ProjectZero (Gestión de una arquitectura de microservicios con Istio)

Raül de Arriba Garcia

Resumen– El proyecto consiste en el estudio de varios casos de uso que nos permite hacer Istio junto con Kubernetes. Para poder realizar estas pruebas, se llevará a cabo la creación de un pequeño *cluster* virtualizado con Virtualbox. Kubernetes será utilizado como orquestador de los contenedores, mientras que Docker contendrá los microservicios que gestionaremos con Istio. El principal objetivo de este trabajo es diseñar y construir un entorno donde poder desplegar microservicios y gestionarlos de forma independiente.

Palabras Clave– Infraestructura, Kubernetes, Istio, Docker, Microservicio

Abstract– The project consists in the study of several use cases that allow us to make Istio together with Kubernetes. In order to perform these tests, the creation of a virtualized *cluster* with Virtualbox will be carried out. Kubernetes will be used as a container orchestrator, while Docker will contain the microservices that we will manage with Istio. The main objective of this work is to design and build an environment where you can deploy microservices and manage them independently.

Keywords– Infrastructure, Kubernetes, Istio, Docker, Microservices

1 Introducción

En los últimos años se escucha en los medios de comunicación, y cada vez con más frecuencia, términos como *cloud* [2], *Big Data* [9], *Blockchain* [10] o microservicios [11]. Estos son campos muy diferentes entre sí, pero todos ellos relacionados y con características en común: no pueden realizarse en un único ordenador personal, es decir, necesitan un *cluster* para poder trabajar y realizar sus objetivos. Según nuestro conocimiento, no existe un ordenador suficientemente potente para operar con petabytes[1] de tamaño (Big Data), ni métodos de verificación únicos para validar los datos de una *Blockchain* por un único individuo.

Cada vez más, las medianas y grandes empresas están dejando atrás la idea de tener un gran centro de procesamiento de datos (CPD) propio por motivos económicos, ya que disponer de tu propio CPD conlleva, entre otros: un gran gasto en electricidad, mantenimiento de los equipos informáticos, personal de mantenimiento, mantenimiento y seguridad del edificio, dificultad para asegurar la alta disponibilidad y gran inversión en climatización del espacio.

Este cambio de mentalidad se ha visto favorecido gracias a grandes empresas como Amazon (AWS), Google (Google Cloud) o Microsoft (Azure). Estas empresas han hecho posible llevar a otras grandes y pequeñas empresas a la nube, como es el caso de Apple y su iCloud, que está parcialmente en la plataforma Amazon S3 [1].

Toda esta combinación de nuevas tecnologías me ha generado un gran interés por entender el funcionamiento a bajo nivel y este nuevo método de organización de las empresas. Con este proyecto pretendo montar una infraestructura para profundizar más en este campo y poder comprender a más bajo nivel su funcionamiento, ventajas e inconvenientes.

Con esos objetivos en mente, el presente documento trata de mostrar todo el proceso del proyecto. En la Sección 2 encontramos el estado del arte donde se contextualiza este proyecto. En la Sección 3 se expondrán los objetivos del proyecto. En la Sección 4 se explicará la metodología utilizada para su desarrollo. La planificación del proyecto se detalla en la Sección 5. Las tecnologías y herramientas se presentarán en la Sección 6, donde veremos el estudio de

• E-mail de contacto: rauldearriba@gmail.com
• Mención realitzada: Tecnologies de la Informació
• Trabajo tutoritzado por: Jordi Casas Roma (DEIC)
• Curso 2019/20

[1] 1 petabyte = 10^{15} bytes

mercado de todas las tecnologías escogidas para el desarrollo del proyecto. A continuación, en las Secciones 7 y 8, se presentará el escenario que se va a crear y los procesos de instalación y configuración del sistema para poder conseguir una plataforma operable y gestionable, así como qué microservicios y cómo se despliegan empleando Kubernetes. En lo referente a la automatización, Sección 9, se mostrará la parte del proyecto automatizado. En la Sección 10 se mostrará la solución de monitorización empleada para el proyecto. En lo referente al código, se encontrará en la Sección 11, donde estarán las funcionalidades utilizadas de GitHub y la estructuración del repositorio. En la Sección 12 se presentará Istio y se mostrará cómo se gestionan las comunicaciones entre los servicios desplegados en la plataforma, para garantizar su correcto funcionamiento y poder conocer las intercomunicaciones de los servicios a partir de los casos de uso. Finalmente, en las Secciones 13, 14 y 15 se expondrán los resultados, trabajos futuros y conclusiones del proyecto, respectivamente.

2 Estado del arte

En la actualidad, la tecnología ya es una parte imprescindible de cualquier empresa. Ya sea la página web de una pequeña tienda o las grandes aplicaciones de gestión de una multinacional.

Además, sobretodo en las grandes empresas, donde pueden llegar a perder millones con cada minuto de inactividad del servicio, aparecen los conceptos de arquitectura monolítica y arquitectura de microservicios.

Actualmente, ninguna empresa invierte recursos económicos en crear una gran aplicación que contenga todas las funcionalidades en un solo programa (concepto de arquitectura monolítica). Esta clase de programas, tienen demasiados inconvenientes: todos los componentes están fuertemente acoplados, la actualización de una funcionalidad debe recompilar todo el código y la puesta en producción de esta nueva versión causa una indisponibilidad del sistema.

En cambio, las empresas si están invirtiendo recursos y tiempo en refactorizar estas aplicaciones y convertirlas en microservicios. Esta nueva metodología de creación de aplicaciones trae ventajas como: la creación de un nuevo servicio o actualización de uno ya existente no provoca indisponibilidad de todo el sistema. Se puede desplegar por partes a producción y evitar colapsos del sistema por sobrecarga.

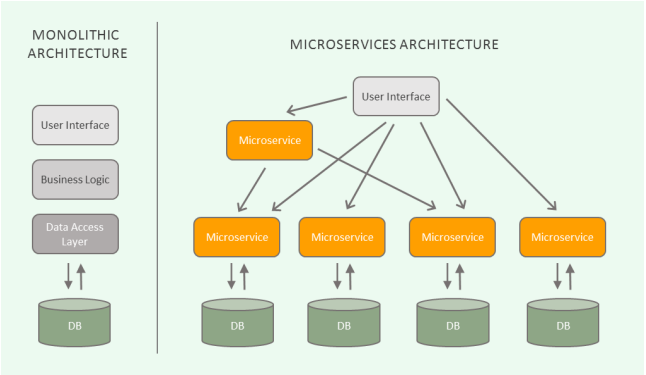


Figura 1: Arquitectura monolítica contra arquitectura microservicios. Fuente Dzone[12].

3 Objetivos

El objetivo principal del proyecto es entender el funcionamiento a bajo nivel de una arquitectura de microservicios. Para llevar a cabo el objetivo principal se proponen tres subobjetivos. Estos se encuentran listados a continuación.

- **Crear una plataforma con Kubernetes e Istio.**

  El primer paso consiste en realizar un estudio de mercado entre las diferentes posibilidades disponibles actualmente en entornos *cloud*.

- **Instalar y configurar la plataforma.**

  Este objetivo consiste en diseñar, implementar y configurar la arquitectura y tecnologías necesarias para poder tener una plataforma con la que poder realizar las pruebas.

- **Caso de uso 1: Limitar las comunicaciones entre los microservicios desplegados en el *cluster*.**

  En un *cluster*, las comunicaciones entre los diferentes servicios no están limitadas, como muestra la figura 2a. En este caso de uso se pretende limitar estas comunicaciones entre servicios utilizando la tecnología Istio, como se muestra en la figura 2b.

- **Caso de uso 2: Introducir una serie de microservicios creados por nosotros y tratar de tener un diagrama de comunicaciones generado por Isitio de forma automática.**

  En este caso buscaremos estudiar si Istio nos permite conseguir un diagrama de todas las comunicaciones dentro del *cluster* para llegar al punto de granularidad que nos permita, en el caso de detectar un problema, saber cómo se propagaría la comunicación.

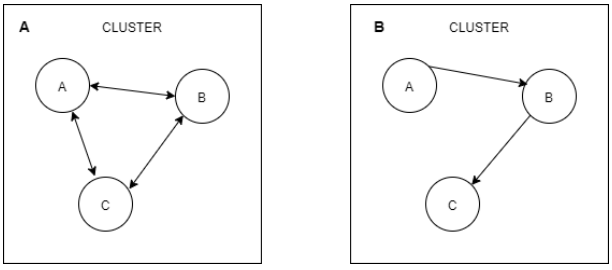


Figura 2: Ejemplo de comunicaciones en un *cluster*. (a) comunicaciones todos con todos. (b) comunicación empleada en el primer caso de uso.

4 Metodología

Para este proyecto utilizaremos una metodología del modelo de cascada [13]. Este proyecto esta pensado de forma secuencial: empieza con la creación del entorno, un primer caso de uso para analizar y limitar las conexiones entre las máquinas del *cluster*, y finalmente, un segundo caso de uso que nos permita aumentar el nivel de granularidad de las comunicaciones y verlas gráficamente.

5 PLANIFICACIÓN

La planificación de este proyecto se ha realizado empleando una herramienta online, llamada canva[3]. La planificación se ha dividido en 4 fases (véase anexo 15).

- La primera fase de planificación y estudio de mercado (figura 8 a).
- La segunda dedicada a la instalación y configuración del entorno base y el primer caso de estudio (figura 8 b).
- La tercera donde introducimos microservicios propios y estudiamos la comunicación entre ellos (figura 8 c).
- Una última fase donde aumentamos el número de *clusters* en 2 y estudiamos las comunicaciones entre ellos (figura 8d).

6 ANÁLISIS DE TECNOLOGÍAS Y HERRAMIENTAS

En este apartado mostraremos las herramientas y tecnologías utilizadas para la realización de este proyecto. Estos recursos los dividiremos en tres grandes grupos: (1) herramientas organizativas y de control de versiones, (2) herramientas de la plataforma y (3) herramientas utilizadas dentro de la plataforma.

6.1. Herramientas organizativas y de control de versiones

Las soluciones utilizadas en la parte organizativa y control de versiones de código son las siguientes:

Canva

Este software *online* que consiste en un marco para crear composiciones de imágenes, lo utilizo para crear la planificación y el diagrama de Gantt por su fácil y rápido aprendizaje. Además, tiene la ventaja de ser muy fácil de compartir con otras personas y es gratuito[3].

Github

A nivel de control de versiones hay diferentes posibilidades, como Bitbucket o Gitlab, pero finalmente se ha utilizado Github, principalmente, por la gran comunidad y las funcionalidades extras que ofrece.

Github es una plataforma de desarrollo que te permite alojar código, mantener un control de versiones del mismo, mantener organizados tus proyectos, poder trabajar desde cualquier ordenador y compartir código con otros desarrolladores. A demás, Github tiene funcionalidades menos conocidas como la sección de *Issues* y la wiki.

Estas dos últimas funcionalidades han facilitado la gestión de incidencias y la documentación del proyecto, quedando integradas con el propio código, y ofreciendo una documentación accesible y de calidad del proyecto.

En el repositorio del proyecto [2] están todos los comandos y explicaciones detalladas de las etapas del proyecto[4].

[2]https://github.com/Radega1993/ProjectZero-TFG-/wiki

6.2. Herramientas de la plataforma

El segundo bloque de herramientas son las utilizadas para la creación de la plataforma.

Virtualbox

La elección de esta herramienta ha resultado especialmente compleja, debido a que representa la base de todo el proyecto. Las opciones aquí eran muchas. Entre ellas he estudiado Amazon Web Services, Google Cloud, Microsoft Azure, IBM Cloud Private y RedHat OpenShift.

En todos estas herramientas se han detectado dos problemas relevantes para la realización de este proyecto:

1. Son de pago por uso.
2. La configuración de estas herramientas se realiza a alto nivel, lo que dificulta o impide entender el funcionamiento a bajo nivel de estos servicios, que es uno de los objetivos de este trabajo.

Por este segundo motivo se abren 2 opciones. La primera, crear un *cluster* de Raspberry Pi pero volvemos a tener el impedimento económico y para empezar el proyecto necesitaríamos mínimo 3, 1 *master* y 2 nodos *workers*. Esto sin hacer un *capacity planning*[14] y tener una previsión de uso de todo los servicios que tendrá desplegados nuestra plataforma.

Finalmente la opción escogida ha sido Virtualbox, ya que cumple todas las necesidades para este proyecto. En primer lugar, es gratuito y *open source*. En segundo lugar, proporciona escalabilidad horizontal[15] al tener la posibilidad de crear tantos nodos como nuestro ordenador sea capaz de soportar[5].

Kubernetes

Kubernetes[16] o k8s es un proyecto *open source* creado para ofrecer la funcionalidad de orquestrador de contenedores.

La función principal de k8s es automatizar la implementación, el escalado y la administración de aplicaciones en contenedores.

Docker

Se ha escogido Docker[6] como opción de contenerización de aplicaciones por los siguientes motivos: el primero es la integración con k8s, el segundo es el amplio soporte de la comunidad, la extensa documentación que hay en internet, y finalmente, porqué a nivel profesional, es la solución más utilizada actualmente, como vemos en la figura 3.

Python

Es un lenguaje de programación interpretado. Se ha escogido este lenguaje para la creación de microservicios por sus librerias Flask [3] y Requests[4] para realizar llamadas HTTP. Además, es un lenguaje que no requiere un alto uso de recursos para desplegar aplicaciones[7].

[3]https://flask.palletsprojects.com/en/1.1.x/
[4]https://requests.readthedocs.io/en/master/

Figura 3: Comparación entre Doker[azul], LXD[rojo] y Podman[amarillo]. Fuente: Google trends

Ansible

Es una tecnología ampliamente utilizada en entornos *cloud*. Permite, de forma rápida y segura, automatizar aplicaciones e infraestructura. En este proyecto lo utilizaremos para automatizar la creación y configración del *cluster*, tanto de los nodos *masters* como de los *workers*. De esta forma, en caso de necesitar mas máquinas, se podrán configurar, instalar e introducir en el *cluster* de forma automática. Se ha escogido Ansible por su rápida curva de aprendizaje, su gran potencial y que es una de las soluciones más utilizadas [18].

Istio

Es una solución que se integra junto a k8s y permite conectarse, proteger, controlar y observar los servicios que están funcionando dentro de la plataforma. En este proyecto vamos a utilizar todas funcionalidades que esta herramienta nos ofrece[8].

6.3. Herramientas dentro la plataforma

El tercer bloque de herramientas utilizadas en este proyecto son las aplicaciones que están corriendo dentro de la plataforma.

Prometheus

Esta tecnología *open source* de monitorización y alerta de eventos se encarga de registrar métricas a tiempo real mediante unos *exporters* y permite tener una visión del estado del sistema en todo momento[19].

Grafana

Es una una solución *open source* para analizar y monitorizar a través de *dashboards* la información recogida en una base de datos y mostrar estos gráficos a tiempo real. Su integración con Prometheus es muy habitual y la utilizaremos como solución de monitorización de la plataforma[20].

Kiali

Esta tecnología propia de Istio te permite ver *dashboards* y gráficos de los servicios del *cluster* además de ver y editar los YAMLs de configuración de Istio[21].

7 Diseño del escenario

En esta sección veremos diseño, configuración y creación de los escenarios hasta conseguir un entorno operativo.

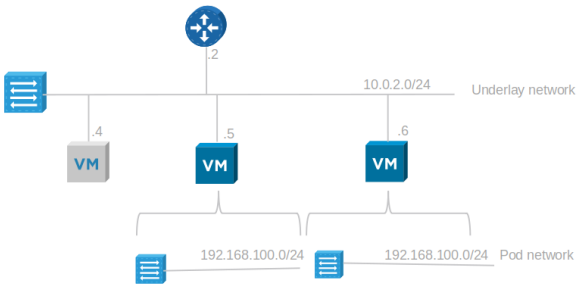


Figura 4: Diagrama de red del *cluster* del proyecto

7.1. Prerequisitos

Para poder implementar este proyecto, necesitamos los siguientes requisitos: un servidor, múltiples servidores o un ordenador con 8 cores; 16GB de RAM y con capacidad para mantener un mínimo de 3 máquinas virtuales (MV); un software de virtualización y la ISO de Centos7, como base de sistema operativo para las MV.

7.2. Estado inicial

Como base donde alojar todo este proyecto, se ha escogido un ordenador DELL XPS 15 con un sistema operativo Linux, un procesador Intel I7 de 7 generación y 16 GB de RAM. Para la parte de virtualización, Virtualbox nos permite configurar los cores y la RAM de cada MV, y nos permite crear redes para comunicar las MV entre ellas.

8 Instalación

En este apartado, procederemos a instalar y configurar el *cluster* de Kubernetes e Istio sobre el escenario mostrado anteriormente.

Antes de poder empezar a instalar y aplicar la configuración de cada una de las máquinas, crearemos una imagen base que la clonaremos tantas veces como máquinas en el *cluster*. Esto lo hacemos para asegurar que todas las máquinas que componen el *cluster* serán homogéneas entre ellas, serán la misma imagen base y con ello las mismas versiones de sistema operativo.

8.1. Máquina Base

Para instalar la máquina base, con la ayuda de Virtualbox, en nuestra maquina *host*, crearemos una nueva máquina virtual y la configuraremos con una capacidad de 40GB de disco duro, le daremos 2 cores de CPU y 4 GB de RAM, que son requisito de Kubernetes. Una vez creada la MV, instalaremos el sistema operativo Centos7[23] sin modo gráfico,

ya que no es necesario, y ganaremos espacio en el disco que podremos utilizar con aplicaciones más útiles para la plataforma. Una vez tengamos Centos7 completamente operativo en nuestra MV, procedemos a crear la red NAT con la que comunicaremos todas las máquinas del *cluster*. Hemos decidido utilizar el rango privado de clase A 10.0.2.0/24.

8.2. Redireccionamiento de puertos

Este paso solo es necesario para trabajar en máquinas virtuales: para acceder desde la máquina *host* al *cluster* vamos a necesitar hacer una redirección de puertos. Esto se debe a que la *IP* que tienen nuestras MV son privadas, asignadas por Virtualbox que actúa como DHCP. Si no hiciéramos estos redireccionamientos de puertos no se podría acceder a determinados servicios desde nuestro *host*, como por ejemplo SSH[24] o servidores web Apache o Nginx.

8.3. Master

Una vez tenemos creada y configurada la máquina base y la red NAT para nuestro entorno, procedemos a clonar esta MV y le ponemos el nombre de "Master"para poder identificarla. Dentro de la MV *master* le asignamos la *IP* estática que hemos definido en nuestro diagrama de red (figura 4), definimos en el fichero de la ruta `/etc/sysconfig/network-scripts/ifcfg-enp0s3` con la *IP* 10.0.2.4/24 y las demás configuraciones de red, como son el *gateway*, el DNS y el protocolo estático.

Se puede encontrar la configuración en el aparatado de instalación de la wiki del proyecto[25]. El primer paso es definir una *IP* estática para nuestro *master* y *workers*. Necesitamos conocer a las máquinas en todo momento para poder acceder a ellas y gestionarlas. Si las *IPs* cambiaran por el protocolo DHCP no seria viable gestionar esta clase de entornos por el gran volumen de nodos que pueden llegar a tener. A continuación, para facilitar el acceso entre máquinas del *cluster*, definiremos en el `/etc/hosts` todas las maquinas de nuestro *cluster* por *IP* y nombre. Así, será mas sencillo recordar o identificar una máquina llamada Worker1 que saber quién es 10.0.2.x.

Una vez terminada la configuración inicial del *master*, vamos a instalar los servicios necesarios para crear el *cluster* de k8s.

En primer lugar, definimos el SELinux[26] en modo permisivo. SELinux es un sistema de control de acceso obligatorio, donde antes de una llamada al sistema, el kernel pregunta a SELinux si el proceso está autorizado a realizar esa operación[26]. Los servicios que necesitamos instalar en el nodo *master* son:

- **kubelet**: Se encuentra en cada nodo del *cluster* y su función principal es crear una comunicación con la API de kubernetes e informar del estado de ejecución de cada nodo y asegurar los contenedores que están corriendo en cada pod. Un pod es uno o más contenedores Docker que comparten red, almacenamiento y especificaciones de funcionamiento en común.

- **kubeadm**: Es una herramienta de Kubernetes que nos permite desplegar de manera más sencilla un *cluster* de Kubernetes. Esta herramienta contiene todas las instrucciones necesarias para crear un *cluster*.

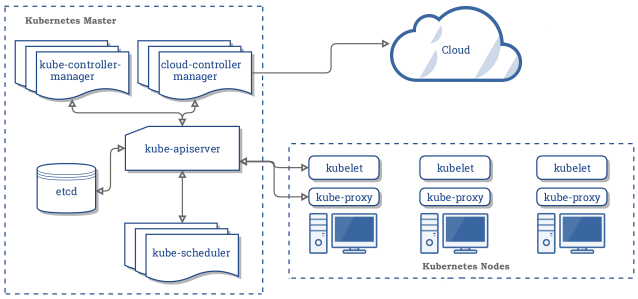


Figura 5: Componentes de un *cluster* k8s. Fuente: Documentación oficial k8s

- **kubectl**: Es la herramienta, basada en linea de comandos, nos permite controlar todo el *cluster*.

- **Docker**: Es la herramienta de contenerización que hemos decidido utilizar en el proyecto. Se encarga de crear contenedores con la aplicación dentro y así permitir ser orquestado por Kubernetes.

Finalmente, antes de iniciar nuestro *cluster* de Kubernetes, debemos deshabilitar el *swap*. Esto lo hacemos porque Kubernetes tiene la posibilidad de limitar los recursos que puede consumir un pod dentro del sistema. Si no se deshabilita esta opción de la configuración de Linux, si una aplicación supera el limite de memoria RAM definido, este acabara por llenarnos la memoria *swap* del sistema. Adicionalmente, deshabilitamos el *firewall* para evitar problemas de conexión dentro del *cluster* e iniciamos el *cluster* con el comando `kubeadm`, con los parámetros de la IP de la API de Kubernetes que se encuentra en el nodo *master* (figura 5) y el rango de *IPs* de los pods que serán las *IPs* que recibirán las aplicaciónes que estén en ejecución dentro de la plataforma. Si todo se ha ejecutado correctamente, por pantalla nos mostrara 3 cosas importantes. Un mensaje *"successfully"* diciendo que la instalación se ha realizado sin errores, un grupo de 3 comandos que servirán para copiar y dar permisos al fichero de configuración de k8s y, finalmente, el comando de `kubeadm join` con un token. Este token lo deberemos guardar y no publicarlo en ningún lugar público porque este comando permite, al ejecutarlo en un nodos, pasan a formar parte del *cluster*.

En este punto ya tenemos nuestro *cluster* de Kubernetes casi operativo. Solo faltaría desplegar nuestro primer pod que será Calico y el segundo que sera MetalLB. Calico[27] es una solución *open source* de red y seguridad. En la red, Calico se encargará de crear los túneles de comunicación dentro de nuestro *cluster* en todas las aplicaciones que se desplieguen, dentro de la red de pods que hemos definido al iniciar el *cluster*. MetalLB[28] es una solución *open source* de balanceador de carga para *cluster*s de Kubernetes no desplegados en plataformas, como Amazon o Google, que te lo configuran por defecto en la instalación. MetalLB nos asigna una *IP* del rango que le hemos configurado al servicio desplegado para poder acceder a él desde fuera del *cluster*.

8.4. Istio

Istio lo instalaremos en el nodo máster del *cluster* y este lo veremos como un pod más corriendo en k8s en su *namespace* 'istio-system'. La instalación base la realizaremos

descargando el *script* que nos proporcionan los desarrolladores de la solución. Este *script* se encargará de descargar todos los ficheros y directorios necesarios para poder lanzar Istio en nuestro *cluster*. Una vez terminada la ejecución de este *script* ejecutaremos, con el comando `kubectl`, todos los ficheros YAML que se encuentran en la ruta `istio-[version]/install/kubernetes`. A continuación debemos configurar la herramienta `istioctl` para poder utilizar todas las herramientas, y con esto finalizaríamos la instalación completa de nuestro primer nodo del *cluster*, el nodo máster.

8.5. Workers

Una vez tenemos el nodo máster operativo, debemos instalar, configurar y agregar los workers al *cluster* para que puedan correr aplicaciones. En primer lugar, debemos realizar el mismo proceso que en el máster. Clonaremos la imagen base y configuraremos la IP estática y añadiremos los nodos al `/etc/hosts`. A continuación, deshabilitaremos el SELinux, el *firewall* y el *swap* de la máquina y habilitamos el *ip forward* para permitir que la red de pods que hay en los workers tengan salida hacia los demás nodos del *cluster*. Con en nodo configurado, instalamos `kubeadm` para que nos permita utilizar el comando de `kubeadm join`, más adelante, y Docker (que será la herramienta con la que contenerizaremos las aplicaciones). A continuación, lanzamos el comando que guardamos cuando iniciamos el *cluster* en el nodo máster y nuestro *worker* pasará a formar parte del *cluster*.

9 ANSIBLE

En esta sección, procedemos a mostrar la automatización disponible en la plataforma.

Ansible trabaja con ficheros YALM, llamados playbooks. Estos ficheros contienen las tareas, configuraciones y instalaciones para que se realicen de forma automática. La estructura escogida de Ansible para automatizar el proyecto ha sido la siguiente: En la raíz, encontramos un fichero donde están definidos todos los nodos del *cluster* y un fichero de *setup* que genera una llamada al *playbook* que realizará la automatización del proceso. Este *playbook* contiene la llamada a todos los roles necesarios para realizar la tarea de forma automática. Finalmente, encontramos un directorio 'roles' donde está definida la implementación de cada tarea necesaria para su automatización.

10 MONITORIZACIÓN

En este apartado se explica la solución de monitorización escogida para la plataforma. Me decidí a utilizar la dupla de aplicaciones de monitorización estándares, que además se integran con Kubernetes e Istio a la perfección ya que se creó para funcionar en este tipo de entornos.

Por un lado, tenemos Prometheus, software de recogida y procesado de métricas y alertas, donde hemos definido, entre otras, una alerta de *ping* a los nodos para saber en todo momento si perdemos la comunicación dentro del *cluster*.

Por el otro lado tenemos Grafana. Esta aplicación recoge los datos de las métricas de Prometheus y con esos datos se crean *dashboards* que se actualizan a tiempo real. Esto nos permite ver el estado del sistema a cada momento.

Finalmente, disponemos de Kiali en la plataforma, que es una aplicación de Istio para monitorizar la red y poder ver y editar algunos ficheros de configuración. Además, permite la creación de un grafo de comunicaciones de los servicios.

11 CÓDIGO FUENTE Y DOCUMENTACIÓN

En este apartado se explica el uso que se le ha dado a Github durante la realización del proyecto.

Github es el software de control de versiones de Microsoft. Para el proyecto se creó un repositorio llamado "ProjectZero-TFG-". En este repositorio podemos encontrar lo siguiente:

- Los ficheros YAML utilizados para el despliegue de las aplicaciones que corren en nuestro *cluster*.
- Los ficheros Ansible para la automatización de algunos procesos del *cluster*.
- El paquete de Istio para su despliegue dentro del *cluster*.
- El informe del proyecto.
- El código de una aplicación Python para ver el estado del *cluster*.

Además, se ha utilizado 2 funcionalidades extras de la plataforma Github. La primera y más importante, ha sido la función de wiki[22], donde se ha realizado toda la documentación del proyecto en lenguaje *markdown*. La segunda funcionalidad es *İssues"*, donde se han ido publicando diferentes problemas encontrados durante la realización del proyecto y su solución una vez encontrada.

12 CASOS DE USO

En este apartado se explican los diferentes casos de uso realizados para estudiar el funcionamiento de Kubernetes e Istio para poder ver todo su potencial.

12.1. caso 1: Limitar las comunicaciones entre los microservicios desplegados en el *cluster*

El primer caso de uso vamos a probar como funciona el enrutamiento proporcionado por Istio[29].

Para desarrollar esta primera prueba de concepto, utilizaremos una aplicación que nos proporciona Istio, que consiste en una plataforma para ver *reviews* de diferentes libros.

La aplicación tiene 6 microservicios.

- El frontend donde se muestra toda la información.
- Tres versiones diferentes de las *reviews*, una sin estrellas, otra con estrellas negras y una ultima con estrellas rojas.
- Un microservicio con los detalles que contiene la información del libro.

- *ratings* que contiene la información del número de estrellas del libro.

Para poder probar el enrutamiento, debemos conocer 3 conceptos de Istio.

- **Subset**: Sería equivalente a un grupo o *namespace*, es un subconjunto de pods de un mismo servicio.
- **DestinationRule**: Son las reglas que tiene que cumplir el tráfico enrutado.
- **VirtualService**: Son los enrutadores que se encargan de dirigir el tráfico hacia el destino y que cumplan las reglas definidas en las *destination rules*.

Una vez tenemos la aplicación funcionando en el *cluster*, definiremos los *subsets* dentro de las *DestinationRules* y, finalmente, crearemos los *virtual-services* donde definimos cada servicio que subset y regla tiene que cumplir.

Gracias a estas funcionalidades que nos ofrece Istio, podemos enrutar el tráfico a diferentes microservicios, podemos enrutarlo por campos del *header* de la petición http (podemos redirigir el trafico dependiendo del navegador o del usuario *logeado*) o podemos balancear la carga entre las diferentes versiones[22].

12.2. Caso 2: Introducir una serie de microservicios creados por nosotros y tratar de tener un diagrama de comunicaciones generado por Isitio de forma automática.

En este caso de uso, hemos creado un conjunto de microservicios para poder ver el estado del *cluster*. Hemos creado 4 microservicios:

- Un Frontend donde se muestra la información.
- Un servicio para ver el estado de los nodos.
- 2 servicios para el estado de los pods: uno con todos y otro solo con los pods del sistema.

Una vez creado el código y generado la imagen con docker, la hemos subido al repositorio de imágenes docker hub para hacerla pública y poder utilizarla en el *cluster*. El siguiente paso consiste en desplegar estas imágenes. Para ello, hemos creado 3 ficheros necesarios:

- Deployment: En este fichero *YAML* se definen parámetros como: la imagen docker a desplegar; el nombre de la aplicación o el puerto donde se expone al exterior(figura 6).
- Service: fichero que sirve para definir como se expone un pod en ejecución como un servicio en la red (figura 7).

Recreando la parte de red del caso 1, si accedemos a *kiali* podremos ver la generación de los grafos como el de la figura 9 del apéndice 2.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
   app: frontend
   version: v1
 spec:
  replicas: 1
  selector:
   matchLabels:
    app: frontend
    version: v1
  template:
   metadata:
    labels:
     app: frontend
     version: v1
   spec:
    serviceAccountName: check-frontend
    containers:
    - name: frontend
     image: radega1993/checker-k8s:frontend
     imagePullPolicy: Always
     ports:
     - containerPort: 8000
```

Figura 6: Fichero: deployment.yaml

```
apiVersion: v1
kind: Service
metadata:
 name: frontend
 labels:
  app: frontend
  service: frontend
spec:
 selector:
  app: frontend
 ports:
  - name: http
    port: 8000
```

Figura 7: Fichero: service.yaml

13 Resultados

En este apartado compararemos los objetivos propuestos en el proyecto con el trabajo realizado para conseguirlos.

Tal y como hemos visto en la sección de creación de la plataforma, este no es un proyecto de creación de una aplicación sino que se trata de un proyecto de investigación, de una prueba de concepto sobre como se creaba una plataforma a partir de la tecnología Kubernetes e Istio. El resultado ha sido satisfactorio, ya que hemos conseguido un entorno operativo. Este objetivo era el más importante, requería un trabajo de investigación y aprendizaje elevados, ya que son tecnologías que utilizar una gran cantidad de conceptos vistos durante el grado y otros que no conocía antes de iniciar el proyecto. Gracias a este proyecto he podido ver como se relacionan herramientas como git o Docker, más orientadas a ingeniería del software. Conceptos de túneles SSH y configuraciones de red y seguridad, entre las aplicaciones de la mención de Tecnología de la Información y las comunica-

ciones y plataformas *cloud*, de sistemas distribuidos.

El primer caso a estudiar una vez el *cluster* estuviera listo, consistía en poder limitar la comunicación entre los servicios.

Este caso de uso sí lo hemos podido realizar de manera satisfactoria. Conseguimos enrutar el tráfico según nuestras necesidades, hacer un servicio inaccesible y limitar el trafico por usuarios, hemos podido ver que el manejo de trafico que nos ofrece Istio es muy potente y no muy complejo de configurar.

El segundo caso a estudiar también se realizó de manera satisfactoria, hemos podido desplegar un conjunto de *pods* en la plataforma y a partir de kiali, hemos podido ver como se generaban los grafos con todas las conexiones (figura 9) y poder ver en todo momento que comunicaciones se están realizando en el *cluster*.

14 Trabajos futuros

En este apartado se expondrán algunas consideraciones que no se han podido realizar, pero que si un sistema como este se quisiera llevar a producción habría que tener presente antes de desplegarlo.

Lo primero serían los recursos, este entorno debería desplegarse en un *cluster* de servidores con memoria RAM, CPU y disco acorde con lo que deba soportar la plataforma.

Por otro lado, habría que disponer de 2 *clusters* idénticos pero en diferentes ubicaciones para garantizar la alta disponibilidad y así poder evitar pérdida de servicio y datos.

Se debería plantear una opción de registro de *logs* como ElasticSearch y Kibana para tener un control interno del *cluster* y poder detectar errores o fallas del sistema.

Finalmente, habría que añadir, a la parte de monitorización, una solución como Alertmanager que te permita enviar las alertas de Prometheus por correo o algún canal de comunicación en caso de fallo para su rápida actuación, en caso de ser necesaria.

A medida que el proyecto crece, surgirán nuevas necesidades para la plataforma, pero estas son las acciones necesarias más críticas detectadas a lo largo del proceso de realización de este proyecto.

15 Conclusiones

En este apartado se exponen las conclusiones extraídas de realizar este proyecto de fin de grado.

Durante el proyecto, he podido aprender nuevas tecnologías y reforzar conocimientos de casi todos los ámbitos del grado por la gran cantidad de temas que se han estudiado en este trabajo. Por una lado, he podido aprender más a fondo el funcionamiento y tecnologías empleadas en los sistemas *cloud*, sobretodo Kubernetes e Istio, que han sido toda la base de este trabajo. Además, he podido entender mejor el funcionamiento de una red y sus componentes, ya que, la potencia de un *cluster* reside en las interconexiones de los nodos y en especial, con Kubernetes, donde la comunicación de los microservicios para que puedan entenderse entre ellos y poder realizar las funciones deseadas, es uno de sus grandes potenciales.

Por otro lado, he podido reforzar lenguajes de programación como python, herramientas de control de versiones como git y todas las funcionalidades que ofrece la plataforma Github y por último, aprender el funcionamiento de Docker y la contenerización de aplicaciones para su despliegue en cualquier entorno.

Agradecimientos

En primer lugar, agradecer a todos los profesores que he tenido a lo largo de mi paso por el grado, ya que sin ellos, este proyecto habría sido algo casi imposible de realizar. Mención especial a mi tutor Jordi Casas, por su seguimiento y disponibilidad para atenderme siempre a pesar de mi difícil horario, y a Porfidio Hernández por ayudarme a orientar alguna parte del proyecto.

En segundo lugar, agradecer a mi familia y pareja por apoyarme durante el proyecto y leerlo muchas veces para dar su opinión sobre el estado del artículo.

Finalmente, agradecer a los compañeros de trabajo que me ayudaron a no quedarme estancado en el proyecto y poder avanzar gracias a sus recomendaciones y consejos durante todo el trabajo.

Referencias

[1] JAVIER PASTOR. “Apple paga más de 30 millones de dólares al mes a Amazon por usar AWS: iCloud es en buena parte la nube de Amazon” `https://bit.ly/2MpsIJX` Accedido en: Setiembre, 2019.

[2] RedHat. “¿Qué son las nubes?” `https://red.ht/2oSw8vX` Accedido en: Noviembre, 2019.

[3] Canva. “oficial” `https://www.canva.com/` Accedido en: Noviembre, 2019.

[4] Github. “oficial” `https://github.com/` Accedido en: Noviembre, 2019.

[5] Virtualbox. “oficial” `https://www.virtualbox.org/` Accedido en: Noviembre, 2019.

[6] Docker. “oficial” `https://www.docker.com/` Accedido en: Noviembre, 2019.

[7] Python. “oficial” `python.org/` Accedido en: Noviembre, 2019.

[8] Istio. “oficial” `https://istio.io/` Accedido en: Noviembre, 2019.

[9] Oracle. “Definición de big data” `https://bit.ly/36GFaNC` Accedido en: Octubre, 2019.

[10] Pradip Kumar Sharma ; Mu-Yen Chen ; Jong Hyuk Park. “A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT” `https://bit.ly/2NlTcNc` Accedido en: Octubre, 2019.

[11] RedHat. “¿Qué son los microservicios?” https://red.ht/2qp6kbp Accedido en: Noviembre, 2019.

[12] DZone. “What Are Microservices, Actually?” https://bit.ly/2ScoAjG Accedido en: Noviembre, 2019.

[13] OBS Business school. “Gestión de proyectos siguiendo el modelo de cascada” https://bit.ly/2ByxnmZ Accedido en: Octubre, 2019.

[14] Wikipedia. “Capacity planning” https://bit.ly/2PQdHmN Accedido en: Octubre, 2019.

[15] Oscar Blancarte. “Escalabilidad Horizontal y Vertical” https://bit.ly/2NREQmy Accedido en: Noviembre, 2019.

[16] Oscar kubernetes. “Orquestación de contenedores para producción” https://kubernetes.io/es/ Accedido en: Noviembre, 2019.

[17] Istio. “What is Istio?” https://bit.ly/33n9smt Accedido en: Noviembre, 2019.

[18] Ansible. “Ansible oficial” https://www.ansible.com/ Accedido en: Diciembre, 2019.

[19] prometheus. “prometheus oficial” https://prometheus.io// Accedido en: Diciembre, 2019.

[20] grafana. “grafana oficial” https://grafana.com/ Accedido en: Diciembre, 2019.

[21] kiali. “kiali oficial” https://kiali.io/ Accedido en: Diciembre, 2019.

[22] Wiki. “wiki” https://bit.ly/2ZY6BzB Accedido en: Enero, 2019.

[23] Centos. “Official” https://www.centos.org/ Accedido en: Octubre, 2019.

[24] Wiki. “wiki ssh” https://bit.ly/397vU6H Accedido en: Octubre, 2019.

[25] Raül. “wiki instalación” https://bit.ly/2SihQ4e Accedido en: Octubre, 2019.

[26] Debian. “Introducción a SELinux” https://bit.ly/34MYSoE Accedido en: Noviembre, 2019.

[27] calico. “Calico oficial” https://bit.ly/392o0vc Accedido en: Noviembre, 2019.

[28] MetalLB. “MetalLB oficial” https://metallb.universe.tf/ Accedido en: Noviembre, 2019.

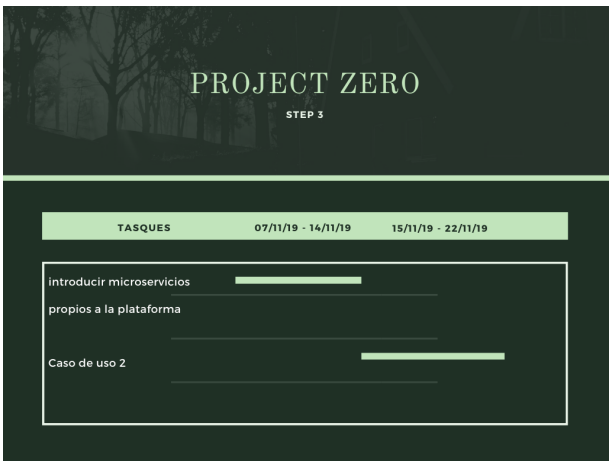
[29] Istio. “Bookinfo Application” https://bit.ly/39KmEpo Accedido en: Enero, 2020.

APENDICES

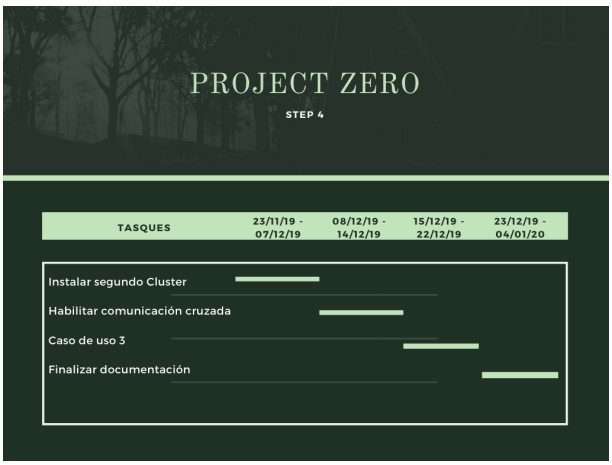
A.1. Fases planificación

(a) Fase 1 del proyecto

(b) Fase 2 del proyecto



(c) Fase 3 del proyecto



(d) Fase 4 del proyecto

Figura 8: Planificación.

A.2. gráfico kiali

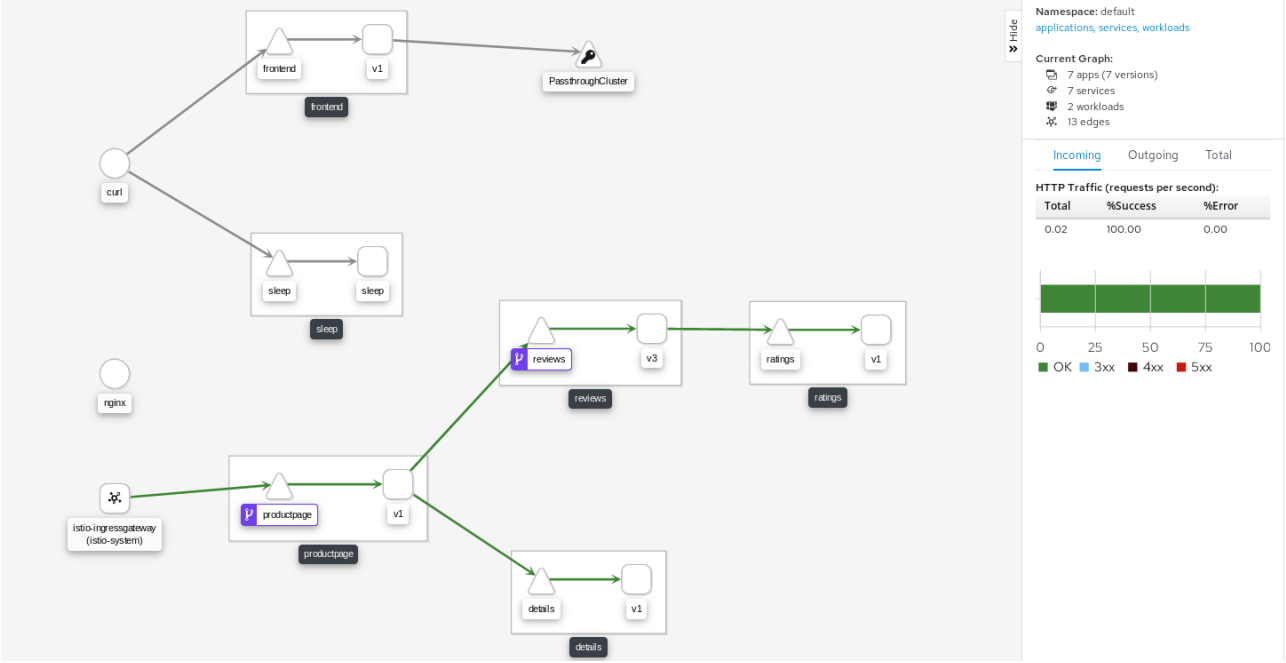


Figura 9: Ejemplo de grafo generado con kiali