Universidad
Rey Juan Carlos

Escuela Técnica Superior en Ingeniería Informática

Despliegue de una aplicación basada en microservicios con autoescalado en la nube de un proveedor de servicios con orquestador de contenedores

Memoria del Trabajo Fin de grado

Ingeniería De Computadores

Autor:

Jaime Oñate Rodríguez-Pardo

Tutor: Maria Teresa González De Lena

Junio 2024

Resumen

En el mercado actual, las aplicaciones, programas y servicios presentan grandes complejidades en su diseño y arquitectura. Esto se relaciona con su rendimiento, y con la capacidad de satisfacer las peticiones de clientes bajo cargas de trabajo inesperadas. Con cargas de trabajo bajas, se destina mas dinero del necesario para el servicio desplegado; y con cargas de trabajo superiores a la esperada, no se satisfacen servicios esenciales (afecta especialmente a la experiencia de usuario).

Debido a estas cargas de trabajo arbitrarias, es necesario invertir en nuevas tecnologías que resuelvan el problema, como en la tecnología del escalado automático. Por otro lado, debido la complejidad implicada es necesario una herramienta que gestione el despliegue de esta tecnología, como los orquestadores de contenedores.

En este trabajo de fin de grado se ha investigado la implementación de escalado automático en el despliegue de productos software utilizando orquestadores de contenedores. A lo largo del proyecto, se han identificado las limitaciones de los gestores de contenedores tradicionales y se ha explorado la viabilidad de emplear tecnologías avanzadas que proporcionen estas capacidades.

El estudio se ha centrado en la transición de un gestor de contenedores a un orquestador de contenedores, esencial para gestionar aplicaciones que requieren escalado automático. Se ha demostrado que orquestadores como Kubernetes no solo facilitan el despliegue y escalado de contenedores, sino que también aseguran balanceo de carga, alta disponibilidad y recuperación ante fallos. Esto contrasta con las capacidades limitadas de los gestores de contenedores, que se restringen a manejar contenedores individuales en un solo host.

En los capítulos siguientes, se presenta un análisis detallado del estado actual de las tecnologías de orquestación, se describe la lógica subyacente en los sistemas de orquestación de contenedores y se despliega un producto en un entorno de orquestación real, realizando pruebas para evaluar su rendimiento y capacidad de escalado. A pesar de la complejidad y coste de emplear un orquestador de contenedores, se vuelve necesario su uso en aplicaciones con cargas de trabajo variables e impredecibles.

Índice general

Índice de tablas

Índice de figuras

Algoritmos

Capítulo 1

Introducción

En este capítulo se hablará de la motivación para desarrollar este trabajo de fin de grado. Después se introducirán las diferentes opciones y maneras de desplegar un producto en la nube de proveedor se servicios.

1.1. Motivación

Una de las asignaturas que más pueden llamar la atención a los estudiantes del grado de ingeniería de computadores es la asignatura de **sistemas distribuidos**. La razón se debe a que se puede aprender el desarrollo de un prototipo de un producto comercial. Pero, en el mercado actual, estos productos no se despliegan de la manera en la que se aprende en el grado. Se ha investigado sobre la manera de desplegar estas aplicaciones en un entorno seguro, fiable y competente para garantizar el servicio a un número indefinido y fluctuante de clientes. Este conocimiento es esencial para las empresas que desean crecer y suelen reinvertir beneficios en el desarrollo de sus propios servicios.

Es por eso que surge la inquietud e interés por estudiar una herramienta que permita solventar el problema de despliegue de manera competitiva. Así que aprender un orquestador de contenedores y desplegarlo en un entorno competitivo otorga un conocimiento valioso para gestionar aplicaciones ya que muestra beneficios en escalabilidad, eficiencia de recursos, seguridad y simplificación. Esto supone una mayor capacidad de innovación y aumento del servicio para los desarrolladores y los clientes, asegurando la competitividad en el mercado.

1.2. Orquestador de contenedores vs gestor de contenedores

Para lograr que una aplicación profesional pueda soportar cargas de trabajo fluctuantes e inesperadas, hay que emplear una herramienta que solvente dicha necesidad. Conocidas las prestaciones de los gestores de contenedores, se ha investigado sobre más tecnologías que solucionen sus problemas y limitaciones. Por eso se ha profundizado sobre la tecnología de la orquestación de contenedores [3].

La distinción entre un orquestador de contenedores y un gestor de contenedores es fundamental para entender cómo se administran y coordinan las aplicaciones en entornos de contenedores. Un gestor de contenedores es una herramienta que se utiliza para manejar el **ciclo de vida de contenedores individuales**, mientras que un orquestador de contenedores es una herramienta más avanzada que gestiona no solo contenedores individuales, sino también **múltiples contenedores** distribuidos en varias máquinas conectadas entre sí (es decir, un clúster) (explicación en el capítulo 2).

- Un gestor de contenedores automatiza el despliegue y escalado de aplicaciones en múltiples contenedores y nodos. Esta automatización está directamente relacionada con proporcionar servicio a gran escala. Sin embargo, un gestor de contenedores se caracteriza por lanzar y detener contenedores basados en imágenes específicas. Si bien es cierto que descargar, almacenar y manejar las imágenes de contenedores es importante, en un orquestador se puede solventar con la conexión a un registro de imágenes.
- Un gestor de contenedores se limita a permitir al usuario configurar redes y volúmenes de almacenamiento para contenedores y monitorizar el estado y rendimiento de contenedores individuales sin ningún tipo de comportamiento automático. Pero un orquestador de contenedores configura y gestiona la red entre múltiples contenedores y nodos y proporciona herramientas para monitorizar el rendimiento y registrar los eventos de los contenedores en todo el clúster. También puede actuar en consecuencia para garantizar eficacia y eficiencia.

El orquestador se ocupa de varias responsabilidades más: distribuye la carga de trabajo entre los contenedores, garantiza que los contenedores sigan funcionando y se reinicien en caso de fallos, y coordina las dependencias y comunicaciones entre diferentes servicios y aplicaciones desplegadas en contenedores. En la tabla 1 se plasman las diferencias de las tecnologías en los distintos ámbitos de la gestión de aplicaciones (ver Tabla 1).

Característica	**Gestor de Contenedores**	**Orquestador de Contenedores**
Nivel de Gestión	Individual (contenedor)	Múltiple (contenedores y nodos)
Despliegue	Manual y específico	Automático y escalable
Escalado	Limitado a contenedores individuales	Gestionado y automático
Balanceo de Carga	No disponible	Disponible
Alta Disponibilidad	No garantizada	Garantizada
Red y Almacenamiento	Configuración básica	Gestión avanzada
Monitorización y Registro	Limitado a contenedores individuales	Centralizado y completo
Orquestación de Servicios	No disponible	Disponible

Tabla 1: Resumen de las principales diferencias entre un gestor y un orquestador de contenedores

En resumen, mientras que un gestor de contenedores se centra en la gestión de contenedores individuales, un orquestador de contenedores se ocupa de la gestión de aplicaciones

complejas y distribuidas en múltiples contenedores y nodos, proporcionando herramientas para despliegue, escalado, balanceo de carga y alta disponibilidad.

1.3. Objetivos

Vamos a definir los objetivos del trabajo y luego presentar en el próximo capítulo el estado de la tecnología respecto a este tema. Los objetivos principales del trabajo son averiguar qué tecnología ofrece la prestación de escalado automático, y desplegar dicha herramienta en una infraestructura actualizada del mercado.

- Será necesario producir escalado automático que garantice el servicio.
- Se requiere entender la lógica y profundizar en el desarrollo de una aplicación ejecutada en un orquestador de contenedores.
- Se precisará utilizar las últimas tecnologías e infraestructura en el ámbito de despliegue de aplicaciones.
- Será necesario desplegar una aplicación en el entorno de orquestación.

1.4. Estructura de la memoria

Este trabajo de fin de grado se estructura en los siguientes capítulos:

1. – **Introducción**: donde se explica la motivación del trabajo, la herramienta empleada, los objetivos y la estructura.
2. – **Estado del arte**: Se explica el estado actual en el despliegue de aplicaciones, por ello se va a hablar de los distintos orquestadores, de Kubernetes, de los diferentes modelos de despliegue de aplicaciones. También se nombrarán los principales proveedores
3. – **Análisis de Requisitos**: donde se enumeran los requisitos funcionales y no funcionales y se explica la metodología.
4. – **Diseño e implementación**: Se realiza una exposición de las tecnologías empleadas para demostrar el diseño y aprendizaje de un entorno de orquestación de contenedores. Se profundiza en la estructura del orquestador elegido (Kubernetes), y se emplea la herramienta en una infraestructura presente en el mercado.
5. – **Pruebas de carga**: donde se explican las mediciones y se valora el rendimiento y comportamiento del producto en la herramienta e infraestructura elegida.
6. – **Conclusiones y trabajo futuro**: qué se ha cumplido, aprendido y qué se podría hacer en trabajos posteriores.

Capítulo 2

Estado del arte

La modulación de los componentes en una arquitectura software está estandarizada como una manera profesional de implementarlos. Antes de diseñar una arquitectura modulada (apartado 3 requisitos) realizaremos unos estudios de mercado para elegir nuestra tecnología de implementación. Hay diferentes maneras de lograr modularidad de los componentes y la principal es la de microservicios desplegados en contenedores, ya que es de las más prominentes en el presente.[8]

2.1. Contenedores

Un contenedor es un entorno de ejecución que proporciona al servicio virtualización a nivel de sistema operativo otorgándole su propio sistema de archivos, procesos, espacio de red y recursos (Ver referencia [9]). Esta virtualización también supone una encapsulación de las herramientas, bibliotecas y dependencias que utilicen.

Los contenedores ofrecen una ejecución consistente y tienen algunas similitudes con las máquinas virtuales, pero también importantes diferencias y ventajas.

Ejecución Consistente

- Los contenedores aíslan las aplicaciones y sus dependencias del sistema operativo subyacente, asegurando que funcionen de manera consistente en diferentes entornos, desde el desarrollo hasta la producción.
- Las aplicaciones en contenedores se empaquetan en archivos de imagen que incluyen todo lo necesario para ejecutarse, lo que garantiza que la aplicación se ejecute de la misma manera independientemente del entorno.

Comparación con Máquinas Virtuales

Similitudes

- **Aislamiento**: Tanto los contenedores como las máquinas virtuales proporcionan aislamiento entre aplicaciones y el entorno subyacente.

- **Portabilidad**: Ambas tecnologías permiten mover aplicaciones entre diferentes entornos de manera relativamente sencilla.

Diferencias y Ventajas de los Contenedores

- **Ligereza**: Los contenedores comparten el mismo sistema operativo del host, lo que reduce significativamente el uso de recursos en comparación con las máquinas virtuales que requieren un sistema operativo completo.
- **Velocidad de Inicio**: Los contenedores se inician mucho más rápido que las máquinas virtuales porque no necesitan arrancar un sistema operativo completo.
- **Eficiencia de Recursos**: Debido a que no requieren un sistema operativo completo para cada instancia, los contenedores utilizan menos memoria y CPU, permitiendo ejecutar más contenedores en el mismo hardware que las máquinas virtuales.

Desventajas de los Contenedores

- **Seguridad**: Aunque el aislamiento de los contenedores es bueno, no es tan robusto como el de las máquinas virtuales. Los contenedores comparten el mismo kernel del sistema operativo, lo que puede ser una preocupación de seguridad.
- **Persistencia de Datos**: Los contenedores están diseñados para ser efímeros, lo que puede complicar la persistencia de datos a menos que se utilicen volúmenes o soluciones de almacenamiento específicas.

Los contenedores ofrecen una ejecución consistente y muchas ventajas en términos de eficiencia y portabilidad en comparación con las máquinas virtuales. Sin embargo, también tienen sus propias desventajas y consideraciones, especialmente en términos de seguridad y gestión de datos persistentes.

2.2. Clústeres y Orquestadores de Contenedores

Un **clúster** es un conjunto de computadoras (también llamadas nodos) que trabajan juntas y se comportan como si fueran una única unidad de cómputo. Los clústeres se utilizan para mejorar la disponibilidad, la capacidad de procesamiento y la eficiencia de los sistemas informáticos. En el contexto de contenedores, un clúster puede contener múltiples nodos que ejecutan aplicaciones en contenedores.

Orquestador de Contenedores

Un **orquestador de contenedores** es una herramienta que automatiza la gestión, el escalado y el mantenimiento de aplicaciones en contenedores en un clúster. Los orquestadores de contenedores proporcionan una serie de funcionalidades clave:

- **Despliegue Automático**: Permiten desplegar y actualizar aplicaciones en contenedores de manera automática.

- **Gestión de la Configuración**: Manejan la configuración de las aplicaciones y sus dependencias.
- **Escalado**: Permiten escalar aplicaciones hacia arriba o hacia abajo según la demanda.
- **Tolerancia a Fallos**: Supervisan los contenedores y los reinician en caso de fallos para asegurar la alta disponibilidad.
- **Redes**: Gestionan la conectividad de red entre contenedores en diferentes nodos.
- **Almacenamiento**: Manejan el almacenamiento persistente requerido por las aplicaciones en contenedores.

Relación entre Clústeres y Orquestadores de Contenedores

La relación entre un clúster y un orquestador de contenedores es fundamental para la gestión eficiente de aplicaciones en contenedores. Aquí se detalla cómo se relacionan:

- **Gestión de Recursos**: El orquestador administra los recursos del clúster (CPU, memoria, almacenamiento) y distribuye los contenedores entre los nodos disponibles para optimizar el uso de los recursos.
- **Distribución de Cargas**: Balancea la carga de trabajo entre los diferentes nodos del clúster para asegurar un rendimiento óptimo y evitar la sobrecarga de un solo nodo.
- **Resiliencia y Recuperación**: En caso de que un nodo falle, el orquestador puede reprogramar los contenedores afectados en otros nodos del clúster, asegurando así la continuidad del servicio.
- **Escalabilidad**: Permite escalar horizontalmente agregando más nodos al clúster y desplegando más instancias de contenedores según la necesidad.
- **Automatización de Tareas**: Facilita la automatización de tareas operativas como el despliegue de aplicaciones, actualizaciones y la gestión del ciclo de vida de los contenedores.

2.3. Orquestadores de contenedores

Actualmente, hay diferentes orquestadores de contenedores que pueden usarse para desplegar productos. Aunque no se hayan probado, existen referencias que garantizan su competencia [12]:

1. – **Kubernetes**: Es el orquestador de contenedores más utilizado y ofrece una gran cantidad de funcionalidades avanzadas para la gestión de contenedores, incluyendo escalado automático, despliegue continuo, y gestión de redes y almacenamiento. A continuación se presenta la herramienta:

- Permite gran margen de trabajo para construir sistemas distribuidos.
- Escrito en Go, es ligero, modular y extensible.
- Liderado por Google, Red Hat y otros.
- Desarrollado por primera vez por ingenieros de Google antes de ser de código abierto en 2014.
- Documentación y comunidad sólidas.
- Gestión de procesos interna con la herramienta "kube-scheduler".
- Arquitectura desacoplada y plataforma extensible.
- Admite revertir implementaciones, automatizar implementaciones y actualizar aplicaciones.
- Equilibrio de carga inherente.
- Utiliza Pods, una unidad atómica de programación. Cada pod tiene su propia dirección IP, no requiere NAT y permite la comunicación dentro del pod a través de localhost.

2. – **Docker Swarm**: Integrado directamente con Docker, es una opción más sencilla que Kubernetes para gestionar contenedores en clústeres. Ofrece funcionalidades como balanceo de carga, escalado y actualizaciones continuas.

3. – **Apache Mesos**: Proporciona una plataforma para ejecutar y gestionar contenedores en un entorno distribuido. Mesos es muy adecuado para grandes organizaciones debido a su capacidad para manejar múltiples tipos de cargas.

4. – **Nomad**: Desarrollado por HashiCorp, es un orquestador ligero que soporta múltiples tipos de cargas de trabajo, no solo contenedores. Por defecto mantiene una baja sobrecarga de recursos.

5. – **OpenShift**: Basado en Kubernetes, es una plataforma empresarial que incluye herramientas adicionales para CI/CD, seguridad y gestión de aplicaciones.

6. – **Rancher**: Ofrece una plataforma de gestión centralizada para Kubernetes, facilitando la gestión de múltiples clústeres Kubernetes en diferentes entornos.

7. – **Portainer**: Proporciona una interfaz de usuario amigable para gestionar contenedores y orquestadores como Docker Swarm, Kubernetes y Nomad.

2.4. Orquestadores de contenedores de proveedores de servicios

Si bien existen estas opciones presentadas en el apartado anterior, también existen otras opciones basadas en las mismas tecnologías, pero esta vez vienen integradas en los principales proveedores de servicios en la nube. Las soluciones ofrecidas para la orquestación de contenedores son las siguientes:

AWS (Amazon Web Services)

1. ECS (Elastic Container Service):
    - Descripción: Un servicio de orquestación de contenedores altamente escalable y de alto rendimiento que admite Docker. Permite ejecutar y escalar aplicaciones en contenedores en AWS [6].
    - Características: Soporte para integración con otros servicios de AWS, facilidad de configuración y administración, opciones para ejecutar tareas y servicios con Fargate o EC2.
2. EKS (Elastic Kubernetes Service):
    - Descripción: Un servicio gestionado de Kubernetes que facilita la ejecución de Kubernetes en AWS sin necesidad de operar el plano de control de Kubernetes.
    - Características: Integración con AWS, alta disponibilidad y seguridad, soporte para actualizaciones automáticas, y escalado.
3. Fargate:
    - Descripción: Una tecnología que permite ejecutar contenedores sin necesidad de gestionar servidores o clústeres. Se puede usar con ECS y EKS.
    - Características: Simplifica la ejecución de contenedores al eliminar la necesidad de provisionar y gestionar la infraestructura subyacente, pago por uso basado en los recursos consumidos por los contenedores.

Google Cloud Platform (GCP)

- GKE (Google Kubernetes Engine):
    - Descripción: Un servicio gestionado de Kubernetes que facilita el despliegue, administración y escalado de aplicaciones en contenedores utilizando la infraestructura de Google.
    - Características: Integración con otros servicios de Google Cloud, soporte para escalado automático, actualizaciones automáticas de clústeres, gestión simplificada y seguridad mejorada.

Microsoft Azure

- AKS (Azure Kubernetes Service):
    - Descripción: Un servicio gestionado de Kubernetes que facilita la implementación, administración y operaciones de Kubernetes en Azure.
    - Características: Integración con servicios de Azure, soporte para escalado automático, actualizaciones automáticas de clústeres, seguridad integrada y monitorización avanzado [4].

En importante abarcar estas herramientas ya que en este trabajo vamos a emplear tres de ellas por distintas razones: Docker Swarm, Kubernetes y GKE.

2.5. Resumen de competencias de los orquestadores de contenedores

En la tabla 2 se resumen las principales características de los orquestadores de contenedores. En concreto, las diferencias en los diferentes aspectos a gestionar en un producto dependiendo de si el orquestador se proporciona como infraestructura como servicio (ver Tabla 2).

2.6. Modelos de implementación de servicios

En el apartado anterior hemos nombrado a los principales proveedores de servicios en la nube. Vamos a presentar la razón de porqué existen estos servicios, y a profundizar en sus prestaciones, ya que es necesario entender sus propuestas de negocio para emplearlas nosotros como usuarios. Para explicar su modelo de negocio hay que explicar los diferentes modelos de implementación de servicios [13]:

1. **On-Premise**: En el modelo On-premise, una empresa instala y gestiona su propia infraestructura y software en sus instalaciones.
    - Infraestructura: La empresa compra y mantiene el hardware (servidores, almacenamiento, redes) y software necesario.
    - Control: Ofrece el máximo control sobre la infraestructura y los datos.
    - Costos: Altos costos iniciales para la adquisición de hardware y software, y costos continuos de mantenimiento y actualización.
    - Seguridad: La empresa es responsable de la seguridad física y digital de sus sistemas.
    - Escalabilidad: Puede ser limitada y requiere inversión adicional en hardware.
2. **IaaS (Infrastructure as a Service)**: IaaS proporciona infraestructura de TI virtualizada a través de Internet. Los proveedores de IaaS ofrecen recursos de computación, almacenamiento y redes bajo demanda.
    - Infraestructura: Hardware virtualizado y servicios de red gestionados por el proveedor.
    - Control: Los usuarios tienen control sobre el sistema operativo, almacenamiento, aplicaciones y redes.
    - Costos: Modelo de pago por uso, reduciendo los costos iniciales de infraestructura.
    - Seguridad: Compartida entre el proveedor (infraestructura) y el cliente (datos, aplicaciones).
    - Escalabilidad: Alta, con capacidad de ajustar recursos según demanda.
    - Ejemplos: Amazon Web Services (AWS) EC2, Microsoft Azure, Google Compute Engine.

Categoría	**Orquestadores de Contenedores**	**Orquestadores de Contenedores de Proveedores de Servicios**
Definición	Software para gestionar, escalar y desplegar contenedores.	Servicios gestionados por proveedores cloud que ofrecen funcionalidades similares a los orquestadores de contenedores.
Ejemplos	Kubernetes, Docker Swarm	AWS ECS, GKE, AKS
Administración	Requiere configuración y gestión manual por parte del usuario.	Gestión automatizada y soporte técnico proporcionado por el proveedor.
Implementación	Puede ser implementado on-premises o en cualquier infraestructura cloud.	Implementación específica en la infraestructura del proveedor de servicios cloud.
Escalabilidad	Alta escalabilidad, pero depende de la infraestructura subyacente.	Alta escalabilidad con recursos bajo demanda gestionados por el proveedor.
Actualizaciones y Mantenimiento	El usuario es responsable de las actualizaciones y mantenimiento.	El proveedor gestiona actualizaciones, parches y mantenimiento.
Integración con Otros Servicios	Puede requerir configuración adicional para integrar con otros servicios.	Integración nativa y simplificada con otros servicios del mismo proveedor (como bases de datos, almacenamiento, etc.).
Seguridad	La seguridad debe ser gestionada por el usuario, incluyendo configuraciones y actualizaciones.	Seguridad gestionada por el proveedor con configuraciones predefinidas y prácticas recomendadas.
Costos	Generalmente más económicos, pero con costos ocultos de gestión y mantenimiento.	Costos por uso con tarifas claras, pero potencialmente más altos a largo plazo.
Flexibilidad	Alta flexibilidad para personalizar y adaptar a necesidades específicas.	Menos flexibilidad debido a la dependencia del ecosistema del proveedor.
Aprendizaje y Conocimiento Técnico	Requiere conocimientos profundos y especializados para la gestión efectiva.	Menor curva de aprendizaje con soporte y documentación proporcionada por el proveedor.
Soporte y Comunidad	Amplia comunidad de código abierto, con foros y contribuciones.	Soporte profesional y servicio al cliente del proveedor, además de la comunidad de usuarios.
Ejemplos de Casos de Uso	Ideal para empresas que buscan control total sobre su infraestructura.	Ideal para empresas que buscan facilidad de uso y gestión simplificada sin preocuparse por la infraestructura subyacente.

Tabla 2: Exposición de Orquestadores de Contenedores y Orquestadores de Contenedores de Proveedores de Servicios

3. **PaaS (Platform as a Service)**: PaaS proporciona una plataforma completa que permite a los desarrolladores crear, desplegar y gestionar aplicaciones sin preocuparse por la infraestructura subyacente.

    - Infraestructura: Incluye hardware y software (sistema operativo, bases de datos, servidores web) gestionados por el proveedor.
    - Control: Los usuarios gestionan las aplicaciones y los datos, pero no la infraestructura subyacente.
    - Costos: Pago por uso, reduciendo costos de desarrollo y despliegue.
    - Seguridad: Gestionada en parte por el proveedor (infraestructura, plataforma) y en parte por el cliente (aplicaciones, datos).
    - Escalabilidad: Alta, con servicios escalables según necesidades.
    - Ejemplos: Google App Engine, Microsoft Azure App Services, Heroku.

4. **SaaS (Software as a Service)**: SaaS proporciona software a través de Internet. Los usuarios acceden a aplicaciones a través de un navegador web sin necesidad de gestionar la infraestructura subyacente.

    - Infraestructura: Todo el hardware y software necesario es gestionado por el proveedor.
    - Control: Los usuarios solo gestionan la configuración de las aplicaciones; no tienen control sobre la infraestructura.
    - Costos: Modelo de suscripción o pago por uso, sin costos iniciales significativos.
    - Seguridad: Principalmente gestionada por el proveedor, aunque los usuarios son responsables de la configuración y el uso seguro de la aplicación.
    - Escalabilidad: Muy alta, con capacidad de ajustarse automáticamente a las necesidades del usuario.
    - Ejemplos: Google Workspace, Microsoft Office 365, Salesforce.

Como se puede observar hay muchas diferencias a tener en cuenta. Para elegir bien qué tecnología y herramienta emplear habría que realizar un estudio sobre las necesidades a cubrir, y analizar si compensa cubrir características contratando al proveedor, o garantizando el servicio por nuestra parte. En la tabla 3 (ver Tabla 3) se resumen las características de los modelos de servicios. La figura 1, que ilustra el modelo de implementación de servicios, junto a la Tabla 3 facilitan la justificación de utilizar un modelo u otro (ver Figura 1).

Característica	On-Premise	IaaS	PaaS	SaaS
Gestión	Empresa	Proveedor (infraestructura)	Proveedor (plataforma e infraestructura)	Proveedor (todo)
Control	Completo	Medio	Limitado	Mínimo
Costos iniciales	Altos	Bajos	Bajos	Muy bajos
Costos operativos	Altos	Pago por uso	Pago por uso	Suscripción/ Pago por uso
Escalabilidad	Limitada	Alta	Alta	Muy alta
Seguridad	Empresa	Compartida	Compartida	Proveedor principal
Ejemplos	Sistemas tradicionales en empresas	AWS EC2, Microsoft Azure, Google Compute Engine	Google App Engine, Microsoft Azure App Services, Heroku	Google Workspace, Microsoft Office 365, Salesforce

Tabla 3: Comparación Resumida de Modelos de Implementación de un Producto en la Nube

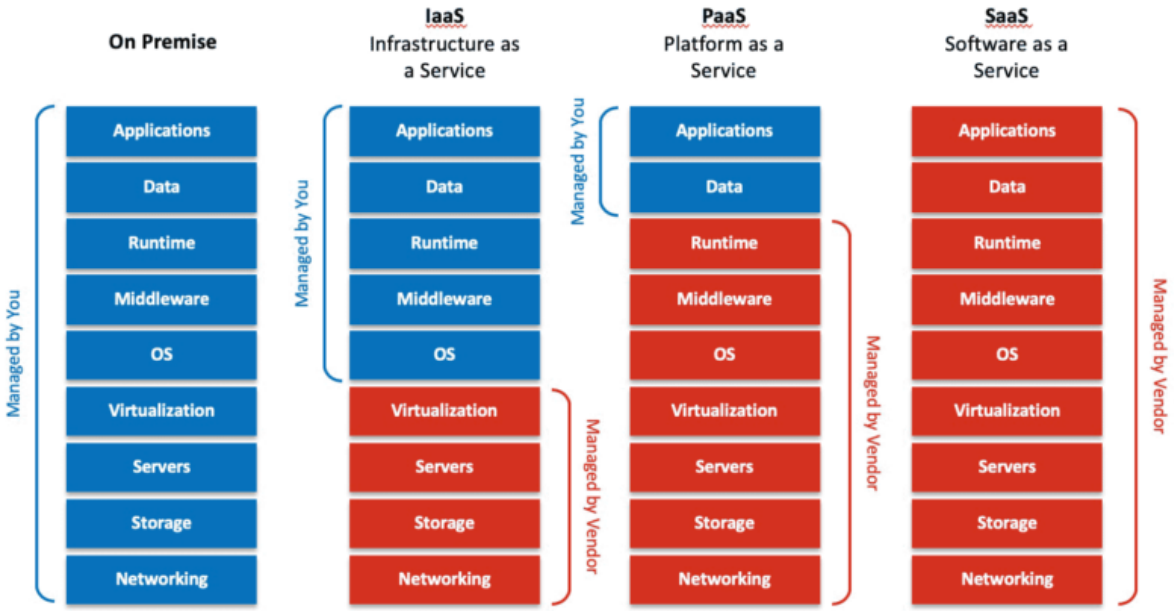

Figura 1: Modelos de servicio divididos según las prestaciones cubiertas por un proveedor y por el usuario.

Capítulo 3

Análisis

En este capítulo se explicará la metodología a seguir durante el trabajo. Luego, se listarán los requisitos funcionales y no funcionales de la aplicación desarrollada en este trabajo. Se ha empleado una notación estándar IEEE 830 para definir y formalizar los distintos requisitos del sistema.

3.1. Metodología

La metodología de trabajo empleada se basa en el cumplimiento de principios y prácticas de **DevOps**. En ella se utilizan herramientas para automatizar la configuración y el aprovisionamiento de infraestructuras y se utilizan scripts y herramientas para automatizar el proceso de despliegue, reduciendo el riesgo de errores humanos y acelerando la entrega de software. De hecho, en este trabajo destacará el despliegue de servicios de manera declarativa [2].

- *DevOps* es una práctica que combina el desarrollo de software (*Dev*) y las operaciones de tecnología de la información (*Ops*). Su objetivo es acortar el ciclo de vida del desarrollo de sistemas y proporcionar una entrega continua con alta calidad. *DevOps* promueve una cultura de colaboración entre los equipos de desarrollo y operaciones, automatizando procesos para mejorar la eficiencia y la fiabilidad del software.
- **Integración continua (CI)** es una práctica de desarrollo de software donde los desarrolladores integran sus cambios de código en un repositorio compartido con frecuencia, preferiblemente varias veces al día. Cada integración es verificada mediante una compilación automatizada y pruebas, lo que permite detectar y corregir errores de manera rápida y eficiente.
- **Despliegue continuo (CD)**, por otro lado, va un paso más allá de la integración continua. No solo implica la integración frecuente del código, sino también su despliegue automático en entornos de producción o de prueba. Esto asegura que el software está siempre en un estado que puede ser desplegado, permitiendo una entrega rápida y fiable a los usuarios finales.

Se utilizan además sistemas de monitorización para observar el comportamiento de las aplicaciones y la infraestructura en tiempo real. También se configuran y emplean sistemas

de registros centralizados que recogen y analizan registros(logs) de diferentes componentes del sistema, permitiendo detectar y resolver problemas rápidamente.

Los beneficios del Enfoque DevOps son varios. Destacamos que la automatización y las prácticas de CI/CD permiten despliegues más rápidos y frecuentes. También destacamos que la integración y pruebas continuas aseguran que los errores se detecten y corrijan rápidamente.

3.2. Requisitos Funcionales

Desde la tabla 4 a la tabla 8 podemos observar los requisitos funcionales que describen las funcionalidades que debe tener la aplicación:

Número	RF-1
Nombre	Despliegue de microservicios en orquestador de contenedores
Descripción	Se debe desplegar un producto basado en microservicios en un orquestador de contenedores.
Motivación	Aprovechar la modularidad y escalabilidad que ofrecen los microservicios y contenedores.
Criterios de aceptación	Los microservicios deben estar completamente operativos y gestionados por el orquestador de contenedores especificado.

Tabla 4: RF-1: Despliegue de microservicios en orquestador de contenedores

Número	RF-2
Nombre	Eficiencia en la orquestación de microservicios
Descripción	Aprovechar la lógica de la orquestación de contenedores para desplegar los microservicios de manera eficiente y eficaz.
Motivación	Optimizar recursos y tiempo de despliegue.
Criterios de aceptación	Los microservicios deben desplegarse y escalar de manera automatizada y bajo una gestión eficiente de recursos.

Tabla 5: RF-2: Eficiencia en la orquestación de microservicios

Número	RF-3
Nombre	Tecnología de orquestación de contenedores
Descripción	Emplear una tecnología de orquestación de contenedores presente en el mercado.
Motivación	Utilizar herramientas robustas y bien soportadas por la comunidad.
Criterios de aceptación	La tecnología seleccionada debe estar documentada y demostrar compatibilidad con los requisitos del sistema.

Tabla 6: RF-3: Tecnología de orquestación de contenedores

Número	RF-4
Nombre	Uso de proveedor de servicios en la nube
Descripción	Utilizar el servicio de un proveedor de servicios en la nube para la implementación y gestión de recursos.
Motivación	Aprovechar la flexibilidad y escalabilidad de los servicios en la nube.
Criterios de aceptación	El sistema debe integrarse correctamente con los servicios ofrecidos por el proveedor de servicios en la nube seleccionado.

Tabla 7: RF-4: Uso de proveedor de servicios en la nube

Número	RF-5
Nombre	Documentación del proceso de despliegue
Descripción	Documentar cada paso del proceso de configuración y despliegue de la aplicación.
Motivación	Garantizar la replicabilidad y mantenibilidad del sistema.
Criterios de aceptación	La documentación debe ser completa, detallada y accesible para los equipos de desarrollo y operaciones.

Tabla 8: RF-5: Documentación del proceso de despliegue

3.3. Requisitos No Funcionales

Los requisitos no funcionales son aquellos que describen otros aspectos y características de la aplicación, en nuestro caso los requisitos están recogidos de la tabla 9 a la tabla 13:

Número	RNF-1
Nombre	Tiempo de respuesta bajo carga
Descripción	Las aplicaciones desplegadas responden dentro de un tiempo aceptable bajo diferentes cargas de trabajo.
Motivación	Garantizar la experiencia de usuario y la eficiencia operativa.
Criterios de aceptación	Las aplicaciones deben mantener un tiempo de respuesta promedio por debajo de cierto umbral bajo carga simulada.

Tabla 9: RNF-1: Tiempo de respuesta bajo carga

Número	RNF-2
Nombre	Escalabilidad horizontal
Descripción	Verificar que las aplicaciones puedan escalar horizontalmente en respuesta a la demanda creciente.
Motivación	Garantizar la disponibilidad y capacidad de respuesta del sistema ante incrementos repentinos en la carga.
Criterios de aceptación	El sistema debe demostrar la capacidad de añadir instancias adicionales automáticamente y distribuir la carga eficientemente.

Tabla 10: RNF-2: Escalabilidad horizontal

Número	RNF-3
Nombre	Comparación de escalabilidad
Descripción	Realizar una comparación de la facilidad y eficacia de escalado entre diferentes tecnologías y configuraciones.
Motivación	Seleccionar la mejor opción para la implementación basada en criterios objetivos de escalabilidad.
Criterios de aceptación	Se debe documentar y presentar un análisis comparativo de escalabilidad entre al menos dos opciones tecnológicas.

Tabla 11: RNF-3: Comparación de escalabilidad

Número	RNF-4
Nombre	Documentación de configuraciones
Descripción	Documentación de todas las configuraciones y scripts utilizados para facilitar el mantenimiento futuro.
Motivación	Garantizar que el mantenimiento y las actualizaciones del sistema sean eficientes y bien gestionadas.
Criterios de aceptación	Todas las configuraciones deben estar documentadas de manera clara y estar disponibles para el equipo de operaciones.

Tabla 12: RNF-4: Documentación de configuraciones

Número	RNF-5
Nombre	Compatibilidad con sistemas operativos actuales
Descripción	Debe poder funcionar en los sistemas operativos empleados en el presente.
Motivación	Garantizar la interoperabilidad y el soporte continuo en entornos operativos actuales.
Criterios de aceptación	El sistema debe ser probado y demostrar su funcionalidad en los sistemas operativos especificados por el cliente.

Tabla 13: RNF-5: Compatibilidad con sistemas operativos actuales

3.4. Restricciones

Las restricciones, recogidas de las tablas 14 a la 16, limitan la realización y desarrollo correcto del proyecto por temas ajenos al producto. Estas limitaciones se formalizan y se exponen a continuación:

Número	R-1
Nombre	Presupuesto limitado para servicios en la nube
Descripción	Uso de créditos o presupuesto limitado para el uso servicios de un proveedor de servicios en la nube.
Motivación	Gestionar los recursos financieros de manera eficiente y evitar sobrecostos.
Criterios de aceptación	Los costos totales deben mantenerse dentro del presupuesto asignado para los servicios en la nube.

Tabla 14: R-1: Presupuesto limitado para servicios en la nube

Número	R-2
Nombre	Tiempo limitado para pruebas exhaustivas
Descripción	Tiempo limitado para realizar pruebas exhaustivas en los tres entornos.
Motivación	Completar el proyecto dentro del plazo establecido sin comprometer la calidad.
Criterios de aceptación	Las pruebas deben ser planificadas y ejecutadas eficientemente dentro del tiempo disponible.

Tabla 15: R-2: Tiempo limitado para pruebas exhaustivas

Número	R-3
Nombre	Consistencia en el entorno de desarrollo y pruebas
Descripción	Necesidad de mantener consistencia en el entorno de desarrollo y pruebas para la comparación de tecnologías.
Motivación	Asegurar resultados confiables y comparables entre diferentes tecnologías y configuraciones.
Criterios de aceptación	Los entornos de desarrollo y pruebas deben estar alineados y documentados para garantizar la consistencia.

Tabla 16: R-3: Consistencia en el entorno de desarrollo y pruebas

Capítulo 4

Diseño e implementación

Desplegaremos una aplicación basada en microservicios de diferentes maneras. Será de manera incremental en cuanto a prestaciones y complejidad, e iremos analizando las ventajas y desventajas que vamos obteniendo.

4.1. Estrategia de trabajo

Vamos a realizar un camino a través de diferentes tecnologías y herramientas. Para cada tecnología habrá un nivel de abstracción y una herramienta que podremos emplear para analizar la lógica del producto desplegado con ella [10].

1. Orquestador de contenedores "de bajo nivel": Esta tecnología permitirá analizar el rendimiento de una aplicación en un clúster en un nivel en el que también se gestionan los contenedores y las imágenes desplegadas.
   - Herramientas: **Docker Swarm** y la imágen de Docker para desplegar contenedores que ejecutan instancias de Docker (DinD).
   - Infraestructura: clúster en una máquina y virtualizado en varias máquinas.
2. Orquestador de contenedores: Esta tecnología permitirá analizar el rendimiento de una aplicación en un clúster con la lógica y costumbres mas conocidas, desde la perspectiva On-Premise.
   - Herramientas: **Minikube** es una herramienta que permite disponer de un entorno sencillo de Kubernetes con la mayor parte de sus funcionalidades. **Kubectl** es una interfaz de línea de comandos para ejecutar comandos sobre despliegues en clúster de Kubernetes.
   - Infraestructura: Un nodo de un cluster real en Kubernetes que permita probar las funcionalidades que nos interesan de estas herramientas y forzar los despliegues que se prueben en el.
3. Orquestador de contenedores de un proveedor de servicios en la nube (IaaS): Esta tecnología permitirá analizar el rendimiento de una aplicación en un clúster real de un proveedor que compite en el mercado.

- Herramientas: Google Compute Engine y **Google Kubernetes Engine**, permitirán desplegar el producto en un entorno real y probarlo con cargas de trabajo reales.
- Infraestructura: dos tipos de clúster real, los que ofrece el proveedor.

En el análisis de cada herramienta, se tendrá en cuenta la manera en la que se cumplen con los requisitos del funcionamiento de la aplicación. Se valorará la eficacia de las herramientas y tecnologías, y al cumplir con las prestaciones, se valorará entonces la eficiencia y el rendimiento.

4.2. Docker Swarm y Docker in Docker, DinD

Docker in Docker (DinD) se refiere a ejecutar instancias de Docker dentro de un contenedor Docker. Esto puede ser útil para escenarios como la construcción de contenedores dentro de un contenedor, pruebas, o ejecución de pipelines de CI/CD. Utilizar DinD en un clúster Docker Swarm permite que las disntinas ejecuciones de Docker en los contenedores puedan hacer de nodos para el clúster, ya que los nodos pueden ser máquinas virtuales.

Bien es cierto que la idea de clúster es ganar potencia de cómputo y de esta misma manera estamos perdiendo la razón de utilizar un clúster, pero podemos observar la lógica del clúster mientras se forma, y hacer determinadas configuraciones de manera manual.

Vamos a implementar la lógica de un cluster en nuestra máquina local, ya que nos permite implementar un modelo híbrido entre Kubernetes y el despliegue de docker-compose (ver Algoritmo 1).

Algoritmo 1 Configuración inicial del clúster en Docker Swarm. Se inician contenedores que se convierten en nodos virtuales que se añaden al clúster

```
//Inicia el clúster

docker swarm init (me da el comando de añadir workers)

//ejecuta un contenedor, aparece una máquina virtual

docker run -d --priviledge --name dind1 -p 9000:9000 nodo-dind

//la máquina se añade al clúster y se vueleve un nodo

docker exec -it dind1 docker swarm join \
    --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39
       trti4wxv-8vxv8rssmk743ojnwacrr2e7c \
    192.168.99.100:2377
```

El comando de añadir workers siempre está disponible ejecutando: docker swarm join -token «mánager».

Emplear el *docker exec* va a ser necesario para trabajar con un clúster de este tipo. Por ejemplo, si queremos que un nodo DinD sea el mánager, todas las operaciones tendrán que empezar por docker exec –it (nombre-del-contenedor).

Y es que es importante conocer los dos tipos de nodos, y distinguir las diferencias (ver Tabla 17):

Capacidad	**Nodo Manager**	**Nodo Worker**
Tareas de Gestión	Sí (gestiona el estado y la configuración del clúster)	No
Programación de Servicios	Sí (asigna tareas a los nodos)	No
Ejecución de Tareas/Contenedores	Sí	Sí
Participación en el Consenso	Sí (participa en el algoritmo de consenso Raft)	No
Almacenamiento del Estado del Clúster	Sí (mantiene el estado del clúster en el almacenamiento de Raft)	No
Aceptación de Comandos del Usuario	Sí (recibe comandos `docker service`, `docker node`, etc.)	No
Monitorización del Estado del Clúster	Sí	No
Escalabilidad de Tareas	No (no se recomienda sobrecargar con tareas de contenedores)	Sí

Tabla 17: Capacidades y comportamiento determinado de los Nodos Manager y Worker en un Clúster.

Teniendo un pequeño clúster configurado, se pueden crear los servicios (ver Algoritmo 2):

Algoritmo 2 Creación y configuración de un servicio de manera manual. En este caso de la base de datos.

```
docker service create --name mysqldb --network pcshop3local_
    default --publish published=9001,target=3306 --placement-
    pref spread=node.role==worker -e MYSQL_ROOT_PASSWORD=
    password -e MYSQL_DATABASE=test mysql
```

Pero lo ideal es proporcionar un archivo de despliegue declarativo. Utilizamos un archivo de docker-compose para desplegar más fácilmente (ver Figura 2):

docker stack deploy -c docker-compose.yml pcshop3swarm

También podemos escalar manualmente los servicios. Pero al ser la única manera de escalar, no vale como opción competitiva.

```
C:\Users\jaime\Desktop\CV\TFG\Tienda Online>docker stack deploy -c docker-compose-swarm.yml pcshop3swarm
Ignoring unsupported options: restart

Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network pcshop3swarm_red1
Creating service pcshop3swarm_grafana_pcshop
Creating service pcshop3swarm_prometheus_pcshop
Creating service pcshop3swarm_node_exporter_pcshop
Creating service pcshop3swarm_cadvisor_pcshop
Creating service pcshop3swarm_cliservicecompose
Creating service pcshop3swarm_pcshop3compose
Creating service pcshop3swarm_mysqldb
```

Figura 2: Despliegue declarativo en Swarm. En la figura se ejecuta el comando que incluye como argumento el archivo declarativo que la configuración para desplegar los servicios.

```
C:\Users\jaime>docker service scale clientservice=3
clientservice scaled to 3
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service clientservice converged

C:\Users\jaime>docker service ls
ID             NAME            MODE         REPLICAS   IMAGE                                PORTS
o57xmfyffo7x   clientservice   replicated   3/3        jaimedeonate/public:clientservicetag   *:8600->8500/tcp
a2uvtd84vt8b   mysqldb         replicated   1/1        mysql:latest                         *:9001->3306/tcp
```

Figura 3: Escalado en Swarm. Se puede observar que el comando es manual y por lo tanto el escalado es manual.

Comprobamos un despliegue fácil gracias a las redes overlay. Resulta que, aunque los workers no conozcan otros nodos ni otra información, si conocen los puertos como si fuera la misma máquina.

Pero el escalado es manual, no hay mejora requerida, solo ejecutar en un sistema físicamente distribuido, donde tenemos la posibilidad de aumentar fácilmente el hardware para ganar potencia en ejecución, pero no hemos automatizado nada (ver Figura 3).

Ademas, ha resultado incómoda el control de versiones, se ha tenido que emplear el repositorio externo, por comodidad, aun cuando las imágenes se encontraban en la máquina local.

Así que vamos a iniciarnos en Kubernetes para paliar estos problemas y finalmente utilizar una herramienta de despliegue presente en el mercado.

4.3. Kubernetes

Comenzamos a analizar la herramienta central del trabajo [7] [11].

4.3.1. minikube

Minikube es una herramienta que facilita la ejecución de Kubernetes en un entorno local en una sola máquina (ver Figura 4). Está diseñado para desarrolladores que desean aprender y

probar aplicaciones basadas en Kubernetes sin necesidad de configurar un clúster completo de Kubernetes.

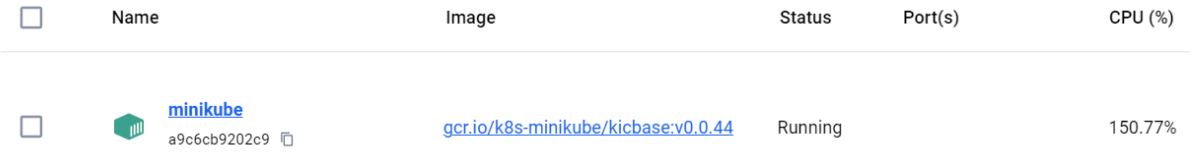

Figura 4: Clúster simulado en local, ejecutando un contenedor de minikube en el entorno de Docker se consigue un programa que permite ejecutar Kubernetes.

Algunas **características clave de Minikube** incluyen:

1. Entorno Local: Permite ejecutar un clúster de Kubernetes en una máquina local, como tu computadora portátil o estación de trabajo.
2. Facilidad de Uso: Simplifica la configuración y gestión de un clúster de Kubernetes, proporcionando un único nodo Kubernetes que se puede iniciar con un comando simple.
3. Desarrollo y Pruebas: Es ideal para desarrolladores que necesitan probar sus aplicaciones en un entorno similar al de producción basado en Kubernetes.

Además vamos a declarar los principales componentes de kuberntes, ya que será obligatorio conocerlos para trabajar con la herramienta.

4.3.2. Principales Objetos de Kubernetes

1. **Pods**: El pod es la unidad básica de despliegue en Kubernetes y representa una única instancia en ejecución de una aplicación. Un pod puede contener uno o más contenedores que comparten el mismo almacenamiento y red.

   Características:

   - Compartición de almacenamiento: Los contenedores en un pod comparten volúmenes.
   - Compartición de red: Todos los contenedores dentro de un pod comparten la misma dirección IP.
   - Ciclo de vida conjunto: Los contenedores en un pod se crean y destruyen juntos.

   Es ideal para casos donde múltiples contenedores necesitan comunicarse estrechamente, como un servidor web y un contenedor de registro.

2. **Deployments**: Los deployments proporcionan una forma declarativa de gestionar pods. Permiten definir el estado deseado de la aplicación y Kubernetes se encargará de actualizar y mantener ese estado.

   Características:

- Actualización Continua: Permiten actualizaciones graduales y rollbacks automáticos si algo sale mal.
- Escalado: Facilitan el escalado automático y manual de pods.
- Gestión de Versiones: Kubernetes rastrea el historial de versiones, permitiendo revertir a versiones anteriores si es necesario.

Es ideal para desplegar aplicaciones de forma segura y gestionada, con capacidad de actualización sin tiempo de inactividad.

3. **Services**: Los services en Kubernetes definen una política lógica de agrupación y un conjunto de políticas para acceder a los pods. Proveen una forma de exponer una aplicación ejecutada en un conjunto de pods como un servicio de red.

   Tipos:

   - ClusterIP: Exposición del servicio solo dentro del clúster.
   - NodePort: Exposición del servicio en cada nodo del clúster en un puerto estático.
   - LoadBalancer: Provisión de un balanceador de carga externo para distribuir el tráfico.

   Características:

   - Descubrimiento de Servicios: Proveen un punto de acceso estable a través de un nombre DNS.
   - Balanceo de Carga: Distribuyen el tráfico de red entre los pods designados.

   Es útil para garantizar que las aplicaciones puedan ser accedidas de manera consistente y equilibrada a través de la red.

4.3.3. Despliegue en minikube

Se despliega el servicio con nuestro archivo declarativo, kubernetes-producto.yaml (ver Figura 5).

```
C:\Users\jaime>docker service ls
ID             NAME                                  MODE         REPLICAS   IMAGE                                          PORTS
sfb135hnk2lm   pcshop3swarm_cadvisor_pcshop          replicated   1/1        gcr.io/cadvisor/cadvisor:latest
o31yl8wg21k7   pcshop3swarm_cliservicecompose        replicated   0/1        clientservice2:latest                          *:9100->8500/tcp
vy4cvkby3obm   pcshop3swarm_grafana_pcshop           replicated   1/1        grafana/grafana:8.0.6                          *:3000->3000/tcp
wekk3wukdujb   pcshop3swarm_mysqldb                  replicated   2/1        mysql:latest                                   *:9001->3306/tcp
yqstigkp60p0   pcshop3swarm_node_exporter_pcshop     replicated   1/1        quay.io/prometheus/node-exporter:latest        *:9099->9100/tcp
tdif0hf2u7av   pcshop3swarm_pcshop3compose           replicated   1/1        pcshop3:latest                                 *:9000->8443/tcp
jvw8qwm86hpg   pcshop3swarm_prometheus_pcshop        replicated   1/1        prom/prometheus:v2.28.1                        *:9090->9090/tcp
```

Figura 5: Despliegue declarativo de la aplicación de todos los servicios que conforman el producto.

Ahora investigamos y averiguamos que Kubernetes ofrece un servicio de métricas y autoescalado que puede integrar pero no viene por defecto. Hablamos del HPA (Horizontal Pod Autoscaler) y tenemos que obtenerlo de la siguiente fuente (ver Algoritmo 3):

Algoritmo 3 Obtención y lanzamiento del servicio de autoescalado proporcionado por Kubernetes.

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
    server/releases/latest/download/components.yaml
```

Se despliegan una serie de procesos que implementan la prestación y se activan el servicio de autoescalado por CPU, que al principio no funciona (ver Figura 6).

```
C:\Users\jaime>kubectl get hpa
NAME               REFERENCE                     TARGETS             MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   cpu: <unknown>/50%   1         10        1          130m
```

Figura 6: Ejecución del autoescalado incluido en el Kubernetes de minikube, que resulta en error.

Descargamos el archivo, lo modificamos y relanzamos nuestra aplicación. Se configura así la prestación con mi servicio (ver Figura 7).

```
C:\Users\jaime\Desktop\CV\TFG\2º TFG>kubectl get hpa
NAME               REFERENCE                     TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   cpu: 2%/50%   1         10        1          173m
```

Figura 7: El servicio de monitorización y autoescalado está operativo, ya que el servicio funciona y recibe el uso de CPU.

Se provoca un aumento de la CPU con pruebas manuales (ver Figura 8).

```
C:\Users\jaime\Desktop\CV\TFG\2º TFG>kubectl get hpa
NAME               REFERENCE                     TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   cpu: 6%/50%   1         10        1          3h1m

C:\Users\jaime\Desktop\CV\TFG\2º TFG>kubectl get hpa
NAME               REFERENCE                     TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   cpu: 55%/50%   1         10        1          3h1m
```

Figura 8: Se monitoriza el consumo de CPU en diferentes estados del contenedor que aloja el servicio que requiere autoescalado.

Se confirma el escalado revisando la salida del HPA, puede verse en el número de réplicas (ver Figura 9).

```
C:\Users\jaime\Desktop\CV\TFG\2º TFG>kubectl get hpa
NAME               REFERENCE                     TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   cpu: 55%/50%   1         10        1          3h2m

C:\Users\jaime\Desktop\CV\TFG\2º TFG>kubectl get hpa
NAME               REFERENCE                     TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   cpu: 6%/50%   1         10        2          3h2m
```

Figura 9: Hay un control claro sobre el consumo de CPU, que permite monitorizar el servicio desde la terminal.

Se observa la réplica del escalado del producto (ver Figura 10).

```
C:\Users\jaime\Desktop\CV\TFG\2º TFG>kubectl get pods
NAME                                  READY   STATUS    RESTARTS        AGE
apache1                               1/1     Running   5 (117m ago)    39h
cadvisor-pcshop-9ddcb79cd-wl2pz       1/1     Running   1 (117m ago)    3h6m
cliservicecompose-7ff7498864-k77j4    1/1     Running   0               4m53s
cliservicecompose-7ff7498864-mpm9z    1/1     Running   0               7m19s
grafana-pcshop-758948d798-grjvl       1/1     Running   0               79m
mysqldb-755b75c6fb-nx984              1/1     Running   2 (117m ago)    5h14m
node-exporter-pcshop-7c4bbbd775-wnbb2 1/1     Running   1 (117m ago)    3h6m
pcshop3compose-c87b955f7-tmz76        1/1     Running   10 (117m ago)   5h30m
prometheus-pcshop-7cb859c56b-px8hm    1/1     Running   0               79m
web-68b899c98d-25gpw                  1/1     Running   5 (117m ago)    39h
```

Figura 10: el servicio cliservice ha sido escalado, comprobado al generar una carga de trabajo de manera manual.

Ahora que observamos una herramienta eficaz y una tecnología válida para satisfacer las necesidades del producto, necesitamos una infraestructura que despliegue el producto de manera real y pueda ofrecer un servicio real en el mercado.

4.4. Google Kubernetes Engine

Hay varias razones sólidas para considerar el uso de Google Kubernetes Engine (GKE) para desplegar y gestionar las aplicaciones en contenedores:

Por un lado GKE facilita la implementación y gestión de clústeres de Kubernetes. Google Cloud se encarga de la gestión de la infraestructura subyacente, lo que permite centrar al usuario más en el desarrollo de aplicaciones y menos en la administración operativa de Kubernetes [1].

Por otro lado GKE está diseñado para escalar automáticamente según las necesidades de las aplicaciones. Se pueden configurar políticas de escalado automático para los pods y los nodos del clúster, lo que garantiza que las aplicaciones se mantengan disponibles y respondan a cambios en la carga de trabajo.

Además hay completa integración con Google Cloud, GKE se integra estrechamente con otros servicios de Google Cloud Platform (GCP).

4.4.1. Clústers disponibles en GKE

En Google Kubernetes Engine (GKE), la elección entre un clúster automático (Autopilot) y un clúster estándar depende de las necesidades y el nivel de control que se desea tener sobre el entorno de Kubernetes. Se presenta una comparación de ambos enfoques [14]:

Clúster Automático (Autopilot)

Ventajas:

- Autopilot gestiona muchas de las tareas operativas, como el escalado, la gestión de nodos y la configuración de la infraestructura.
- Autopilot se factura según los recursos consumidos por los pods, lo que puede ser más eficiente en términos de costos para cargas de trabajo variables.
- GKE Autopilot incluye configuraciones predeterminadas que mejoran la seguridad y el cumplimiento normativo.
- ofrece menor sobrecarga operativa, ideal para equipos que desean centrarse más en el desarrollo de aplicaciones y menos en la gestión de la infraestructura.

Desventajas:

- Al automatizar muchas tareas, se pierde algo de control sobre la infraestructura subyacente y las configuraciones detalladas.
- Puede haber limitaciones en cuanto a la personalización y ajustes específicos de la configuración del clúster.

Clúster Estándar

Ventajas:

- Permite una personalización completa de la configuración del clúster, incluidas las configuraciones de nodos, redes, políticas de seguridad, etc.
- Esta flexibilidad es ideal para casos en los que se requieren configuraciones específicas o integraciones complejas con otros sistemas.
- Se Pueden controlar cómo se escalan los nodos y optimizar el rendimiento según las necesidades de la aplicación.

Desventajas:

- Requiere más tiempo y conocimientos para gestionar y mantener el clúster.
- Puede ser más costoso si no se optimiza adecuadamente, ya que se factura según los recursos de nodo provisionados, independientemente del uso real.

En este trabajo vamos a probar los dos (ver Figura 11), y a decidir dónde desplegaríamos la aplicación.

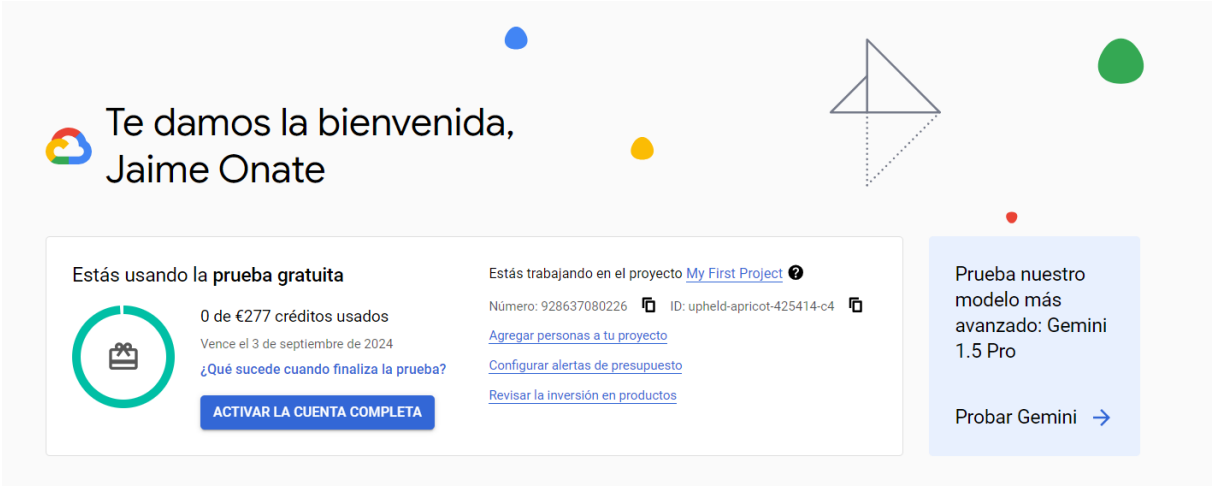

Figura 11: Primer contacto con GCP: Google Cloud Platform. Se recibe la información de los primero pasos a tomar y el regalo de 300 dólares para probar sus servicios.

Para aprovechar al máximo los servicios de un proveedor en el mercado tan potente y competitivo como Google, habría que dedicar otro trabajo de investigación y práctica sobre todas sus funcionalidades. Además, se inicia una relación usuario-proveedor que comienza a implicar derechos y responsabilidades (ver Figura 12).

Figura 12: En seguida se aprende que hay un gran número de herramientas que pueden servir para mejorar el rendimiento de cualquier producto.

4.4.2. Configuración del clúster Autopilot

Dadas todas las herramientas disponibles, se elige el servicio de Google Kubernetes Engine y se inicia un clúster disponible. Solo es necesario subir un archivo de despliegue declarativo de la aplicación para que funcione (ver Figura 13).

Figura 13: Se ilustra cómo conectar con el clúster desplegado.

Tras iniciar el clúster tenemos la opción de abrir una termial en línea desde el navegador, o bien una terminal desde nuestra máquina que nos conecte a nuestro clúster con solo identificarnos. Una vez conectados, tenemos la opción de subir nuestro archivo de declaración, un archivo que descargará las imágenes del registro en línea (ver Figura 14).

```
jaime_onate_rp@cloudshell:~ (siliconavalley-shop)$ dir
kubernetes-pcshop2.yaml                kubernetes-pcshop-autopilot_(2).yaml
kubernetes-pcshop-autopilot_(1).yaml   kubernetes-pcshop-autopilot_(3).yaml
```

Figura 14: Se han ejecutado varias versiones del archivo de despliegue de la aplicación.

El funcionamiento del producto es inmediato (ver Figura 15).

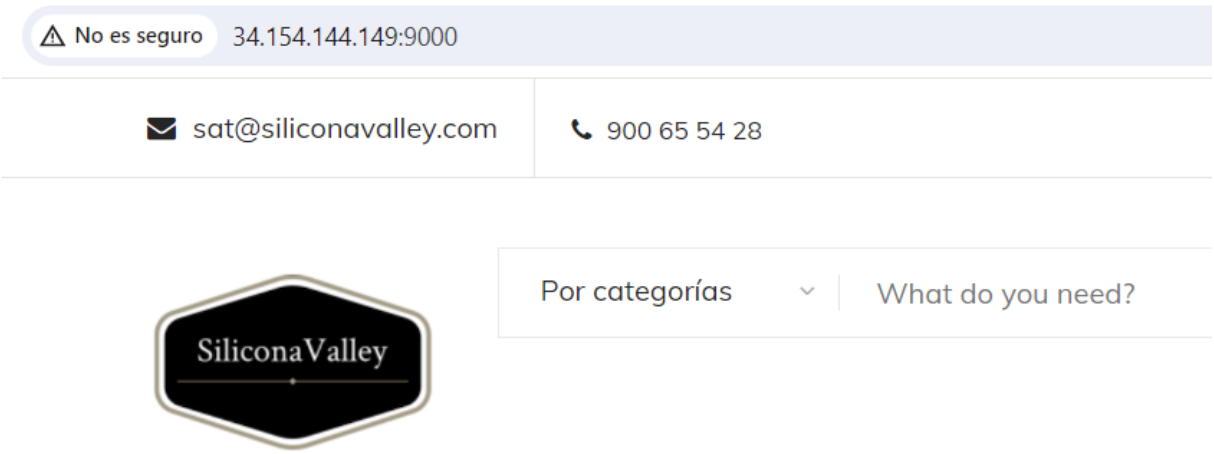

Figura 15: Desplegamos nuestra aplicación en la nube, nos dan una dirección IP que comprobamos que funciona correctamente.

Desplegamos la aplicación y tratamos de generar carga de trabajo.

```
Events:
  Type     Reason            Age    From
  ----     ------            ----   ----
  Warning  FailedScheduling  3m38s  gke.io/optimize-utilization-scheduler
n victims found for incoming pod..
  Normal   TriggeredScaleUp  3m30s  cluster-autoscaler
nceGroups/gk3-autopilot-cluster-1-pool-2-e35ed944-grp 0->1 (max: 1000)}]
  Warning  FailedScaleUp     3m8s   cluster-autoscaler
g scheduled.
jaime_onate_rp@cloudshell:~ (siliconavalley-shop)$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
cliservicecompose   1/1     1            1           141m
mysqldb             1/1     1            1           141m
pcshop3compose      2/4     4            2           141m
```

Figura 16: Hay cierto escalado pero ocurre algo que impide un comportamiento esperado.

Parece que hay algún problema, ya que el escalado pide cuatro servicios pero solo despliega dos (ver Figura 16).

```
Message
-------
0/2 nodes are available: 2 Insufficient cpu, 2 Insufficient memory. preemption: 0/2 nodes are available: 2 No preemptio

pod triggered scale-up: [{https://www.googleapis.com/compute/v1/projects/siliconavalley-shop/zones/europe-west8-c/insta

Node scale up in zones europe-west8-c associated with this pod failed: GCE out of resources. Pod is at risk of not bein
```

Figura 17: El proveedor de servicios notifica un problema en el servicio de GKE.

Se investiga la causa del problema a través de las notificaciones (ver Figuras 17 y 18) y la documentación proporcionada por el proveedor de servicios.

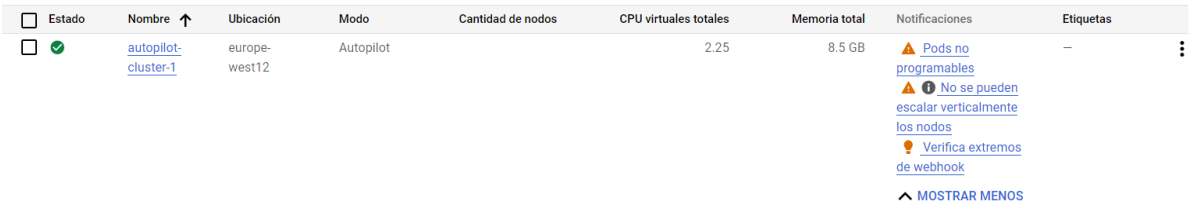

Figura 18: Se observan problemas en el clúster, ya que hay ciertas notificaciones que aparecen de manera interactiva y también en el buzón de notificaciones.

Se descubre el problema investigando y profundizando en la documentación del proveedor (ver Figura 19).

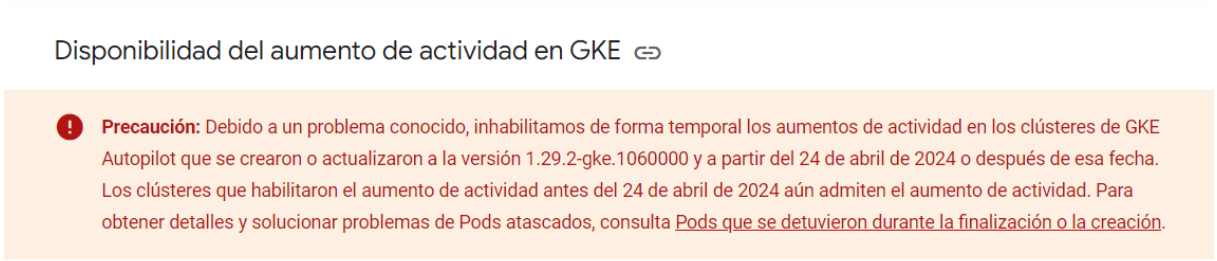

Figura 19: Error de GKE, el proveedor del servicio. Hay un error que persiste en los clúster Autopilot e impide una correcta evaluación de nuestro producto.

Hay un grave problema con el servicio que proporciona Autopilot, podemos concluir que no es una buena opción para desplegar nuestro producto.

4.4.3. Ejecución en el clúster Estándar

Pero vistos estos problemas se procede a probar el clúster estándar. Los clústeres estándar permiten una mayor personalización y control sobre la configuración de los nodos y el clúster en general. Se pueden ajustar parámetros específicos que pueden no estar disponibles o ser limitados en un clúster Autopilot, lo que hace que algunas aplicaciones o configuraciones específicas pueden no ser completamente compatibles con las restricciones y políticas predeterminadas. También hay acceso completo a los nodos y se pueden realizar diagnósticos más detallados.

Finalmente podemos comprobar un escalado muy sensible y eficaz (ver Figura 20).

```
jaime_onate_rp@cloudshell:~ (siliconavalley-shop)$ kubectl get hpa
NAME                REFERENCE                      TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   1%/50%    1         2         1          137m
pcshop3compose      Deployment/pcshop3compose      94%/50%   1         6         1          8m41s
jaime_onate_rp@cloudshell:~ (siliconavalley-shop)$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
cliservicecompose-6b6b9654c5-p8f9g   1/1     Running   0          137m
mysqldb-57f97ffc75-scw4p             1/1     Running   0          137m
pcshop3compose-6b7cf4bb7b-kqlq5      1/1     Running   0          13s
pcshop3compose-6b7cf4bb7b-xkxll      1/1     Running   0          3m46s
jaime_onate_rp@cloudshell:~ (siliconavalley-shop)$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
cliservicecompose-6b6b9654c5-p8f9g   1/1     Running   0          138m
mysqldb-57f97ffc75-scw4p             1/1     Running   0          138m
pcshop3compose-6b7cf4bb7b-h7jcc      0/1     Pending   0          63s
pcshop3compose-6b7cf4bb7b-kqlq5      1/1     Running   0          2m3s
pcshop3compose-6b7cf4bb7b-m22xq      0/1     Pending   0          63s
pcshop3compose-6b7cf4bb7b-xkxll      1/1     Running   0          5m36s
jaime_onate_rp@cloudshell:~ (siliconavalley-shop)$ kubectl get hpa
NAME                REFERENCE                      TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
cliservicecompose   Deployment/cliservicecompose   1%/50%    1         2         1          139m
pcshop3compose      Deployment/pcshop3compose      51%/50%   1         6         4          10m
jaime_onate_rp@cloudshell:~ (siliconavalley-shop)$
```

Figura 20: Escalabilidad inmediata en el clúster estándar, prueba satisfactoria del servicio de autoescalado en el producto.

Capítulo 5

Pruebas de carga

El escalado en el clúster estándar de GKE satisface los objetivos planteados para el trabajo (ver Figura 21). Aunque en el primer clúster se pudo apreciar escalado, este estaba muy limitado.

No se ha podido realizar pruebas de autoescalado con grandes cargas de trabajo porque suponen un gasto directo en la contratación de los servicios.

```
NAME                                    READY   STATUS    RESTARTS
cliservicecompose-55b547cb7f-6nflg      1/1     Running   0
cliservicecompose-55b547cb7f-6skh4      1/1     Running   0
cliservicecompose-55b547cb7f-m5f7l      1/1     Running   0
cliservicecompose-55b547cb7f-v5dc5      1/1     Running   0
cliservicecompose-55b547cb7f-zm9jj      1/1     Running   0
mysqldb-77b4f469dc-h66mc                1/1     Running   0
pcshop3compose-698d9b9ddb-62ppr         1/1     Running   1 (2m20s ago)
```

Figura 21: Alerta activada en el producto del clúster estándar, comportamiento del autoescalado apreciado de manera inmediata.

5.1. Servicio de carga de trabajo en Google Cloud Engine

Estas pruebas de carga ayudan a evaluar cómo se comporta el sistema bajo diferentes niveles de estrés y cómo responde el autoescalado. A continuación se exponen las principales herramientas más populares en siguiente tabla (ver Tabla 18), donde se presenta las características mas prácticas. En concreto se va a describir la configuración, pruebas de carga y monitorización para poder elegir la herramienta adecuada según las circunstancias de cada caso.

Consideradas las herramientas, se decide utilizar la herramienta integrada en Google Cloud. Se prueba la funcionalidad de generar carga de trabajo para probar el rendimiento del clúster (ver Figuras 22 y 23).

Se consiguen efectos parecidos, ya que la versión gratuita del servicio es bastante pequeña para nuestros casos de prueba.

Herramienta	Configuración	Pruebas de Carga	Monitorización
Google Cloud Load Testing	Configuración integrada en Google Cloud	Generación de carga a gran escala	Integrado con Google Cloud Monitoring
Apache JMeter	Configuración manual y Test Plan en JMeter	Simulación de carga y análisis de rendimiento	Requiere configuración adicional para integración con monitorización
Locust	Script en Python para definir el comportamiento del usuario	Interfaz web para iniciar y controlar las pruebas de carga	Puede integrarse con herramientas de monitorización como Prometheus

Tabla 18: Comparación resumida de Herramientas de Pruebas de Carga en GCE, tanto la configuración como las pruebas de carga y la monitorización.

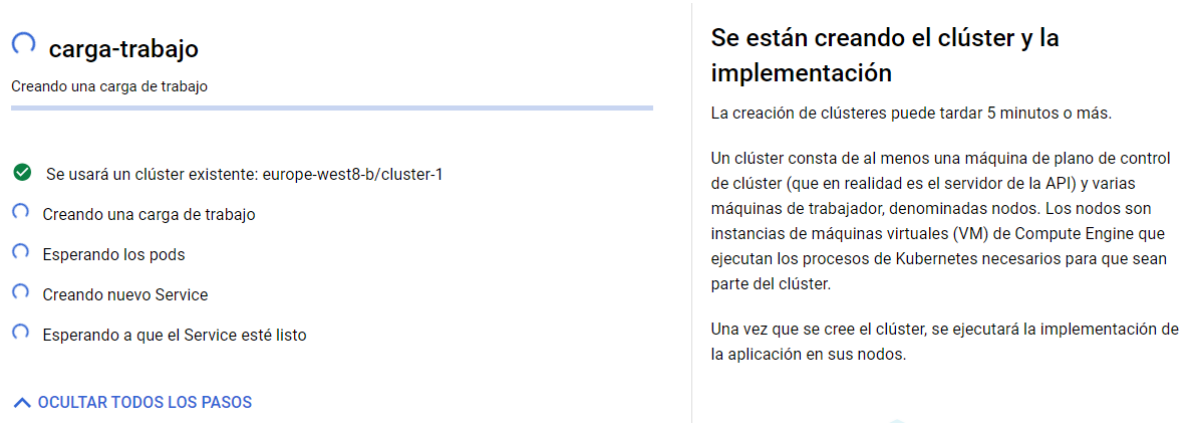

Figura 22: Simulación carga de trabajo en Google Cloud Load Testing.

Pods administrados

Revisión	Nombre	Estado	Reinicios
1	carga-trabajo-8f9fddb58-7rvcb	Running	0
1	carga-trabajo-8f9fddb58-47sjq	Running	0
1	carga-trabajo-8f9fddb58-hh5c8	Running	0

Figura 23: Despliegue de pods que realizan la carga solicitada.

Figura 24: Comportamiento de la carga de trabajo que informa de que el servicio de Kubernetes (el HPA Horizontal Pod Autoscaler) no ha podido leer todas las métricas. Eso es el comportamiento deseado ya que buscamos autoescalado en base al uso de CPU.

En el comportamiento final (ver Figura 24) observamos un servicio replicado del servicio interno que permite al producto ofrecer las prestaciones necesarias. Podemos confiar en el proveedor de servicios de que mantendrá y asegurará el servicio de nuestra aplicación.

5.2. Recopilación de datos observados, diferencias en el rendimiento final de las herramientas probadas

Se exponen las primeras conclusiones sobre las herramientas y tecnologías probadas y se resumen en una tabla (ver Tabla 19):

1. **Docker Swarm**

   - **Capacidades de Autoescalado**: Docker Swarm no tiene un soporte nativo para el autoescalado de servicios basado en la carga de trabajo. Se pueden implementar scripts personalizados y herramientas de terceros, pero no es tan robusto ni integrado como en Kubernetes o GCE.
   - **Rendimiento y Eficiencia**: Debido a la falta de autoescalado nativo, es posible que se tenga que preveer con mas recursos para manejar picos de carga, resultando en una utilización ineficiente de los recursos. La escalabilidad manual puede afectar el rendimiento y la capacidad de respuesta bajo cargas fluctuantes.

2. **Kubernetes**

   - **Autoescalado Horizontal de Pods (HPA)**: Kubernetes ofrece el Autoescalado Horizontal de Pods (HPA), ajustando automáticamente el número de pods en función de métricas como la CPU y la memoria. Se pueden definir políticas de autoescalado basadas en métricas personalizadas.
   - **Cluster Autoscaler**: Kubernetes tiene el Cluster Autoscaler, que ajusta el número de nodos en el clúster según la demanda de recursos, garantizando suficiente capacidad para manejar la carga sin intervención manual.
   - **Rendimiento y Eficiencia**: El autoescalado avanzado de Kubernetes mejora el rendimiento y la eficiencia operativa, adaptándose rápidamente a cambios en la carga de trabajo y asegurando una utilización óptima de los recursos y alta disponibilidad.

3. **Google Cloud Engine (GCE)**

   - **Autoescalado de Instancias de VM**: GCE ofrece autoescalado para grupos de instancias administrados, ajustando automáticamente el número de instancias basándose en métricas como la CPU, las solicitudes por segundo y otras métricas personalizadas.
   - **Integración con Google Kubernetes Engine (GKE)**: GKE proporciona los beneficios del autoescalado de Kubernetes junto con la infraestructura optimizada de Google Cloud. GKE integra HPA y Cluster Autoscaler, ofreciendo una solución completa y eficiente para el autoescalado.
   - **Rendimiento y Eficiencia**: La combinación de GCE y GKE ofrece una capacidad de autoescalado altamente eficiente, asegurando que la aplicación pueda manejar picos de carga sin problemas. La infraestructura de Google Cloud proporciona alta disponibilidad y escalabilidad, maximizando el rendimiento y minimizando el tiempo de inactividad.

Capacidad	**Docker Swarm**	**Kubernetes**	**Google Cloud Engine**
Autoescalado	No nativo, requiere soluciones personalizadas	Soporte robusto con HPA y Cluster Autoscaler	Integrado y fácil de configurar para instancias de VM, con HPA y Cluster Autoscaler en GKE
Rendimiento	Menos eficiente y más propenso a subutilización o sobreprovisión de recursos	Muy eficiente, adapta dinámicamente los recursos a la carga de trabajo, optimizando el uso y asegurando alta disponibilidad	Altamente eficiente, escalabilidad dinámica y recursos optimizados, especialmente con la integración de GKE

Tabla 19: Comparación de Rendimiento como orquestador de contenedores y comportamiento general del Autoescalado entre Docker Swarm, Kubernetes y Google Cloud Engine. Tabla propuesta como primeras conclusiones.

Capítulo 6

Conclusiones

6.1. Objetivos logrados

En este trabajo se ha logrado desplegar una aplicación basada en microservicios en un proveedor de servicios en la nube. La aplicación se sirve de autoescalado configurado y proporcionado por las herramientas de última generación.

- Gracias al HPA (Auto escaling Horizontal de Pods) se logra el escalado automático de los servicios de la aplicación.
- Se han empleado las herramientas de DinD y minikube para desplegar la aplicación con diferentes herramientas que aportan diferentes prestaciones. Gracias a desplegar paso a paso se ha podido mantener la compresión sobre lo que se ha ido desplegando.
- Se han empleados las herramientas proporcionadas por Google, donde participan en el mercado de tecnologías e infraestructura en el ámbito de despliegue de aplicaciones.
- Se ha aprovechado la aplicación desarrollada en la universidad y mejorada de manera particular para probar el autoescalado con una aplicación basada en microservicios.

6.2. Reflexiones

Trabajar como desarrollador DevOps es una opción atractiva, ya que hay una alta demanda de profesionales con este perfil. Principalmente por la creciente adopción de prácticas ágiles y de entrega continua en las organizaciones. Las empresas buscan integrar desarrollo y operaciones para mejorar la eficiencia y la calidad del software.

Además, los roles DevOps suelen estar bien remunerados debido a la combinación de habilidades técnicas en desarrollo, automatización, y gestión de infraestructura. Y aunque trabajar en DevOps implica mantenerse al tanto de las últimas tecnologías y herramientas, puede ser muy enriquecedor para un desarrollo profesional.

Sin embargo, es importante estar no subestimar los desafíos que conlleva este rol, como la necesidad de aprender continuamente nuevas tecnologías, lidiar con entornos complejos y a menudo impredecibles, y colaborar eficazmente en equipos de personas.

6.3. Trabajos futuros

La iniciación a Google Cloud Computing abre las puertas a una gran cantidad de servicios de última generación para emplear sobre una aplicación profesional o comercial. Por lo tanto el primer trabajo sería seguir investigando, probando y desarrollando estos servicios y aplicarlos [5].

A continuación, se describen algunos de los servicios más relevantes para una aplicación como la que hemos visto en este trabajo:

- **Máquinas Virtuales (VMs)**: Creación y gestión de máquinas virtuales personalizables en términos de CPU, memoria y almacenamiento.
- **Grupos de Instancias**:
  - **Grupos de Instancias Administrados**: Facilitan el escalado automático, la actualización de instancias y la gestión de la disponibilidad.
  - **Grupos de Instancias No Administrados**: Proporcionan un conjunto de VMs sin las características avanzadas de los grupos administrados.
- **Discos Persistentes**: Almacenamiento de alta disponibilidad y rendimiento que se puede adjuntar a las VMs.
- **Redes y Seguridad**: Configuración de redes privadas virtuales (VPC), cortafuegos, y reglas de red para controlar el tráfico de entrada y salida.
- **Balanceo de Carga**: Distribución del tráfico de red entre varias VMs para asegurar una alta disponibilidad y rendimiento.
- **Autoscaler**: Escalado automático de las VMs en función de la demanda de la carga de trabajo.

Servicios Relevantes para Pruebas de Carga de Trabajo

Para probar una aplicación con diferentes cargas de trabajo variables y arbitrarias, los siguientes servicios de Google Compute Engine son particularmente útiles:

- **Grupos de Instancias Administrados con Autoscaler**:
  - Permiten definir políticas de escalado automático basadas en métricas como el uso de CPU, el tráfico de red, o métricas personalizadas.
  - Proporcionan alta disponibilidad al distribuir la carga entre múltiples instancias y asegurarse de que siempre haya un número adecuado de instancias ejecutándose.
- **Balanceo de Carga**:
  - Distribuye el tráfico de red entrante entre las instancias de VM en el clúster, mejorando la disponibilidad y el rendimiento.

- Facilita la gestión del tráfico durante picos de carga y asegura una experiencia de usuario consistente.

- **Cloud Monitoring y Cloud Logging**:
  - Proporcionan monitorización y registro en tiempo real de las métricas de rendimiento y los registros de las aplicaciones.
  - Ayudan a identificar cuellos de botella, errores y a realizar un seguimiento del comportamiento de la aplicación bajo diferentes cargas de trabajo.

- **Cloud Pub/Sub**:
  - Sistema de mensajería asíncrona que permite desacoplar los componentes de la aplicación.
  - Útil para manejar cargas de trabajo variables distribuyendo las tareas de manera eficiente.

- **Discos Persistentes SSD**:
  - Ofrecen alto rendimiento de E/S, lo cual es crucial para aplicaciones que requieren acceso rápido a los datos bajo cargas de trabajo intensivas.

- **Cloud Load Testing (servicio de terceros)**:
  - Herramientas como Apache JMeter o servicios en la nube como BlazeMeter pueden integrarse para simular diferentes cargas de trabajo y probar el rendimiento de la aplicación.

Servicios de Google Cloud Platform

Hemos probado algo de lo que queda dentro del marco de trabajo de Google Cloud Engine, pero la nube de Google ofrece muchas más servicios con los que se puede experimentar y entender cómo funcionan, ya que podrían ser servicios que se considerara implementar en un futuro:

- **Cloud Storage:** Cloud Storage es un servicio de almacenamiento en la nube escalable, confiable y seguro para empresas de todos los tamaños. Ofrece una amplia gama de características, incluyendo almacenamiento de objetos, almacenamiento de archivos y almacenamiento en bloque, y está diseñado para satisfacer las necesidades de las aplicaciones más exigentes.

- **Cloud SQL:** Google Cloud SQL es un servicio de base de datos completamente gestionado que ayuda a implementar, gestionar y escalar fácilmente las bases de datos relacionales. Cloud SQL incluye los siguientes servicios: MySQL, PostgreSQL, SQL Server.

- **BigQuery:** BigQuery es un almacén de datos analíticos a escala de petabytes completamente gestionado que permite a las empresas analizar todos sus datos de forma muy rápida. BigQuery incluye los siguientes servicios: Ediciones, On-Demand, BigQuery ML, BI Engine.

- **Cloud TPU:** Google Cloud TPUs son aceleradores de IA diseñados a medida, optimizados para el entrenamiento e inferencia de grandes modelos de IA.
- **Cloud GPU:** Se pueden adjuntar una o más GPUs a las instancias de máquinas virtuales (VM) para acelerar cargas de trabajo específicas o descargar trabajo de las vCPUs.
- **Vertex AI GenAI Models:** Se pueden probar, ajustar y desplegar los grandes modelos de IA generativa de Google para su uso en las aplicaciones impulsadas por IA. Vertex AI GenAI Models incluye los siguientes servicios: Modelos de Código, Modelos de Imágenes, Modelos de Lenguaje, Modelos Multimodales, Modelos de Embedding.
- **Vertex AI Agent Builder:** Vertex AI Agent Builder permite a los desarrolladores, incluso aquellos con habilidades limitadas en aprendizaje automático, aprovechar el poder de los modelos fundamentales de Google, la experiencia en búsqueda y las tecnologías de IA conversacional para crear aplicaciones de IA generativa de nivel empresarial. Vertex AI Agent Builder incluye los siguientes servicios: Vertex AI Search, Vertex AI Conversation.
- **Persistent Disk:** Persistent Disk es un servicio de almacenamiento en bloque que proporciona almacenamiento confiable, duradero y escalable para las aplicaciones. Persistent Disk incluye los siguientes servicios: Persistent Disk, Hyperdisk.
- **Google Kubernetes Engine:** Analizada en este trabajo, Google Kubernetes Engine proporciona un entorno completamente gestionado para implementar, escalar y operar las aplicaciones en contenedores en Google Cloud. Google Kubernetes Engine incluye los siguientes servicios: GKE Standard, Backup for GKE.
- **Document AI:** Extrae datos de, clasifica y divide documentos a través de modelos preentrenados y personalizados. Luego utiliza Warehouse para buscar y almacenar documentos. Document AI incluye los siguientes servicios: Document AI Workbench, Procesadores Generales, Procesadores de Identidad, Procesadores de Préstamos, Procesadores de Adquisiciones, Warehouse.
- **Cloud Translation:** Cloud Translation permite a los sitios web y aplicaciones traducir texto dinámicamente a través de una API. Cloud Translation incluye los siguientes servicios: Cloud Translation - Basic, Cloud Translation - Advanced.
- **App Engine:** Construye sitios web monolíticos renderizados en el servidor. App Engine soporta lenguajes de desarrollo populares con una gama de herramientas para desarrolladores. App Engine incluye los siguientes servicios: Entorno Estándar, Entorno Flexible.
- **Cloud Functions:** Ejecuta un código en la nube sin servidores ni contenedores que gestionar con nuestro producto de funciones como servicio (FaaS) escalable y de pago por uso.
- **AlloyDB:** AlloyDB para PostgreSQL es un servicio de base de datos PostgreSQL nativo de la nube completamente gestionado que ofrece características de nivel empresarial, alto rendimiento, escalabilidad y seguridad.

- **Cloud Vision:** Google Cloud Vision es una suite de APIs de aprendizaje automático que ayuda a los desarrolladores a entender fácilmente el contenido de imágenes, videos y texto.
- **Networking:** Virtual Private Cloud (VPC) Networking permite crear una red privada en la nube con los mismos controles que haya en las instalaciones. Networking incluye los siguientes servicios: Dirección IP, Transferencia de Datos, NAT Gateway, Cloud Load Balancing.
- **Pub/Sub:** Pub/Sub es un servicio de mensajería en tiempo real completamente gestionado que permite enviar y recibir mensajes entre aplicaciones. Pub/Sub incluye los siguientes servicios: Pub/Sub, Pub/Sub Lite.
- **Speech-to-Text v2:** Convierte con precisión el habla en texto con una API impulsada por la investigación y tecnología de IA de Google.
- **Text-to-Speech AI:** Convierte texto en voz natural utilizando una API impulsada por las mejores tecnologías de IA de Google.
- **Firestore:** Desarrolla fácilmente aplicaciones utilizando una base de datos de documentos completamente gestionada, escalable y sin servidor.
- **Filestore:** Las instancias de Filestore son servidores de archivos completamente gestionados en Google Cloud que pueden conectarse a múltiples tipos de clientes.
- **Vertex AI Training:** Vertex AI proporciona un servicio de entrenamiento gestionado que permite supervisar el entrenamiento de modelos de aprendizaje automático a gran escala.
- **Vertex AI Prediction:** Vertex AI Prediction despliega modelos de aprendizaje automático entrenados para servir solicitudes de predicción.
- **Cloud Run:** Cloud Run permite ejecutar servicios de frontend y backend, trabajos por lotes, desplegar sitios web y aplicaciones, y procesar cargas de trabajo en cola sin la necesidad de gestionar infraestructura.
- **Cloud CDN:** Cloud CDN (Content Delivery Network) utiliza la red global de borde de Google para servir contenido más cerca de los usuarios, lo que acelera los sitios web y aplicaciones.
- **Cloud BigTable:** Servicio de base de datos NoSQL compatible con HBase, de nivel empresarial, con baja latencia de milisegundos, escalabilidad ilimitada y 99.99 % de disponibilidad.
- **Apigee:** La herramienta nativa de gestión de APIs de Google Cloud para construir, gestionar y asegurar APIs para cualquier caso de uso, entorno o escala.
- **Gemini Code Assist:** Desarrollo asistido por IA para aplicaciones. Aumenta la velocidad de desarrollo y entrega de software con Gemini Code Assist para Desarrolladores, impulsado por modelos fundamentales de Google, con protección de seguridad y privacidad de nivel empresarial.

- **Cloud Memorystore:** Reduce la latencia con un servicio en memoria escalable, seguro y altamente disponible para Redis Cluster, Redis y Memcached. Cloud Memorystore incluye los siguientes servicios: Memorystore para Redis, Memorystore para Redis Cluster, Memorystore para Memcached.
- **Cloud Spanner:** Cloud Spanner es un servicio de base de datos relacional completamente gestionado que ofrece alta disponibilidad, escalabilidad y consistencia fuerte.
- **Cloud VPN:** Cloud VPN conecta de forma segura la red peer a la red de Virtual Private Cloud (VPC) a través de una conexión VPN IPsec.
- **Bare Metal Solution:** Google Cloud Bare Metal Solution permite ejecutar cargas de trabajo especializadas, como bases de datos Oracle, directamente en hardware gestionado por Google dentro de los centros de datos de Google, ofreciendo acceso de baja latencia a los servicios en la nube.
- **Cloud Interconnect:** Cloud Interconnect extiende la red externa a la red de Google a través de una conexión de alta disponibilidad y baja latencia. Cloud Interconnect incluye los siguientes servicios: Partner Interconnect, Dedicated Interconnect.
- **VMware Engine:** Transforma rápidamente las aplicaciones basadas en VMware a Google Cloud sin cambios en las aplicaciones, herramientas o procesos.
- **Cloud Armor:** Ayuda a proteger las implementaciones en Google Cloud de múltiples tipos de amenazas, incluyendo ataques DDoS, cross-site scripting (XSS) e inyección SQL.
- **Vertex AI:** Vertex AI incluye los siguientes servicios: Vertex AI Workbench, Vertex AI Pipelines, Vertex AI Feature Store, y Vertex AI Vector Search.
- **NetApp Volumes:** NetApp Volumes es un servicio de almacenamiento de archivos completamente gestionado en Google Cloud, que proporciona alto rendimiento, escalabilidad y gestión avanzada de datos para cargas de trabajo empresariales.
- **Cloud DNS:** DNS confiable, resiliente y de baja latencia servido desde la red mundial de Google con todo lo que necesitas para registrar, gestionar e implementar los dominios.
- **Dataflow:** Tecnología de procesamiento de datos en flujo y por lotes unificado.

Bibliografía

[1] Aravind Agarwal y P Raj. *Learning Google Kubernetes Engine: A Comprehensive Guide to GKE and Its Applications in the Industry*. Packt Publishing, 2020 (vid. pág. 28).

[2] Brian Boyd y Pramod Sadalage. *Refactoring Databases for DevOps: Evolutionary Database Design and Continuous Delivery for Kubernetes and AWS*. Addison-Wesley Professional, 2020 (vid. pág. 15).

[3] Brendan Burns y col. "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade". En: *ACM Queue* 14.1 (2016), págs. 70-93 (vid. pág. 1).

[4] Shital Chauhan. *Azure Kubernetes Services with Microservices: Practical Usage of AKS for Real-World Applications*. Packt Publishing, 2021 (vid. pág. 9).

[5] JJ Geewax Davis. *Google Cloud Platform in Action*. Manning Publications, 2019 (vid. pág. 42).

[6] Bernard Golden. *Amazon Web Services for Dummies*. John Wiley & Sons, 2013 (vid. pág. 9).

[7] Kelsey Hightower, Brendan Burns y Joe Beda. *Kubernetes: Up and Running*. O'Reilly Media, 2017 (vid. pág. 24).

[8] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment". En: *Linux Journal* 2014.239 (2014), pág. 2 (vid. pág. 5).

[9] Liz Rice Morris. *Container Security: Fundamental Technology Concepts that Protect Containerized Applications*. O'Reilly Media, 2020 (vid. pág. 5).

[10] Michael Pohl. *Docker Management Design Patterns: Swarm Mode on Amazon Web Services*. Packt Publishing, 2018 (vid. pág. 21).

[11] Gigi Sayfan Ross. *Mastering Kubernetes: An Advanced Guide to Container Orchestration and Microservices*. Packt Publishing, 2020 (vid. pág. 24).

[12] Rohan Shetty. *Kubernetes and OpenShift for Developers: A CoreOS Quick Start*. Packt Publishing, 2019 (vid. pág. 7).

[13] N Sultan. *Cloud Computing: PaaS, SaaS, Public, Private, and Hybrid Clouds*. Pearson, 2017 (vid. pág. 10).

[14] Vitthal White, Harihara Deshmukh y Piyush Chaganti. *Google Cloud Platform for Architects: Design and Manage Powerful Cloud Solutions*. Packt Publishing, 2020 (vid. pág. 28).