Creación de un clúster para un entorno CI/CD de desarrollo de software

Gerard Falcó Sánchez

Resumen– Cuando hay que desarrollar una aplicación o software se pueden llegar a necesitar una gran cantidad de dependencias como librerías, versiones del sistema operativo, compilador, etc. Disponer de todas estas dependencias en todos los ordenadores de cada desarrollador del equipo es complejo e ineficaz, por eso, aparecieron las tecnologías de la nube. Gracias a los clústeres y los contenedores se puede crear todo un entorno de desarrollo compartido por los desarrolladores y accesible desde cualquier parte. En este proyecto, se mostrará como a partir de la práctica CI/CD e Infraestructura como Código (IaC), a través de Terraform, se puede desarrollar un entorno de producción de *software* en un clúster gestionado por Kubernetes. Este clúster dispondrá de diferentes microservicios como Jenkins, SonarQube y PostgreSQL.

Palabras clave– Kubernetes, computación en la nube, SonarQube, Terraform, Docker, Linux, Raspberry Pi, Jenkins pipeline, integración continua, despliegue continuo

Abstract– Developing an application or software could require a large number of dependencies such as libraries, operating system versions, compilers, etc. Having all these dependencies available on all the computers of each developer in the team is complex and inefficient in time, that is why cloud technologies appeared. Thanks to clusters and containers, it is possible to create an entire development environment shared by developers and accessible from anywhere. In this project, it will be shown how from the Continuous Integration and Continuous Development (CI/CD) practice an infrastructure will be developed as code through Terraform, creating a software production environment in a cluster managed by Kubernetes. This cluster will have different microservices such as Jenkins, SonarQube and PostgreSQL.

Keywords– Kubernetes, cloud computing, SonarQube, Terraform, Docker, Linux, Raspberry Pi, Jenkins pipeline, continous integration, continous deployment

1 Introducción

Kubernetes [1] se ha convertido en una herramienta casi imprescindible para poder crear y gestionar un clúster con una gran cantidad de nodos, es por eso que la mayoría de proyectos ya la usan en sus infraestructuras de desarrollo. Al ser Kubernetes uno de los protagonistas de este proyecto, en el Capítulo 5 se ha realizado una breve introducción a esta tecnología y sus conceptos más importantes.

Como introducción a este proyecto, en el Capítulo 2, se pretende mostrar el estado actual de las tecnologías que se emplearán a lo largo de su desarrollo. Posteriormente, se presentan, en los Capítulos 3 y 4, todos los objetivos a cumplir junto a la planificación y metodología a seguir a lo largo del desarrollo del trabajo.

Para agilizar la creación de entornos de desarrollo, una de las opciones más usadas es la combinación de las técnicas Infraestructura como Código (IaC) [2] junto a la Integración Continua y Desarrollo Continuo (CI/CD) [3].

Al implementar CI/CD con herramientas como Jenkins [4], se pueden juntar las diferentes acciones manuales en una secuencia de fases automatizadas ligadas entre ellas dando como resultado una *pipeline*. Estas *pipelines* pueden compilar y ejecutar el código que crea la infraestructura y, entre otras acciones, añadir todo tipo de pruebas de calidad y seguridad.

Para contrastar las decisiones tomadas lo largo del proyecto, en el Capítulo 6.1, se presentan los análisis de coste y diseños realizados entre las distintas opciones a elegir para poder desarrollar el entorno CI/CD. Asimismo, se muestra detalladamente como se ha configurado a una Raspberry Pi

• E-mail de contacto: gerardfalco7@gmail.com
• Mención realizada: Tecnologías de la Información
• Trabajo tutorizado por: Miguel Carpio Miranda (DEiC)
• Curso 2021/22

[5] y a un ordenador portátil para actuar como nodo esclavo y máster del clúster. Seguidamente, en el Capítulo 6.2, se efectúa la creación del clúster al conectar los nodos entre ellos y empezar a ser gestionados con Kubernetes.

Para el despliegue del cúster, se configurará una primera instancia de Jenkins con unas *pipelines* encargadas de analizar el código Terraform [6] generado para crear y mantener la infraestructura usando IaC y ejecutarlo para activar el clúster. De esta manera, se estarán combinando las técnicas IaC y CI/CD para el despliegue ágil del clúster.

En el Capítulo 6.3, se muestra como se ha configurado el clúster y sus microservicios. Dentro de la instancia de Jenkins del clúster, se crearán un conjunto de *pipelines* con el fin de replicar un entorno CI/CD de producción. Estas *pipelines* interconectarán diferentes aplicaciones como GitHub y SonarQube [7] para automatizar todo el proceso de compilación, pruebas de seguridad y/o calidad de código y almacenamiento de *software*.

Una vez el clúster esté listo, en el Capítulo 6.4, se explica como se ha conseguido implementar la práctica CI/CD dentro del clúster y para su propio despliegue y/o configuración.

Finalmente, en el Capítulo 7 se presentan las conclusiones finales de este proyecto y, en el Capítulo 8, un conjunto de posibles mejoras a implementar al resultado obtenido.

2 Estado del arte

La práctica CI/CD se ha convertido en un pilar para la gran mayoría de equipos de desarrolladores y sistemas, este fenómeno es debido a que permite mejorar constantemente el estado del *software* o la infraestructura en todas sus etapas.

Existen varias herramientas para implementar la práctica CI/CD como por ejemplo Bamboo, Circle CI o Gitlab CI/CD, pero una de las que ha manifestado más protagonismo en los últimos años ha sido Jenkins. Esta herramienta destaca sobre las demás ya que es de código abierto, de uso gratuito y permite crear *pipelines* con facilidad. Gracias a esta importante diferencia Jenkins consigue disponer de muchos más usuarios activos.

Por otro lado, a causa de la necesidad de automatizar el despliegue de entornos de desarrollo como clústeres y sus microservicios, apareció la técnica denominada Infraestructura como Código y es implementada a partir de tecnologías como Terraform.

Para terminar, es relevante mencionar que hay una gran cantidad de proyectos e investigaciones basados en la práctica CI/CD, pero, uno de los que destaca es el artículo *Modelling CI/CD Pipeline Through Agent-Based Simulation* [8]. Esta investigación realizada por Qianying Liao, pretende demostrar la importancia de configurar correctamente las *pipelines* para obtener los beneficios de esta tecnología en lugar de perjudicar la evolución de un desarrollo.

3 Objetivos

El objetivo principal de este proyecto es crear un entorno automatizado de desarrollo de aplicaciones. Esta tarea consiste en proporcionar a los desarrolladores de *software* todas las herramientas necesarias para facilitar sus funciones y agilizar los procesos de compilación, pruebas, etc. De esta forma, cuando la infraestructura esté terminada y lista para llevar a producción, los usuarios dispondrán de todos los recursos para que ellos solo deban encargarse de generar el código fuente, puesto que todos los pasos intermedios y posteriores ya se realizarán de manera automática.

Para completar totalmente este proyecto, hay tres grupos principales de objetivos que debería alcanzar al finalizar este trabajo. Estos retos se muestran a continuación, están ordenados según como se ha diseñado la planificación y como deberá ir avanzando hasta el resultado final.

3.1. Preparar la Raspberry Pi 4

Uno de los objetivos más básicos y pilar de este trabajo es la puesta a punto de la *Raspberry Pi 4*, este computador debe ser configurado para poder ser añadido como nodo del clúster, tener conectividad con los otros nodos y alojar distintos microservicios.

3.2. Crear un clúster

El siguiente objetivo será la creación y mantenimiento de un clúster utilizando Terraform para conseguir infraestructura como código y orquestado con Kubernetes. Dicho clúster estará formado por dos nodos y dispondrá de diferentes microservicios como Jenkins, SonarQube y otros que estarán comunicados entre sí con el fin de ofrecer un servicio público para el desarrollo de aplicaciones siguiendo la práctica CI/CD.

3.3. Utilizar la práctica CI/CD para la creación del clúster

Dado que el objetivo principal es ofrecer una infraestructura de desarrollo automatizada mediante la práctica CI/CD, uno de los retos que se pretende lograr con este proyecto es aplicar esta práctica también en la creación del entorno a partir de una instancia de Jenkins externa al clúster y así demostrar su verdadero potencial.

3.4. Implementar la práctica CI/CD dentro del clúster

La práctica CI/CD tiene un papel muy importante en este proyecto, ya que el principal objetivo del trabajo es ofrecer un entorno de desarrollo de *software* automatizado. Para poder automatizar todas las etapas posibles una vez el código fuente se ha finalizado, y como objetivo final del proyecto, será necesario aplicar la práctica CI/CD a través de la herramienta Jenkins y un conjunto de *pipelines*.

4 Planificación y Metodología

Para llevar a cabo este proyecto se diseñó un diagrama de Gantt, presentado en el Apéndice A.1, y así mostrar todas las fases y objetivos que se esperan conseguir en un intervalo de tiempo específico.

Las fases en las que se ha dividido el proyecto se han agrupado en diferentes grupos según su relación entre ellas, por ejemplo, como se puede observar en la Figura 1 del Apéndice 1, se encuentra la sección “Investigación de las

tecnologías a usar". En este grupo de tareas, se encuentran todas aquellas relacionadas con la familiarización con las herramientas que se utilizarán a lo largo del proyecto.

Cada una de estas fases dura alrededor de una semana y para lograr finalizar todas las etapas a tiempo se va a seguir la metodología ágil llamada Scrum [9]. Se ha decidido optar por esta metodología porque es muy abierta a cambios y en caso de tener que modificar la prioridad de las tareas y su orden de ejecución no supondría un gran problema para el desarrollo del proyecto.

5 Introducción a Kubernetes

A lo largo del documento se ha comentado repetidamente el nombre Kubernetes sin llegar a concretar que es o su utilidad, para facilitar la comprensión de este proyecto se explicará brevemente la herramienta Kubernetes y sus conceptos clave.

Kubernetes es una herramienta de código abierto para administrar y crear aplicaciones en un clúster de nodos de cómputo, las aplicaciones y todas sus dependencias se encuentran dentro de contenedores con el fin de que estas sean portables a cualquier entorno de desarrollo. Asimismo, Kubernetes ayuda a controlar el ciclo de vida de los contenedores usando distintos métodos que permiten tener un alto nivel de disponibilidad y escalabilidad.

En lo que se refiere a Kubernetes y su usabilidad, hay cinco conceptos esenciales que se nombran constantemente a lo largo del desarrollo del proyecto que deben ser comprendidos. El primero es el llamado Plano de Control o *Control Plane*, este objeto se aloja en el nodo máster del clúster y se encarga de realizar las llamadas a la API de Kubernetes para la gestión de los recursos del entorno.

El segundo objeto fundamental es el *Pod*, un *Pod* es la unidad mínima de ejecución de Kubernetes y representa uno o más contenedores ejecutándose en el clúster. El siguiente recurso imprescindible de Kubernetes es el llamado *Deployment*, este objeto de la API de Kubernetes está a cargo de gestionar una aplicación o microservicio y crear los *Pods*.

En cuarto lugar, se encuentra el objeto catalogado como *Namespace* y es vital para una correcta organización interna del clúster. Los *Namespace* permiten clasificar por grupos los distintos contenedores según su finalidad, tipo o equipos de desarrollo en múltiples clústeres virtuales.

Finalmente, en Kubernetes existe un objeto en su API llamado Servicio o *Service* que tiene la función de describir como se accede a una aplicación o *Pod* y el conjunto de puertos y/o balanceadores de carga.

6 Desarrollo

En este capítulo, se muestra el desarrollo de cada una de las etapas de este proyecto y como se han resuelto los problemas encontrados. La organización para llevar a cabo estas fases se ha hecho siguiendo la planificación comentada en el Capítulo 4.

6.1. Puesta a punto del entorno

Cuando este proyecto fue diseñado, la idea inicial era crear y mantener un clúster con sus diferentes microservicios en uno de los principales proveedores de cómputo en la nube como Microsoft Azure, Amazon Web Services (AWS) y Google Cloud Platform (GCP). Una vez evaluado los costes de los diferentes servicios y sus precios, se tomó la decisión de buscar una alternativa debido a los altos costes de mantener esos recursos para un solo usuario.

6.1.1. Diseño en Amazon Web Services

Con el objetivo de reflejar aproximadamente el coste que tendría mantener el clúster y sus aplicaciones en un proveedor de cómputo en la nube, se ha elegido a Amazon Web Services para realizar un presupuesto de los recursos necesarios. En la Tabla 1, se muestran los servicios contratados en AWS, su precio de uso por hora (en USD) para un total de cinco meses y el coste final al terminar este proyecto. Conjuntamente, en la Figura 1 se muestra una representación del ecosistema que se iba a construir a través del proveedor AWS y los recursos contratados.

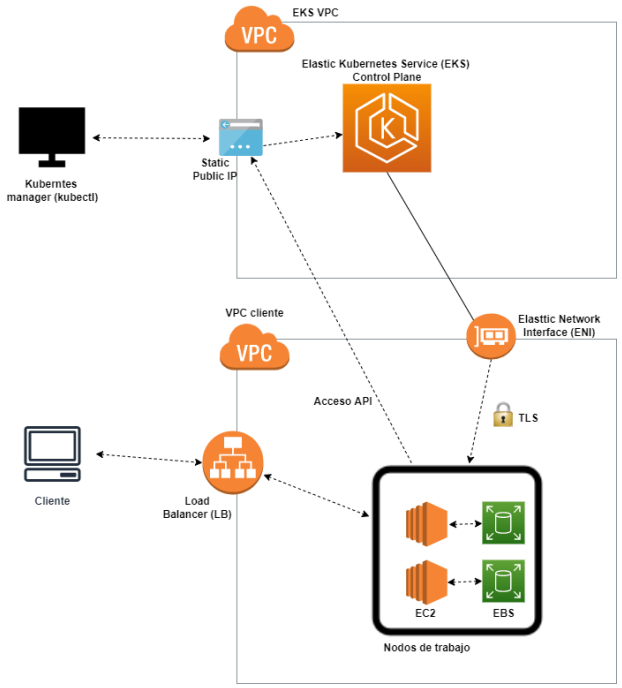


Fig. 1: Entorno desarrollado en AWS

AWS ofrece una infinidad de recursos para poder alojar y gestionar aplicaciones en la nube, para este proyecto se ha estimado el coste total de los servicios mínimos requeridos para cumplir los resultados esperados.

El primer servicio que se debería contratar es el llamado Elastic Kubernetes Service (EKS), este recurso ofrece la posibilidad de generar el *Control Plane* de Kubernetes en un nodo y poder gestionar todo el entorno y sus microservicios.

Para poder crear un clúster con varios nodos y así repartir la carga de trabajo, será necesario disponer del recurso Elastic Compute Cloud (EC2). Cada instancia EC2 es una máquina virtual en la nube totalmente personalizable según los requerimientos que el usuario imponga (Sistema Operativo, procesador, memoria, etc.). Se han configurado dos instancias, ya que en una se encontrarían las tres aplicaciones del clúster y la otra se convertiría en un nodo "esclavo" de Jenkins. El nodo "esclavo" sería el encargado de ejecu-

tar todas las *pipelines*, puesto que requieren una mayor capacidad de cómputo. Además, no es aconsejable correr las *pipelines* en el mismo nodo donde se encuentra el servidor de Jenkins.

Un recurso complementario para los nodos del clúster es el llamado Elastic Block Store (EBS), este servicio de AWS permite crear volúmenes de almacenamiento y asociarlos a las instancias de Amazon EC2. Una vez los EBS han sido asociados a los nodos, se puede manejar un sistema de archivos sobre estos volúmenes o ejecutar una base de datos. En ese nuevo espacio de memoria, se almacenaría toda la configuración de los distintos microservicios del clúster y los datos que generasen sobre las aplicaciones ejecutadas.

Finalmente, para gestionar la conectividad en la red del clúster sería necesario contratar dos servicios más. El primer recurso es una Red Privada en la Nube (VPC), gracias a un VPC las diferentes aplicaciones del clúster se podrán comunicar entre ellas y gestionar la seguridad en el tráfico de datos. El segundo elemento a contratar al proveedor AWS para configurar las peticiones al clúster, es un Balanceador de Carga (LB) y así manipular los datos que entran al clúster, redirigir el tráfico entre los diferentes nodos y proporcionar un punto público de acceso a las aplicaciones del entorno desarrollado.

A continuación se muestra la tabla que se ha comentado al inicio del capítulo, se puede observar que el coste total al finalizar este proyecto hubiese sido de 1545,10 USD.

Coste de AWS					
Servicios	EKS	EC2	VPC	LB	EBS
Instancias	1	2	2	1	2
Precio/hora	0,1	0,30	0,9619	0,0264	0,22
Horas/mes	100	100	100	100	100
Coste/mes	10	60	192,19	2,64	44
Meses trabajo	5	5	5	5	5
Coste servicio	50	3000	961,90	13,23	220
Coste total	1545,10 USD				

TABLA 1: ESTIMACIÓN COSTES DE AWS

Una vez finalizada la evaluación de recursos, económicamente se escapa del presupuesto del proyecto, por esa razón, se ha adquirido una Raspberry Pi 4 para hacer una aproximación del entorno que se quería desplegar en AWS. El procedimiento que se ha seguido para su adquisición y preparación para alojar un clúster de Kubernetes se explica en los capítulos 6.1.2 y 6.1.3.

6.1.2. Diseño sobre un entorno local

Después de descartar la opción de llevar a cabo este proyecto en un proveedor de cómputo en la nube, se ha decidido simular el clúster en un entorno local de desarrollo. Para ello, se va a trabajar en un ordenador portátil y una Raspberry Pi 4 (RPi4). Este ordenador de placa única, dispone de un procesador *Quad-core* con una frecuencia de 1.5GHz y arquitectura ARM v8, una memoria RAM de 4GB, soporte para almacenaje externo (mediante una tarjeta Micro SD) y conectividad WiFi, Ethernet y Bluetooth.

A continuación, se encuentra la Figura 2 donde se representa un diagrama del entorno CI/CD desarrollado a lo largo del proyecto. En ella, se puede observar los dos nodos de cómputo, la red común, las aplicaciones que se ejecutan en cada computador y el puerto de red por el cual acceder a ellas.

Al crear un clúster, es necesario establecer que rol va a tener cada nodo, ya que uno debe ser el máster y el o los demás esclavos. El rol máster se ha asignado al ordenador portátil porque dispone de mayor capacidad de cómputo, por lo tanto, tendrá más facilidades para ejecutar el *Control Plane* de Kubernetes.

Si se observa al máster en la Figura 2, este está alojando tres microservicios llamados Jenkins, PostgreSQL y SonarQube. La instancia de Jenkins de este nodo, será la encargada de asignar la ejecución de las *pipelines* en la que está desplegada en la RPi4 con el fin de distribuir la carga de trabajo, generar un entorno CI/CD y conectar con SonarQube. Por otro lado, para que SonarQube pueda realizar su trabajo, necesita estar conectado con una base de datos como PostgreSQL para almacenar todos los datos de los análisis realizados.

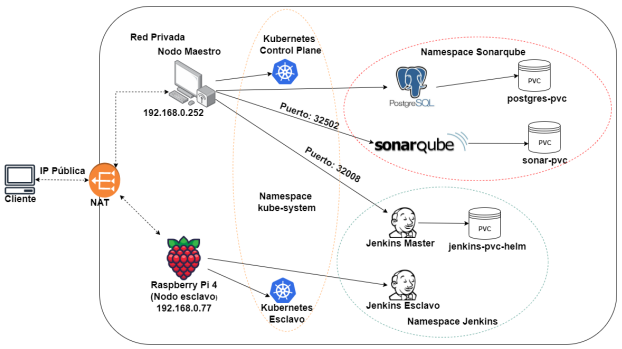


Fig. 2: Entorno desarrollado en RPi4 y ordenador portátil

Finalmente, en la Figura 2, se encuentra la RPi4 actuando como nodo esclavo en el clúster y ejecutando la instancia trabajadora de Jenkins además del servicio de Kubernetes para recibir las órdenes del máster.

Ambos nodos de cómputo se encuentran bajo una red local, esto implica que para acceder a las aplicaciones desde el exterior se hace a través de la misma IP pública de un router, siendo este mismo el encargado de redirigir el tráfico a cada nodo según unas reglas de red previamente configuradas.

Para poder utilizar la RPi4, es necesario disponer de algunos recursos extra como una fuente de alimentación específica, memoria MicroSD, etc. En la Tabla 2 se muestran todos los productos adquiridos para establecer el entorno de trabajo. Junto a cada recurso, se refleja el coste de compra y finalmente el coste de la totalidad del proyecto. Si se comparan los costes finales entre las tablas 1 y 2, destaca la gran diferencia entre ellos, ya que el presupuesto de este proyecto en AWS sería de 1545,10 USD y el precio final empleando una RPi4 es 110,35 USD.

6.1.3. Configurar la Raspberry Pi 4

El primer paso para poder utilizar una Raspberry Pi 4 es la selección e instalación del sistema operativo que se va a

Coste Raspberry Pi 4				
Recurso	RPi4	Fuente de Alimentación	MicoSD	Caja
Coste	73,13	9,90	14,32	13
Coste total	110,35 USD			

TABLA 2: COSTE TOTAL DEL PROYECTO

ejecutar en el equipo. Debido a que el controlador de Kubernetes está diseñado para ser usado en Linux y por cierta familiarización con la distribución Ubuntu, el sistema operativo que se usará en la RPi4 es Ubuntu Server.

Una vez escogido el sistema operativo, se empieza a realizar su montaje en la memoria MicroSD que se va a usar en el nodo. La compañía Raspberry Pi Foundation ofrece una manera rápida de efectuar el montaje del sistema operativo, a través de su herramienta *Raspberry PI Imager* es posible seleccionar que distribución de Linux queremos instalar y en que espacio de memoria.

Cuando ha finalizado la instalación del sistema operativo, es momento de hacer unas pequeñas modificaciones personales para facilitar su conectividad y gestión. Dado que siempre se accederá a la Raspberry Pi de manera remota y ofrecerá unas aplicaciones públicas, se le asignará una dirección IP estática dentro de una red local para facilitar la redirección de tráfico siempre hacia un mismo punto.

Hay varias formas de establecer una IP estática en un computador, pero la configuración que se ha implementado en este proyecto es sobre *Netplan* y la modificación del fichero *50-cloud-init.yaml* tal y como se muestra en la Figura 1 del Apéndice 2. Este documento ofrece una manera sencilla de realizar la configuración de la conexión a Internet, ya sea vía WiFi o Ethernet. En la Figura 1 del Apéndice 2, se puede apreciar que hay algunas zonas pixeladas debido a que son las credenciales de la red local y que la IP estática establecida para la RPi4 es 192.168.0.77. A través de esta dirección se podrá acceder al servidor, configurar el nodo y utilizar sus aplicaciones.

Una vez la RPi4 dispone de conexión a Internet, ya se puede acceder a ella en remoto con SSH y empezar a desarrollar todo el proyecto. Es posible conectarse por primera vez al servidor a través del usuario genérico *ubuntu*, ya que el sistema operativo es Ubuntu Server, pero una vez realizado el primer acceso se creará un nuevo usuario y se eliminará el utilizado en la primera conexión. De esta forma, ya no habrá ningún usuario público, se generará una pareja de llaves criptográficas (pública y privada) para el nuevo usuario y se establecerán los privilegios específicos para mejorar la seguridad del servidor. Como resultado, tal y como se muestra en la Figura 2 del Apéndice 2, disponemos de un servidor operativo identificado como *gfalco-raspi* ejecutándose en una Raspberry Pi 4.

6.2. Crear el clúster

Hay varias herramientas para inicializar el controlador de un clúster local (sin proveedor de cómputo en la nube) de Kubernetes como Minikube, Kind o K3s donde cada uno tiene sus propias características y están diseñados para ciertos entornos. En este proyecto, se ha escogido a MicroK8s [12] como la distribución de Kubernetes para la creación de nuestro clúster y todos sus microservicios. MicroK8s es ideal para la implementación del proyecto porque está diseñado para ejecutarse en computadores de pocas prestaciones, como la RPi4 donde se está desarrollando el clúster. Asimismo, una de las características de MicroK8s que ha ayudado a ser la candidata principal para este proyecto es su compatibilidad con las arquitecturas ARM, la misma sobre la que trabaja uno de nuestros nodos.

Una vez instalado MicroK8s en la RPi4 y en el ordenador portátil, ya se puede inicializar la herramienta dando como resultado la creación de dos clústeres de un solo nodo cada uno. El siguiente paso consiste en enlazar los dos nodos para dar lugar a un único clúster, donde el ordenador con más capacidad de cómputo sea el máster y la RPi4 el esclavo. Existen varias formas para añadir un nodo a un clúster, pero MicroK8s ofrece una vía más sencilla donde con un comando en la terminal de cada nodo se consigue el resultado esperado.

Seguidamente, para terminar con la configuración inicial del clúster, únicamente queda añadir unas etiquetas a cada nodo a través de Kubernetes. Esta configuración extra es de gran ayuda para identificar de manera rápida que rol desempeña cada nodo, asimismo, será de utilidad para clasificar que microservicios deben ejecutarse en cada nodo a través de sus ficheros de configuración. En la Figura 3, se muestran los nodos del clúster, su estado, el rol que desempeñan, la versión de Kubernetes que están usando y su identificador o *hostname*.

```
gerard@gerard-master:~$ kubectl get nodes
NAME            STATUS   ROLES    AGE   VERSION
gfalco-raspi    Ready    worker   15d   v1.21.7-3+4da79ac15211ad
gerard-master   Ready    master   15d   v1.21.7-3+7700880a5c71e2
```

Fig. 3: Nodos del clúster

Para poder visualizar a mayor detalle el estado del clúster y de las aplicaciones que se están ejecutando en él, se ha configurado una funcionalidad llamada *Kubernetes Dashboard*. Al activar el *Kubernetes Dashboard* es posible ver gráficamente, desde el navegador, los recursos utilizados como la CPU o memoria de cada nodo, donde se aloja cada microservicio, etc. En la Figura 1 del Apéndice 3, se muestran algunos de los *pods* que están actualmente activos y contienen los contenedores que están ejecutando aplicaciones en el clúster.

6.3. Creación del entorno CI/CD

Esta sección es una de las más importantes del proyecto, ya que es donde se muestra como se han desplegado cada una de las aplicaciones que se alojan en el clúster. Cada una de estas aplicaciones están compuestas por varios componentes como los *Services*, *Deployments*, los *Persistent Volume* (PV) o los *Persistent Volume Claim* (PVC) explicados en el Capítulo 5.

Para desplegar microservicios en un clúster existen distintas formas de hacerlo, según las necesidades de la aplicación, entorno y conocimientos de los administradores se usará una u otra. En este proyecto se han utilizado dos vías de creación de aplicaciones, se ha optado por usar dos tipos de despliegue con el fin de comparar los procesos y las dificultades de cada una.

En primer lugar, se ha utilizado el método tradicional de Kubernetes basado en la creación de todos los componentes de la aplicación por separado. Al emplear esta vía de despliegue, tienes una gran visión del estado de todos los objetos que has creado y el usuario sabe toda la configuración de la aplicación. Por otro lado, tiene una gran desventaja y es que dificulta el mantenimiento de la aplicación y su automatización. Este inconveniente es debido a que hay que ir manualmente componente por componente actualizando los valores de configuración.

En segundo lugar, se ha usado Helm [13] para desplegar algunos de los microservicios y poder comparar resultados con el anterior método. Helm es una herramienta para gestionar paquetes o aplicaciones en un clúster orquestado por Kubernetes, gracias a esta herramienta es muy sencillo para los administradores de sistemas instalar, modificar o actualizar las aplicaciones y sus valores de configuración. Helm dispone de unas *cartas de navegación* llamadas *Helm Charts*, estas cartas son plantillas predefinidas para el despliegue de aplicaciones donde solo es necesario configurar los valores de las aplicaciones según las necesidades del equipo. Este método es extremadamente útil para la automatización de creación de microservicios, actualizaciones, control de versiones, etc.

Finalmente, tal y como se ha comentado a lo largo del documento, la configuración del clúster y el despliegue de los microservicios, ya sea con Helm o el método tradicional de Kubernetes, se ha hecho a partir de Terraform, y así, poder desplegar y mantener el clúster mediante la técnica CI/CD.

6.3.1. Namespaces

La creación de los *Namespaces* es el primer paso a realizar antes del despliegue de los microservicios, esto nos ayudará a separar cada aplicación y sus componentes para mantener un orden total del clúster. En este proyecto, se ha decidido crear un *Namespace* por cada microservicio, excepto en el caso de SonarQube y PostgreSQL porque son aplicaciones complementarias entre ellas.

Para generar cada *Namespace*, se ha generado un nuevo fichero llamado *namespace.tf* y en él se han declarado los tres objetos a partir del recurso de Terraform denominado *kubernetes_namespace*. Este recurso nos permite establecer un nombre para cada *Namespace*, etiquetas usadas como identificadores y otras anotaciones.

En la Figura 4, se muestran todos los *Namespaces* ocasionados en el clúster a través de Terraform junto con los generados por defecto por el *Control Plane* de Kubernetes.

```
gerard@gerard-master:~$ kubectl get ns
NAME                 STATUS   AGE
kube-system          Active   16d
kube-public          Active   16d
kube-node-lease      Active   16d
default              Active   16d
container-registry   Active   16d
jenkins              Active   14d
sonarqube            Active   14d
nginx                Active   14d
```

Fig. 4: Namespaces del clúster

6.3.2. Jenkins

Jenkins ha sido el primer microservicio en ser desplegado en el clúster, esta aplicación ha sido configurada a través de Helm e implementada con código Terraform. Al usar Helm es necesario crear dos tipos de ficheros para montar este microservicio, el primero es el llamado *jenkins-values.yaml*. Dicho fichero se emplea para declarar toda la configuración de la aplicación como los usuarios iniciales, imagen y versión de Jenkins que se usará en el contenedor, puerto de red, nodo donde se aloja, asignación de PVC y mucho más.

Uno de los factores más importantes al configurar Jenkins, es la selección del nodo donde desplegar la aplicación. Tal y como se ha comentado anteriormente, Jenkins dispone de una instancia máster y otra esclava donde se ejecutan las *pipelines*, con el objetivo de distribuir la carga de trabajo en el clúster se ha configurado el fichero *jenkins-values.yaml* para que Jenkins máster se aloje en el nodo máster y Jenkins esclavo en la RPI4 como nodo trabajador.

La gran ventaja de usar Helm, es que a través de este fichero puedes mantener la configuración de todo el microservicio, ya que en él también se declaran los servicios de comunicación y se puede modificar toda la aplicación cambiando unos pocos valores de este documento.

Por otro lado, en el segundo fichero es donde se crea la aplicación a partir de código Terraform y se utilizan los valores del primero para su configuración interna. En este fichero, se usa el recurso *helm_release* de Terraform para ocasionar las dos instancias de Jenkins, además, a partir de este recurso también se puede configurar en que *Namespace* situar el microservicio, variables de entorno, etc.

En la Figura 1 del Apéndice 4, se muestran todos los componentes que conforman la aplicación de Jenkins como sus servicios de conexión que escuchan el puerto 32008, los PVC donde se almacena toda la información y los *pods* donde se están ejecutando los contenedores con la imagen de Jenkins.

Durante el desarrollo de esta aplicación, se han encontrado varios problemas debido a ser la primera en ser desplegada y la falta de experiencia investigando los errores de Kubernetes. Uno de los errores que frenó más el progreso del proyecto fue la administración de permisos sobre una carpeta interna del nodo, ya que Jenkins necesita disponer de unos permisos especiales sobre el directorio donde almacenará los datos. Encontrar este error requirió varios días de investigación a bajo nivel de Kubernetes y revisar detalladamente los ficheros de registro conocidos como *logs*. Además, la necesidad de proporcionar estos permisos a Jenkins cada vez que se creaba el clúster frenaba la posibilidad de automatizar el proceso de creación. Para solucionar este problema, se ha generado un programa con instrucciones, conocido como *script*, que automáticamente asigna los permisos necesarios a los directorios que guarden los datos de la aplicación.

Finalmente, en la Figura 5 aparece la ventana principal de Jenkins listo para ser usado en un entorno CI/CD.

6.3.3. PostgreSQL

PostgreSQL es un proyecto de código abierto utilizado como sistema de bases de datos relaciones, esta aplicación es vital para nuestro entorno, ya que SonarQube requiere estar conectado a una base de datos como esta. SonarQube,

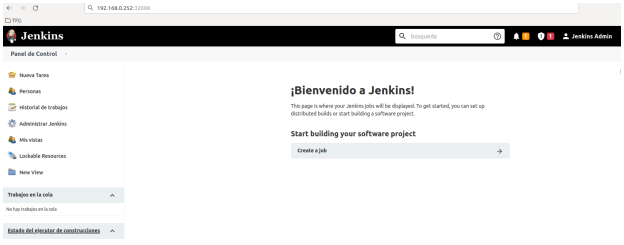


Fig. 5: Jenkins

a diferencia de Jenkins, sin PostgreSQL los contenedores de configuración no podrían iniciar la aplicación y generarían un error de conectividad con la base de datos. Por esa razón, PostgreSQL se ha configurado para que sea desplegada en el mismo *Namespace* que SonarQube, de esta forma estarán todos los recursos organizados según su propósito final.

De lo contrario a Jenkins, en este caso no se ha empleado Helm para la instalación de este microservicio, sino que se ha hecho a partir del método tradicional de Kubernetes. Emplear este método implica la creación específica de cada componente que se quiere incorporar al microservicio, es por eso, que a través de Terraform y sus recursos específicos se ha generado un PVC, un servicio para su conectividad en la red, su propio *Kubernetes configmap* para gestionar las variables de entorno y, finalmente, un *deployment* para desplegar la aplicación en el clúster.

A continuación, en la Figura 2 del Apéndice 4, se muestran todos los componentes generados en el clúster para la utilización de PostgreSQL.

Mientras se realizaba la configuración de esta aplicación se ha tropezado con algunos problemas, el más relevante ha sido la necesidad de cambiar parte del proyecto y desplegar este microservicio en el nodo máster en lugar del esclavo. El problema de utilizar PostgreSQL en una RPi4 es su arquitectura, ya que la aplicación está diseñada para trabajar sobre una arquitectura AMD, en cambio, las RPi4 usan ARM. Resolver este error ocupó gran parte del tiempo invertido en los inicios del proyecto, para descubrirlo, fue necesario revisar gran parte de los ficheros de registro y revisar el código fuente de la imagen de Docker de PostgreSQL.

6.3.4. SonarQube

SonarQube es una herramienta para realizar análisis estáticos de seguridad, es decir, analizar el código fuente de las aplicaciones para detectar vulnerabilidades y otros fallos de ejecución. Esta herramienta es vital para evitar la producción de *software* con fallos de seguridad, además, es un proceso que si se automatiza con herramientas como Jenkins no suponen ningún coste sobre el tiempo de desarrollo del producto final.

Al igual que con PostgreSQL, para desplegar SonarQube en el clúster se ha usado el método tradicional de Kubernetes. Como resultado, se ha creado varios componentes a través de código Terraform como el PVC de la aplicación, servicio de red para poder acceder a través de un navegador web, etc. Todos los recursos que forman este microservicio, pueden verse en detalle en la Figura 3 del Apéndice 4.

A medida que avanzaba la configuración de esta herramienta, se iban encontrando problemas que frenaban el progreso del proyecto. Hay varios errores que se tuvieron que solucionar para utilizar SonarQube, pero hay dos que fueron los que más tiempo llevaron encontrar una solución. El primero es el mismo que ocurrió con PostgreSQL, SonarQube está diseñado sobre una arquitectura AMD, por lo tanto, no podía ser configurada en la RPi4, provocando el cambio de nodo donde desplegar el microservicio.

El segundo error al cual se dedicó más tiempo en referencia a SonarQube, fue conectar correctamente la aplicación con su base de datos. Este microservicio dispone de un contenedor encargado de su configuración inicial, en él se realiza la conexión con la base de datos, si no se puede realizar este lazo SonarQube no puede iniciarse. La relación entre las dos aplicaciones se hace en el *Kubernetes configmap* de SonarQube, ahí se añade la IP interna del contenedor que está ejecutando la aplicación PostgreSQL. Esta IP es dinámica, en consecuencia cada vez que se reinicia todo el microservicio PostgreSQL dicha IP se modifica. Para resolver este problema, a través de Terraform, se ha ideado guardar la IP del contenedor una vez generada en una variable privada del entorno, de este modo, cada vez que se reinicia la base de datos se envía al *Kubernetes configmap* de SonarQube la nueva IP interna.

Finalmente, en la Figura 6 se presenta la página principal de Sonarqube mostrando que está totalmente operativo, conectado a la base de datos y listo para ejecutar sus análisis de seguridad y calidad.

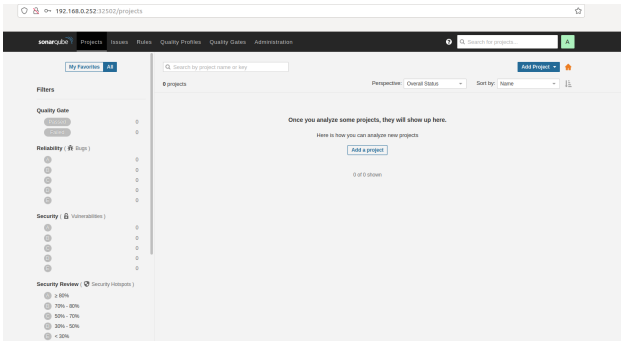


Fig. 6: Página principal de SonarQube

6.4. Implementación de la práctica CI/CD

La utilización de la técnica CI/CD es uno de los objetivos principales de este proyecto, es por eso, que se ha creado el entorno comentado a lo largo de este documento. Para demostrar como se configura, usa y sus ventajas, esta práctica se ha aplicado en dos situaciones totalmente distintas.

6.4.1. CI/CD para el despliegue del clúster

Crear un clúster con diferentes microservicios es un trabajo que requiere una gran inversión de tiempo a lo largo de toda su configuración, si además se añade el tiempo que se tarda en aplicar los nuevos cambios, la dificultad del proyecto aumenta considerablemente.

Para agilizar el despliegue del entorno y aplicar la técnica CI/CD, se ha originado un nuevo Jenkins externo al clúster a través de un contenedor de Docker. Esta nueva instancia se ha configurado para que todos los despliegues y/o modificaciones del clúster sean totalmente automáticas, para

conseguir dicha funcionalidad se han seguido las siguientes dos fases.

En primer lugar, es necesario generar una nueva *pipeline* a través del lenguaje Groovy. Todo el código donde se configura la *pipeline* debe escribirse en un fichero específico llamado *Jenkinsfile*, este documento se ha subido al repositorio de GitHub donde se encuentra el código que despliega el clúster. La *pipeline* dispone de seis etapas que representan todo el trabajo manual que se debería realizar si no se usase la práctica CI/CD. Estas seis fases se basan en recoger el código del repositorio de GitHub, instalar y seleccionar Terraform para utilizarlo en la *pipeline*, inicializar Terraform en la carpeta donde se encuentra el código fuente, generar un *Terraform Plan* donde se muestran todos los cambios introducidos en las últimas modificaciones del código y finalmente ejecutar un *Terraform Apply*. Antes de esta última fase, hay una pequeña etapa después de efectuar el *Terraform Plan* donde se pedirá la aprovación al usuario para continuar la ejecución. Esta fase de control fue añadida porque al realizar un *Terraform Apply* es cuando se aplicarán todos los cambios en la infraestructura que se han mostrado anteriormente en el *Terraform Plan*. Una vez se acepta el *Terraform Apply*, todas las modificaciones se empiezan a introducir en el entorno gracias a la configuración de la IP de los nodos del clúster a través de Terraform.

En segundo lugar, es necesario crear un nuevo elemento en la instancia de Jenkins llamado *pipeline multibranch*. Este objeto permite ejecutar el código Groovy desarrollado en la fase anterior, de esta manera el servidor de Jenkins sabrá que instrucciones realizar en esta *pipeline* en específico. La *pipeline* es tipo *multibranch* porque al conectarla con un repositorio de GitHub cada ejecución de la *pipeline* se iniciará de manera automática y será clasificada según la rama del repositorio donde se haya subido el código del proyecto. Para ello, se ha configurado un *webhook* en el repositorio de GitHub donde cada vez que ocurra un cambio se mandará una señal a Jenkins y este ejecutará automáticamente la *pipeline* codificada anteriormente. Como resultado, obtenemos que con solo subir el código a GitHub se ha desplegado nuevamente el clúster con todos los cambios aplicados.

A lo largo del desarrollo de esta etapa del proyecto se han encontrado varios problemas, un ejemplo es la conexión entre el repositorio de GitHub y la instancia de Jenkins. Jenkins se encuentra dentro de una red privada y detrás de un router que se encarga de direccionar el tráfico que entra, como GitHub es un servicio público de Internet se ha necesitado abrir un puerto del router y encaminar las peticiones que entren por ese puerto hacia la IP privada del Jenkins. Como resultado, se consigue una conexión bidireccional entre los dos servicios.

Otro inconveniente encontrado ha sido la configuración del código Terraform para que Jenkins pueda desplegar el clúster a distancia, este entorno debe poder ser creado desde cualquier ordenador hacia unos nodos específicos. Para solucionar este problema, se ha generado un fichero de Terraform dedicado a configurar las IPs de los nodos y las credenciales para acceder a ellos. Para añadir un grado más de seguridad en el entorno, este nuevo fichero no está añadido al repositorio de GitHub, sino que se ha subido a la instancia de Jenkins a través de un sistema de credenciales encriptadas que proporciona este servicio. Como resultado, Jenkins sabe donde desplegar el clúster y solo se puede acceder a este fichero de configuración a través de las credenciales de la aplicación o del ordenador donde se está ejecutando el contenedor de Jenkins.

A continuación, en la Figura 7, se muestra un conjunto de ejecuciones finalizadas con éxito de la *pipeline* conectada con el repositorio de GitHub. En este ejemplo, se está trabajando sobre la rama máster del proyecto porque es la encargada de aplicar todos los cambios finales en el clúster. En la siguiente imagen, se pueden apreciar las seis fases de la *pipeline* y el tiempo que se ha tardado en finalizar cada una de ellas.

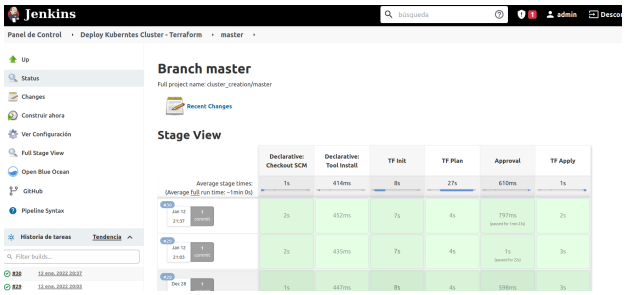


Fig. 7: Pipeline para el despliegue del clúster

6.4.2. CI/CD para el desarrollo ágil de software

Una vez están todos los microservicios del clúster activos y configurados, se puede proceder a la implementación de las *pipelines* de Jenkins para automatizar todo el proceso de pruebas del *software* desarrollado.

Antes de generar las *pipelines*, se ha instalado un *plugin* en el servidor de Jenkins llamado *Sonar-Scanner*. Esta herramienta permite conectar el servidor de SonarQube del clúster con la instancia de Jenkins. Además, es necesario generar un nuevo *webhook* desde SonarQube para que este pueda comunicar a Jenkins la finalización de los análisis de calidad y seguridad.

De igual modo que en el Capítulo 6.4.1, para conseguir la ejecución automática de las *pipelines* en Jenkins, es necesario crear un *webhook* en cada repositorio de GitHub con el fin de permitir la conexión entre ambos servicios. Además, se requiere habilitar un puerto del router de la red privada para que direccione las peticiones de GitHub hacia el servidor Jenkins del clúster.

Las *pipelines* que se han codificado constan de tres fases, estas serán comunes para todos los proyectos que se quiera automatizar el proceso de pruebas de calidad de código con SonarQube.

El primer paso es recoger automáticamente el código de cada repositorio de GitHub, después se instala y configura la herramienta *Sonar-Scanner* para el proyecto. En la tercera fase, se comunica a SonarQube que se quiere realizar los análisis y se le proporciona el código fuente. En este instante, SonarQube empieza las pruebas automáticamente y evalúa si el *software* es seguro o no a partir de unas condiciones a cumplir configuradas anteriormente, todas estas reglas o condiciones se resumen en un resultado final del análisis llamado *Quality Gate*. Finalmente, SonarQube devuelve los resultados a Jenkins y este comunica al repositorio del proyecto en GitHub el estado final de los análisis de seguridad y de la *pipeline*. En la Figura 8, se presenta como es la vista de una *pipeline* y todas sus fases desde la

interfaz de Jenkins. Hay varios detalles que resaltan en esta imagen, entre ellos encontramos que una *pipeline* llamada *Mail server - Automatic Security analysis* se está ejecutando sobre la rama *máster* del proyecto *Mail Server*, las tres fases han finalizado con éxito y el código del proyecto ha superado las pruebas de calidad, ya que la *Quality Gate* de SonarQube se encuentra en *Passed*.

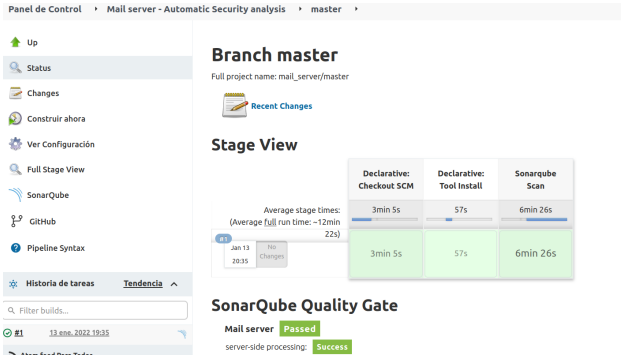


Fig. 8: Pipeline con análisis de código automático

Si el proyecto ha pasado con éxito las pruebas, la *pipeline* habrá finalizado correctamente, de lo contrario, esta aparecerá como fallida en Jenkins y en GitHub. A continuación, tal y como se puede apreciar en la Figura 9, en GitHub se muestra como la ejecución de la anterior *pipeline* en la última modificación del proyecto ha terminado con éxito, pero, en la antecedente los tests han fallado.

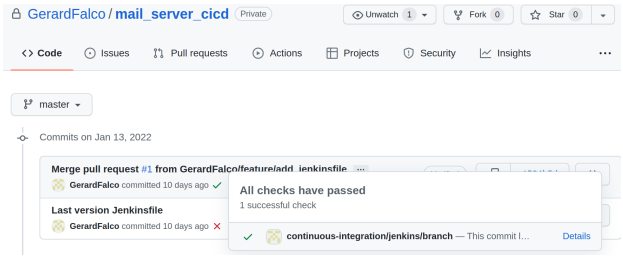


Fig. 9: Resultados de Jenkins recibidos en GitHub

Para comprobar el funcionamiento de las *pipelines*, se han usado tres proyectos distintos. Estos tres trabajos están escritos en diferentes lenguajes de programación como Python, Java, C y más. Al utilizar varios lenguajes, se pueden ver distintos fallos, posibles vulnerabilidades o detalles a mejorar en el estilo de cada código. En la Figura 10, se muestra como SonarQube presenta los resultados obtenidos en un análisis realizado al proyecto *Mail Server* mencionado anteriormente. Las conclusiones más importantes que se pueden extraer de estos resultados, es que no hay ninguna vulnerabilidad detectada en el código fuente ni ningún *Bug* o error de desarrollo. Por otro lado, SonarQube ha detectado la existencia de cuatro errores de calidad de código o *Code Smells*, como consecuencia, habría que invertir aproximadamente cinco minutos en solucionar estos errores y seguir los reglas de calidad. Además, SonarQube también ha detectado un posible fragmento de código sensible en términos de seguridad, dicho error se denomina *Security Hotspot* y deberá ser revisado para catalogarlo como falso positivo o fallo de seguridad y remediarlo.

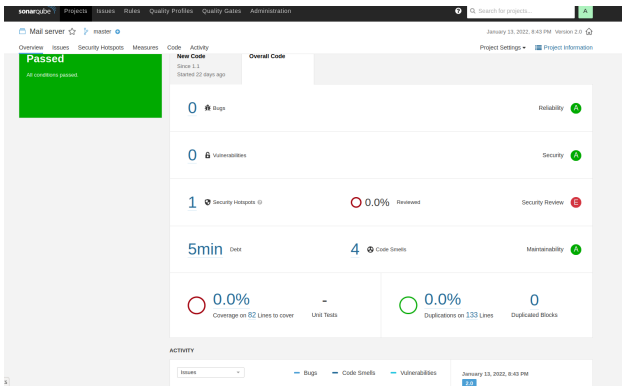


Fig. 10: Resultados de SonarQube

7 Conclusiones

Crear un entorno automatizado de producción de *software* era el objetivo principal de este proyecto, tal y como se ha podido observar a lo largo del documento, este propósito se ha podido cumplir gracias a la correcta configuración de todos los microservicios del clúster.

Para lograr el desarrollo de este entorno, se ha seguido de cerca la planificación comentada en la sección 4 pudiendo alcanzar los diferentes grupos de objetivos como la configuración de la RPi4 actuando como un nodo del clúster.

Una de las conclusiones que se pueden extraer de este proyecto, es la necesidad de una gran capacidad de cómputo para los nodos del clúster según el propósito de este. Según la planificación inicial, el clúster debía disponer de un microservicio más llamado Nexus Repository Manager 3, actuando como repositorio de artefactos, esta aplicación no pudo ser desplegada en el clúster debido a que los dos nodos no disponían de suficientes recursos como para soportar un microservicio más ejecutándose en ellos. Este problema supuso cambiar la planificación, añadir una aplicación menos, quitar una fase de las *pipelines* y en consecuencia variar los resultados esperados del proyecto.

Por otro lado, en este proyecto, se ve reflejado la importancia de una buena distribución de la carga de trabajo en un clúster. El hecho de separar las dos instancias de Jenkins entre los distintos nodos supuso una mejora absoluta de rendimiento del máster, con esa mejora de rendimiento las otras aplicaciones empezaron a funcionar con más fluidez y efectividad.

Uno de los objetivos más importantes de este proyecto era demostrar las ventajas de usar la práctica CI/CD para desarrollar *software*, después de crear el entorno y configurar las *pipelines*, se ha podido aplicar CI/CD sobre proyectos personales antiguos como el *Mail Server* elaborados inicialmente sin usar esta técnica. Después de ver los resultados, puedo afirmar que el tiempo de desarrollo de esos trabajos habría sido inferior si desde un inicio hubiese dispuesto del entorno de producción elaborado a lo largo de este proyecto. Gracias a SonarQube y las *pipelines* de Jenkins, todo el trabajo de compilación y búsqueda de errores ha quedado automatizado y la única preocupación del desarrollador es arreglar y generar código fuente del *software*.

Finalmente, la última conclusión de este proyecto se basa en la utilización conjunta de las técnicas CI/CD e IaC. En los inicios de la investigación, mientras se desplegaban distintos clústeres de prueba para adaptarse a las tecnologías,

el trabajo manual que se realizaba en cada configuración o modificación en el entorno era lento y repetitivo. Con la introducción de Terraform, todos los nuevos despliegues de los entornos eran más ágiles y sencillos, pero, aún se seguía efectuando una gran cantidad de trabajo manual como el *Terraform Plan* y otras tareas. Cuando se desplegó la instancia de Jenkins externa al clúster, al juntar IaC con CI/CD, la velocidad de progreso del proyecto aumentó considerablemente debido a que todas las acciones repetitivas y manuales habían sido automatizadas. Como conclusión, al disponer de la automatización de tareas no esenciales, todos los esfuerzos estaban enfocados en la mejora continua del entorno dando lugar a unos resultados de mayor calidad.

8 Posibles mejoras

La mejora más importante que se puede hacer en este proyecto, si se dispone del presupuesto, es mover todo el entorno que se ha creado a un proveedor de cómputo en la nube como AWS. A lo largo del desarrollo, se han encontrado muchos problemas con la accesibilidad a las aplicaciones, debido a que estas no están públicas a Internet ni disponen de su propio nombre de servicio. Con AWS, gran parte de estas dificultades se solucionan de manera automática, se puede escalar los recursos necesarios según las peticiones recibidas de los usuarios, más seguridad gracias a los recursos y políticas del proveedor y mucho más.

Una mejora significativa que se puede añadir a este proyecto es el grado extra de seguridad a través de Istio y Falco, usar estas dos herramientas era el último objetivo de este proyecto que finalmente no se ha podido llevar a cabo por falta de tiempo. Con Istio correctamente integrado en el clúster, se puede aumentar de manera considerable la seguridad del entorno ya que toda la información que navegase entre los distintos microservicios sería controlada por esta herramienta. Además, gracias a Falco, se podría monitorizar el estado de cada uno de los *Pods* y contendores del clúster, de esta manera, cualquier acción inusual sería reportada a los administradores del clúster.

Agradecimientos

Para finalizar este proyecto, me gustaría agradecer a varias personas todo el soporte que me han brindado a lo largo del desarrollo de este trabajo.

La primera persona a la que quiero darle las gracias es a Miguel Carpio, él se ha encargado de realizar el seguimiento del proyecto durante todos estos meses e indicarme siempre como mejorar cada uno de los detalles del informe final.

En segundo y último lugar, me gustaría agradecerle a la compañía SQLI Barcelona la oportunidad que me brindó en noviembre de 2020 al contratarme para empezar mi carrera laboral y descubrir estas apasionantes tecnologías. En especial, estoy muy agradecido a mi mentor y amigo Emanuel Doiny por todos los consejos que me ha proporcionado y formarme como ingeniero pero sobre todo como persona.

Referencias

[1] The Kubernetes Authors, *What is Kubernetes?*, The Linux Foundation, Julio 2021. Acceso: Setiembre 2021. [En línea]. Disponible en: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[2] Ed Kaim y Mike Jacobs, *What is Infrastructure as Code?*, Microsoft, Junio 2021. Acceso: Setiembre 2021. [En línea]. Disponible en: https://docs.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code

[3] Red Hat Team, *What is CI/CD?*, Red Hat, Enero 2018. Acceso: Setiembre 2021. [En línea]. Disponible en: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[4] Martin Heller, *What is Jenkins?*, Red Hat, Marzo 2020. Acceso: Setiembre 2021. [En línea]. Disponible en: https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html

[5] Javier Pastor, *Raspberry Pi 4 Model B, análisis: doble de potencia para un mini PC milagroso, pequeñito, pero matón*, Xataka, Julio 2019. Acceso: Setiembre 2021. [En línea]. Disponible en: https://www.xataka.com/ordenadores/raspberry-pi-4-analisis-caracteristicas-precio-especificaciones

[6] Brian Bensky, Nate Baker y Mary Goodhart, *What is Terraform and Why is it Important?*, Fairwinds, Julio 2020. Acceso: Setiembre 2021. [En línea]. Disponible en: https://www.fairwinds.com/blog/what-is-terraform-and-why-is-it-important

[7] Kheenvraj Lomror, *SonarQube: What it is and why to use it?*, Login Radius, Julio 2020. Acceso: Noviembre 2021. [En línea]. Disponible en: https://www.loginradius.com/blog/async/sonarqube/

[8] Qianying Liao, *Modelling CI/CD Pipeline Through Agent-Based Simulation*, University of Coimbra, 2020. Acceso: Octubre 2021. [En línea]. Disponible en: https://doi.ieeecomputersociety.org/10.1109/ISSREW51248.2020.00059

[9] Ben Lutkevich, *What is Scrum?*, Tech Target, Octubre 2021. Acceso: Noviembre 2021. [En línea]. Disponible en: https://searchsoftwarequality.techtarget.com/Scrum

[10] The Istio Authors, *The Istio service mesh*, Istio, Marzo 2020. Acceso: Setiembre 2021. [En línea]. Disponible en: https://istio.io/latest/docs/concepts/security/

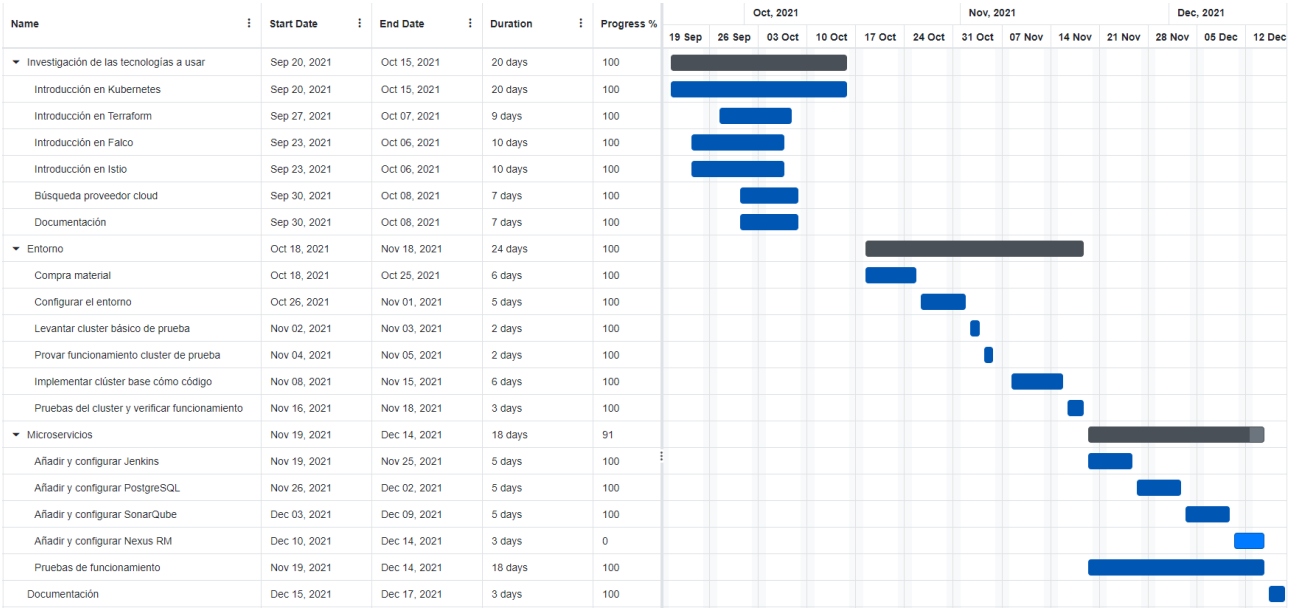
[11] The Falco Authors, *An Introduction to Kubernetes Security using Falco*, Linux Foundation, Diciembre 2020. Acceso: Setiembre 2021. [En línea]. Disponible en: https://falco.org/blog/intro-k8s-security-monitoring/

[12] Anastasia Valti, *What can you do with MicroK8s?*, Ubuntu, Julio 2020. Acceso: Noviembre 2021. [En línea]. Disponible en: https://ubuntu.com/blog/what-can-you-do-with-microk8s

[13] Vladimir Fedak, *What is Helm and why you should love it?*, Hackernoon, Junio 2018. Acceso: Diciembre 2021. [En línea]. Disponible en: https://hackernoon.com/what-is-helm-and-why-you-should-love-it-74bf3d0aafc

APÉNDICE

A.1. Diagrama de Gantt

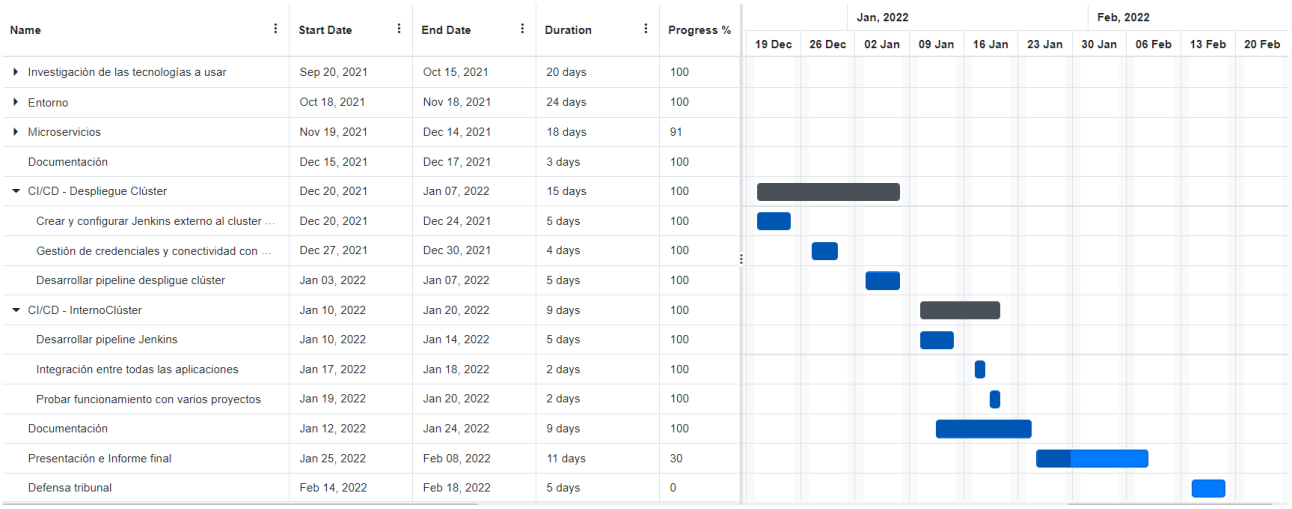


Name	Start Date	End Date	Duration	Progress %
Investigación de las tecnologías a usar	Sep 20, 2021	Oct 15, 2021	20 days	100
Introducción en Kubernetes	Sep 20, 2021	Oct 15, 2021	20 days	100
Introducción en Terraform	Sep 27, 2021	Oct 07, 2021	9 days	100
Introducción en Falco	Sep 23, 2021	Oct 06, 2021	10 days	100
Introducción en Istio	Sep 23, 2021	Oct 06, 2021	10 days	100
Búsqueda proveedor cloud	Sep 30, 2021	Oct 08, 2021	7 days	100
Documentación	Sep 30, 2021	Oct 08, 2021	7 days	100
Entorno	Oct 18, 2021	Nov 18, 2021	24 days	100
Compra material	Oct 18, 2021	Oct 25, 2021	6 days	100
Configurar el entorno	Oct 26, 2021	Nov 01, 2021	5 days	100
Levantar cluster básico de prueba	Nov 02, 2021	Nov 03, 2021	2 days	100
Provar funcionamiento cluster de prueba	Nov 04, 2021	Nov 05, 2021	2 days	100
Implementar clúster base cómo código	Nov 08, 2021	Nov 15, 2021	6 days	100
Pruebas del cluster y verificar funcionamiento	Nov 16, 2021	Nov 18, 2021	3 days	100
Microservicios	Nov 19, 2021	Dec 14, 2021	18 days	91
Añadir y configurar Jenkins	Nov 19, 2021	Nov 25, 2021	5 days	100
Añadir y configurar PostgreSQL	Nov 26, 2021	Dec 02, 2021	5 days	100
Añadir y configurar SonarQube	Dec 03, 2021	Dec 09, 2021	5 days	100
Añadir y configurar Nexus RM	Dec 10, 2021	Dec 14, 2021	3 days	0
Pruebas de funcionamiento	Nov 19, 2021	Dec 14, 2021	18 days	100
Documentación	Dec 15, 2021	Dec 17, 2021	3 days	100

Oct, 2021 | Nov, 2021 | Dec, 2021

19 Sep | 26 Sep | 03 Oct | 10 Oct | 17 Oct | 24 Oct | 31 Oct | 07 Nov | 14 Nov | 21 Nov | 28 Nov | 05 Dec | 12 Dec

Fig. 1: Diagrama de Gantt Setiembre 2021 - Diciembre 2021



Name	Start Date	End Date	Duration	Progress %
Investigación de las tecnologías a usar	Sep 20, 2021	Oct 15, 2021	20 days	100
Entorno	Oct 18, 2021	Nov 18, 2021	24 days	100
Microservicios	Nov 19, 2021	Dec 14, 2021	18 days	91
Documentación	Dec 15, 2021	Dec 17, 2021	3 days	100
CI/CD - Despliegue Clúster	Dec 20, 2021	Jan 07, 2022	15 days	100
Crear y configurar Jenkins externo al cluster ...	Dec 20, 2021	Dec 24, 2021	5 days	100
Gestión de credenciales y conectividad con ...	Dec 27, 2021	Dec 30, 2021	4 days	100
Desarrollar pipeline despligue clúster	Jan 03, 2022	Jan 07, 2022	5 days	100
CI/CD - InternoClúster	Jan 10, 2022	Jan 20, 2022	9 days	100
Desarrollar pipeline Jenkins	Jan 10, 2022	Jan 14, 2022	5 days	100
Integración entre todas las aplicaciones	Jan 17, 2022	Jan 18, 2022	2 days	100
Probar funcionamiento con varios proyectos	Jan 19, 2022	Jan 20, 2022	2 days	100
Documentación	Jan 12, 2022	Jan 24, 2022	9 days	100
Presentación e Informe final	Jan 25, 2022	Feb 08, 2022	11 days	30
Defensa tribunal	Feb 14, 2022	Feb 18, 2022	5 days	0

Jan, 2022 | Feb, 2022

19 Dec | 26 Dec | 02 Jan | 09 Jan | 16 Jan | 23 Jan | 30 Jan | 06 Feb | 13 Feb | 20 Feb

Fig. 2: Diagrama de Gantt Diciembre 2021 - Febrero 2022

A.2. Acceso a la RPi4

Open — 50-cloud-init.yaml [Read-Only] — writable /media/gfalco/writable/etc/netplan — Save

```
# This file is generated from information provided by the datasource.  Changes
# to it will not persist across an instance reboot.  To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    version: 2
    ethernets:
        eth0:
            optional: true
            dhcp4: true
    # add wifi setup information here ...
    wifis:
        wlan0:
            optional: true
            access-points:
                [redacted]:
                    password: [redacted]
            # for static IP
            dhcp4: no
            addresses: [192.168.0.77/24]
            gateway4: [redacted]
            nameservers:
                addresses: [8.8.8.8,8.8.4.4,127.0.0.53]
```

Fig. 1: Configuración IP estática en RPi4

gfalco@gfalco-raspi: ~

Archivo Editar Ver Buscar Terminal Ayuda

```
gerard@gerard-master:~$ ssh gfalco@192.168.0.77
gfalco@192.168.0.77's password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-1046-raspi aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Jan 23 18:03:16 UTC 2022

  System load:  0.35               Processes:               170
  Usage of /:   10.3% of 114.41GB  Users logged in:         0
  Memory usage: 12%                IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                 IPv4 address for wlan0:   192.168.0.77
  Temperature:  35.5 C

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

34 updates can be applied immediately.
15 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Sun Jan 23 18:03:02 2022 from 192.168.0.252
microk8s is not running, try microk8s start
[sudo] password for gfalco:
gfalco@gfalco-raspi:~$ hostname
gfalco-raspi
gfalco@gfalco-raspi:~$
```

Fig. 2: RPi4 configurada

A.3. Panel de control de Kubernetes

Pods

	Name	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created	
✓	calico-kube-controllers-f7868dd95-6twq9	k8s-app: calico-kube-controllers pod-template-hash: f7868dd95	gerard-master	Running	13	-	-	17 days ago	⋮
✓	metrics-server-8bbfb4bdb-9fzjb	k8s-app: metrics-server pod-template-hash: 8bbfb4bdb	gerard-master	Running	12	-	-	17 days ago	⋮
✓	coredns-7f9c69c78c-x5n97	k8s-app: kube-dns pod-template-hash: 7f9c69c78c	gerard-master	Running	11	-	-	17 days ago	⋮
✓	hostpath-provisioner-6bf5c6cf9-bmhfd	k8s-app: hostpath-provisioner pod-template-hash: 6bf5c6cf9	gerard-master	Running	11	-	-	17 days ago	⋮
✓	dashboard-metrics-scraper-78d7698477-5jpxs	k8s-app: dashboard-metrics-scraper pod-template-hash: 78d7698477	gerard-master	Running	11	-	-	17 days ago	⋮
✓	kubernetes-dashboard-85fd7f45cb-p5bz5	k8s-app: kubernetes-dashboard pod-template-hash: 85fd7f45cb	gerard-master	Running	13	-	-	17 days ago	⋮
✓	calico-node-nq5xk	controller-revision-hash: 7f54746f96 k8s-app: calico-node Show all	gerard-master	Running	7	-	-	a day ago	⋮
✓	calico-node-xc9r6	controller-revision-hash: 7f54746f96 k8s-app: calico-node Show all	gfalco-raspi	Running	2	-	-	a day ago	⋮

Fig. 1: Kubernetes Dashboard

A.4. Componentes de los microservicios

```
gerard@gerard-master:~$ kubectl get all -n jenkins
NAME            READY   STATUS    RESTARTS   AGE
pod/jenkins-0   2/2     Running   2          15m

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/jenkins-agent   ClusterIP   10.152.183.72    <none>        50000/TCP        14d
service/jenkins         NodePort    10.152.183.202   <none>        8080:32008/TCP   14d

NAME                       READY   AGE
statefulset.apps/jenkins   1/1     14d
gerard@gerard-master:~$ kubectl get pvc -n jenkins
NAME               STATUS   VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS        AGE
jenkins-pvc-helm   Bound    jenkins-pv   20Gi       RWO            microk8s-hostpath   14d
gerard@gerard-master:~$
```

Fig. 1: Componentes de Jenkins

```
gerard@gerard-master:~$ kubectl get all -n sonarqube | grep -e postgres
pod/postgres-58dd7b8d4c-g6dln    1/1     Running   11         15d
service/postgres    ClusterIP   10.152.183.249   <none>        5432/TCP         15d
deployment.apps/postgres    1/1     1            1           15d
replicaset.apps/postgres-58dd7b8d4c    1         1         1       15d
gerard@gerard-master:~$ kubectl get  pvc -n sonarqube | grep -e postgres
postgres-pvc   Bound    pvc-0202b1eb-4f42-4426-a7d5-1619d61dc2d3   50Gi       RWO            microk8s-hostpath   15d
```

Fig. 2: Componentes de PostgreSQL

```
gerard@gerard-master:~$ kubectl get all -n sonarqube | grep -e sonar
pod/sonarqube-657875cd4f-tj2cw   1/1     Running   11         15d
service/sonarqube   NodePort    10.152.183.160   <none>        9000:32502/TCP   15d
deployment.apps/sonarqube   1/1     1            1           15d
replicaset.apps/sonarqube-657875cd4f   1         1         1       15d
gerard@gerard-master:~$ kubectl get  pvc -n sonarqube | grep -e sonar
sonar-pvc      Bound    pvc-fe36cf2d-fa2e-4973-af30-8e674161acba   50Gi       RWO            microk8s-hostpath   15d
```

Fig. 3: Componentes de SonarQube