entorno, y es independiente del entorno en el que se ejecute. Está cualidad permite que las aplicaciones empaquetadas en contenedores se puedan ejecutar en diferentes plataformas y OS.

En la siguiente figura se muestra las diferencias que supone ejecutar una aplicación empaquetada en un contenedor y esa misma aplicación ejecutada sobre una máquina virtual [2].

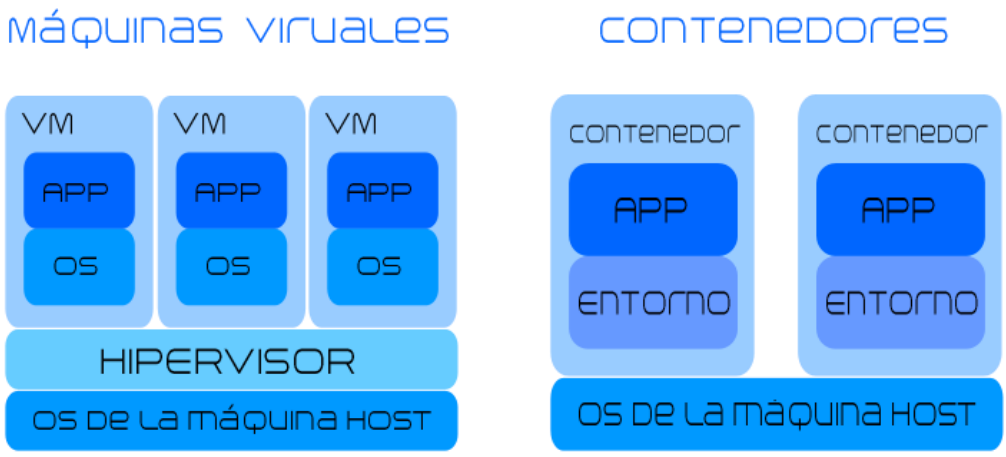


Figura 2. Imagen de la comparación de capas entre virtualización y contenedores

2.2.1 Ventajas principales del uso de contenedores

El uso de contenedores nos proporciona ventajas relacionadas con sus características. Las más destacadas son las mencionadas a continuación y que facilitan tanto el desarrollo de aplicaciones como su instalación.

Desarrollo

Para el desarrollo de nuevas aplicaciones podemos necesitar instalar nuevas herramientas en el entorno donde queremos ejecutar la aplicación, que pueden ser un inconveniente para otra aplicación ya en ejecución.

Así con el uso de contenedores, que proporcionan aislamiento, podemos trabajar de forma segura y posteriormente, si no nos gusta el resultado, borrar el contenedor de nuestra máquina.

Pruebas

El uso de contenedores ofrece varias ventajas importantes de cara a desarrollar o probar nuevas aplicaciones, ya que el contenedor al estar dotado de su propio entorno se ejecuta de forma aislada y, por lo tanto, permite ejecutar varias versiones de una aplicación simultáneamente, o probar nuevo código sin tener que dejar de ejecutar el anterior.

Distribución

De la propiedad de aislamiento que tienen los contenedores, y al disponer de las herramientas necesarias para la ejecución de las aplicaciones empaquetadas, es fácil almacenarlas en un sitio accesible para su despliegue. Estos sitios se llaman repositorios, e incluso disponen de control de versiones de las aplicaciones que se actualizan. Así para desplegar una aplicación empaquetada en un contenedor se descarga del repositorio correcto.

Para desarrollar contenedores podemos utilizar diferentes tecnologías que nos permiten crear aplicaciones y ejecutarlas, como Linux Containers y Docker [3].

2.2.2 Linux Containers (LXC)

LXC es una interfaz para la creación y gestión de contenedores de Linux. Y presta el servicio de un OS de forma aislada compartiendo el kernel del OS Linux de la máquina sobre la que se ejecutan. Por lo que, aunque los contenedores pueden pertenecer a distribuciones de Linux diferentes deben de ser distribuciones que usen el mismo kernel, como ya se mencionó anteriormente.

Los contendores lanzados deben de ejecutarse de manera aislada, de forma que LXC usa Cgroups (grupos de control) y Namespaces (espacios de nombres) para manejar esta cualidad. Además, LXC usa librerías y otras herramientas necesarias para la administración de los contenedores.

Hay que destacar que son contenedores ágiles y permiten que su creación y destrucción sea muy rápida, ya que comparten de las capacidades del kernel del OS Linux sobre el que se ejecutan, haciendo que sea un contenedor muy ligero [3], [4], [5].

2.2.3 Docker

Docker es un proyecto de código abierto y una plataforma de contenedores, que permite automatizar el despliegue de aplicaciones empaquetadas en contenedores.

Docker permite independencia entre las aplicaciones y el entorno de ejecución, ya que como anteriormente se mencionó, un contenedor empaqueta tanto la aplicación como herramientas del sistema necesarias. Por esto, Docker ofrece abstracción de virtualización de las aplicaciones sobre el OS utilizado en la máquina que se ejecuta.

Esto dota a las aplicaciones de portabilidad, ya que un contenedor docker funcionando en una máquina Linux, funciona también en una máquina con Windows o macOS.

A continuación, se muestra una imagen que representa los componentes que usa el motor de contenedores para Docker (Docker Engine) donde se ve como funciona como middleware ente los contenedores y el OS [2], [6].

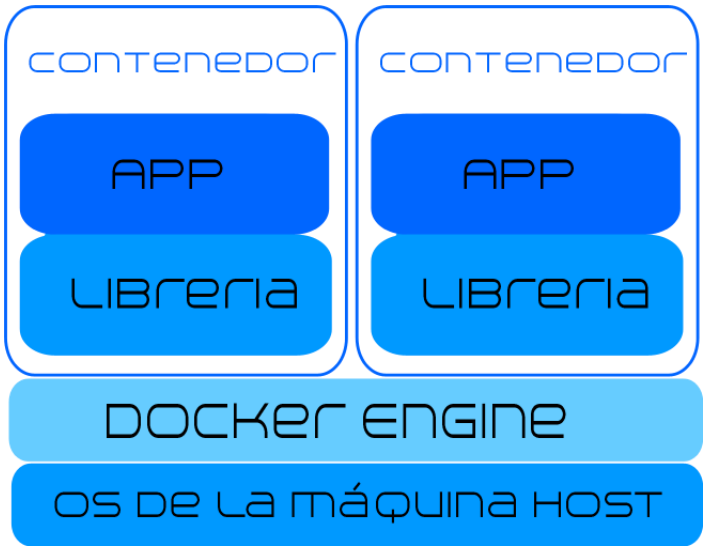


Figura 3. interfaces para acceder a las capacidades de virtualización del kernel Linux.

A continuación, se estudian los componentes necesarios para crear un contenedor docker:

- Imagen Docker

Una imagen Docker es la plantilla que se usa para crear el contenedor de una aplicación. Así una imagen Docker contiene una imagen base, además de la aplicación, que puede ser de un OS como Ubuntu o Alpine.

- Linux Alpine

Es interesante destacar esta distribución de Linux, ya que para la creación de contenedores está muy optimizada, al tratarse de una distribución muy ligera y minimalista. Esto se muestra en la siguiente imagen que nos permiten tener contenedores más rápidos.

Figura 4: Imagen Ubuntu Vs Imagen Alpine

Con un ahorro de tamaño tan considerable, y lo que conlleva, como mejor tiempo de descarga y despliegue, hay empresas que están cambiando las imágenes base de sus contenedores, para funcionar sobre Linux Alpine.

Con esta presentación es lógico pensar que nos interesa crear nuestros contenedores sobre imágenes basadas en Linux Alpine, pero dado que conocemos mejor Ubuntu, a lo largo del proyecto también lo usaremos como imagen base para la creación de nuestros contenedores [7].

- Fichero Dockerfile

Dockerfile es el documento que sirve para crear una imagen Docker, y por lo tanto contendrá la imagen base y la información para la composición de dicha imagen, es decir, un Dockerfile contiene las instrucciones necesarias para componer una imagen Docker [8].

- Docker Hub

Se aloja en la nube y es útil para almacenar repositorios de los ficheros Dockerfile y disponer de las imágenes, que se han creado, en todo momento.

Docker Hub además nos permite acceder a los repositorios de otras personas, ya que se pueden crear tanto repositos públicos como privados.

Para acceder a Docker Hub se puede hacer identificándonos con nuestro identificador de Docker (Docker ID). Además, se puede visualizar si es una imagen oficial, como se muestra en la siguiente figura, tras la búsqueda del repositorio para Ubuntu [9].

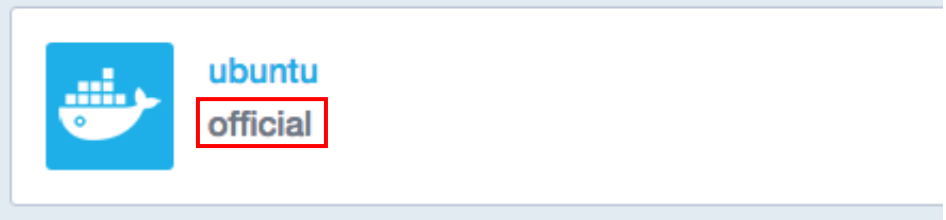


Figura 5. Repositorios de Docker Hub, de contenedores Ubuntu

En la siguiente figura se muestra una imagen de un repositorio público donde se almacena un contenedor Docker creado para ejecutar un servidor "iperf". Y se puede ver como los repositorios tienen información de la descripción, tanto corta como detallada, donde el propietario escribe la información que cree precisa, además de la información del propietario y el comando Docker para lanzar el contenedor a través de CLI.

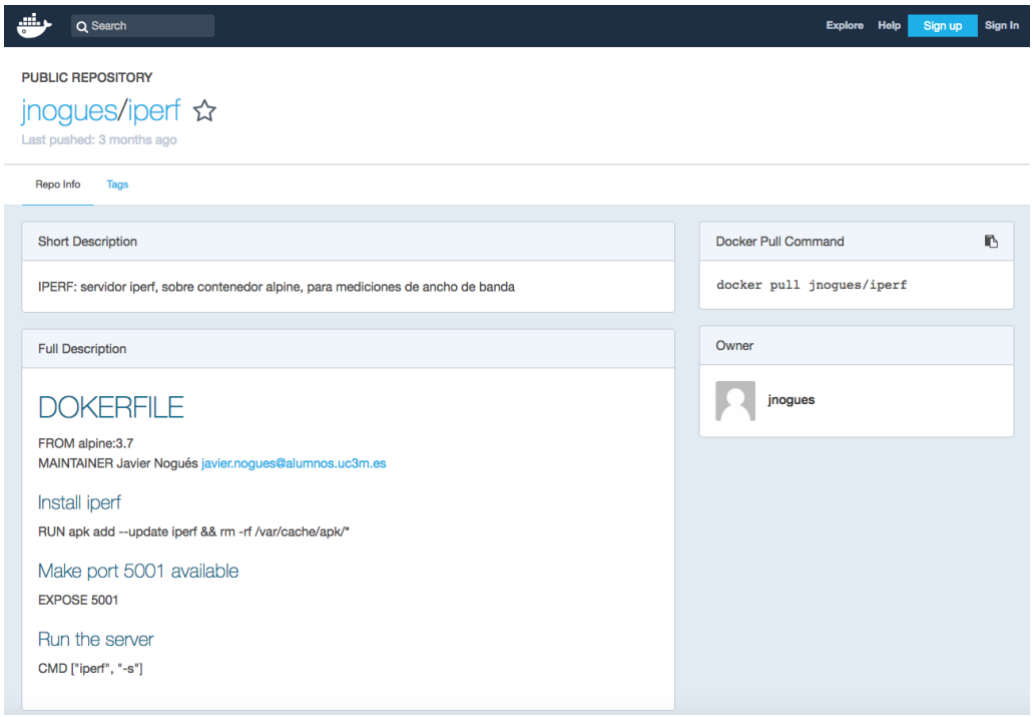


Figura 6. Imagen de un repositorio alojado en Docker Hub

2.3 Kubernetes

Figura 7. Logo de Kubernetes

Kubernetes es una plataforma de código abierto desarrollada por Google, y que posteriormente a sido mejorada a través de su comunidad, ya que fue donado a la "Cloud Native Computing Foundation". Es una evolución del proyecto "Borg" de Google.

A nivel de funcionamiento Kubernetes es un orquestador de la ejecución de aplicaciones que se ejecutan en contenedores, y permite gestionar estas aplicaciones.

Kubernetes significa timonel en griego, y provee de tecnología de despliegue, mantenimiento y escalado de aplicaciones entre sus funciones de orquestación de contenedores. En muchos sitios podemos ver que se refieren a Kubernetes como "K8s".

Admite varios motores de ejecución de contenedores entre los que tenemos Docker, el cual se usa para el desarrollo del proyecto, ya que Kubernetes fue diseñado, principalmente, para orquestar contenedores Docker al ser tecnología de Google [10], [11].

2.3.1 Arquitectura de Kubernetes

Kubernetes distribuye los contenedores en pods, así estos pueden estar en varios nodos. A su vez los nodos forman un clúster, completando la estructura que tiene Kubernetes.

A continuación, se explica de manera más detallada los elementos de Kubernetes [12].

Pods de Kubernetes

En Kubernetes los contenedores se agrupan en pods, por lo que todos los contenedores que se ejecutan en un pod lo harán en la misma máquina o host, ya que no se pueden separar.

Debido a esto se agrupan en el mismo pod a contenedores que usarán y necesitarán los mismos recursos, por lo que puede decirse que forman un host lógico. Si entendemos que podemos decir que un pod es un host lógico dentro del clúster, es fácil entender que un pod tiene una dirección IP compartida por los contenedores que lo forman. Además, todos los Pods se alcanzan entre ellos, dado que comparten una red privada.

Así un nodo puede tener varios pods ejecutándose, y por lo tanto el encargado de administrarlos es máster. Hay que destacar que existen pods con un solo contendor "*one-container-per-Pod*", donde el pod es como un envoltorio del contendor [13].

Nodo de Kubernetes

Aunque se puede decir, a efectos prácticos, que las aplicaciones se ejecutan en el clúster (conjunto de nodos), realmente la aplicación se ejecuta en los nodos, siendo los contenedores distribuidos por Kubernetes de manera automática.

Estos nodos pueden ser un bien un ordenador, bien una máquina virtual, o una máquina en la nube. Y están administrados por un agente que se llama Kubelet, que más adelante veremos.

Los nodos pueden ser de dos tipos: los nodos esclavos y los nodos maestros. Por simplicidad a los nodos esclavo se les llama simplemente nodos y a los maestros, nodos máster, o simplemente máster [14].

Nodo máster de Kubernetes

Kubernetes utiliza el nodo máster para las labores de administrar y planificar los pods que se ejecutan en los nodos del clúster y por lo tanto de los contenedores que estos contienen. Esto se realiza a través de controladores de Kubernetes.

En función de la carga de trabajo, podríamos disponer de varios nodos máster, que dotan al sistema de más resistencia ante fallos [14].

Clúster de Kubernetes

Un clúster se forma por los dos elementos explicados anteriormente, como también se puede deducir de la introducción de este apartado.

De esta forma Kubernetes tiene que coordinar contenedores en un sistema formado por varios nodos, haciendo que todo funcione como una sola unidad orquestada por el máster, logrando que las aplicaciones no estén vinculadas a una

sola máquina. Esto evita la necesidad de instalar una aplicación directamente en una máquina o host, realizándose sobre el clúster.

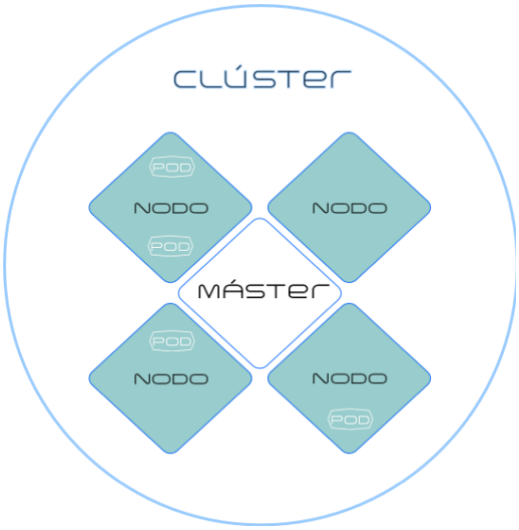


Figura 8. Clúster de Kubernetes

En la figura que se muestra arriba, se ve la estructura de un clúster. Como mínimo un clúster tiene que estar formado por tres nodos, es decir, un máster y dos nodos esclavos, a excepción de Minikube que veremos más adelante y que usa uno para todo.

La comunicación de los nodos con el máster, y del usuario con el clúster se hace a través de la API de Kubernetes [15].

2.3.2 Componentes de Kubernetes

Para que la arquitectura descrita antes funcione correctamente Kubernetes utiliza una serie de componentes en cada elemento del sistema dotando al sistema de las funcionalidades necesarias.

Componentes del máster

- Servidor de API (API server)

El servidor de la API de Kubernetes, sirve para acceder a la API de Kubernetes en nuestro clúster.

Por eso, es el front-end del plano de control de Kubernetes, es decir, es donde llegan las peticiones al clúster correspondientes a un servicio determinado. Estas pueden llegar desde un nodo o desde una petición por parte del administrador conectado al máster, y lo redirige a los componentes que correspondan.

Se encarga de preparar, validando y configurando, los datos de los objetos api (que más tarde explicaremos) que necesitamos para el clúster. Y se encarga de dar servicio de publicación de recursos del clúster.

También prepara la interfaz a través la cual los componentes del clúster interactúan con los demás componentes.

- Etcd

Es una base de datos que almacena y guarda la configuración del clúster, almacenando también información de los servicios que están disponibles.

Etcd estará replicado en todos los nodos máster del clúster para asegurar una alta disponibilidad de la información. Esta información será también usada por el API server para que los servicios desplegados en los contenedores mantengan sus características de forma coherente.

- Planificador (Scheduler)

Asigna a los nuevos pods un nodo en el que ejecutar su trabajo, es decir, se encarga de repartir los recursos disponibles en el clúster, para la ejecución de los pods tras valorar los requisitos que necesitan para desarrollar su trabajo.

También es el responsable de monitorizar la utilización de recursos de cada host para asegurar que los pods no sobrepasen los recursos disponibles una vez ya estén en funcionamiento.

- Gestor de controladores (Controller-manager)

Es un componente que como su nombre indica se encarga de gestionar los controladores de los nodos. Estos controladores son:

  - Controlador de réplicas (Replication Controller)

    Se encarga de controlar la ejecución correcta de réplicas deseadas para una aplicación.

  - Controlador de nodo (Node Controller)

    Se encarga de controlar que los nodos pertenecientes a un clúster funcionan de manera correcta y detectar si un nodo ha dejado de funcionar.

  - Controlador de puntos finales (End-points Controller):

    Gestiona los puntos finales (end-points) de los servicios desplegados.

- Controlador de la nube

Es un controlador más o menos moderno, es decir, de las últimas versiones de Kubernetes, y gestiona la conexión con el proveedor de servicios en la nube que se contrata para nuestro clúster.

Componentes de los nodos

- Kubelet

Se ejecuta en cada nodo gestionando los pods y su contenido a través de los ficheros que describen cada pod (objeto YAML o JSON), y juntos forman las especificaciones de un pod.

Hay tener en cuenta que Kubelet no administra contenedores que no pertenecen a Kubernetes, es decir, que no hayan sido creados por Kubernetes.

- Kube-proxy

Se ejecuta en cada nodo y proporciona abstracción de servicios al realizar el reenvío de conexión. Así, cuando llega una petición de servicio a un nodo por parte de un usuario desde el exterior, a un nodo donde no se está ejecutando algún pod de la aplicación que se encarga de ello, *Kube-proxy* se ocupa de hacer que la conexión se reenvíe al nodo correcto.

- cAdvisor

Se dedica a recoger, información del uso de los recursos que se usan en los nodos vigilando la CPU, la memoria, sistemas de ficheros y el uso de la red. La información recogida se usa para informar al nodo maestro.

Complementos de los nodos

Los complementos dotan a la estructura del clúster de funcionalidades adicionales.

- DNS

  Pese a ser un complemento, su funcionalidad es necesaria para el correcto funcionamiento del clúster. Será usado por *kube-proxy*, para reenviar la información.

En la siguiente figura se muestra la estructura completa de la arquitectura con los componentes en cada elemento del clúster. También la representación de las conexiones entre los nodos y el máster, los usuarios con el clúster y el administrador con el clúster [16].

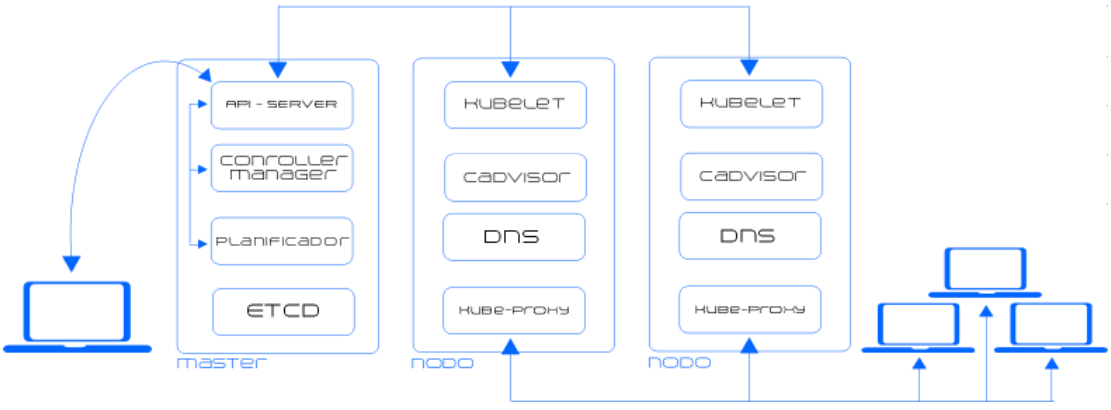


Figura 9. Imagen de los componentes en la estructura K8s

2.3.3 Objetos API de Kubernetes

Todos los elementos vistos hasta ahora, como nodos, controladores, pods, y otros que veremos más adelante, son considerados objetos API para ser administrados por Kubernetes.

Para esto se definen ficheros en formato JSON o YAML que se llaman ficheros de definición, para los objetos de nuestro clúster, con los que posteriormente podremos crear el objeto en el sistema. También se puede crear un objeto y configurarlo directamente en el clúster, utilizando Kubectl.

En Kubernetes los objetos describen el funcionamiento deseado para un objeto, por lo que la suma de objetos conformara el estado del sistema

En la API de Kubernetes, donde cada objeto tiene su representación nos podemos encontrar tres tipos de objetos en función de su desarrollo [17], [18]:

- **Alpha:** versiones de objetos API recién definidos, apenas probada y por lo tanto usarlos nos podrían dar problemas, así que si no es necesario es bueno evitar usarlos.

- **Beta:** versiones ya más probadas, pero no definitivas. Hay que evitarlas para aplicaciones que se necesiten a niveles profesionales.

- **Estable:** es una versión del objeto que ha sido probada con éxito.

2.3.4 Etiquetas

Están definidas en los objetos y son una dupla de clave y valor separados por dos puntos, es decir, con la forma “clave : valor”. Se usan para la gestión de los objetos por parte de los controladores del clúster.

Hay que tener en cuenta que una etiqueta debe ser única en un objeto pero puede tener varios valores incluso estar vacío.

En el objeto las etiquetas se organizan a continuación de la etiqueta “labels” de los metadatos. En la siguiente imagen se muestra un ejemplo de las etiquetas “run” y “app”, con sus valores.

```
labels:
  run: iperf
  app: server
```

Figura 10. Ejemplo de etiquetas

Las etiquetas no están previamente definidas, pudiendo cada desarrollador establecer los nombres que más convengan.

Selectores de etiquetas

Antes se ha visto que no pueden existir dos etiquetas con el mismo nombre en un objeto, pero si puede haber dos objetos con la misma etiqueta y el mismo valor. Esto hace posible agrupar un conjunto de objetos, en función de una etiqueta, usando los selectores de etiquetas y comprobando los valores que estas tienen. Por ejemplo, se utiliza para ejecutar pods para prestar un servicio en el sistema.

Tipos de selectores

- ### Selectores de igualdad

Busca igualdades que cumplan la condición especificada en el operador definido de la expresión implementada.

Ejemplos:

1. app = server
2. app != zone1

El primero selecciona objetos que tienen la etiqueta “app” y el valor “server”, y el segundo ejemplo excluye los objetos que tengan el valor “zone1”.

- ### Selectores de conjunto

Este tipo de selectores busca que una etiqueta tenga un valor que satisface un rango de opciones, es decir, no hay una condición especifica de selección como en los selectores de igualdad.

Ejemplos:

1. type in (server, client1)
2. zone notin (zone1, zone2)

El primer ejemplo selecciona los objetos que tienen la etiqueta “type” con los valores “server” y “cliente1”. El segundo ejemplo selecciona los objetos que no tengan el valor “zone1”, “zone2” y “zone3”. Como se ve en ambos ejemplos, puede haber varios valores para configurar la regla del selector.

- Selectores de clave

Estos selectores solo comprueban la existencia de una etiqueta, sin importa el valor que tiene.

Ejemplos:

1. app
2. type
3. !zone

Los dos primeros ejemplos seleccionan los objetos que tengan la etiqueta, mientras que el tercer ejemplo excluye a los objetos que contengan la etiqueta “zone”. Como no importa el valor de las etiquetas, no se implementan en la regla, es más una condición de existencia [19].

2.3.5 Recursos en Kubernetes

Las maquinas que se usan en un clúster de Kubernetes tienen recursos limitados y por lo tanto es importante conocer como son usados por las aplicaciones, es decir, como se asignan los recursos de los nodos en la ejecución de pods.

Los dos principales recursos por nodo son los recursos de memoria y los recursos de CPU o lo que es lo mismo, recursos de computación.

Asignación de recursos

Para el control de estos se pueden establecer limites de consumo en los contenedores y los pods. Estos límites se establecen en los objetos con la etiqueta “limits”.

De igual modo que los límites de memoria y CPU, podríamos querer establecer unos requisitos de lanzamiento de los contenedores y pods. En este caso en el objeto se configura una etiqueta “request”, que hace referencia a los recursos solicitados.

Para establecer el valor de los límites y de los recursos requeridos, en los pods y contenedores, se fija el valor de las etiquetas especificas de cada recurso. Para lo cual se utilizan unidades de memoria y CPU, usando etiquetas “memory” y “CPU” respectivamente.

Así para el recurso de memoria la unidad es el byte, y para el recurso de computo la unidad es la CPU. Por ejemplo, si se quiere asignar 1GB de memoria se establece el valor “1G” o “1Gi”, y si queremos asignar la mitad de la CPU se establece el valor 0.5 [20].

Comportamiento en caso de exceder recursos

Es muy importante conocer como se comporta Kubernetes en caso de que los recursos sean excedidos. Para ello vemos los siguientes casos [21], [22]:

- **Un contenedor o pod solicita más recursos que su límite:** en este caso el contenedor o pod es finalizado por exceder los recursos.

- **Un contenedor o pod solicita más recursos de los disponibles:** en este otro caso el contenedor o pod se queda a la espera de un nodo con los recursos suficientes para su ejecución, en estado “Pending”, es decir, pendiente de asignación de nodo.

2.3.6 Espacio de nombres (Namespaces)

El espacio de nombres sirve para dividir el clúster físico de manera virtual, y podría decirse que crea clústeres virtuales. Concretamente los recursos que son divididos, como los nodos, no pertenecen a ningún namespaces a diferencia del resto, por ejemplo, cualquier pod desplegado.

Además, esta división lógica crea aislamiento de nombres entre los objetos de diferentes espacios de nombres. Por ejemplo, podríamos tener un pod en un nodo con un nombre, y en el mismo nodo, pero en un namespace diferente otro pod con el mismo nombre. También se pueden asignar permisos de creación o modificación de recursos a cada espacio de nombres para diferentes usuarios.

A continuación, se muestra una imagen que representa cómo sería esta división, de cada componente físico.

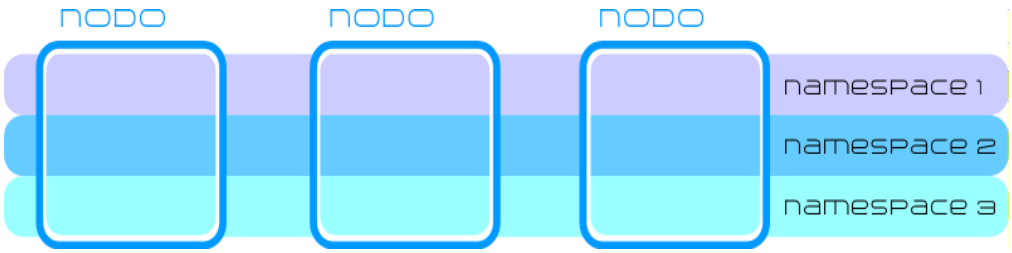


Figura 11. Separación del sistema físico, por el uso de namespaces

Cuando se crea un clúster de Kubernetes, se crean por defecto tres namespaces en el sistema:

- **Default:** cuando se crean objetos a los que no se le ha especificado un espacio de nombres concreto se le asigna el namespaces “default”.

- **Kube-system:** Kubernetes necesita objetos para funcionar y desempeñar su trabajo. Estos objetos se crean en este espacio de nombres

- **Kube-public:** como su nombre indica es público y por lo tanto todos los usuarios pueden acceder a su contenido. Puede ser un espacio que esté vacío o que contenga información pública que interese mostrar.

Adicionalmente, a estos tres espacios de nombres, se pueden crear nuevos espacios para dividir el sistema de manera conveniente. Por ejemplo, sería interesante crear un nuevo espacio de nombres dedicado a pruebas de las aplicaciones que queramos testear previamente.

Las cuotas en los espacios de nombres

Los recursos también se pueden regular en los espacios de nombres para asegurarse de que un grupo de servicios no consuman todos los recursos. Por ejemplo, es muy útil configurar nuestro clúster para que un namespace dedicado a ejecutar aplicaciones y servicios en prueba no consuman recursos necesarios de los servicios que ya estén lanzados.

Para configurar cuotas se usa un objeto *ResourceQuota* y se pueden configurar no solo los recursos de memoria y CPU, también el numero de pods, a través de etiquetas [23].

El objeto LimitRange

Otro recurso de control de recursos que se puede configurar en los espacios de nombres es un objeto *LimitRange.* Este objeto asigna valores de limites de recursos y recursos requeridos cuando no están especificados en los objetos contenedores [24].

2.3.7 Objetos controladores

Al igual que los objetos que definen recursos del clúster, como los pods, también existen objetos de los controladores encargados de que este funcione correctamente para que pueda gestionar los pods y por lo tanto la orquestación de contenedores. A continuación, vemos los esenciales para el proyecto.

Deployment

Es el controlador de despliegues de contenedores que necesitamos para nuestra aplicación, es decir, se encarga de que esta se ejecute en función de unas características concretas. Por ejemplo, el numero de pods que queremos que se ejecuten.

En la siguiente figura se muestra gráficamente el clúster y cómo se situaría una aplicación empaquetada en contenedor Docker tras el despliegue de una implementación a la que pertenece [25].

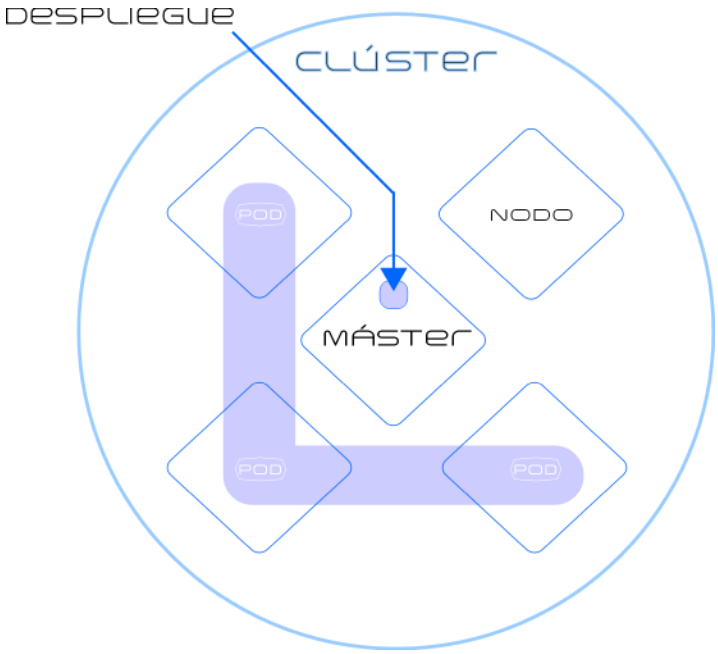


Figura 12. Imagen de una implementación desplegada en un clúster.

ReplicaSet

Es un controlador de réplicas de pods de la aplicación desplegada y conlleva la creación de un objeto *ReplicaSet.* Por lo que es la propia implementación del despliegue la que se encarga de la administración de este objeto.

Estas cantidades especificadas de réplicas se vigilan y en caso de que no se cumplan *ReplicaSet* se encarga de recuperar el estado deseado del número de réplicas. Por ejemplo, si hemos especificado que queremos tres réplicas de un pod y uno de los nodos de nuestro clúster deja de funcionar, el *ReplicaSet* se encargará de solicitar la creación nuevos pods y así estos se alojaran en otros nodos manteniendo la configuración deseada.

De todos modos, se puede crear un objeto *ReplicaSet* si la implementación no lo contempla, pero hay que tener cuidado con los selectores para que los pods que queremos que se repliquen lo hagan de la manera correcta.

ReplicaSet esta sustituyendo a otro objeto controlador, el *ReplicationController* que hace lo mismo, pero sólo permite usar selectores de igualdad [26].

2.3.8 Almacenamiento

Las aplicaciones que se ejecutan en contenedores pueden necesitar usar algún tipo de almacenamiento para su correcta ejecución, ya sea solo durante la ejecución o recibiendo información del sistema de almacenamiento. Así Kubernetes especifica el uso de volúmenes capaces de almacenar datos.

Volúmenes

Para este almacenamiento se crea un objeto volumen, llamado "*volumen",* que se ejecuta en el pod y accesible por todos los contenedores ejecutados en el mismo pod.

Este tipo de almacenamiento es temporal, es decir, lo que se almacena en un pod solo permanece durante la ejecución del pod. Por lo que si tanto un pod como un nodo que ejecuta ese pod deja de funcionar, Kubernetes puede lanzar otro pod para reponer el servicio pero el volumen vuelve a tener la información especificada en la configuración de la aplicación para ser desplegada.

EmptyDir

Monta un volumen vacío en el pod del contenedor en la ruta especificada en la configuración del contenedor con la etiqueta "emptyDir".

Volúmenes persistentes

Para usar almacenamiento persistente en un pod se utiliza un volumen de tipo persistente llamado *persistentVolume.* Este tipo de volumen cargará una ruta de la máquina física y con una petición de volumen llamada *persistentVolumeClaim* se monta en el contenedor.

Cuando se modifica algo en este volumen también se modifica en la máquina física. Así si un nodo se reinicia sigue teniendo la información que modificó.

HostPath

Es un tipo de volumen persistente que monta una ruta del sistema de ficheros del nodo en la ruta indicada en la configuración del contenedor. Esta

información se da como valor de la etiqueta “path” que estará anidada debajo de la etiqueta “hostPath”.

Volúmenes de proveedores de nube pública

Tanto Google como Amazon dispone de sus propios sistemas de almacenamiento persistente. Particularmente en el caso de Google el volumen se llama “gcePersistentDisk”.

Estos volúmenes solo se pueden usar en los contenedores de la misma zona geográfica. Por lo que, en caso de uso, es importante usar mecanismos como los selectores para que los pod que quieran usar estos volúmenes cumplan con este requisito de zona [27], [28].

2.3.9 Exponer una la aplicación en Kubernetes

Cuando la aplicación ya es desplegada en un clúster es importante saber como conectarse a la aplicación para que un cliente pueda realizar la petición de un servicio.

Una forma es a través del CLI de Kubernetes, es decir, usando comandos de Kubectl abriendo un proxy en el nodo donde tengamos el pod de la aplicación, exponiendo un puerto para redirigir el tráfico a ella. Esta forma es solo interesante para probar la aplicación, pero ya que se quiere tener prestando servicio para peticiones externas, hay otra forma de acceder a través de los servicios en Kubernetes.

Servicios en Kubernetes

Los servicios exponen la aplicación de todas las réplicas existentes, en todo el clúster, de una aplicación de manera unificada. Para lo que se usan las etiquetas y selectores. Además, los servicios hacen posible redirigir las conexiones ofreciendo balance de red, es decir, distribuyendo la carga.

Tipos de servicios:

- ClusterIP

Define el servicio asignando una IP interna de clúster y por lo tanto solo se puede acceder al servicio desde el propio clúster. De esta manera no hay que preocuparse de las variaciones de las direcciones IP de los nodos del clúster que se puedan dar por sustitución de nodos o reasignación de direcciones.

Es un tipo de servicio útil para aplicaciones dentro de nuestro clúster que quieren acceder a un servicio back-end.

- NodePort

Asigna el mismo puerto en cada nodo para cada servicio, así con la dupla de NodeIP (IP del nodo) y NodePort (puerto asignado par el servicio) se puede acceder al servicio desde fuera del clúster. *Kube-proxy* se encargará de enrutar, el tráfico usando el servicio DNS de Kubernetes, al nodo correcto.

Para que el usuario que acceda a los servicios no tenga la necesidad de conocer un bloque de direcciones IP se puede crear un balanceador externo.

- LoadBalancer

Si usamos un proveedor de servicios en nube para nuestro clúster de Kubernetes no necesitamos crear un balanceador de carga, ya que la mayoría de los proveedores tienen plugins para esta tarea.

Así a estos servicios se denominan de tipo *LoadBalancer* y crea una IP fija y externa al servicio para que el balanceador se encargue de encaminar el tráfico.

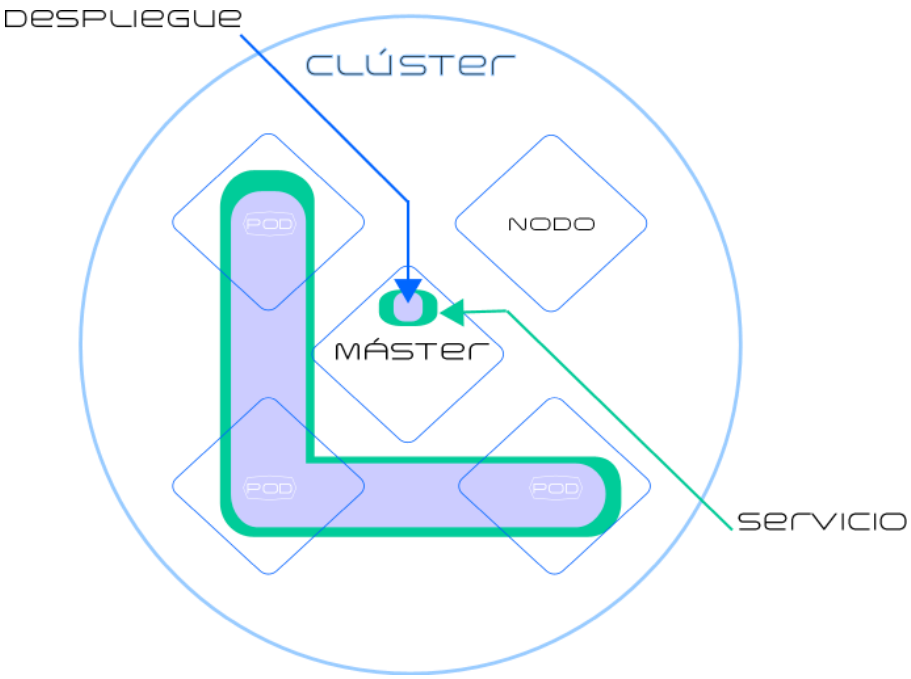


Figura 13. Figura de la estructura de un clúster con una implementación expuesta como un servicio.

En la figura de arriba se representa gráficamente como queda en servicio un despliegue de contenedores de una aplicación (deployment) que tiene tres pods distribuidos en tres nodos [29].

2.3.10 Networking de Kubernetes

Un pod es la unidad más básica que nos encontramos en la arquitectura de Kubernetes. Por lo tanto, cada pod tiene una dirección IP perteneciente a una red privada del clúster. Así los contenedores que pertenecen a un mismo pod comparten

esta dirección IP, siendo las conexiones diferenciables por los puertos que usa cada contenedor. Por lo que un end-point, en Kubernetes, hace referencia a un pod.

Para que la conectividad funcione correctamente en el clúster tiene que cumplirse:

- Que todos los pods de un mismo espacio de nombres puedan alcanzarse entre ellos.
- Que todos los nodos puedan alcanzarse entre ellos.
- Que todos los pods puedan alcanzar todos los nodos.

Así en la siguiente figura se muestran la configuración de la estructura de Kubernetes en una red privada (10.244.0.0/16) para los pod.

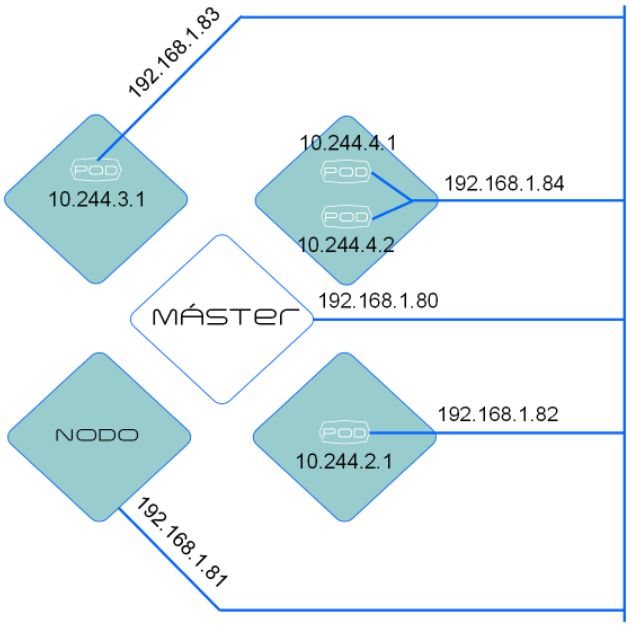


Figura 14. Direccionamiento de red de un clúster

Pese a ser importante que la conectividad de la red funcione adecuadamente Kubernetes no se encarga de ello por defecto. Así permite que el administrador del clúster decida cual es la mejor solución para su red, instalando un plugin de red (plugin CNI) que crea conveniente. Kubernetes, en función del plugin instalado, tiene varios modelos de red.

Algunos plugins CNI:

- **Flannel:** por su simplicidad, y demostrada eficacia en su funcionamiento, le hacen el plugin a instalar en el clúster casi por defecto.
- **GCE plugin de enrutamiento avanzado:** es el plugin de Google, y se usa en su estructura de solución en nube.
- **Kube-router:** esta pensado para Kubernetes y diseñado para dotar al sistema de alto rendimiento.
- **Calico:** es apropiado para redes escalables. Usa BGP distribuir las rutas.

Hay otros, pero no es necesario describirlos, como: Climium, CNI-Genie de Huawei, Contrail, ACI (de Cisco), OVN, Romana [30].

2.3.11 Clúster HA

Kubernetes permite descentralizar un sistema para que se pueda evitar el principal problema de un sistema centralizado, es decir, la existencia de un único punto de fallo.

No obstante, al crear un clúster normal se desplaza este problema al coordinador, es decir, al nodo máster encargado de orquestar los servicios. Para lo cual existe la solución de crear clústeres de alta disponibilidad (Hight Avality) de dos formas.

La primera es creando un clúster con más de un máster, para lo que es necesario un componente balanceador de carga entre los nodos y los nodos máster, como se muestra en la siguiente figura [31].

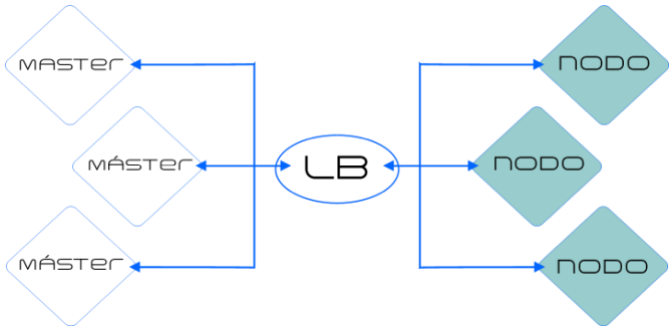


Figura 15. Clúster HA con múltiples nodos máster

Otra forma que hay de crear alta disponibilidad es usando federaciones de Kubernetes. Esto consiste en crear varios clústeres y añadirlos a la misma federación. La siguiente imagen muestra un esquema de esta configuración [32].

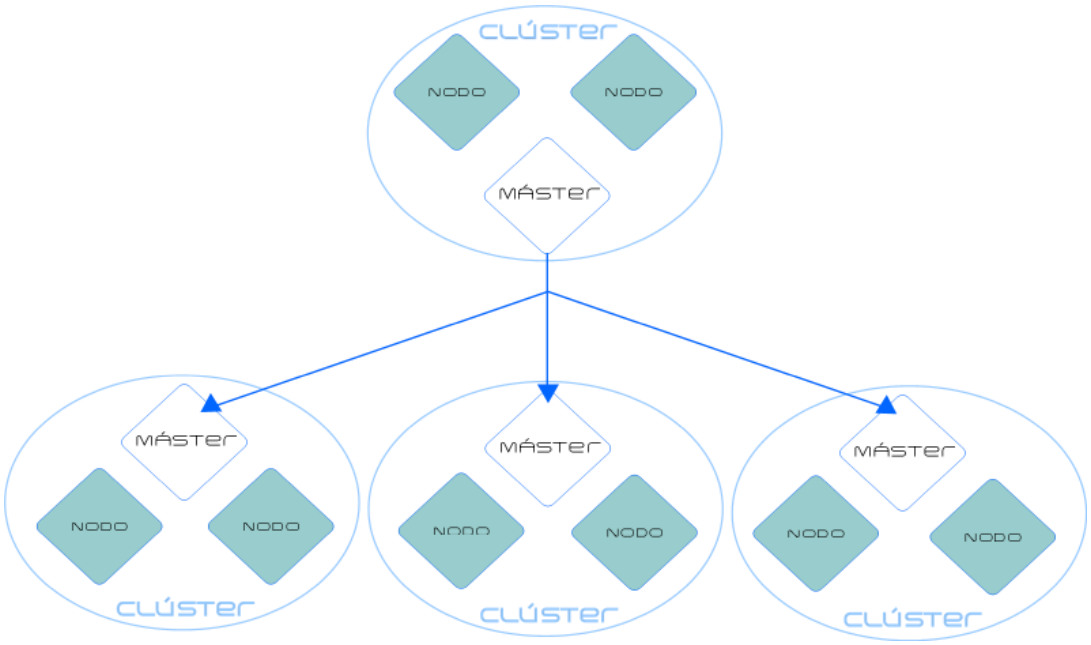


Figura 16. Federación de clústeres

La alta disponibilidad de esta forma llega a ser una solución fiable si cada clúster se configura en diferentes zonas geográficas o regiones, ya que se entiende que dos zonas diferentes rara vez van a estar inaccesibles al mismo tiempo. Así una región se divide en zonas.

Google Cloud Platform permite administrar con un máster nodos de diferentes zonas, siendo también un clúster de alta disponibilidad. En la siguiente figura se muestra este concepto [33].

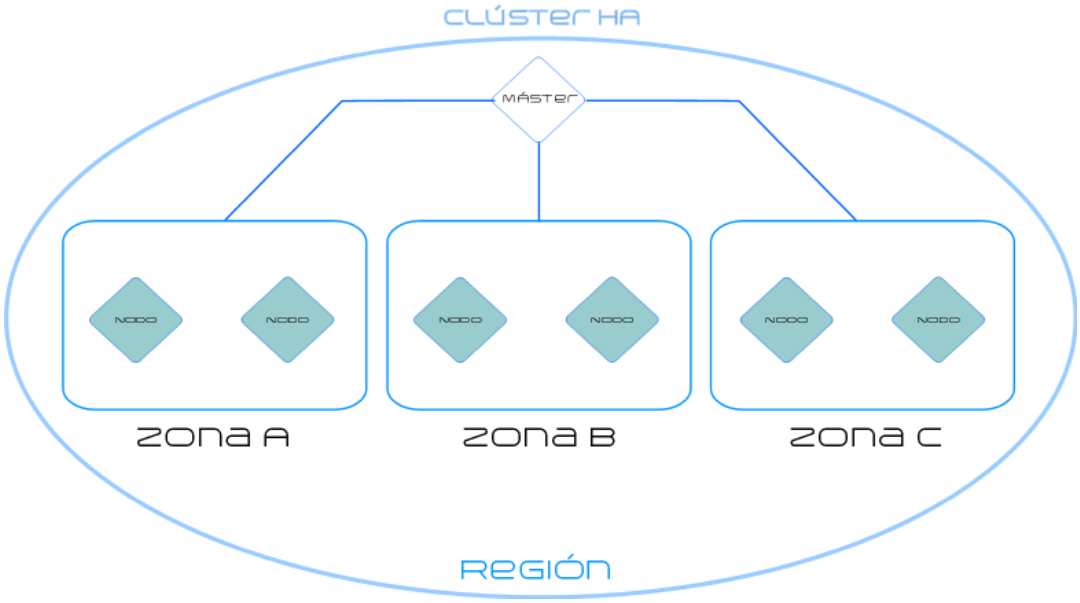


Figura 17. Clúster HA con nodos en diferentes zonas.

2.3.12 Características para estudiar de Kubernetes

Kubernetes ofrece caracterizas que aportan las ventajas que buscamos para el proyecto y que cubren las motivaciones del proyecto. Así se identifican para poder estudiarlas en apartados posteriores, y ver su funcionamiento [10]:

- **Despliegue automático:** permite configurar un despliegue encargado de ejecutar los contenedores necesarios para un servicio.

- **Balanceo de carga:** al configurar servicios “LoadBalancer” se distribuye la carga entre los end-point del despliegue optimizando el sistema al repartir las peticiones a un servicio.

- **Auto-reparación:** los controladores del clúster controlan que los contenedores que se han desplegado funcionan correctamente. En el caso de que no sea así

se encargan de reemplazar los contenedores que generan el problema. Esta auto-reparación puede llegar a nivel de nodo, con algunos proveedores de Kubernetes en nube pública como Google Cloud Platform, donde si detecta que queremos tener tres nodos en un clúster y uno falla se genera un nuevo nodo.

- **Escalado:** permite que una aplicación ya desplegada pueda aumentar o reducir el número de réplicas que tiene en ejecución.

- **Actualizar una aplicación con control de versiones:** permite actualizar una aplicación sin parar el servicio, creando primero un nuevo contenedor actualizado de la aplicación y posteriormente cuando ya está funcionando, elimina el antiguo. Además, el control de versiones permite deshacer una actualización a una versión anterior.

- **Gestión de recursos:** permite configuración de gestión de los recursos que se usa en el clúster para controlar y proteger el correcto funcionamiento.

2.3.13 CLI de Kubernetes: Kubectl

Kubectl es una herramienta de línea de comandos que usando el API de Kubernetes ejecuta los comandos para administrar el clúster. Esto permite al usuario, ya sea administrador o desarrollador, comunicarse con las aplicaciones del clúster y gestionar tanto la tecnología de Kubernetes como las aplicaciones desplegadas a través de un terminal, en la máquina local, que sea el encargado de conectarse con el máster [34].

2.4 Tecnologías alternativas

Además de las tecnologías estudiadas en este capítulo, y suficientes para el desarrollo del objetivo, hay otras formas de empaquetar aplicaciones en contenedores que pueden orquestarse con Kubernetes. Y de igual forma hay otras de orquestar contenedores que no son Kubernetes.

2.4.1 Alternativas de contenedores

Kubernetes nos permite gracias a una interface abstraer su funcionamiento del tipo de contenedores utilizados. Esta interfaz se llama “Container Runtime Interface” (CRI) [35]. Así tenemos:

- **RKT**: que se conoce como tecnología de “rocketnetes”, es tecnología estandarizada.
- **CRI-O**: son contenedores ligeros de RedHat, para aplicaciones ligeras.
- **Hypernetes**: basado en hipervisores.
- **Clear Containers**: contenedores ligeros y seguros de Intel.
- **LXD**: complemento hipervisor para los contenedores LXC, es decir, realmente es un administrador de contenedores LXC. Los LXD están enfocados para simular VMs con contenedores LXC.

2.4.2 Alternativas de orquestación de contenedores

No solo disponemos de Kubernetes para poder orquestar contenedores, así tenemos las siguientes [36].

Docker Swarm

Es la aplicación nativa de docker que permite la creación de un clúster, que en Docker Swarm se llama enjambre, añadiendo los nodos que queremos que formen parte de este.

Figura 18. Logo de Docker Swarm

El enjambre es la conversión de un conjunto de hosts, los que deseados en el enjambre, en un host virtual para ejecutar los contenedores de Docker [37].

Apache Mesos

Desarrollado por la Universidad de Berkeley, para gestionar grandes cargas de cálculo y proporciona aplicaciones como Hadoop y Spask (para Big Data) [38].

2.4.3 Conjure-up

No es realmente una alternativa de Kubernetes, es una alternativa a la ejecución de Kubernetes con contenedores docker. Conjure-up usa contenedores LXD y permite el despliegue de Kubernetes guiado paso a paso [39].

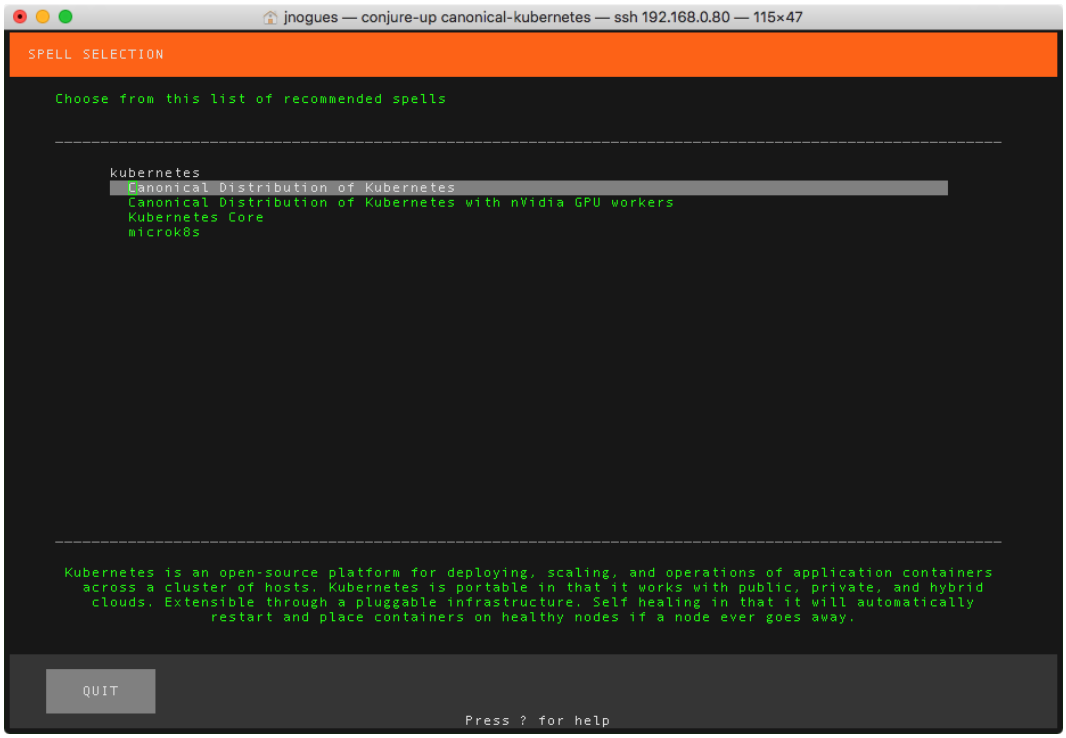


Figura 19. Captura de pantalla de la instalación de Conjure-up

Capítulo 3

Estudio y análisis de Kubernetes

Este capítulo aborda las características a probar que hacen de Kubernetes una tecnología interesante. Se plantean distintos escenarios y se estudian sus capacidades para desarrollar Kubernetes.

Para lo cual primero se compara Docker con LXC, para abordar la decisión de que tecnología de contenedores escoger para el proyecto. Luego se estudian las posibilidades de desplegar un escenario tanto en nube pública como en privada, estudiando las opciones de cada caso y las características que ofrece cada escenario para poder elegir el adecuado.

También se despliega el escenario elegido para Kubernetes. Para el que se configura el escenario adecuadamente para que se puedan analizar y se determinan las características, completando el estudio con la conclusión del análisis, analizando si cumple con lo esperado.

3.1 Escenarios

Para la creación de los escenarios en Kubernetes se valoran las tecnologías de contendores que hemos considerado para el estudio del proyecto, después se consideran los diferentes escenarios para la tecnología de orquestación con Kubernetes.

3.1.1 Comparativa de Docker con LXC

Docker parte de LXC pero con el paso del tiempo se fue desarrollando de manera diferente aportando facilidades de gestión, incluso se han desarrollado mecanismos de orquestación de contenedores como Docker Swarm.

Otra de las características que diferencian ambos sistemas de creación de contenedores es la forma de administrarlos, ya que Docker busca dividir la aplicación en procesos para facilitar su gestión, sin tener que tomar decisiones que afecten a la aplicación como una unidad, sino a partes de ella. Lo que mejora, por ejemplo, una posible restauración. En la siguiente figura se muestran dos imágenes que comparan esta característica.

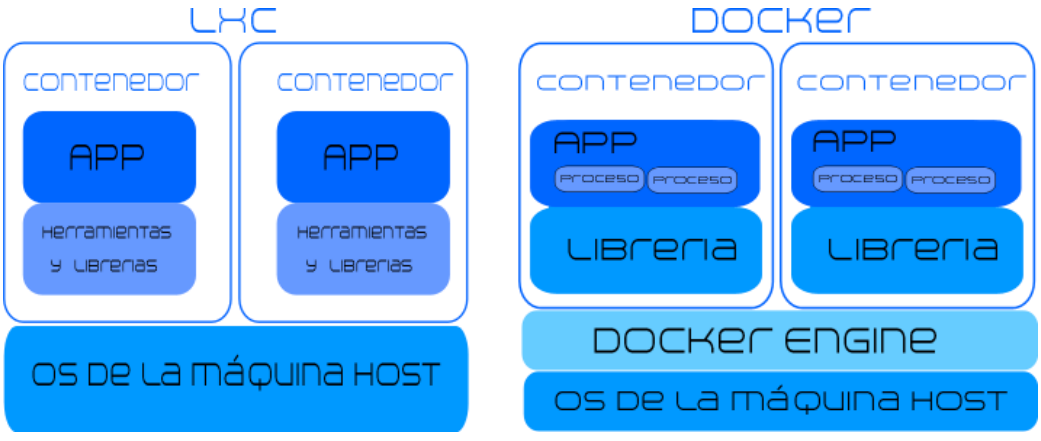


Figura 20. Imagen de la comparativa de capas usadas para el uso de contenedores.

Como diferencia notable hay que destacar la posibilidad de instalar docker CE en entornos Windows y macOS, lo que hace que la portabilidad de los contenedores Docker sea mayor. Siendo esta característica muy importante considerando que muchos usuarios utilizan Windows y que la portabilidad es una de las razones que motivan el proyecto [6].

Después de este análisis vamos a desplegar contenedores Docker en nuestros clústeres de Kubernetes.

Ejemplo de la abstracción respecto al SO

En los siguientes ejemplos se muestra como Docker aporta abstracción respecto al OS, dotando de portabilidad al contenedor. Concretamente se muestra viendo como el mismo contenedor funciona tanto en macOS como en Ubuntu.

- Lanzamiento de contenedor iperf en macOS.

Primero descargamos la imagen del contenedor del repositorio (jnogues/iperf) en “Docker Hub” como se muestra en la siguiente figura.

```
MacBook-Pro-de-JAVIER:~ jnogues$ docker pull jnogues/iperf
Using default tag: latest
latest: Pulling from jnogues/iperf
ff3a5c916c92: Pull complete
aa941bf58106: Pull complete
Digest: sha256:bbf0b3a45863332ef9e4afebb1b6a8ba701b36abada0c0378ecc382b65a4b2ed
Status: Downloaded newer image for jnogues/iperf:latest
```

Figura 21. Captura del terminal de descarga de contenedor iperf

Luego se ejecuta el contenedor y se ve como el servidor queda a la escucha teniendo ya el contenedor funcionando como se muestra en la siguiente figura.

```
MacBook-Pro-de-JAVIER:~ jnogues$ docker run jnogues/iperf
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Figura 22. Captura del terminal de creación y ejecución de contenedor iperf

- Lanzamiento de contenedor en Ubuntu

Ahora con el mismo contenedor descargado del mismo repositorio (jnogues/iperf), se lanza en una máquina con Ubuntu OS como se muestra en la siguiente figura, que a diferencia del caso anterior se crea sin descargar la imagen, pero al detectar que no existe, se descarga automáticamente.

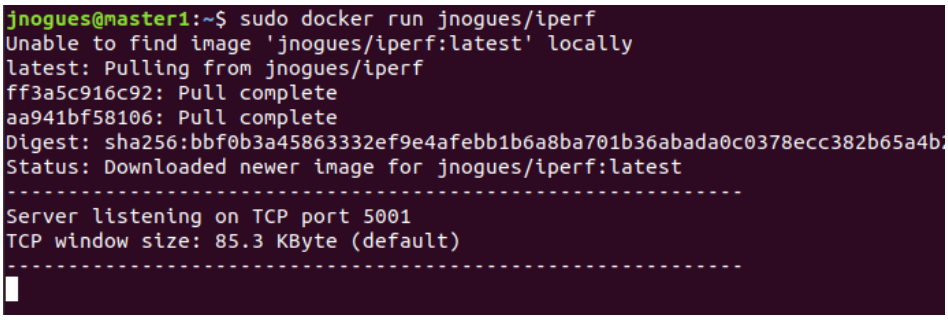


Figura 23. Captura de pantalla de creación y ejecución de contenedor iperf

Como se puede ver con la misma imagen de iperf, descargada del repositorio, se han lanzado dos servicios de medición de ancho de banda en dos OS distintos, sin problema alguno. Esto nos confirma la portabilidad de este sistema.

3.1.2 Kubernetes en nube privada.

Esta opción permite crear un clúster en nuestra red privada usando cualquier tipo de máquinas ya sean virtuales o físicas incluso contenedores que virtualizan máquinas.

Minikube

Figura 24. Logo de minikube

Como su nombre indica (“mini”) es una versión ligera de Kubernetes, y lo que hace es crear una máquina virtual en el ordenador para ejecutar un clúster con un solo nodo.

En la siguiente figura se muestra como al arrancar Minikube se crea un clúster de un único nodo con las funcionalidades básicas del sistema, configurando el clúster de Kubernetes y arrancando los componentes.

```
[MacBook-Pro-de-JAVIER:~ jnogues$ minikube start
There is a newer version of minikube available (v0.28.2).
  Download it here:
https://github.com/kubernetes/minikube/releases/tag/v0.28
.2

To disable this notification, run the following:
minikube config set WantUpdateNotification false
Starting local Kubernetes v1.10.0 cluster...
Starting VM...
Getting VM IP address...
Moving files into cluster...
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
[Kubectl is now configured to use the cluster.
Loading cached images from config file.
MacBook-Pro-de-JAVIER:~ jnogues$ kubectl get node
NAME       STATUS    ROLES     AGE       VERSION
minikube   Ready     master    87d       v1.10.0
```

Figura 25. Captura de arranque de Minikube y obtención de nodos.

Este escenario es muy útil para empezar a usar Kubernetes ya que dispone de las operaciones básicas de manejo de un clúster como son: arranque, detención, estado y eliminación. Lo que hace que, de una manera sencilla, sea fácil acostumbrarse al uso de los comandos. Así ha sido una herramienta muy útil en los comienzos prácticos del proyecto y por eso es importante conocer de su existencia antes de empezar a usar Kubernetes.

Además, es una herramienta básica y útil de cara desarrollar aplicaciones, antes de ponerlas en un entorno más potente o avanzado, ya que asegura un funcionamiento básico correcto [40].

Kubeadm

Es un escenario más avanzado que Minikube, usando varios nodos, con el que se pueden probar las capacidades de Kubernetes básicas y suficientes para asegurar que un despliegue funcione de manera correcta.

Gracias a esta forma de implementar un entorno de Kubernetes, se puede crear un clúster en nuestra red local, para lo que es necesario tener creados nodos previamente. Así para formar un clúster con el que se puedan probar las características de Kubernetes, como mínimo, se necesitan dos nodos y un máster. Cada nodo se añade al clúster a través del terminal al clúster.

Ya que se busca simular un escenario real, este escenario se plantea en base a nodos de maquinar virtuales, aunque podría realizarse con contenedores.

Kubeadm está en contante evolución, actualmente en su versión beta y aún no dispone de servicios de tipo *LoadBalancer*. Esto provoca que cuando se crea un servicio de este tipo la dirección IP externa (EXTERNAL IP) se queda a la espera de ser asignada, como se demuestra en la siguiente figura.

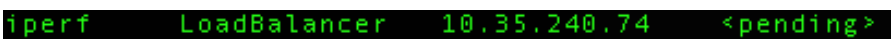


Figura 26. Captura de pantalla del servicio LoadBalancer en kubeadm

Otra característica que está preparándose para trabajar de forma estable, es la implementación de federaciones, actualmente en versión alpha [41] [42] [43].

3.1.3 Kubernetes en la nube pública

Se comparan, aunque hay más opciones, dos plataformas de nube pública. Siendo estas las plataformas de Google y Amazon, ya que son las más populares en este momento.

Google Cloud Platform (GCP)

Google Cloud Platform (GCP) es el escenario que ofrece Google para implementar Kubernetes en su nube pública. Proporciona drivers de balanceo, para poder prestar servicios del tipo “LoadBalancer”, ofreciéndonos una IP externa con la que acceder al servicio desde fuera de la nube. También al desplegar un clúster, proporciona plugin de red para que la comunicación dentro del clúster sea posible. Además, ofrece tecnología multi-zona con la que se puede crear clúster con nodos en diferentes zonas geográficas que, junto a herramientas de comprobación de actividad, ofrecen alta disponibilidad.

Otra gran característica es la facilidad de añadir nuevos nodos al clúster automáticamente, en caso de necesitarlos. Por ejemplo, si se da el caso de un pod que encuentra recursos en los nodos disponibles. Para lo cual se indican previamente los límites de este escalado, como se muestra en la siguiente figura.

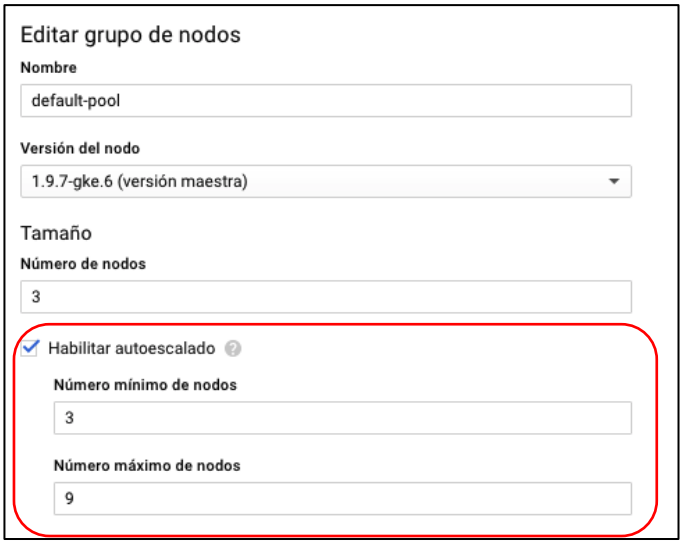


Figura 27. Menú de GCP para editar grupo de nodos

Google usa Google Compute Engine (GCE), que gestiona las instancias de las máquinas virtuales para los nodos necesarios en los clústeres, y Google Kubernetes Engine (GKE) para la gestión de los clústeres [44].

Amazon Web Services AWS

Es la solución de Amazon para prestar servicio en nube de Kubernetes. Amazon permite que los clústeres creados puedan acceder no solo a la API de Kubernetes, también a los servicios de Amazon con su propia API.

AWS no dispone, en principio, de herramientas de administración de clústeres pero soporta la instalación de la herramienta “kops” que permite solventar esta deficiencia. Sin embargo, actualmente existe el “Servicio de Contenedores Elásticos para Kubernetes” (EKS) que como se muestra en la siguiente figura, hace que se pueda administrar un clúster en AWS [45].

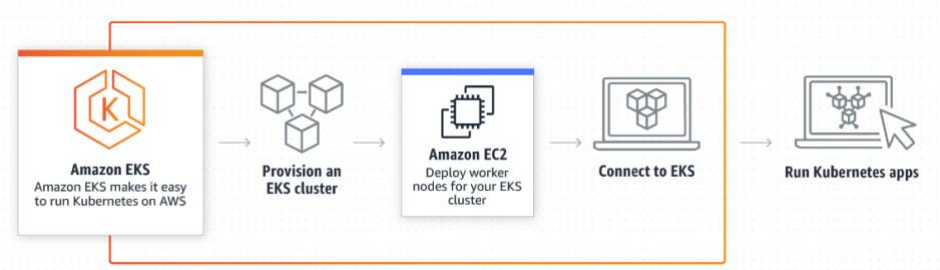


Figura 28. Diagrama del funcionamiento de EKS en AWS [45] .

Al igual que Google, AWS también ofrece clústeres de alta disponibilidad creando nodos en múltiples zonas geográficas. AWS crea además un máster por zona, que luego tienen que coordinarse [46].

Figura 29. Logo de Amazon EKS.

3.1.4 Comparativa de las características de los escenarios

A continuación, se compara las características de los escenarios estudiados, para así poder elegir el escenario correcto.

Para la siguiente tabla se usa un sistema donde:

-  : dispone de la característica.
-  : dispone de la característica, pero a través de complementos.
-  : no dispone de la característica.

	LoadBalancer	Auto escalado de nodos	Clúster HA	Herramienta de consola	Servicio de instalación de clústeres
Minikube					
Kubeadm					
GCP					
AWS					

Tabla 1. Tabla de comparativa de prestaciones de diferentes escenarios en Kubernetes.

Viendo la tabla, se puede ver que los escenarios Minikube y Kubeadm, parten en desventaja de cara a elegir uno de ellos, pero hay que entender que son escenarios de aprendizaje y en desarrollo, respectivamente, por comunidades de desarrolladores.

En cuanto a los escenarios de nube pública (GCP y AWS) se puede ver que, como plataformas formales y profesionales, cuidan al detalle que Kubernetes se ofrezca con las prestaciones optimas y deseadas.

3.2 Preparación del escenario escogido

Para el análisis de las características de Kubernetes se ha escogido el escenario en la nube pública de Google, ya que, entre los escenarios de nube pública, nos ofrece todas las prestaciones de forma nativa.

Esta decisión se basa en que este escenario nos permite hacer pruebas con multitud de nodos sin necesidad de sobrecargar un ordenador con máquinas virtuales, que además no permite estudiar realmente un sistema descentralizado con fidelidad, algo que las plataformas de nube pública si permite, ya que se puede desplegar un clúster en diferentes zonas geográficas. Además, si en caso de necesitar crear un entorno parecido, necesitaríamos contar con varios equipos, infraestructura de la que no se dispone[1].

Dentro de las opciones de nube pública, Google Cloud Platform, ofrece la seguridad de un funcionamiento correcto, ya que Kubernetes es una tecnología creada por Google. Esto hace que la conozcan mejor que otras compañías y, además, hace que sea interesante estudiar el escenario de la compañía que creó este modo de orquestación de contenedores.

3.2.1 Antes de empezar

Primero se necesita preparar algunas herramientas software, tanto en la máquina a usar, como registros en cuentas de plataformas de nube pública para el escenario escogido, y herramientas Docker para repositorios de contenedores.

[1] La Universidad presta equipos informáticos en laboratorios de acceso restringido, pero no nos ofrece exclusividad para las pruebas. Además, ante la posible necesidad de realizar alguna prueba de última hora durante periodos de inactividad de la Universidad, se ha descartado la opción de su uso.

Herramientas hardware

Es necesario disponer de un ordenador con conexión a internet, ya que se necesita acceder a los servicios de Google Cloud para nuestro despliegue.

El equipo tiene que cumplir con las especificaciones de requisitos de las herramientas software se utilizan para el proyecto, eligiendo así el más riguroso.

Para este proyecto, se usa un portátil con macOS, lo que analizando los requisitos tenemos que es necesario como mínimo una máquina con 4GB de RAM y modelo de 2010 o posterior. Ambos requisitos se cumplen, ya que se dispone de modelo de 2011 y de 16GB de RAM.

Docker CE

Tenemos dos distribuciones de Docker para elegir la instalación. Docker Comunity Edition (Docker CE) y Docker Enterprise Edition (Docker EE).

Docker EE, está orientada a empresas que necesiten una plataforma de contenedores para sus aplicaciones. Docker CE, es la que usaremos y creará el entorno de desarrollo necesario para nuestras aplicaciones empaquetadas en contenedores Docker.

Así para la instalación debemos conocer los requisitos que nos exige el programa que son y de donde descargarse el programa, en función del SO elegido. En nuestro caso se ha usado macOS.

- Requisitos del sistema [47]:

  - 4GB de memoria RAM
  - Modelo de 2010 o posterior.
  - macOS El Capital (10.11) o posterior.
  - VirtualBox posterior a la versión 4.3.30

A la hora de descargar el programa se pide identificación para poder comenzar la descarga, así que primero hay que tener creada una cuenta Docker ID. Después se ejecuta el fichero descargado y se arrastra como indica la siguiente figura.

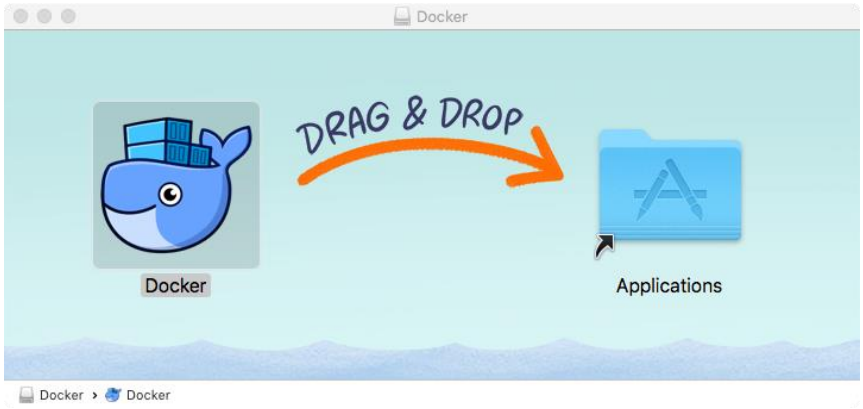


Figura 30. Instalación de Docker CE en macOS.

Ya solo hay que ejecutar la aplicación e identificarse con Docker ID. Para esto hay que ir al icono que aparece a la derecha en la barra superior [48].

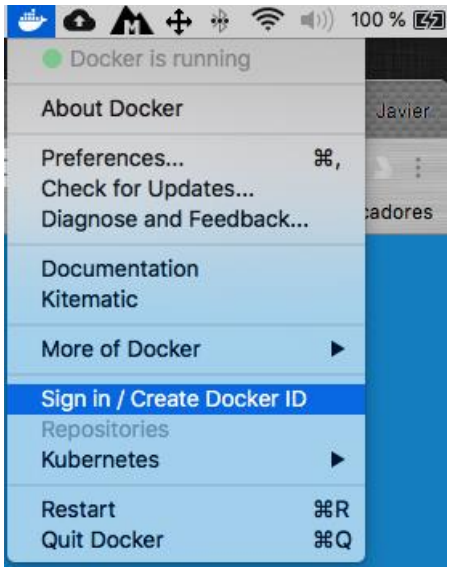


Figura 31. Menú de Docker CE en barra superior de macOS.

Kubectl

Para instalar la CLI de Kubernetes solo hay que introducir en el terminal [49]:

```
$ brew install Kubernetes-cli
```

Cuenta GCP

A través de la web de Google Cloud, se elige probar gratis. Esto lleva a un menú para acceder o crear cuenta nueva usando un correo de Gmail[2] [50].

Una vez ya dentro de GCP aparece la opción de activar el periodo de prueba, se acepta y se siguen los pasos. Se advierte, en la misma página, de las condiciones del periodo de prueba como se muestra en la siguiente figura.

Figura 32. Imagen de la oferta de prueba de GCP.

[2] Se usa uno nuevo personal, ya que el de la Universidad no permite el periodo de prueba gratuito.

SDK de Google Cloud

SDK se utiliza para el usar la shell de Google en la máquina local usada para la gestión del clúster. Conlleva la instalación de la CLI para los sistemas de GCP, como se especifica a continuación.

Instalación común para Linux y macOS:

- Así primero en el terminal se escribe:

```
$ curl https://sdk.cloud.google.com | bash
```

Y se acepta las preguntas como se muestra en las siguientes capturas.

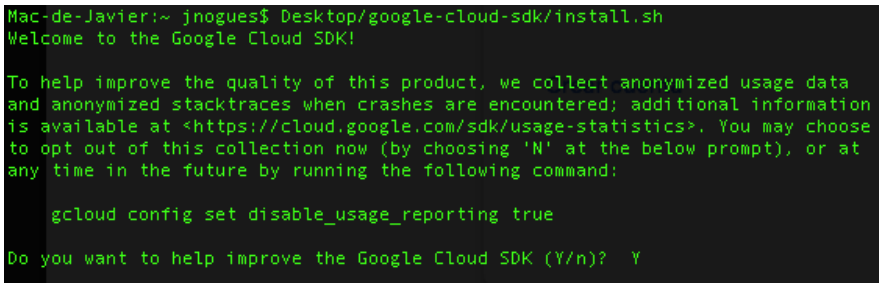


Figura 33. Captura de la instalación de SKD de Google Cloud.

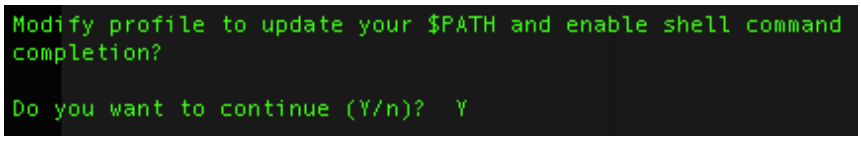


Figura 34. Captura de la modificación de shell para GCP.

- Luego tenemos que reiniciar el shell para poder usar la CLI.

```
$ exec -l $SHELL
```

- Ya solo queda acceder a la cuenta con:

```
$ gcloud auth login
```

Así se lanza automáticamente un navegador web para iniciar sesión, o crear una cuenta nueva. Con la cuenta de GCP, se puede iniciar sesión [51].

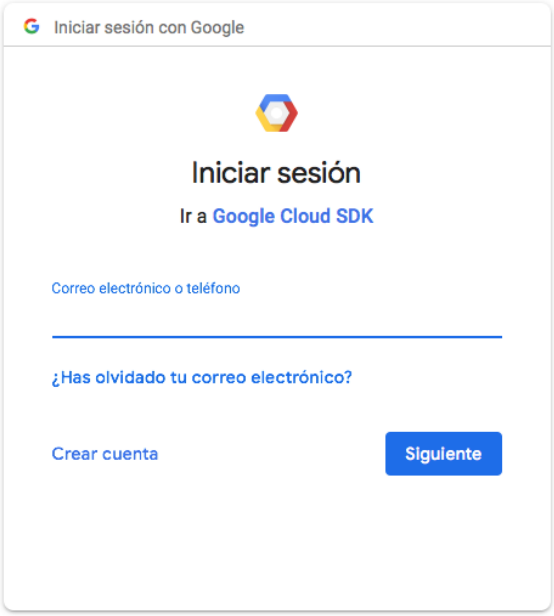


Figura 35. Menú para inicial sesión.

3.2.2 Aprendiendo a usar las herramientas

Ya con las herramientas necesarias instaladas, hay que entender como funcionan para poder desarrollar el estudio.

Así para el uso de contenedores se necesita aprender el uso de Dockerfile y CLI de Docker. De igual modo para orquestar estos contenedores en Kubernetes se necesita conocer el funcionamiento del CLI de Kubernetes, es decir, de Kubectl.

DockerFile

Para la creación de la imagen, en la línea de comandos de un terminal, en la máquina donde tengamos instalado Docker se introduce la instrucción “Docker build”.

Esta instrucción usa el fichero Dockerfile para crear la imagen usando otros ficheros que se encuentren en la misma ruta para crear un contexto en el caso de que fuera necesario. La ruta puede pertenecer a nuestra máquina o una URL de repositorios. A continuación, se muestra un ejemplo del formato de un fichero Dokerfile.

```
# Comentario

INSTRUCTION argumentos

....

INSTRUCTION argumentos
```

Figura 36. Formato del fichero Dockerfile

Instrucciones básicas:

- **ARG:** define argumentos a usar en futuras instrucciones.
  Por ejemplo: `ARG VERSION=latest`

- **FROM:** indica la imagen base que se usará para construir el contenedor, es la primera instrucción que hay en un Dockerfile, aunque puede ir precedida por ARG.

- **RUN:** ejecuta los comandos que se establecen en el argumento de esta instrucción.

- **CMD:** solo puede haber una por fichero y se encarga de aportar valores para un contenedor en ejecución, es decir, ejecuta la instrucción pasada como argumento cuando el contenedor es lanzado.

- **WORKDIR:** establece el directorio de trabajo para las instrucciones que le preceden, y en caso de no existir lo crea.

- **ADD:** esta instrucción tiene dos argumentos y añade el contenido de una ruta (primer argumento) a la ruta en el contenedor (segundo parámetro).

- **USER:** establece el usuario y el grupo que se usarán para ejecutar el contenedor. Es importante destacar que si no se establece esta instrucción, el contenedor trabajará en el grupo de root.

- **EXPOSE:** habilita un puerto del contenedor para conexiones.

- **ENV:** establece variables de entorno para el contenedor. Esta instrucción también tiene dos argumentos, siendo el primero el nombre de la variable y el segundo el valor de la variable.

En la siguiente imagen, a modo de ejemplo, se muestra el contenido de un fichero Dockerfile, con algunas de las instrucciones definidas anteriormente. En este ejemplo se define un contenedor que funcionará como un servidor iperf, para la medición del ancho de banda de redes IP [52].

```
FROM alpine:3.7
MAINTAINER Javier Nogués <javier.nogues@alumnos.uc3m.es>

# Install iperf
RUN apk add --update iperf

# Make port 5001 available
EXPOSE 5001
```

```
# Run the server
CMD ["iperf", "-s"]
```

Fichero 1. Ejemplo de fichero Dockerfile

CLI de docker

Se usa la siguiente sintaxis:

```
$ docker [opciones] comando [opciones del comando]
```

Argumentos de la sintaxis:

- Opciones

Así en el campo de opciones se configuran opciones del modo de ejecución del comando, por ejemplo, si se quiere cambiar la dirección donde docker almacena los ficheros de configuración para el comando “run” este primer argumento se configura de la siguiente forma:

```
$ docker –config ~/directorio/ run
```

Para ver las posibles opciones escribimos “docker” en el CLI y se muestran como, se ve en la siguiente figura, la lista de estas.

```
MacBook-Pro-de-JAVIER:~ jnogues$ docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/Users/jnogues/.docker")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/Users/jnogues/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/Users/jnogues/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/Users/jnogues/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit
```

Figura 37. Captura de pantalla de la lista de opciones del CLI de docker

- Comando y opciones del comando

Para cada comando introducido hay varios flags a usar y la sintaxis difiere. Para conocer la correcta sintaxis de cada comando podemos introducir el flag de ayuda:

```
$ docker comando --help
```

Por ejemplo, en la siguiente figura se muestra un trozo de la salida de la ayuda, del comando run.

```
MacBook-Pro-de-JAVIER:~ jnogues$ docker run --help

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:
      --add-host list                  Add a custom host-to-IP mapping (host:ip)
  -a, --attach list                    Attach to STDIN, STDOUT or STDERR
      --blkio-weight uint16            Block IO (relative weight), between 10 and 1000, o
      --blkio-weight-device list       Block IO weight (relative device weight) (default
      --cap-add list                   Add Linux capabilities
      --cap-drop list                  Drop Linux capabilities
      --cgroup-parent string           Optional parent cgroup for the container
      --cidfile string                 Write the container ID to the file
      --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler) period
      --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler) quota
      --cpu-rt-period int              Limit CPU real-time period in microseconds
      --cpu-rt-runtime int             Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int                 CPU shares (relative weight)
      --cpus decimal                   Number of CPUs
      --cpuset-cpus string             CPUs in which to allow execution (0-3, 0,1)
      --cpuset-mems string             MEMs in which to allow execution (0-3, 0,1)
  -d, --detach                         Run container in background and print container ID
      --detach-keys string             Override the key sequence for detaching a containe
      --device list                    Add a host device to the container
      --device-cgroup-rule list        Add a rule to the cgroup allowed devices list
      --device-read-bps list           Limit read rate (bytes per second) from a device (
      --device-read-iops list          Limit read rate (IO per second) from a device (def
      --device-write-bps list          Limit write rate (bytes per second) to a device (d
      --device-write-iops list         Limit write rate (IO per second) to a device (defa
      --disable-content-trust          Skip image verification (default true)
      --dns list                       Set custom DNS servers
      --dns-option list                Set DNS options
      --dns-search list                Set custom DNS search domains
```

Figura 38. Captura de pantalla de las opciones del comando "run" del CLI de docker.

Como se ve en esta figura nos muestra su sintaxis y las opciones del comando, donde solo se muestran algunas [53].

CLI de Kubernetes: kubectl

Kubectl usa la sintaxis base:

```
$ kubectl [comandos] [tipo de objeto] [nombre] [flags]
```

En los siguientes apartados, se explica cada argumento de la herramienta.

- Comandos

Donde “comandos” especifica la acción que queremos realizar. Hay que destacar que, en función del comando, puede que la estructura de la sintaxis explicada anteriormente varíe.

En la siguiente figura se muestra, con la ayuda de un terminal y la función autocompletar de Kubectl, una lista de los comandos

Figura 39: Captura de terminal de los comandos para Kubectl.

- Tipo de objeto

En este argumento se selecciona el tipo de recurso que sufrirá la acción especificada por el argumento anterior.

Puede seleccionarse cualquier objeto que se explica anteriormente en este documento u otros que no son necesarios para el proyecto.

A continuación, se muestra una figura donde se muestran para el comando “get” (listar recurso), los recursos disponibles.

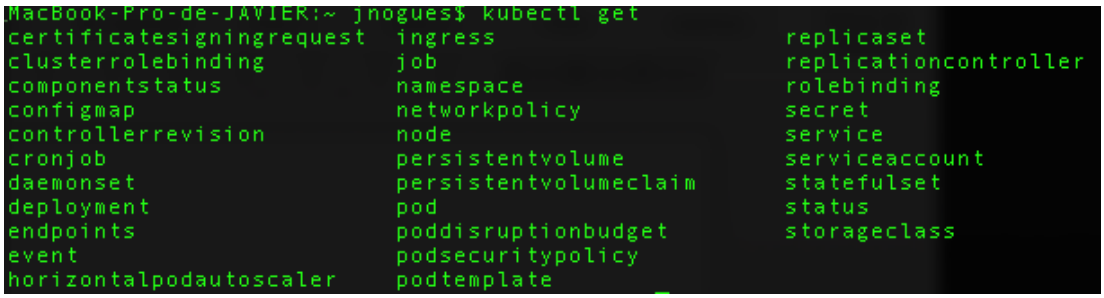


Figura 40. Captura de terminal de los recursos para el comando “get”.

- Nombre

Argumento que selecciona el nombre del objeto que se quiere que sea afectado por la acción seleccionada en el argumento “comando”.

Para seleccionar un fichero de configuración hay que introducir como argumento previo -f, por ejemplo:

```
$ kubectl create -f service.yaml
```

Precisamente en este objeto se ve como no se selecciona el tipo de objeto, ya que “service.yaml” ya lo tiene definido, y al ejecutarse “create” se adapta la sintaxis al comando.

- Flags

Son opciones, por ejemplo, para seleccionar los pods que se ejecutan en un espacio de nombres. Por ejemplo, se puede añadir un flag “--namespace”, como se ve en la siguiente figura, en la que se muestras los flags disponibles para el comando “get”.

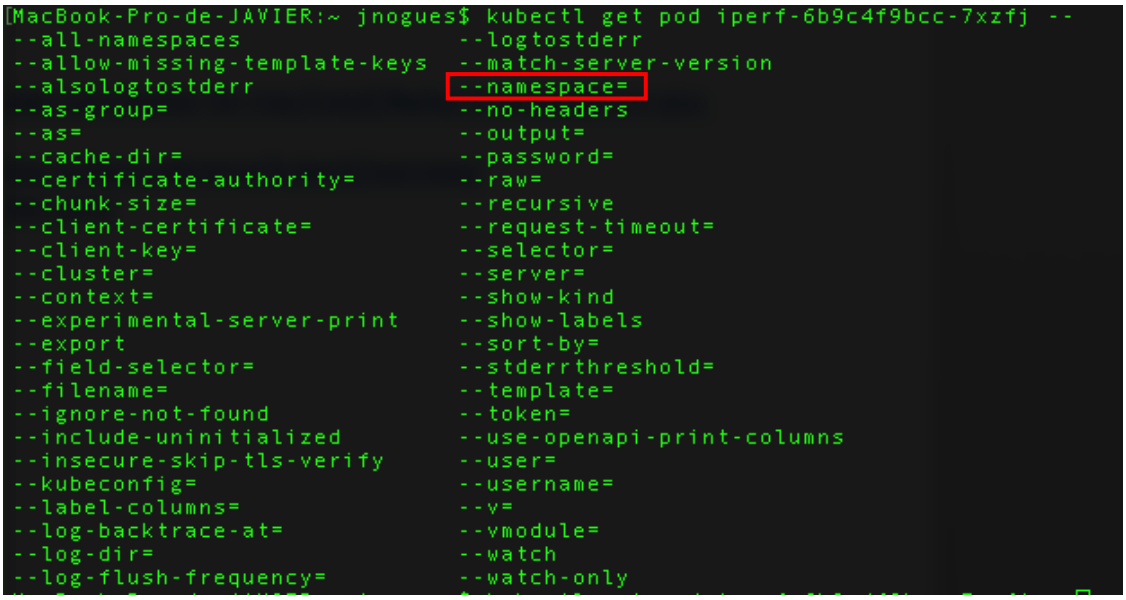


Figura 41. Captura de terminal de los flags para el comando "kubectl get pod iperf".

- Añadir formato de salida de la información

Se puede controlar la forma en la que Kubectl muestra la información, añadiendo a la sintaxis de entrada de una petición "-o" quedando la sintaxis de la siguiente forma:

```
$ kubectl [comandos] [tipo de objeto] [nombre] [flags] -o=<formato de salida>
```

Esta opción viene muy bien para parametrizar implementaciones y servicios que se están ejecutando y que hemos ido configurando con Kubectl, dándole el formato de salida YAML [34].

3.2.3 Creación del clúster

Se accede a Google Cloud Platform iniciando sesión con la ID creada anteriormente, y accediendo en “Selecciona un proyecto”, seleccionamos la opción de nuevo proyecto como se muestra en los cuadros rojos de la siguiente figura.

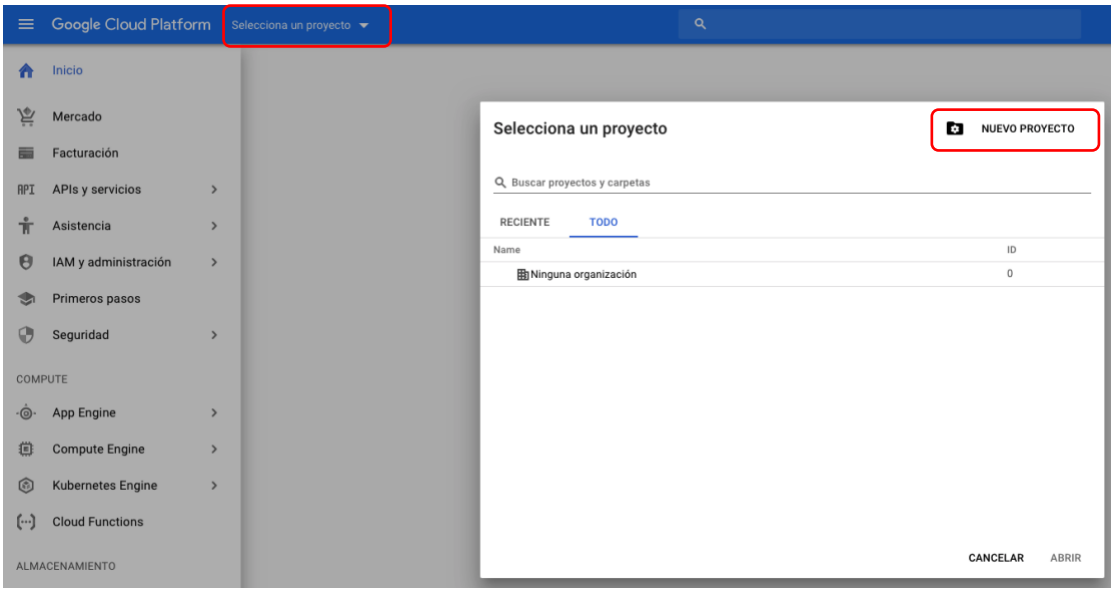


Figura 42.Menú de GCP para crear un proyecto.

Para crear el proyecto se rellenan los campos como se muestra en la siguiente figura y se selecciona “crear”.

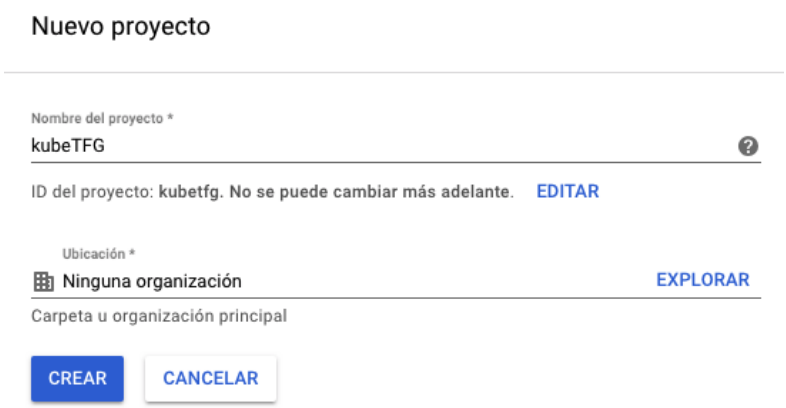


Figura 43. Menú de GCP para crear un proyecto nuevo (kubeTFG).

Ya con el proyecto creado, se crea el clúster que se usará para probar las características de Kubernetes estudiadas y las motivaciones del proyecto.

El clúster que se creará es de alta disponibilidad, configurándolo para que reparta los pods en una región (europe-west3), que dispone de tres zonas. Se configura también para que se desplieguen un nodo por zona y que, en caso de ser necesario se escale automáticamente hasta tres nodos por zona con auto reparación de nodos, los cuales serán máquinas con CoreOS[3] y discos de 100GB por nodo.

Así en el terminal del ordenador que se usa se escribe el siguiente comando:

```
$ gcloud beta container --project "kubetfg" clusters create "cluster-tfg" --region "europe-west3" --username "admin" --cluster-version "1.9.7-gke.6" --machine-type "n1-standard-1" --image-type "COS" --disk-type "pd-standard" --disk-size "100" --scopes "https://www.googleapis.com/auth/compute","https://www.googleapis.com/auth/devstorage.read_only","https://www.googleapis.com/auth/logging.write","https://www.googleapis.com/auth/monitoring","https://www.googleapis.com/auth/servicecontrol","https://www.googleapis.com/auth/service.management.readonly","https://www.googleapis.com/auth/trace.append" --num-nodes "1" --enable-cloud-logging --enable-cloud-monitoring --network "projects/kubetfg/global/networks/default" --subnetwork "projects/kubetfg/regions/europe-west3/subnetworks/default" --enable-autoscaling --min-nodes "1" --max-nodes "3" --addons HorizontalPodAutoscaling,HttpLoadBalancing,KubernetesDashboard --no-enable-autoupgrade --enable-autorepair
```

[3] CoreOS es un OS ligero que ha sido creado para nodos de clústeres, por su seguridad y ligereza.

Esto nos proporciona la siguiente salida:

```
Creating cluster cluster-tfg...done.
Created [https://container.googleapis.com/v1beta1/projects/kubetfg/zones/europe-west3/clusters/cluster-tfg].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud
/europe-west3/cluster-tfg?project=kubetfg
kubeconfig entry generated for cluster-tfg.
NAME         LOCATION      MASTER_VERSION  MASTER_IP       MACHINE_TYPE   NODE_VERSION  NUM_NODES  STATUS
cluster-tfg  europe-west3  1.9.7-gke.6     35.234.121.45   n1-standard-1  1.9.7-gke.6   3          RUNNING
MacBook-Pro-de-JAVIER:~ jnogues$ kubectl get node
NAME                                          STATUS    ROLES     AGE       VERSION
gke-cluster-tfg-default-pool-3e5d23b7-1fnd    Ready     <none>    1m        v1.9.7-gke.6
gke-cluster-tfg-default-pool-4e3013ab-nmgb    Ready     <none>    1m        v1.9.7-gke.6
gke-cluster-tfg-default-pool-be83c988-2nmg    Ready     <none>    1m        v1.9.7-gke.6
MacBook-Pro-de-JAVIER:~ jnogues$
```

Figura 44. Captura de la salida en terminal de la creación de un clúster.

Como se muestra en la figura anterior, se asigna un nodo extra que hará la función de master y coordinará cada zona, formando entre todas las zonas un clúster.

Nombre	Zona	Recomendación	IP interna	IP externa	Conectar
gke-cluster-tfg-default-pool-3e5d23b7-1fnd	europe-west3-b		10.156.0.4 (nic0)	35.242.214.216	SSH
gke-cluster-tfg-default-pool-4e3013ab-nmgb	europe-west3-c		10.156.0.2 (nic0)	35.242.242.39	SSH
gke-cluster-tfg-default-pool-be83c988-2nmg	europe-west3-a		10.156.0.3 (nic0)	35.242.214.8	SSH

Figura 45. Imagen de los nodos del clúster en GCP.

3.3 Análisis de características de Kubernetes

En esta sección se aborda el análisis de las características estudiadas, comprobando que se producen en el escenario escogido, según lo estudiado en el capítulo del estado del arte y que cubren la motivación del proyecto.

3.3.1 Balanceo de carga y escalado

Para probar el funcionamiento de balanceo de carga se instala un servidor web Nginx y se configura para que tenga tres réplicas en tres nodos diferentes. Luego se expone para prestar servicio, como se muestra a continuación.

```
1  $ kubectl create deployment web --image=nginx:alpine
2  $ kubectl scale deployment web --replicas=3
3  $ kubectl expose deployment web --type="LoadBalancer" --port=80
```

En la línea 1 se crea el “deployment” web cargando la imagen docker de Nginx con imagen base de Alpine.

En la línea 2 se escala la aplicación para que ejecuten tres réplicas del pod que ejecuta la aplicación Nginx. En esta línea se ve como una aplicación que solo dispone de una réplica pasa a prestar servicios a través de tres réplicas con tan solo una línea en el CLI de Kubernetes. Lo que nos da la oportunidad de observar que en caso de que con un número de replicas iniciales nuestro servicio esta saturado, aumentar las réplicas para que entre todas se preste el servicio de manera correcta.

Y en la línea 3 se expone el servicio de tipo LoadBalancer para que el balanceador de carga de GCP reparta el tráfico cuando se solicite el servicio, entre los nodos que tienen pods del servicio solicitado, y se asigne una IP externa para poder acceder desde fuera del clúster. En la siguiente figura se muestra como esto se cumple.

```
MacBook-Pro-de-JAVIER:~ jnogues$ kubectl get service web
NAME      TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
web       LoadBalancer   10.35.243.109   35.233.61.107   80:32397/TCP   2h
```

Figura 46. Captura del terminal de servicio LoadBalancer en GCP.

LoadBalancer sabe que nodos disponen de pods del servicio solicitado a través de la etiqueta “end-points”, donde sus valores son las direcciones IP de los pods.

Ya que tenemos el clúster vacío y el servicio no requiere de recursos de manera excesiva, cada réplica se ejecuta en un nodo diferente, como se muestra en la siguiente figura.

```
MacBook-Pro-de-JAVIER:~ jnogues$ kubectl get pod -o wide
NAME                   READY     STATUS    RESTARTS   AGE       IP
web-5d4cd76d78-lrlm9   1/1       Running   0          3h        10.32.0.6
web-5d4cd76d78-tdwxl   1/1       Running   0          3h        10.32.2.6
web-5d4cd76d78-thxx7   1/1       Running   0          3h        10.32.1.7

IP           NODE
10.32.0.6    gke-proyecto-kubernetes-default-pool-be400273-qf94
10.32.2.6    gke-proyecto-kubernetes-default-pool-9424e2fe-5qc4
10.32.1.7    gke-proyecto-kubernetes-default-pool-74d2df24-86ws
```

Figura 47[4]. Captura del terminal de los pods del clúster.

A continuación, se modifica la web que cada réplica sirve bajo el mismo servicio, para que al acceder a este se balancee la carga y se detecte el balanceo en función de la web cargada. Primero se accede al contenedor de un pod, no es necesario especificar a que contenedor porque los pods de Nginx solo tienen uno. A continuación, se muestra como.

```
1  $ kubectl exec -it web-5d4cd76d78-lrlm9 -- /bin/sh
2  $ /usr/share/nginx/html/
3  # vi index.html
```

[4] En la figura se ha partido la imagen para que se muestre de manera legible, pero la imagen inferior es la continuación de la imagen superior. Se ha mantenido en la imagen inferior el campo IP para que sirva de unión, a la hora de entender la imagen completa

Así en la línea 1 y 2 se accede al contenedor y a la ruta donde está el fichero HTML a servir y en la línea 3 se muestra como acceder al fichero para modificarlo con el editor de texto “vi” como se muestra a continuación.

```
<!DOCTYPE html>
<html>
<head>
<title>web1</title>
<style>
   body {
      width: 35em;
      margin: 0 auto;
      font-family: Tahoma, Verdana, Arial, sans-serif;
   }
</style>
</head>
<body>
<h1>Welcome to nginx 1!</h1>
</body>
</html>
```

Fichero 2. index.html

Para los otros dos nodos se hace lo mismo, pero cambiando los números del documento 1 por 2 y 3.

Para probar que se realiza el balanceo de carga de forma efectiva solo queda acceder al servicio, con su dirección externa (*External-IP*) y probar que se van alternando las webs servidas. Así se ha probado tanto a través de un navegador, como del terminal usando el comando “curl”, como se muestra en las siguientes figuras.

```
MacBook-Pro-de-JAVIER:~ jnogues$ curl 35.233.61.107:80
<!DOCTYPE html>
<html>
<head>
<title>Web1</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx 1!</h1>
</body>
</html>
MacBook-Pro-de-JAVIER:~ jnogues$ curl 35.233.61.107:80
<!DOCTYPE html>
<html>
<head>
<title>Web2</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx 2!</h1>
</body>
</html>
MacBook-Pro-de-JAVIER:~ jnogues$ curl 35.233.61.107:80
<!DOCTYPE html>
<html>
<head>
<title>web3</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx 3!</h1>
</body>
</html>
MacBook-Pro-de-JAVIER:~ jnogues$
```

Figura 48. Captura del terminal de la salida de petición web de servicio de tipo LoadBalancer..

Figura 49. Captura del navegador web de servicio web de tipo LoadBalancer.

3.3.2 Gestión de recursos

Se dispone de varias formas de gestionar los recursos, así primero se creará un nuevo espacio de nombres para gestionar los pods como se muestra en la línea 1.

```
1  $ kubectl create namespace tfg
2  $ kubectl config set-context $(kubectl config current-context) --namespace=tfg
```

En la línea 2 se configura el contexto para que de ahora en adelante Kubectl realice las operaciones por defecto en el nuevo espacio de nombres.

Para gestionar los recursos de la zona creamos el siguiente fichero de definición del tipo de objeto *ResourceQuota.*

```
1  apiVersion: v1
2  kind: ResourceQuota
3  metadata:
4      name: tfg
5  spec:
6      hard:
7          pods: "5"
```

Fichero 3. rq_tfg.yaml

Este fichero hay que destacar la línea 7, donde se especifica el límite de pods que se ejecutarán en el espacio de nombres definido en la línea 4, es decir, en el “namespace tfg”.

Para gestionar los recursos de los contenedores que se ejecuten sin límites, se configura el límite de recursos de memoria a través del objeto *LimitRange* como se muestra en el siguiente fichero de definición:

```
1  apiVersion: v1
2  kind: LimitRange
3  metadata:
4     name: mem-limit-range
5  spec:
6     limits:
7     - default:
8           memory: 1Gi
9       defaultRequest:
10          memory: 512Mi
11      type: Container
```

Fichero 4. lr_tfg.yaml

En el fichero en la línea 4 se especifica el tipo de recurso que va a ser gestionado, siendo el de memoria.

En la línea 6, se comienza con las especificaciones que afectarán a los contenedores, como se indica en la línea 11, que no tengan configurados gestión de recursos de memoria.

Así en la línea 6 se indica que se van a definir los límites, siendo el límite de memoria especificado en la línea 8 (1GB). Y como se indica en la línea 9, también se definen los recursos asignados en la creación de los contenedores, que se especifica en la línea 10 (512MB).

Para el proyecto no se usará *LimitRange* para controlar los recursos de CPU, ya que se quiere que use lo necesario.

Así para crear estos objetos, encargados de la gestión de los recursos se ejecutan en un terminal los siguientes comandos, cargando así los ficheros descritos anteriormente:

```
1  $ kubectl create -f rq_tfg.yaml
2  $ kubectl create -f lr_tfg.yaml
```

Para ver si se han creado correctamente, se muestran las descripciones de los objetos creados. Así para el objeto *ResourceQuota*:

```
$ kubectl describe resourcequota tfg
```

Lo que nos da la salida de la siguiente figura:

```
Name:       tfg
Namespace:  tfg
Resource    Used  Hard
--------    ----  ----
pods        1     5
```

Figura 50. Captura del terminal de la descripción de ResourceQuota.

Se puede ver como el objeto existe y se ha configurado correctamente el límite de pods que se pueden ejecutar en el namespace tfg.

Para comprobar que funciona, se escala el despliegue para que tenga más de 7 réplicas, con el comando:

```
$ kubectl scale deployment web --replicas=7
```

Para comprobar cuantas réplicas se ejecutan, introducimos el comando:

```
$ kubectl get replicaset
```

Lo da la salida mostrada en la siguiente figura, donde se puede ver que, aunque se desean siete réplicas solo se están ejecutando cinco, luego “ResourceQuota” está funcionando correctamente para gestionar este recurso.

```
NAME             DESIRED   CURRENT   READY     AGE
web-5d4cd76d78   7         5         5         11h
```

Figura 51. Captura del terminal del estado del control de réplicas del clúster.

Para ver que se ha creado el objeto *LimitRange*, se introduce el comando:

```
$ kubectl describe limitrange mem-limit-range
```

Así se obtiene la salida de la siguiente figura, donde se ve que se ha configurado según lo especificado en el fichero de definición.

Figura 52. Captura del terminal de la configuración de los recursos de memoria.

Para observar que funciona se vuelve a realizar el despliegue de Nginx:

```
1  $ kubectl delete deployment web
2  $ kubectl create deployment web --image=nginx:alpine
```

En la línea 1 se ha eliminado el existente y en la línea 2 se ha vuelto a crear. Ahora para ver si se han asignado de forma correcta la asignación de los recursos se muestra la configuración del pod en ejecución con el comando:

```
$ kubectl describe pod web-5d4cd76d78-frw8j
```

En la siguiente figura se muestra (parcialmente) la salida que, del comando, donde se resalta en los cuadros rojos que *LimitRange* ha gestionado los recursos del pod. Así en el primer cuadro rojo se muestra la información de que *LimitRange* ha configurado los recursos de limite de memoria en el pod y el segundo muestra la configuración de los límites de memoria.

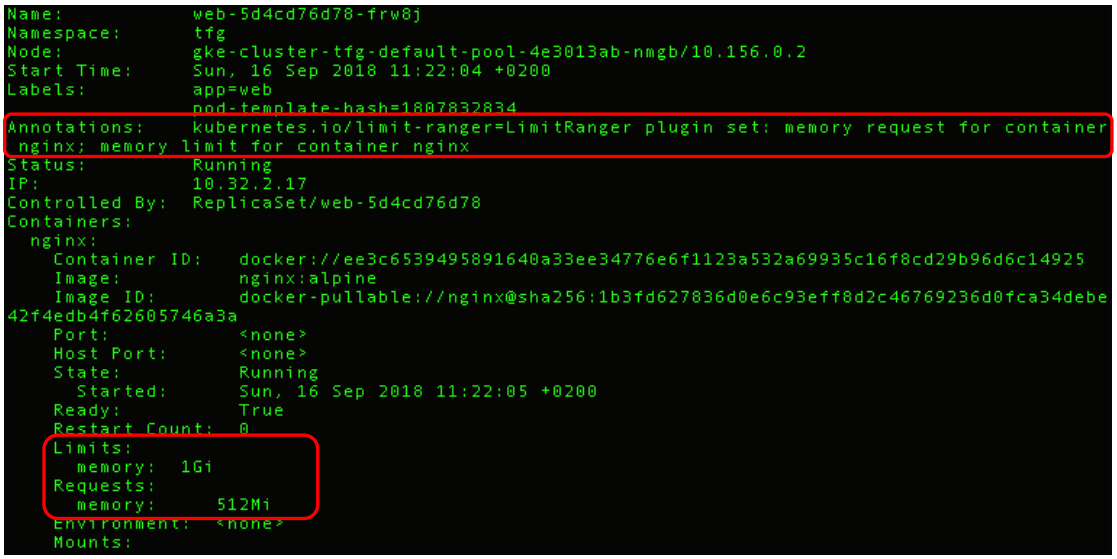


Figura 53. Captura del terminal de la descripción del pod "web".

3.3.3 Auto-recuperación

Se va a comprobar que *ReplicaSet* es capaz de controlar el número de réplicas que se ejecutan del despliegue.

Se comprueba la configuración actual de este objeto:

```
$ Kubectl get replicaset
```

Así se obtiene la siguiente salida, donde se muestra que hay tres réplicas ejecutándose.

Figura 54. Captura del terminal del estado del control de réplicas del clúster.

Comprobado el número de réplicas deseado, vamos a emular que un pod ha sido finalizado eliminándolo. Del mismo modo se monitorea por pantalla los cambios de

los pods de los nodos. Para esto se ejecuta en dos terminales diferentes los siguientes comandos:

Terminal 1	$ kubectl get pod -w
Terminal 2	$ kubectl delete pod web-5d4cd76d78-5k8gh

En el terminal 1 se obtiene la siguiente figura, de donde se puede leer el nombre de los pods que actualmente se están ejecutando en el clúster. Así se selecciona uno de ellos para en el terminal 2 eliminarlo para ver la reacción en el primer terminal, como se muestra en la siguiente figura.

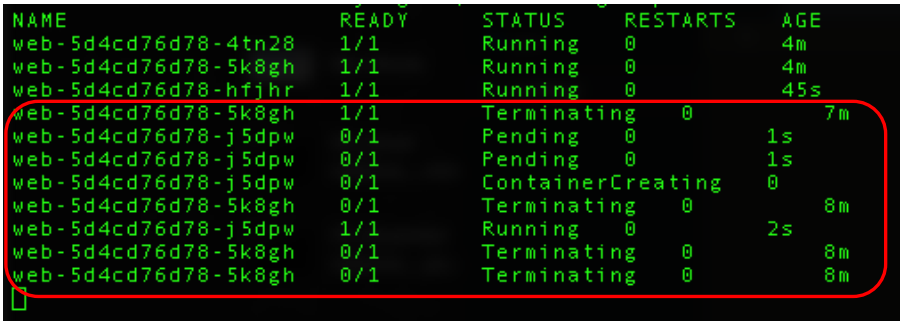


Figura 55. Captura del terminal de los pod que hay en el clúster.

En la figura se ha enmarcado en rojo la reacción a la eliminación de un pod. En la primera línea del cuadro, se ve como el pod elegido termina su ejecución y como se ha comprobado, instantáneamente el controlador *ReplicaSet* ha detectado que no se ejecutaban el número correcto de pod, planificando el despliegue de un nuevo pod, y asignando un nodo a este para crearlo como se muestra en las tres siguientes líneas de la misma figura.

Así vemos como se ha curado el clúster, a lo que adicionalmente GCP nos da la herramienta de recuperación de nodos, lo que hace que del clúster se mantenga bajo la configuración deseada.

3.4 Descentralización y actualización

En el apartado anterior no se ha estudiado como se actualizan las aplicaciones que se lanzan en el clúster, pero se estudiará en este apartado, después de analizar la descentralización de un servidor de hora.

Así partimos de un escenario donde disponemos de un servidor de hora con MQTT, centralizado en nuestra nube privada. El servidor se forma por dos componentes, el publicador, que informara de la hora del sistema al componente bróker, que se encarara de distribuir esa información a todos los suscriptores que estén suscritos a el servicio de hora. Para ello se utiliza el protocolo de mensajes MQTT, que es un protocolo de mensajes ligero pensado para la comunicación entre dispositivos IoT (Internet de las cosas). En la siguiente figura se muestra un esquema del servidor de hora centralizado.

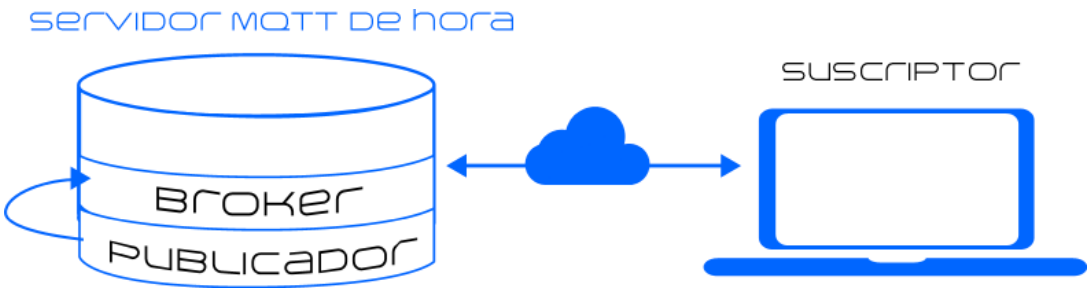


Figura 56. Esquema de la organización del servidor de hora

Para descentralizar y poder usar el servidor en Kubernetes y aprovechar el clúster creado, primero tenemos que empaquetar el programa “publish.c”[5] en docker y crear el contenedor docker del servicio bróker de MQTT.

[5] Programa disponible en el apartado de Anexos de este documento.

Así para crear los contenedores, primero se definen los Dokerfile asociados a cada uno:

```
1  # Se usa como imagen base una oficial de Ubuntu
2  FROM ubuntu
3
4  # Crea el directorio /app en el contenedor
5  WORKDIR /app
6
7  # Copia los ficheros del directorio actual al directorio del contenedor /app
8  ADD . /app
9
10 # Instala los paquetes que necesitamos para ejecutar el programa
11 RUN apt-get update
12 RUN apt-get install gcc -y
13 RUN apt-get install mosquitto -y
14 RUN apt-get install mosquitto-clients -y
15 RUN gcc -Wall -o publish publish.c
16
17 # Cuando el contenedor arranca se ejecuta ./publish
18 CMD ["./publish"]
```

Fichero 5. Dockerfile del contenedor del publicador.

Este fichero muestra como se configura el contenedor para la parte del programa encargado de publicar la hora.

```
1  # Se usa como imagen base una oficial de Ubuntu
2  FROM ubuntu
3
4  # Instala los paquetes que necesitamos para ejecutar MQTT broker
5  RUN apt-get update
6  RUN apt-get install mosquitto -y
7
8  # Se expone el puerto 1883 para acceder al broker
9  EXPOSE 1883
10
11 # Cuando el contenedor arranca se ejecuta MQTT
12 CMD ["mosquitto", "-v"]
```

Fichero 6. Dockerfile del contenedor del bróker.

Este fichero se encarga de crear el contenedor con las herramientas necearías para ejecutar el bróker, del servidor de hora.

Después de tener los ficheros Dokerfile preparados, se construyen las imágenes y se suben al repositorio para luego poder descargarlas en el clúster. Así se introducen los siguientes comandos para crear la versión 1.0 de ambos:

```
1  $ sudo docker build -t jnogues/brokermqtt:1.0 .
2  $ sudo docker build -t jnogues/publishmqtt:1.0 .
3  $ sudo docker push jnogues/brokermqtt:1.0
4  $ sudo docker push jnogues/publishmqtt:1.0
```

En las líneas 1 y 2 se crean las imágenes de los contenedores y en las líneas 3 y 4 se crean los repositorios en Docker Hub.

Ahora se configura el servidor en el clúster:

1	$ kubectl create deployment broker --image=jnogues/brokermqtt:1.0
2	$ kubectl expose deployment broker --port=1883 --type="LoadBalancer" --cluster-ip="10.35.247.167"
3	$ Kubectl create deployment publish --image=jnogues/publishmqtt:1.0

En la línea 1, se muestra la creación del despliegue del bróker y en la línea 2 se expone el servicio configurando la dirección IP del servicio en el clúster a 10.35.247.167, para que internamente se pueda acceder al bróker.

En la línea 3 se crea el despliegue del publicador de hora (publish), que desplegará un contenedor con la imagen del programa “prublish.c”, que se ha modificado para que publique la hora actual en un servidor localizado en la IP 10.35.247.167.

```
MacBook-Pro-de-JAVIER:~ jnogues$ kubectl get service
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
broker       LoadBalancer   10.35.247.167   35.242.206.232   1883:32273/TCP   15m
```

Figura 57. Captura del terminal de los servicios expuestos en el cúster.

En la figura anterior se muestra como el servicio está lanzado y expuesto externamente con la IP 35.242.206.232.

Para probar que funciona correctamente, desde fuera del clúster, hacemos suscripción al tópico del publicador (time), como se muestra en la siguiente figura, conectando con el servidor a través de la IP externa creada para el servicio del bróker.

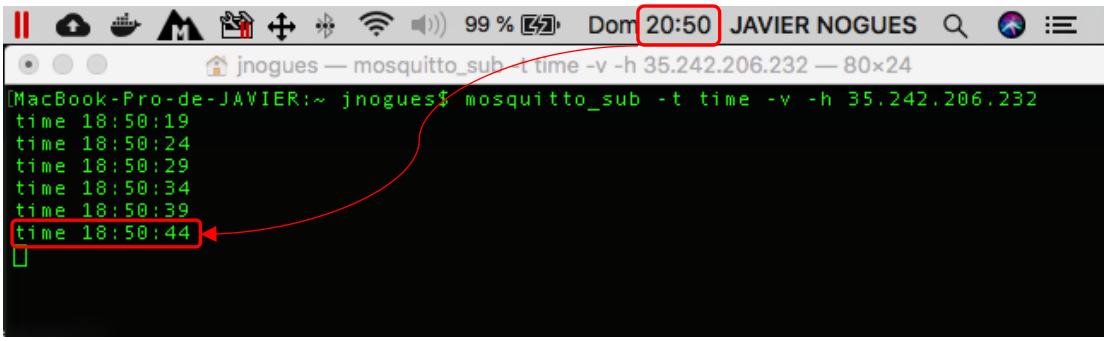


Figura 58. Captura de pantalla del terminal del suscriptor.

Se ve en la figura anterior que el servidor está funcionando correctamente, sirviendo la hora actual cada cinco segundos, como está implementado en “publish.c”

3.4.1 Actualización con control de versiones

Para estudiar esta característica primero se va a crear una actualización del repositorio “publismqtt:1.0”, siendo la nueva “publishqmtt:1.1”, donde solo se mejora el mensaje. Así tenemos que modificar “publish.c” y crear en nuevo repositorio introduciendo los siguientes comandos:

1	$ sudo docker build -t jnogues/publishmqtt:1.1 .
2	$ sudo docker push jnogues/publishmqtt:1.1

El comando de la línea 1 crea la nueva imagen, y el comando de la línea 2 sube la imagen al repositorio. En la siguiente figura se muestra el repositorio de Docker Hub para el contenedor y como están disponibles las dos versiones.

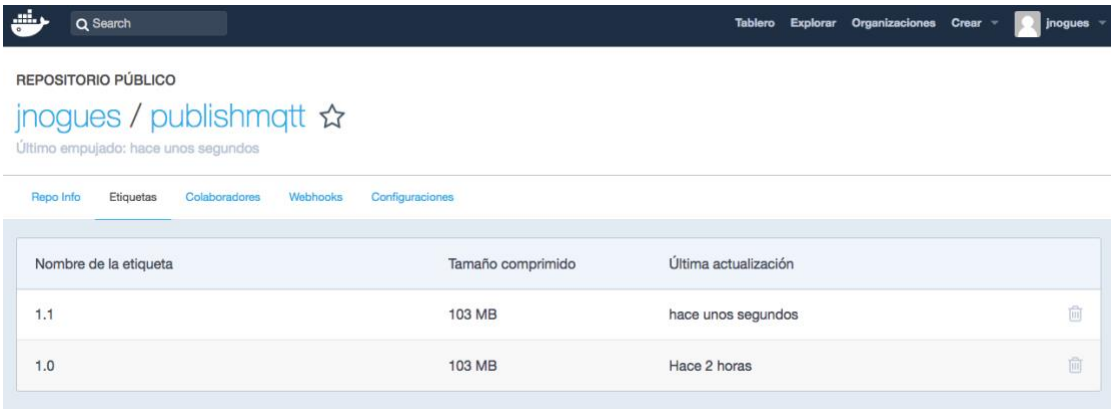


Figura 59. Repositorio Docker Hub del contenedor publishmqtt.

Ahora se actualiza el despliegue existente de “publish” editando este de la siguiente manera, introduciendo en el terminal el siguiente comando:

```
$ kubectl edit deployment publish
```

Esto hace que se lance un editor “vi” para modificar el fichero de configuración del objeto *deployment*. En este objeto vemos como se registra la revisión, es decir la versión del objeto deployment que corre actualmente en el clúster, siendo en este caso la 1, como se muestra en el primer cuadro rojo de la siguiente figura.

Se modifica la versión de la imagen que se ejecuta en los contenedores del despliegue al valor 1.1 como se ve en el segundo cuadro rojo de la siguiente figura.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2018-09-16T18:01:09Z
  generation: 1
  labels:
    app: publish
  name: publish
  namespace: tfg
  resourceVersion: "194864"
  selfLink: /apis/extensions/v1beta1/namespaces/tfg/deployments/publish
  uid: 7d5a2ab8-b9da-11e8-83ed-42010a9c0012
spec:
  replicas: 1
  selector:
    matchLabels:
      app: publish
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: publish
    spec:
      containers:
      - image: jnogues/publishmqtt:1.1
        imagePullPolicy: IfNotPresent
        name: publishmqtt
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 1
  conditions:
  - lastTransitionTime: 2018-09-16T18:01:09Z
    lastUpdateTime: 2018-09-16T18:01:09Z
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
  updatedReplicas: 1
```

Figura 60. Captura del terminal de la modificación descripción del despliegue del publicador.

Automáticamente, al guardar y salir del fichero, se aplican los cambios y podemos ver como se aplican con el siguiente comando:

```
$ kubectl rollout status deployment publish
```

Donde se da la salida en el terminal que se muestra en la siguiente figura, y que informa de que los cambios se han producido de forma exitosa.

```
Waiting for rollout to finish: 0 out of 1 new replicas have been updated...
Waiting for rollout to finish: 0 of 1 updated replicas are available...
Waiting for rollout to finish: 0 of 1 updated replicas are available...
deployment "publish" successfully rolled out
```

Figura 61. Captura del terminal del estado de la actualización de un despliegue.

Para comprobar estos cambios introducimos el comando:

```
$ kubectl describe deployment publish
```

Y como se muestra en la siguiente figura se ha aumentado el valor de la revisión que se esta ejecutando, siendo ahora la segunda y como la imagen que se ejecuta en los contenedores es la correspondiente a “publishmqtt:1.1”

```
Name:                   publish
Namespace:              tfg
CreationTimestamp:      Sun, 16 Sep 2018 20:01:09 +0200
Labels:                 app=publish
Annotations:            deployment.kubernetes.io/revision=2
Selector:               app=publish
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:  app=publish
  Containers:
   publishmqtt:
    Image:        jnogues/publishmqtt:1.1
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   publish-67d84c7dd8 (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  2m    deployment-controller  Scaled up replica set publish-67d84c7dd8 to 1
  Normal  ScalingReplicaSet  2m    deployment-controller  Scaled down replica set publish-77c8d4485f to 0
```

Figura 62. Captura del terminal de la descripción del despliegue del publicador después de actualizarse.

Aunque en teoría antes de destruir el viejo pod, crea el nuevo para no interrumpir el servicio, se ha apreciado una interrupción dado que ha habido una pequeña demora en el establecimiento del nuevo pod con la actualización, siendo de 30 segundos. Esto es debido a la configuración por defecto provoca que un pod antes de pasar el trabajo al pod nuevo tenga un periodo de espera para finalizar.

Lo podemos consultar en el fichero de definición con el siguiente comando:

```
$ kubectl get deployment publish -o yaml | grep 30
```

Que da la salida de la siguiente imagen.

Figura 63. Captura del terminal del tiempo de actualización de un despliegue.

Si existieran problemas con esta versión se puede volver a la anterior con:

```
$ kubectl rollout undo deployment publish
```

Y comprobamos estos cambios con:

```
$ kubectl describe deployment publish
```

Se puede observar en la salida, como se muestra en la siguiente figura, que la revisión ahora es la tercera y que la imagen que ejecuta el contenedor es la misma que la anterior a la actualización, es decir, “publishmqtt:1.0”

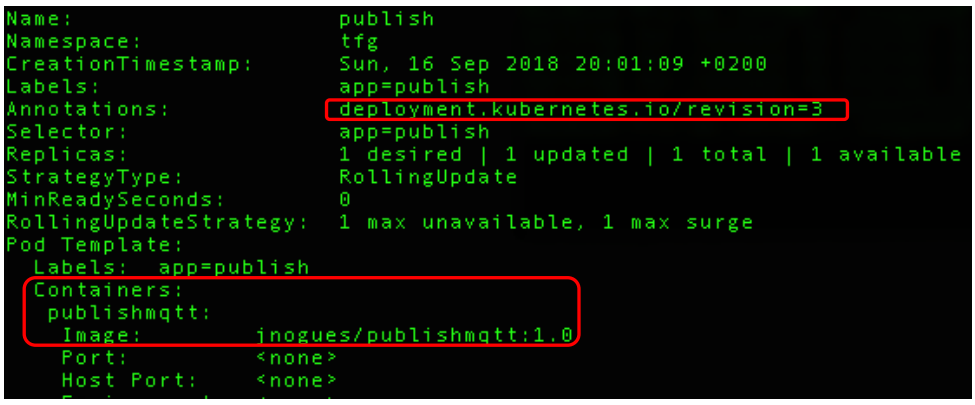


Figura 64. Captura del terminal de la descripción del despliegue del publicador.

3.5 Conclusiones

De todo lo visto en este capítulo del documento, se ve a modo de conclusión, que Docker es una tecnología de contenedores que ofrece mejor portabilidad de las aplicaciones que se empaquetan en contenedores que la ofrecida por tecnología LXC, que solo permite portabilidad entre máquinas con kernel de Linux compatible con el del contenedor. Además, esta portabilidad se comprueba en dos OS (Linux y macOS), con total éxito, demostrando con esto también que es fácil de distribuir mediante el uso de repositorios para las imágenes de los contenedores, lo que corrobora lo estudiado.

Se concluye también que la puesta en marcha de un clúster en Google Cloud Platform es una labor muy fácil de realizar, ejecutando un comando, además de la web de la plataforma, que resulta muy intuitiva. Incluso las herramientas usadas son muy fáciles de utilizar ya que resultan también intuitivas, y tiene disponibles opciones de ayuda, para ver que opciones de comandos existen para cada sintaxis en la CLI.

Del estudio de las características de Kubernetes en el escenario escogido, se concluye que todas estas satisfacen lo estudiado anteriormente, cumpliendo así con los motivos que provocan el desarrollo del proyecto, aunque en el caso de la actualización de la aplicación del servidor de hora, se ha interrumpido el servicio, pero lo ha hecho de manera controlada y rápida, con lo cual se puede decir que como el cliente suscrito ha recuperado el servicio automáticamente, no incumple del todo esta premisa.

Capítulo 4

Entorno socio-económico y marco regulador

4.1 Entorno socio-económico

Cada día el uso de Kubernetes es más sencillo para el administrador inexperto, ya que cada vez hay más tutoriales y conferencias que informan tanto de sus ventajas y de como usar esta tecnología.

Desde que Kubernetes es expuesto como código abierto por Google en 2015 hay una comunidad de Kubernetes y un grupo de desarrolladores de Kubernetes que se encargan de mantener y actualizar esta tecnología. A través de twitter se pueden recibir las novedades de Kubernetes y además se organizan conferencias llamadas “KubeCon”. Incluso se puede participar en su desarrolla a través del proyecto “GitHub” [54].

Kubernetes ofrece gestión para el uso de los recursos de la nube, lo que está revolucionando las empresas. Hoy en día son muchas empresas las que han desarrollado sistemas en Kubernetes por su simplicidad a la hora de aplicar una actualización, o corrección, de las aplicaciones lanzadas sobre esta tecnología.

Al poder utilizar Kubernetes a través de empresas que la ofrecen como un servicio, a precios realmente económicos, como Google (GCP) y Amazon (AWS), se convierte en una opción cada vez más usada por empresas que no pueden disponer de grandes infraestructuras privadas por diversos motivos, principalmente el económico. Además, permite disponer de solo lo necesario en cada momento y pagar solo por los recursos consumidos.

Por todo esto cada vez son más las empresas y personas que deciden usar Kubernetes con compañías que lo ofrecen como un servicio, ya que luego con los servicios, que despliegan sobre Kubernetes, obtienen beneficios que lo hacen rentable. Así el mediano y pequeño empresario no se preocupa de mantener sistemas informáticos complejos y puede ofrecer servicios estables, de alta disponibilidad y fiables, ya que en momentos de necesidad pueden ampliar los recursos necesarios para su negocio.

Según una encuesta de “Cloud Native Computing Foundation” en 2017, Kubernetes es la plataforma de orquestación preferida de los desarrolladores [36].

Además, las grandes empresas y multinacionales también han encontrado en Kubernetes una solución para administrar contenedores, ya que así optimizan los recursos de su infraestructura. En la siguiente figura se muestran compañías cliente de Kubernetes.

Figura 65. Imagen de empresas que usan Kubernetes [55]

Kubernetes esta avanzando, y mejorando las herramientas para la nube privada, y se prevé que se añadan funcionalidades de nube hibrida, lo que supone que, conjunto a las guías cada vez más completas, y múltiples blogs con tutoriales gratuitos, que usuarios particulares cada vez vean más atractiva la idea de conocer esta tecnología.

4.2 Marco regulador

Actualmente hay muchos proveedores de infraestructuras en nube y por lo tanto también de Kubernetes, por lo que, si un cliente quiere contratar estos servicios podría no estar seguro de si van a funcionar correctamente y cumplir con las especificaciones de Kubernetes.

Para solucionar este problema, y garantizar que las migraciones de los clientes, entre proveedores, permitan que las aplicaciones sigan funcionando, y por lo tanto sean portables, “Cloud Native Computing Foundation” (CNCF) tiene un programa para certificar a las empresas que prestan este servicio. De este modo las empresas que ofrecen Kubernetes son certificadas como “Proveedor de Servicios Certificado de Kubernetes” (KCSP) [55]. Además, existe la certificación de Kubernetes, que certifica que un determinado software funciona sobre Kubernetes [56].

Figura 66. Logo del certificado de Kubernetes por CNCF.

No solo las empresas pueden obtener el certificado de Kubernetes, ya que la CNCF también dispone de un programa de certificados para los desarrolladores y administradores y así asegurarse de una correcta expansión de Kubernetes con gente preparada.

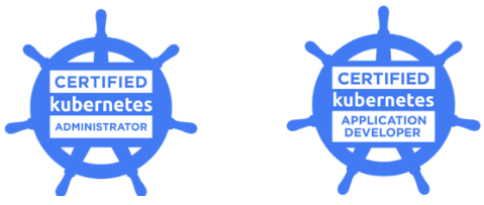


Figura 67. Logos de los certificados para administradores y desarrolladores por CNCF.

Kubernetes es un proyecto de código libre tras ser liberado por Google en 2015 lo que permite que los desarrolladores puedan acceder al código fuente y mejorarlo, adaptarlo o crear complementos eficientes. Esto permite que se pueda colaborar con el proyecto y fomenta la participación de comunidades sin problemas legales.

Así puede distribuirse de forma gratuita y desarrollarse de forma legal por cualquier usuario o desarrollador experto. Por lo que los certificados mencionados anteriormente son importantes para mantener un producto que ofrezca garantías [57], [58].

Kubernetes es una tecnología de orquestación y el uso en si de ella no exige cumplir alguna ley. Pero si usamos nuestro producto con clientes lo normal es que estos estén registrados y se deba cumplir las leyes de protección de datos. De igual manera, si usamos plataformas de Kubernetes en nube pública, los datos para acceder a ella también se regirán por leyes de protección de datos, concretamente la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal que ha sido actualizada el 30 de julio de este año (2018) [59].

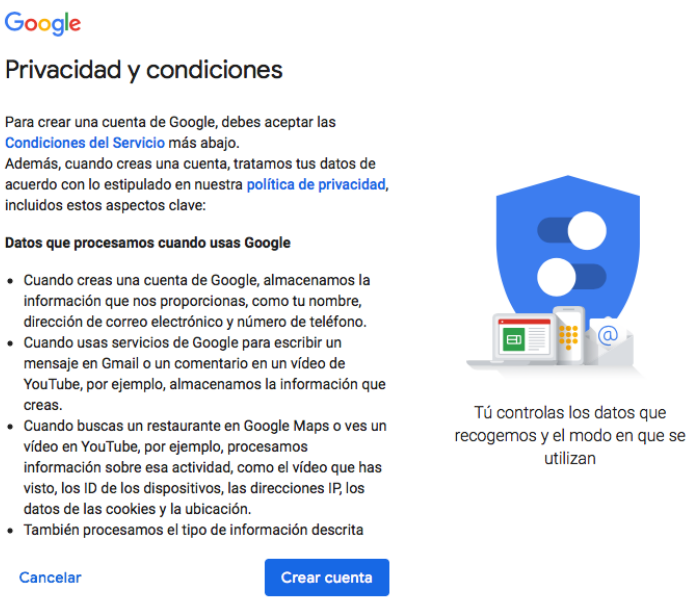


Figura 68. Privacidad y condiciones de Google para una cuenta.

Además, Kubernetes funciona ejecutando contenedores para ejecutar las aplicaciones que expone como servicios, lo que no deja de ser virtualización ligera en nube, pudiendo virtualizar funciones de red y servidores. Por lo que también se rige por estándares NFV (Network Functions Virtualization), que busca tecnologías estandarizadas que se puedan aplicar a cualquier fabricante que quiera desplegar su plataforma y ofrecer Kubernetes como un servicio, ofreciendo clústeres virtualizados.

ETSI (European Telecommunications Standards Institute) creó un grupo para el control de estos estándares que afectan a Kuberntes. En este grupo se encuentra MANO (Management and Orquestration) que se encarga de orquestar el escenario NFV. También NVF y NFVI (Network Functions Virtualization Infrastructure), esta última contiene la infraestructura hardware para alojar el sistema virtualizado [61].

Capítulo 5

Planificación y presupuesto del proyecto

5.1 Planificación del proyecto

Para abordar el proyecto se determinan varias etapas, es decir, se lleva a cabo una planificación de las fases o etapas para completar con éxito el proyecto. Así se definen las siguientes cuatro etapas:

1ª etapa: planificación

La etapa de planificación tiene como finalidad organizar el desarrollo del proyecto.

- **Alcance**: estudio del alcance del proyecto, donde se especifica el tema del proyecto y la motivación que conlleva la elección del proyecto.

- **Requisitos**: en este punto se determina las necesidades que tienen que cumplir la tecnología elegida para cubrir las motivaciones del proyecto. Así como el análisis de las normas a cumplir (marco regulatorio).
- **Recursos**: se analiza los recursos disponibles para la realización del proyecto, realizando un análisis previo de soluciones alojadas en terceros (como la nube pública), y los disponibles en la Universidad y sus laboratorios.
- **Definición de objetivos**: se definen las tareas que son necesarias para afrontar el desarrollo del proyecto, las tecnologías a estudias y escenarios donde se probaran.

2ª etapa: estudio

Esta etapa se centra en el estudio de los conocimientos necesarios para la realización del proyecto y las alternativas de cada tecnología. Es la etapa donde se recaba la información del estado del arte de este documento.

- **Estudio de tecnologías**: se estudian las tecnologías que son necesarias para cubrir las motivaciones del proyecto y sus principales alternativas.
- **Estudio de escenarios**: se estudian los posibles escenarios sobre los que probar la tecnología que se ha estudiado.

3ª etapa: desarrollo

Etapa donde se desarrollan los escenarios y las pruebas que permiten estudiarlos buscando cumplir con los objetivos que cubren las motivaciones del proyecto.

- **Creación de escenario**: después del estudio de escenarios se elige el más favorable para nuestro estudio.
- **Pruebas**: en el escenario se prueba que se cumple lo estudiado de las tecnologías y que cubren la motivación del proyecto.
- **Evaluación**: se valora el funcionamiento, a modo de conclusión, de las pruebas.

4ª etapa: documentación

Etapa final del proyecto donde se documenta la realización del proyecto, las pruebas, tecnologías estudiadas, presupuestos, entorno socio-económico, marco regulatorio y conclusiones obtenidas.

- **Elaboración de la estructura**: se crea la estructura del documento distribuyéndolo en diferentes capítulos.
- **Redacción de la memoria**: se realiza este documento, donde se refleja todo el trabajo realizado para la realización del proyecto.
- **Preparación de la presentación**: se desarrolla una presentación para explicar el proyecto.

Diagrama de Gantt

Para el proyecto se ha dispuesto de un cuatrimestre académico, siendo el segundo del curso 2017-2018, extendiéndolo hasta la fecha de la defensa, es decir hasta el 30 de septiembre de 2018. Hay que destacar que el proyecto se ha llevado a cabo, coordinándolo con otras actividades, como el curso académico y jornada laboral.

Así para organizar las tareas en función del tiempo disponible para el proyecto se define un diagrama de Gantt en las siguientes tablas.

	ENERO				FEBRERO				ABRIL				MAYO			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1ª etapa: planificación																
Alcance																
Requisitos																
Recursos																
Definición de objetivos																
2ª etapa: estudio																
Estudio de tecnologías																
Estudio de escenarios																
3ª etapa: desarrollo																
Creación del escenario																
Pruebas																
Evaluación																
4ª etapa: documentación																
Elaboración de estructura																
Redacción de memoria																
Preparación de presentación																

Tabla 2. Primea parte del diagrama de Gantt

	JUNIO				JULIO				AGOSTO				SEPTIEMBRE			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1ª etapa: planificación																
Alcance																
Requisitos																
Recursos																
Definición de objetivos																
2ª etapa: estudio																
Estudio de tecnologías																
Estudio de escenarios																
3ª etapa: desarrollo																
Creación del escenario																
Pruebas																
Evaluación																
4ª etapa: documentación																
Elaboración de estructura																
Redacción de memoria																
Preparación de presentación																

Tabla 3. Segunda parte del diagrama de Gantt

5.2 Presupuesto del proyecto

En este apartado se detallan los gastos del proyecto, para lo que se diferencian dos tipos, Capex y Opex. Así los gastos Capex son los destinados a los recursos que no se consumen, es decir, gastos materiales e inversiones para proveer de recursos a la realización del proyecto. Y los gastos Opex son los gastos destinados a recursos necesarios para la realización del proyecto y que se consumen.

Para identificas los gastos se ha considerado que el interesado en desarrollar el proyecto no es una empresa con infraestructura previa, o la propia Universidad, siendo un montaje desde cero, es decir, sin localización donde desarrollar el proyecto. Aunque se considera que este ya está dado de alta en Hacienda, Seguridad Social y otras que se pudieran necesitar.

Descripción de los gastos

Gastos Capex

- **Hardware (HW)**: para el desarrollo del proyecto se ha utilizado un MacBook Pro de 2011, se tomará el precio de compra como referencia de gasto (1.500€).

Gastos Opex

- **Fijos:**
    - **Proveedor de Servicio de Internet (ISP**): es necesario disponer de conexión a internet para el uso de plataformas de nube pública. Para lo que consideraremos una tarifa orientada a autónomos.
    - **Espacio de trabajo**: es necesario un lugar con posibilidad de contratar servicios de luz e internet, en este caso en la localidad de Leganés.

Después de un sondeo en la web Idealista, se considera una oficina de 28m^2.

- **Personal**: se necesita contratar a una persona con capacidades técnicas, por eso se toma como referencia un estudiante a punto de titularse, trabando a media jornada de desarrollador.
- **Software**: para el desarrollo de la memoria se ha usado Word de Microsoft Office y para la presentación PowerPoint. Su la licencia de uso es una mensualidad fija, a pagar cada mes.

- **Variables**

  - **Electricidad, agua**: durante el transcurso del proyecto, los aparatos eléctricos consumen electricidad, por eso necesitaremos contratar este servicio. Así hay que cumplir un mínimo de condiciones, como un sanitario, donde es necesario suministros de agua.
  - **Google Cloud Platform (GCP)**: recurso necesario para alojar el clúster en nube pública. Solo se gasta en función de los recursos usados y el tiempo que se dispone de ellos[6].

Cálculo de los gastos y el presupuesto

Tras la identificación de los gastos que genera el proyecto, en la siguiente tabla se muestra el cálculo del total que hay que afrontar para desarrollar el proyecto tanto en gastos como en presupuesto. En la siguiente tabla se ve el gasto, lo que cuesta la unidad o el precio mensual, las cantidades o meses que hay que mantener el gasto y el total del precio.

[6] Para el calculo del presupuesto no se tiene en cuenta el periodo de prueba gratuito.

Gasto	Precio/mes	Cantidad	Total
Amortización HW	31,25 €	8 meses	250 €
Software	9 €	8 meses	70 €
ISP	62 €	8 meses	496 €
Espacio de trabajo	500 €	8 meses	4.000 €
Personal	450 €	8 meses	3.600 €
Luz	75 €	8 meses	600 €
Agua	20 €	8 meses	160 €
GCP	190 €	1,5 meses	285 €
			9.461 €

Tabla 4. Gastos totales de la realización del proyecto.

Para el gasto ocasionado por la depreciación de los activos adquiridos en los gastos Capex, que en este caso es el hardware adquirido, se ha estimado que tendrá una vida útil de cuatro años, calculando la cuota mensual de la amortización del hardware (1.500 € / 48 meses). Se obtiene, por lo tanto, un gasto de 9.461 € estimados durante la realización del proyecto.

Aunque el equipo ya estaba comprado se considera que se parte de cero, por lo que hay que asumir este gasto Capex como parte de la inversión inicial para el cálculo del presupuesto. En la siguiente tabla se muestra el dinero necesario para realizar el proyecto.

Gasto	Precio/mes	Cantidad	Total
Hardware	1.500 €	1 unidad	1.500 €
Software	9 €	8 meses	70 €
ISP	62 €	8 meses	496 €
Espacio de trabajo	500 €	8 meses	4.000 €
Personal	450 €	8 meses	3.600 €
Luz	75 €	8 meses	600 €
Agua	20 €	8 meses	160 €
GCP	190 €	1,5 meses	285 €
			10.711 €

Tabla 5. Presupuesto total de la realización del proyecto.

Se ve como el presupuesto es de 10.711 € para realizar el proyecto. La diferencia con el gasto es el valor de los activos al finalizar el proyecto. Esta diferencia es de 1.250€ que es lo mismo que restar al precio del hardware su amortización tras los 8 meses, es decir 1.500 € menos 250 €.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones

En este proyecto se ha desarrollado un entorno de orquestación de contenedores usando tecnología de Google (Kubernetes y Docker). Este estudio ha sido motivado por la necesidad actual de prestar servicios en nube, con las herramientas y tecnologías más ligeras y eficaces posibles, de cara a cubrir la demanda actual de servicios. Todo esto ha provocado que los servidores tradicionales hayan necesitado evolucionar para optimizar sus sistemas llegando, hoy en día, a tecnologías de basadas en orquestación de contenedores.

En el proyecto se han estudiado las principales características que nos ofrece Kubernetes para orquestar contenedores, administrarlos y mantenerlos. Para lo cual se crea un escenario donde teóricamente se cumplen los objetivos a estudiar, y que son consecuencia de los motivos que llevan a desarrollar este proyecto.

Se han identificado varios escenarios para desplegar Kubernetes, siendo estos de nube pública y privada. Y se ha elegido el desarrollo en nube pública por las ventajas que nos ofrece el no necesitar una potente infraestructura de red, ya que, estas

plataformas ofrecen soluciones que se encargan de ello. En nuestro proyecto Google Compute Platform nos provee de la infraestructura necesaria para la creación de clústeres a precio competitivo, en el que solo se paga por lo consumido.

Esto ha conllevado el estudio de las ventajas que nos ofrece Kubernetes y las características de cada entorno para elegir la opción más conveniente para el desarrollo del proyecto.

En cuanto a las características que nos ofrece el sistema de orquestador de contenedores Kubernetes, el estudio de cada una de ellas ha supuesto un gran aprendizaje de lo que se puede hacer con tecnologías de virtualización ligera, dando solución a las motivaciones que fomentan la realización del proyecto, enfocadas en las necesidades actuales de prestación de servicios en nube.

Además, se ha comprobado que el gasto para el desarrollo del proyecto en nube pública no es muy elevado en comparación con las alternativas en nube privada ya que, como se puede ver en el apartado de presupuestos de este documento, el precio para desplegar el entorno en la nube de Google (GCP) no supera tan siquiera el precio de un ordenador.

El proyecto además ha supuesto un desafío a la hora de establecer la planificación, ya que el tiempo es limitado y ha habido que compaginarlo con trabajo y estudios y alguna interrupción por motivos de viaje. Todo esto ha sido beneficioso para el desarrollo como estudiante en última etapa de la carrera, ya que es un proyecto individual y de gran alcance, partiendo desde el desconocimiento de lo que es la base de la tecnología usada, es decir, un contenedor.

Así concluyo que la elección, por los conocimientos que me ha aportado la realización del proyecto, ha sido acertada. Incluso finalizado mis estudios y este proyecto, viendo lo que ofrece el mundo de la orquestación de contenedores, espero poder seguir indagando y ampliando mis conocimientos acerca de Kubernetes.

6.2 Trabajos futuros

El estudio de Kubernetes motiva seguir indagando y estudiando las opciones que nos da esta tecnología. Por lo que es interesante plantear trabajos futuros que mejoren el estudio de Kubernetes.

Dicho esto, se plantean dos trabajos futuros que podrían realizarse:

- **Kubernetes en OpenStack**: es capaz de gestionar un híbrido de nube privada y pública, por lo que puede estudiar Kubernetes en nube híbrida.

- **Kubernetes con Conjure-up**: se puede estudiar el escenario de nube privada implementando Kubernetes con contenedores LXC.

Bibliografía

Todos los enlaces están disponibles a 15 de septiembre de 2018.

[1] Wikipedia, «Hipervisor - Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Hipervisor.

[2] Docker, «What is a container,» [En línea]. Available: https://www.docker.com/resources/what-container.

[3] jPablo, «¿Por qué usar LXC?,» [En línea]. Available: https://www.jpablo128.com/por-que-usar-lxc-linux-containers/.

[4] Linux Containers, «Linux Containers,» [En línea]. Available: https://linuxcontainers.org/lxc/introduction/.

[5] redhat, «¿Qué es un contendor de Linux?,» [En línea]. Available: https://www.redhat.com/es/topics/containers/whats-a-linux-container.

[6] redhat, «¿Qué es Docker?,» [En línea]. Available: https://www.redhat.com/es/topics/containers/what-is-docker.

[7] M. Wheatley, «siliconANGLE,» [En línea]. Available: https://siliconangle.com/2016/02/09/docker-gets-minimalist-with-plan-to-migrate-images-to-alpine-linux/.

[8] B. Cessa, «¿Cómo escribir un buen Dockerfile?,» [En línea]. Available: https://github.com/mxabierto/hackea-el-sistema/wiki/%C2%BFC%C3%B3mo-escribir-un-buen-Dockerfile%3F.

[9] Docker, «Docker Hub,» [En línea]. Available: https://www.docker.com/products/docker-hub.

[10] Kubernetes, «Kubernetes,» [En línea]. Available: https://Kubernetes.io/.

[11] Wikipedia, «Kubernetes,» [En línea]. Available: https://es.wikipedia.org/wiki/Kubernetes.

[12] Kubernetes, «Concepts,» [En línea]. Available: https://Kubernetes.io/docs/concepts/.

[13] Kubernetes, «Pods,» [En línea]. Available: https://Kubernetes.io/docs/concepts/workloads/pods/pod/.

[14] Kubernetes, «Nodes,» [En línea]. Available: https://Kubernetes.io/docs/concepts/architecture/nodes/.

[15] Kubernetes, «Using Minikube to create a cluster,» [En línea]. Available: https://Kubernetes.io/docs/tutorials/Kubernetes-basics/create-cluster/cluster-intro/. [Último acceso: 2018].

[16] Kubernetes, «Kubernetes components,» [En línea]. Available: https://Kubernetes.io/docs/concepts/overview/components/.

[17] Kubernetes, «Understanding Kubernetes objects,» [En línea]. Available: https://Kubernetes.io/docs/concepts/overview/working-with-objects/Kubernetes-objects/.

[18] Kubernetes, «Kubernetes API overview,» [En línea]. Available: https://Kubernetes.io/docs/reference/using-api/api-overview/#api-versioning.

[19] Kubernetes, «Labels and Selectors,» [En línea]. Available: https://Kubernetes.io/docs/concepts/overview/working-with-objects/labels/.

[20] Kubernetes, «Manage Compute Resources Container,» [En línea]. Available: https://Kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/.

[21] Kubernetes, «Assign Memory Resource,» [En línea]. Available: https://Kubernetes.io/docs/tasks/configure-pod-container/assign-memory-resource/.

[22] Kubernetes, «Assign CPU Resource,» [En línea]. Available: https://Kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/.

[23] Kubernetes, «Resource Quotas,» [En línea]. Available: https://Kubernetes.io/docs/concepts/policy/resource-quotas/.

[24] Kubernetes, «Memory Default Namespac,» [En línea]. Available: https://Kubernetes.io/docs/tasks/administer-cluster/manage-resources/memory-default-namespace/.

[25] Kubernetes, «Deployments,» [En línea]. Available: https://Kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[26] Kubernetes, «ReplicaSet,» [En línea]. Available: https://Kubernetes.io/docs/concepts/workloads/controllers/replicaset/.

[27] Kubernetes, «Persistent Volumes,» [En línea]. Available: https://Kubernetes.io/docs/concepts/storage/persistent-volumes/.

[28] Kubernetes, «Volumes,» [En línea]. Available: https://Kubernetes.io/docs/concepts/storage/volumes/.

[29] Kubernetes, «Using a Service to Expose Your App,» [En línea]. Available: https://Kubernetes.io/docs/tutorials/Kubernetes-basics/expose/expose-intro/.

[30] Kubernetes, «Cluster Networking - Kubernetes,» [En línea]. Available: https://Kubernetes.io/docs/concepts/cluster-administration/networking/.

[31] Kubernetes, «Highly Available Master,» [En línea]. Available: https://Kubernetes.io/docs/tasks/administer-cluster/highly-available-master/.

[32] Kubernetes, «Federation,» [En línea]. Available: https://Kubernetes.io/docs/concepts/cluster-administration/federation/.

[33] Kubernetes, «Multiple Zones,» [En línea]. Available: https://Kubernetes.io/docs/setup/multiple-zones/.

[34] Kubernetes, «Overview of kubectl,» [En línea]. Available: https://Kubernetes.io/docs/reference/kubectl/overview/.

[35] Y.-J. Hong, «CRI: the Container Runtime Interface,» [En línea]. Available: https://github.com/Kubernetes/Kubernetes/blob/242a97307b34076d5d8f5bbeb154fa4d97c9ef1d/docs/devel/container-runtime-interface.md.

[36] J. M. Fiz, «Por qué todos apuestan por Kubernetes,» [En línea]. Available: https://www.paradigmadigital.com/techbiz/por-que-todos-apuestan-por-Kubernetes/.

[37] Docker docs, «Swarm mode overview,» [En línea]. Available: https://docs.docker.com/engine/swarm/.

[38] The Apache Software Foundation, «Apache Mesos,» [En línea]. Available: http://mesos.apache.org/.

[39] Canonical Ltd. Ubuntu y Canonical, «Conjure-Up,» [En línea]. Available: https://conjure-up.io/.

[40] Kubernetes, «Minikube,» [En línea]. Available: https://Kubernetes.io/docs/setup/minikube/.

[41] Git Hub, «Releases,» [En línea]. Available: https://github.com/Kubernetes/federation/releases.

[42] Kubernetes, «Install Kubeadm,» [En línea]. Available: https://Kubernetes.io/docs/setup/independent/install-kubeadm/.

[43] Kubernetes, «Create Cluster Kubeadm,» [En línea]. Available: https://Kubernetes.io/docs/setup/independent/create-cluster-kubeadm/.

[44] Google, «Documentación de Google Cloud,» [En línea]. Available: https://cloud.google.com/docs/.

[45] Amazon, «EKS,» [En línea]. Available: https://aws.amazon.com/es/eks/.

[46] Amazon, «AWS,» [En línea]. Available: https://aws.amazon.com/es/.

[47] Docker docs, «What to know before you install,» [En línea]. Available: https://docs.docker.com/docker-for-mac/install/#what-to-know-before-you-install.

[48] Docker Strore, «Docker CE for Mac,» [En línea]. Available: https://store.docker.com/editions/community/docker-ce-desktop-mac.

[49] Kubernetes, «Install Kubectl,» [En línea]. Available: https://Kubernetes.io/docs/tasks/tools/install-kubectl/.

[50] Google, «Google Cloud Platform,» [En línea]. Available: https://cloud.google.com.

[51] Google, «Usar el instalador de Google Cloud SDK,» [En línea]. Available: https://cloud.google.com/sdk/docs/downloads-interactive.

[52] Docker docs, «Dockerfile reference,» [En línea]. Available: https://docs.docker.com/engine/reference/builder/#usage.

[53] Docker docs, «Use the Docker command line,» [En línea]. Available: https://docs.docker.com/engine/reference/commandline/cli/.

[54] The Linux Foundation, «The Linux Foundation Events,» [En línea]. Available: https://events.linuxfoundation.org/events/kubecon-cloudnativecon-europe-2018/.

[55] Kubernetes, «Kubernetes User Case Studies,» [En línea]. Available: https://Kubernetes.io/case-studies/.

[56] CNCF, «Kubernetes Certified Service Provider,» [En línea]. Available: https://www.cncf.io/certification/kcsp/.

[57] CNCF, «Software Conformance,» [En línea]. Available: https://www.cncf.io/certification/software-conformance/.

[58] CNCF, «Certified Kubernetes Application Developer (CKAD) Program,» [En línea]. Available: https://www.cncf.io/certification/ckad/.

[59] CNCF, «Certified Kubernetes Administrator (CKA) Program,» [En línea]. Available: https://www.cncf.io/certification/cka/.

[60] Agencia Estatal BOLETÍN OFICIAL DEL ESTADO, «BOE,» [En línea]. Available: https://www.boe.es/buscar/act.php?id=BOE-A-1999-23750.

[61] Wikipedia, «Virtualización de funciones de red,» [En línea]. Available: https://es.wikipedia.org/wiki/Virtualizaci%C3%B3n_de_funciones_de_red.

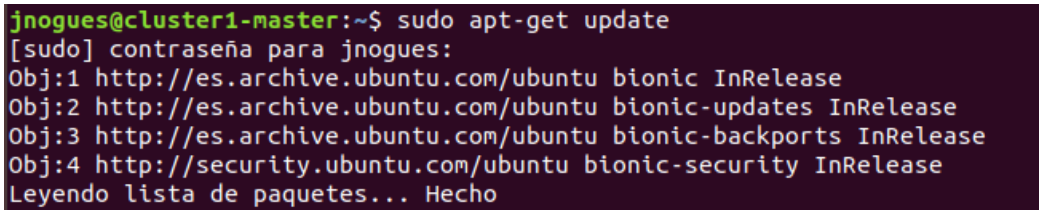
[62] Kubernetes, «Namespaces,» [En línea]. Available: https://Kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/.

Anexos

Anexo I. Instalación de Docker en Ubuntu

Los requisitos para instalar Docker en Ubuntu solo exigen que ser tenga una versión de 64 bits para: Biónico 18.04 (LTS), Artful 17.10, Xenial 16.04 (LTS), Trusty 14.04 (LTS).

Para que las actualizaciones se instalen automáticamente, al actualizar el sistema se instala a través de los repositorios.



Actualización de paquetes de repositorios

Para la instalación de los repositorios, primero actualizamos los existentes como se muestra en la imagen anterior, con el comando “apt-get update”, y se instalan los paquetes que permitirán usar el nuevo repositorio para Docker. De este modo en el terminal se escribe:

```
$ sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
software-properties-common
```

Luego se agrega la clave oficial GPG y se verifica la clave con la huella digital, 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

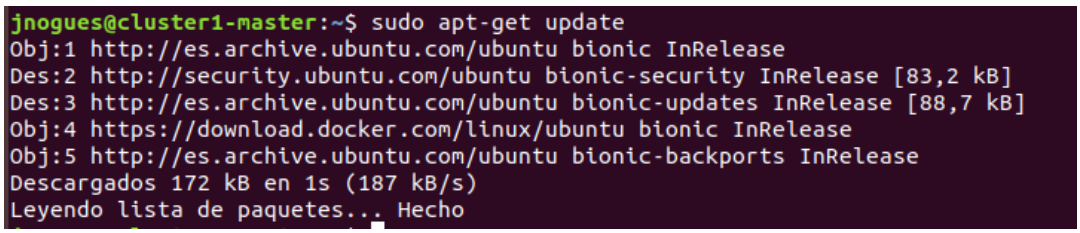
Se verifica la clave con:

```
$ sudo apt-key fingerprint 0EBFCD88
```

Se añade el repositorio oficial:

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Se puede comprobar que se ha añadido correctamente actualizando los repositorios y viendo que ahora tenemos uno más, como se muestra en la siguiente figura.



Actualización de paquetes de los nuevos repositorios

Ya solo queda instalar con:

```
$ sudo apt-get install docker-ce -y
```

Y ya estaría Docker CE instalado[7].

[7] Kubernetes, «Get Docker for Ubuntu» [En línea]. Available: https://docs.docker.com/install/linux/docker-ce/ubuntu/.

Anexo II. Programa del publicador

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <time.h>
5  #include <unistd.h>
6
7  int main(int argc, char const *argv[])
8  {
9
10         while(1)
11         {
12                 sleep(5);
13                 time_t tiempo = time(NULL);
14                 struct tm *tm;
15                 tm=localtime(&tiempo);
16
17                 char mensaje[100];
18                 strftime(mensaje, 100, "mosquitto_pub -t time -h
   10.35.247.167 -p 1883 -m \"La hora actual es: %H:%M:%S\"", tm);
19
20                 system(mensaje);
21                 printf("%s\n", mensaje);
22         }
23
24         return 0;
25 }
```

Código del programa publish.c

La diferencia entre el código usado para la versión 1.1 y 1.0 de los contenedores creados con publish.c solo es la forma de enviar el mensaje que se envía en la línea 18 después de la opción -m.