UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERÍA DE SISTEMAS

"IMPLEMENTACIÓN DE UN SISTEMA RECOMENDADOR PARA UN COMERCIO ELECTRÓNICO UTILIZANDO GRAPHQL COMO MIDDLEWARE PARA EL CONSUMO DE MICROSERVICIOS EN INFRAESTRUCTURAS BASADAS EN CÓDIGO"

Trabajo de titulación previo a la obtención

del título de Ingeniero de Sistemas

AUTORES: JORGE RENÉ ARÉVALO PAÑI

FRANKLIN GUSTAVO GUALLPA GIÑIN

TUTOR: ING. DIEGO FERNANDO QUISI PERALTA, M.Sc.

Cuenca - Ecuador

2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Jorge René Arévalo Pañi, con documento de identificación N°0105345193, y Franklin Gustavo Guallpa Giñin, con documento de identificación N°0106305477; manifestamos que:

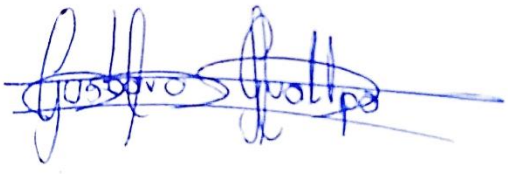
Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 29 de abril del 2022

Atentamente,

Jorge René Arévalo Pañi

0105345193



Franklin Gustavo Guallpa Giñin

0106305477

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Jorge René Arévalo Pañi con documento de identificación N°0105345193 y Franklin Gustavo Guallpa Giñin con documento de identificación N°0106305477, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Implementación de un sistema recomendador para un comercio electrónico utilizando GraphQL como middleware para el consumo de microservicios en infraestructuras basadas en código", el cual ha sido desarrollado para optar por el título de: Ingeniero de Sistemas, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

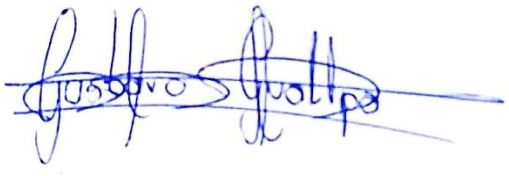
En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 29 de abril del 2022

Atentamente,

Jorge René Arévalo Pañi

0105345193



Franklin Gustavo Guallpa Giñin

0106305477

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Diego Fernando Quisi Peralta con documento de identificación N° 0104616461, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: "IMPLEMENTACIÓN DE UN SISTEMA RECOMENDADOR PARA UN COMERCIO ELECTRÓNICO UTILIZANDO GRAPHQL COMO MIDDLEWARE PARA EL CONSUMO DE MICROSERVICIOS EN INFRAESTRUCTURAS BASADAS EN CÓDIGO", realizado por Jorge René Arévalo Pañi con documento de identificación N°0105345193 y por Franklin Gustavo Guallpa Giñin con documento de identificación N°0106305477, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 29 de abril del 2022

Atentamente,

_____

Diego Fernando Quisi Peralta

0104616461

DEDICATORIA

Dedico el presente trabajo de titulación a Dios y a la Virgen por llenarme de sabiduría y guiarme para culminar con este proceso educativo. A mis padres, Jorge y Digna, por el apoyo incondicional desde mis primeros pasos en esta carrera y por el sacrificio que han realizado para convertirme en un hombre de bien y cumplir con mi objetivo. A mis hermanos por los buenos y malos momentos, por la paciencia y los consejos que me motivaron durante toda mi trayectoria. A mi compañera de vida quien ha sido partícipe en este camino apoyándome en cada una de mis decisiones sin dejarme solo en ningún momento.

DEDICATORIA

Dedico este trabajo de titulación a Dios y su madre la santísima virgen María en su advocación de María Auxiliadora porque como mi madre ha estado allí en todos los momentos de mi vida y permitirme llegar a esta etapa tan importante de mi formación profesional. A mi querida Mamá Rosa Giñin que a pesar de la distancia ha sido el pilar más importante y por siempre demostrarme su apoyo y cariño incondicional, gracias a su paciencia y esfuerzo que me han permitido llegar a cumplir hoy un sueño más.

AGRADECIMIENTO

Agradezco a Dios y a la Virgen por haberme cuidado durante toda esta etapa de mi estudio y por las bendiciones recibidas. A mi mamá Digna por darme la vida y por ser una persona de luz quien me apoyó, apoya y apoyará infinitamente sin pedir nada a cambio. A mi papá Jorge por motivarme y guiarme a cumplir mis sueños y por ser partícipe de los mejores momentos de mi vida. A Mercy, Priscila y Antonio por confiar en mi y ayudarme cuando más apoyo necesitaba. A Alexandra por ser mi apoyo incondicional, por su amor infinito y por ser mi guía personal en este presente proyecto. Y, a Magdalena por convertirse en mi mayor inspiración para toda la vida.

Para terminar, agradezco a cada uno de mis profesores quienes nos guiaron con sus conocimientos durante esta etapa académica para convertirnos en personas de éxito. De manera especial al Ing. Diego Quisi por su acompañamiento como tutor y amigo quien con su colaboración e interés nos llenó de confianza para llevar a cabo este proyecto.

AGRADECIMIENTO

Agradezco a Dios por haberme dado la vida y la santísima Virgen María que ha sido mi compañera de batalla, ya lo decía Don Bosco, este gran santo italiano ¡Siempre he puesto toda mi confianza en María Auxiliadora ¡, y nunca he salido defraudado, por protegerme durante toda mi vida y darme las fuerzas para superar los obstáculos de mi vida.

A mi mamá Rosa, a mis hermanas Karolina, Johanna, a mi hermano Pedro y a mis abuelitos José y Juana por su cariño y apoyo incondicional, durante todo este proceso, por estar conmigo en todo momento, oraciones, consejos y palabras de aliento hicieron de mí una mejor persona y de una u otra forma me acompañan en todos mis sueños y metas.

Finalmente quiero expresar mi más grande y sincero agradecimiento a todos mis docentes durante mi proceso estudiantil y de manera especial al Ing. Diego Quisi, tutor en este proceso, quien con su dirección, conocimiento, enseñanza y colaboración permitió el desarrollo de esta tesis.

RESUMEN

En la infraestructura como código su objetivo es visualizar el despliegue en el proyecto mediante un proveedor en la nube como Azure brindando una solución horizontal. Para tener una infraestructura como código se puede usar varias herramientas que permiten escribir líneas de código, en este caso se utilizó Terraform el cual permite modificar, construir y llevar un control de la infraestructura. A través de la infraestructura, se brinda la posibilidad de tener una infraestructura versionada de debido a la flexibilidad y alto rendimiento. Para tener una infraestructura optima se implementó el servicio de Kubernetes el cual permite orquestar la infraestructura. Con Kubernetes se va a orquestar cada servicio en donde se va a agregar características de alta disponibilidad. Los entornos en los que se desliga los servicios son escalables y óptimos para producción. Los tiempos de respuesta del servidor son de 1840 ms, en cuanto a la desviación el tiempo es de 1622 ms. El resultado del error que presenta el servidor es de 5,45 %, estos valores se obtuvieron al realizar pruebas utilizando la herramienta JMeter.

Al desarrollar aplicaciones web se puede evidenciar que el proceso implica el uso de varias tecnologías entre las que tenemos librerías, frameworks tanto para el Fronted como Backend y los más importante la base de datos. La tecnología hoy en día evoluciona de una forma muy rápida con ello se ha podido integrar dentro de un conjunto para facilitar el proceso de desarrollo. En las combinaciones tenemos el Stack MEAN que integrar a Mongo DB como base de datos, Express como Framework para Backend, Angular como Framework para el Fronted y Node como entorno de ejecución para el servidor utilizado para desarrollar el comercio electrónico.

Para fortalecer el comercio electrónico se ha implementado un sistema recomendador con un porcentaje de predicción del 60% que podría mejorarse a medida que se realicen más compras.

ABSTRACT

In infrastructure as code, its objective is to visualize the deployment in the project through a cloud provider such as Azure, providing a horizontal solution. To have an infrastructure as code, you can use several tools that allow you to write lines of code, in this case Terraform was used, which allows you to modify, build and keep track of the infrastructure. Through the infrastructure, the possibility of having a versioned infrastructure is provided due to the flexibility and high performance. To have an optimal infrastructure, the Kubernetes service was implemented, which allows orchestrating the infrastructure. With Kubernetes, each service is going to be orchestrated where high availability features are going to be added. The environments in which the services are unlinked are scalable and optimal for production. Server response times are 1840ms, deviation time is 1622ms. The result of the error presented by the server is 5.45%, these values were obtained when testing using the JMeter tool.

When developing web applications, it can be seen that the process involves the use of various technologies, among which we have libraries, frameworks for both the Fronted and Backend, and the most important, the database. Today's technology evolves very quickly, so it has been possible to integrate it into a set to facilitate the development process. In the combinations we have the MEAN Stack that integrates Mongo DB as a database, Express as a Backend Framework, Angular as a Frontend Framework and Node as an execution environment for the server used to develop electronic commerce.

To strengthen electronic commerce, a recommender system has been implemented with a prediction percentage of 60% that could be improved as more purchases are made.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

INTRODUCCIÓN

Al realizar proyectos con mayor cantidad de procesos y tareas se debe realizar el aprovisionamiento de la infraestructura necesaria y tener las tecnologías actualizadas en la empresa brindando agilidad, trazabilidad, seguridad, economía entre otras opciones más. Se tiene varias herramientas y servicios para crear infraestructuras, las más utilizadas son Terraform y Ansible. Al incorporar estas tecnologías en una empresa se restructura su arquitectura a partir de Kubernetes incorporando por servicios para llevar una alta disponibilidad en rendimientos de las aplicaciones grandes que manejen las empresas. (Ekaterina Novoseltseva, 2020).

En la actualidad existen empresas que trabajan virtualizando servicios en diferentes contendores el cual al tener varios servidores y al existir sobre carga, uno de los servicios cae y deja de funcionar el sistema, por tal motivo que el servidor no posee alta disponibilidad. Al hacer uso de una infraestructura con buenas prácticas se puede tener servicios orquestados con Kubernetes, el cual mediante archivos manifiestos se pueden configurar cada servicio y asignar replicas para llevar una alta disponibilidad evitando sobrecarga en los servicios. En caso de que una Deployment de un servicio cayera, el máster de Kubernetes se encarga de levantar una nueva instancia sin interrumpir el rendimiento, así manteniendo al servicio en estado activo.

El objeto es completar el desarrollo de una aplicación web empleando para ello un conjunto de tecnologías MEAN, la aplicación se trata de un comercio electrónico de venta de Libros, que consta de dos partes: la teórica en la que se explica a detalle el uso de cada una de las tecnologías; la otra toda la parte práctica en la que se desarrolla lo referente a los requisitos, diseño, arquitectura e implementación.

CASOS DE ÉXITO

CASO DE ÉXITO INFRAESTRUCTURA COMO CÓDIGO

CASO DE ÉXITO DE PRISA

Prisa es una de las principales compañías de medios de comunicación que administra cientos de sistemas de información construidas con tecnologías nativas de la nube ayudando a escalar su plataforma.

Para Prisa el rendimiento es la clave, si tiene un problema de rendimiento tienen un impacto perjudicial en el rendimiento de negocio y a la percepción del consumir su marca. A pesar de que la empresa contrataba varias soluciones de monitorización, Prisa carecía de la visibilidad de extremo a extremo de sus aplicaciones, infraestructura y servicios subyacentes que necesitaban para garantizar el rendimiento de las aplicaciones. Un encargado de Prisa decía “Teníamos una idea de cómo funcionaba nuestra infraestructura. Pero no teníamos visión de los procesos de las tecnologías superpuestas. Conectar los puntos y entender las relaciones fue la parte difícil.” (Hopla-software, 2022).

Luego buscaron nuevas soluciones, mientras el grupo de Prisa trabajaba son su socio, Hopla Sofware, Prisa introdujo en Instana y obtuvo la respuesta que requerían para sus proyectos. Con la sencillez de despliegue de un agente de Instana, Prisa obtuvo resultados inmediatos, el desligue fue sencillo, presentaba un costo razonable e indexado al volumen de la infraestructura, para Prisa era importante administrar columnas con gran cantidad de tráfico. (Hopla-software, 2022).

Prisa agradeció el descubrimiento automático y continuo de Instana para romper la complejidad de su pila Cloud-Native.

CASO DE ÉXITO MEAN STACK

Let's Get Fitness es una aplicación web en forma de red social fitness en la que los usuarios comparten sus rutinas de entrenamiento, desarrollada como proyecto de titulación por parte de José Coalla de la Universidad Politécnica de Madrid , el autor menciona que se decidió utilizar este Stack por el uso del Lenguaje JavaScript ya que dentro de este Stack tanto de lado del cliente y del servidor las 4 herramientas están escritas JavaScript , lo que ayuda muchísimo al desarrollador porque no se tiene que especializar en diferentes lenguajes para la realización del proyecto lo que ayuda a ganar rapidez a la hora de desarrollar , además menciona que otro punto fuerte es la manipulación de datos , JavaScript utiliza JSON como formato de intercambio de datos para realizar el intercambio entre las capas que lo conforman , por lo que el cliente y el servidor pueden intercambiar información de una manera sencilla.

Al concluir su trabajo menciona que MEAN Stack es el punto de partida perfecto para cualquier desarrollador que quiera inicializarse en el desarrollo Web con ello ratificando muchas de las ventajas que tiene este Stack. (Coalla, 2018).

CASO DE ÉXITO GRAPHQL

GRAPHQL: UNA HISTORIA DE ÉXITO PARA PAYPAL CHECKOUT.

PayPal la empresa estadounidense de alcance mundial que brinda el servicio de pago en línea, mencionan que "GraphQL ha cambiado completamente la forma en que se piensa acerca de los datos, se recupera los datos y se crean aplicaciones".

Los productos de PayPal se distribuyen en muchas aplicaciones webs y móviles, al inicio sus Apis en REST eran bastante limpias, pequeñas y atómicas, todo iba muy bien, pero REST no consideraba las necesidades de las aplicaciones web, móviles y la de sus usuarios. Esto se notaba

en la transacción de Checkout, cada viaje de ida y vuelta le costaba 700 ms en tiempo de red, pero sin contar el tiempo de procesamiento de la solicitud en el servidor a medida que se iba agregado funcionalidades el rendimiento se veía afectado y los usuarios pagaba el precio ya que al agregar más campos la API se volvía más pesada.

Los desarrolladores sabían que el rendimiento era un problema. Sentían que no eran tan productivos como querían.

En ese momento, comenzaron a escuchar más y más sobre GraphQL. Pasaron una semana analizando de cerca GraphQL y quedaron impresionados y decidieron usar GraphQL

Mientras un equipo desarrollaba la interfaz de usuario, otro equipo construía la API en paralelo. Eso solo fue el inicio crearon Mobile SDK. Ahora están renovando sus productos insignia de PayPal Checkout sobre GraphQL y React, GraphQL le permitió tener: Rendimiento

Flexibilidad y Productividad a los desarrolladores. (ICHI.PRO).

CASO DE ÉXITO DE SISTEMAS RECOMENDADORES

CASO DE ÉXITO DE HENNEO

Henneo es uno de los mayores grupos empresariales involucrados en el periodismo, dueño de varias empresas periodísticas y webs. Este grupo agrego un mecanismo de recomendación para recomendar las noticas el cual este sistema era autónomo, escalable fidelizando a los usuarios. Este sistema fue el éxito el cual hacía que las ventas maximicen con un fit perfecto entre el cliente, ya que el recomendador al mostrar noticias llamativas y con posibles gustos del usuario, impactan llevando a cabo a la compra del producto.

Henneo cuenta con un recomendador de noticias aprovechando datos obtenidos por los usuarios visitantes. El proyecto de esta empresa se formó en 5 fases; 1) Investigación de problema de negocio, algoritmos, y diseño de arquitectura, 2) Investigación de problema de negocio,

algoritmos, y diseño de arquitectura, 3) Evaluación y optimización offline de prototipo funcional, 4) Puesta en producción: de prototipo funcional a producto mínimo viable (MVP) y 5) Mejora continua y exploración de nuevos algoritmos. (Benco, 2020)

Esta empresa realizo varias investigaciones antes de integrar el sistema recomendador. Tenían muchas preguntas en cuanto al usuario como, por ejemplo, "¿Puede un usuario, que lee un artículo en un momento dado, estar más interesado en ese momento de leer otros artículos de la misma temática?". Luego de varias investigaciones y estudios se decidieron implementar el sistema recomendador de noticas. Con la implementación fueron poco a poco atrayendo a clientes, esto les ayudo a ir mejorando el sistema recomendador. Al final optimizaron el recomendador y los resultados del sistema recomendador en producción empezó lanzar resultados positivos y mejoras en ventas.

Esta integración del recomendador de noticias les genero más ingresos y visitas para la empresa así generando una mejor utilidad en ventas.

OBJETIVOS

OBJETIVO GENERAL

- Diseñar e implementar un sistema recomendador de productos para un comercio electrónico utilizando GraphQL como middleware para el consumo de microservicios en infraestructuras basadas en código.

OBJETIVOS ESPECÍFICOS

- Revisar el estado del arte sobre infraestructuras basadas en código, GraphQL y sistemas recomendadores para comercios electrónicos.
- Definir una arquitectura basada en código para dar soporte al sistema recomendador y transaccional de comercio electrónico.
- Diseñar, desarrollar e implementar una aplicación desplegada sobre una plataforma en nube privada que permita la gestión del comercio.
- Desarrollar un sistema recomendador basado en perfiles de usuarios utilizando técnicas de inteligencia artificial.
- Diseñar y ejecutar un plan de experimentación que permita validar el sistema recomendador dentro del aplicativo del comercio electrónico.
- Implementar y desplegar un sistema de comercio electrónico sobre infraestructura basada en código
- Validar y probar la infraestructura basada en código en base a métricas de rendimiento y confiabilidad.

CAPÍTULO 1 MARCO TEORÍCO

1.1 INFRAESTRUCTURA COMO CÓDIGO

Se refiere a la automatización de infraestructura y la gestión de la infraestructura, incluyendo el hardware, los recursos virtuales, los sistemas de contenedores, los servicios, las plataformas y las topologías en lugar de hacerlo mediante procesos. Con la infraestructura se crean archivos de configuración que contiene configuraciones específicas que se necesitan la cual facilita la edición y la distribución de las configuraciones ya que esto me permite reducir el tiempo para desplegar arquitecturas. Las principales ventajas son: reducción de costos, disminución de la cantidad de errores, aumento de velocidad y mayor uniformidad de la infraestructura.

Las herramientas de aprovisionamiento son utilizadas para realizar instalaciones de la infraestructura de TI, redes, almacenamiento, servidores y otros recursos más. Existen varias herramientas que pueden ayudar a automatizar la infraestructura y las más usadas son las siguientes: Terraform, Chef, Saltstack, Ansible y Packer.

1.1.1 TERRAFORM

Es una herramienta de código abierto desarrollada por HashiCorp, sirve para la administrar, desarrollar, modificar y versionar infraestructura como código en archivos declarativos. Terraform está formado por recursos, providers, state y módulos. Esta herramienta se puede usar para desplegar infraestructura en cualquier proveedor de nube ya sea Azure, AWS, Digital Ocean GCP, etc. (Ekaterina Novoseltseva, 2020). Ver anexo 14.

Terraform presenta cinco componentes:

- Providers: es el que interpreta la interacciones con la API y es el encargado de exponer los recursos.
- Resource: conjunto de atributos configurables como lectura, creación, borrado y actualización.
- Data: permite recuperar o calcular datos para usar en configuraciones de Terraform, estos son esencialmente un subconjunto de recursos de solo lectura.
- Provisioner: es el que inicializa un recurso desde un determinado script local o remoto. Se usan para ejecutar scripts en máquinas remotas o locales.
- Terraform State: Almacena el estado de la gestión de la infraestructura y configuración. Se usa para mapear recursos, hacer seguimientos de metadatos. Este estado es almacenado por defecto de un fichero local llamado terraform.tfstate

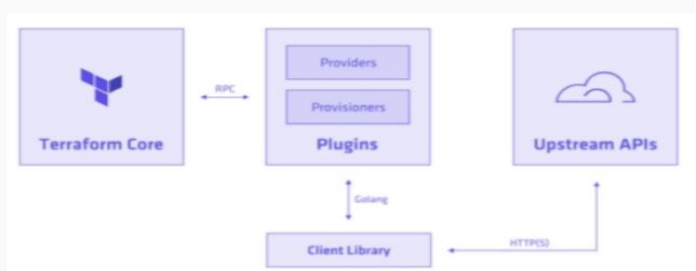


Fig. 1 Arquitectura Terraform (Aída Díaz, 2019).

1.1.2 CHEF

Es una herramienta agnóstica de la nube que permite describir y gestionar la infraestructura en forma de código y trabaja con varios proveedores como AWS, AZURE, Google Cloud y otros más. Esta tecnología fue escrita en Ruby que automatiza el procedimiento de aprovisionamiento. Presenta un modelo de trabajo que es maestro-cliente. (Aída Díaz, 2019).

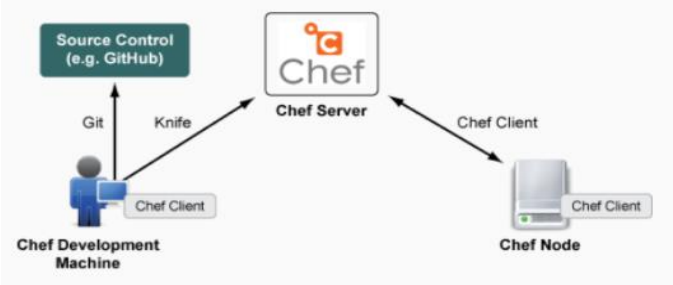


Fig. 2 Arquitectura Chef (Aída Díaz, 2019).

1.1.3 SALTSTACK

Es una herramienta de código abierto que es escalable para administrar decenas de miles de servidores y rápida en comunicación en milisegundos. Se utiliza para el control y la vigilancia automatizada de los sistemas de servidores.

1.1.4 ANSIBLE

Es una herramienta de software libre de automatización y configuración de infraestructura. El componente importante de Ansible son los módulos que sirven para controlar distintos recursos, ejemplo, el módulo user, este permite conducir los usuarios de un sistema.

1.1.5 PACKER

Es una herramienta para automatizar y estandarizar la creación de máquinas virtuales en diferentes formatos. Es muy útil cuando se quiere desplegar instancias de maquina completamente aprovisionadas y configuradas. El proceso de configuración es en una plantilla con formato JSON. Los ficheros JSON se divide en tres secciones:

- Variables: compartir credenciales de AWS como una variable.
- Constructores: descripción del entorno en el que se encuentre trabajando.
- Aprovisionadores: definición de procesos con configuraciones que se desean que la máquina virtual realice después que esté lista.

1.2 CONTENEDORES

Los contenedores son ambientes aislados dentro de un servidor web donde se empaqueta el código, las configuraciones de la aplicación junto con todas las dependencias que contenga dicha aplicación en bloques fáciles de usar. Algunos beneficios son: eficiencia operativa, productividad para los desarrolladores y control de versiones. Los contenedores se pueden ejecutar en cualquier lugar y por ello es necesario organizar, gestionar y monitorear. En este contexto es cuando interviene el papel fundamental de los Docker, crear imágenes en contenedores pues estos son esenciales para provisionar y remover contendores de acuerdo con la demanda de peticiones. (Saffirio, 2018).

1.2.1 DOCKER

Docker es una plataforma de software escrito en el lenguaje de programación Go para desarrollar y a su vez permite empaquetar, aprovisionar y desplegar cualquier aplicación como un contenedor ligero, independientemente del sistema operativo. Un contenedor con relación a la ejecución de una aplicación está conformado de sistemas de ficheros, procesos, memoria y todo con referencia al sistema operativo. Docker utiliza funciones del kernel de Linux tales como: los grupos de control y los espacios de nombre. Estos se despliegan con el propósito de ejecutar varios procesos y aplicaciones por separado aprovechando la infraestructura. En este sentido, con esta tecnología podemos tener varios contenedores con imágenes ya sea de base de datos, aplicaciones api rest que soporten gran carga de peticiones, agentes para test de carga y algunas tareas que el desarrollador realice llevando a producción las imágenes sin problemas. En estos escenarios de producción elaborados sobre arquitecturas distribuidas y orientadas a microservicios deben implementarse en clústeres de orquestador con el objetivo de reducir la complejidad de la implantación. (Devops Latam, 2021).

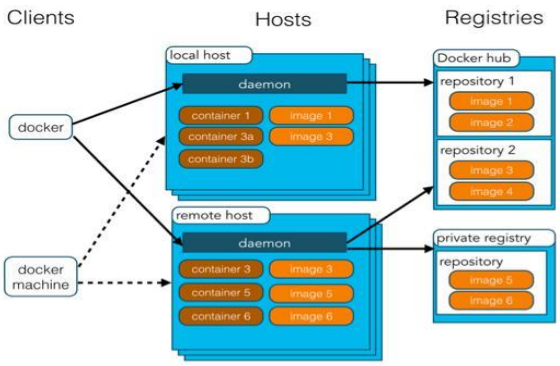


Fig. 3 Arquitectura Docker (Yanagishita, 2017).

El Daemon de Docker como servidor acepta peticiones y procesos de clientes ejecutándose en máquinas que se comunican a través de sockets o API RESTful. Según (Cuervo,2019) la arquitectura Docker se deduce al siguiente apartado:

> "La arquitectura Docker es una arquitectura cliente-servidor, dónde el cliente habla con el servidor mediante el API para poder gestionar el ciclo de vida de los contenedores y así poder construir, ejecutar y distribuir los contenedores."

Es decir, a través de esta arquitectura cliente-servidor se puede establecer una comunicación entre servidores gestionando el funcionamiento de cada uno de los procesos que se lleven a cabo en los contenedores.

1.2.2 FUNCIONAMIENTO BÁSICO

Docker usa el aislamiento de recursos en el Kernel del sistema operativo para ejecutar varios contenedores en la misma máquina virtual dependiendo de la memoria y procesador que disponga el servidor.

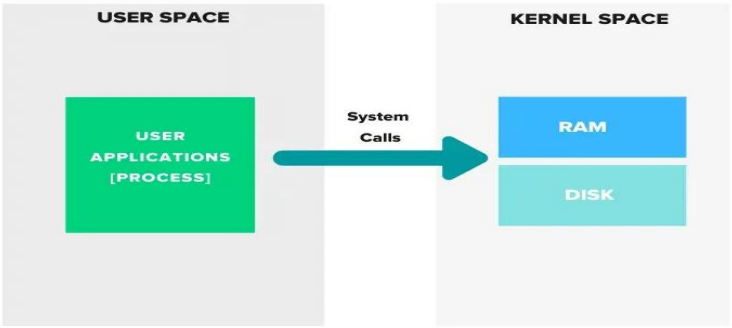


Fig. 4 Composición básica Docker. (Devops Latam, 2021).

- Userspace: traducido al español significa espacio de usuario y hace referencia a el código que se utiliza para desplegar programas de usuario como aplicaciones y procesos. (Devops Latam, 2021).
- Kernel Space: este es el centro del sistema operativo en donde se encuentra el código del kernel el cuál actúa con el hardware del sistema, almacenamiento, etc. (Devops Latam, 2021).

1.2.3 CARACTERÍSTICAS

Docker presenta características de velocidad por su optima configuración únicamente usando el sistema de archivos mínimos para que el funcionamiento de los servicios sea óptimo. Cada contenedor que se crea es ejecutado de manera aislada en el kernel de Linux, tales como groups y namespace para que estos autoricen qué contenedor se ejecute dentro de una sola instancia evitando sobrecarga. Docker presenta tres principales características: a) la portabilidad, hace referencia a que se puede desplegar en cualquier sistema sin tener que realizar configuraciones de la aplicación, pues todas las dependencias son empaquetadas en el contenedor; b) ligereza, aquí el contenedor solo abarca las características necesarias del sistema operativo, es decir no se virtualiza todo un sistema operativo y c) autosuficiencia, el contendor abarca solo librerías, archivos y configuraciones para que la aplicación se despliegue con todas sus funcionalidades necesarias.

1.2.4 DOCKERFILE

El Dockerfile es un archivo de texto simple de tipo manifiesto con una serie de comandos o instrucciones específicas para crear imágenes personalizadas con aplicaciones propias que serán utilizadas para un determinado propósito dentro de un sistema operativo optimo. Las imágenes se crearán mediante el comando docker build, la cual se ejecutará después de haber configurado el archivo. (Rodríguez, 2021).

1.2.4.1 SINTAXIS DOCKERFILE

Las principales instrucciones de dockerfile se pueden ver en el anexo 15.

1.2.5 DOCKER COMPOSE

Es una herramienta que permite definir y ejecutar aplicaciones dockerizadas y al mismo tiempo se puede crear distintos contenedores y en cada uno de ellos formar diferentes servicios. Ahora bien, teniendo en cuenta la arquitectura de microservicios, cada elemento se ejecutará en un contenedor separado, Docker compose simplifica la administración de todos estos contenedores en lugar de administrarlos individualmente. Para usar compose, primero se necesita un archivo para describir múltiples servicios del contenedor con sus respectivas relaciones y, por último, iniciar todos los contenedores de acuerdo con las configuraciones realizadas a través de comandos. (Maxtor, 2019).

Ver ejemplo de sintaxis en el anexo 16.

Los principales apartados del fichero se pude ver en el anexo 17.

1.3 KUBERNETES

Es una plataforma open source extensible de código abierto que se usa para administrar contenedores de Linux en entornos de nube privada, pública e híbrida. Esta herramienta organiza

procesos tales como: programar el despliegue, escalar, monitorizar los contenedores y administrar aplicaciones en grupos de nodos. Así mismo, Kubernetes tiene una ventaja principal, la cual consiste en programar y ejecutar contenedores en clústeres de máquinas virtuales. Esta tecnología orquesta la infraestructura de redes, cómputo y almacenamiento dando soporte a un número diverso de cargas de trabajo que incluyen en aplicaciones. (Cabrera, 2015).

1.3.1 CLÚSTER KUBERNETES

El clúster de Kubernetes es un conjunto de varias máquinas de nodos que ejecutan aplicaciones en contenedores de Linux. Los clústeres pueden contener hosts locales y nubes públicas. De tal manera, un clúster contiene dos procesos: a) plano de control y b) nodos Workers. El primero está encargado de mantener el estado deseado del clúster y el segundo, ejecuta las aplicaciones desarrolladas por el programador. (RedHat, 2020).

1.3.1.1 NODO MÁSTER

También se le conoce como plano de control, es un conjunto de procesos que tiene como objetivo controlar o administrar los nodos y los pods de kubernetes. El plano de control es tan preciso que puede detectar cuando un Worker ha caído. Este controla las aplicaciones que se ejecutan y las imágenes de contenedores que se utilizan. Además, recibe instrucciones del administrador y luego las transmite a las máquinas informáticas proporcionando tolerancia a errores y alta disponibilidad. Ejemplo, cuando necesite que el clúster genere un contenedor, el nodo maestro será quien designe o distribuya las tareas y al final iniciará un nuevo contenedor. Los componentes del máster son los siguientes:

- Kube-apiserver: es el responsable de procesar operaciones como la creación y configuración de pods y servicios, puede ejecutar varias instancias y equilibrar el tráfico entre esas instancias. También permite que distintas librerías y herramientas puedan comunicarse de manera más fácil. (Cabrera, 2015).
- Etcd: es una base de datos de alta disponibilidad que almacena el estado del sistema y, la información de servicios, pods, redes, etc. (Cabrera, 2015).
- kube-scheduler: se encarga de asignar los pods entre los nodos, lee los requisitos del pod, analiza el clúster y selecciona los nodos que son aceptables. Tiene una comunicación con el apiserver el cual busca pods no desplegados para luego ejecutarlos en el nodo que mejor cumpla los requerimientos. (Cabrera, 2015).
- kube-controller-manager: este servicio permite controlar el proceso de replicación que fueron definidos en las tareas de este proceso. (Cabrera, 2015).

1.3.1.2 NODO WORKER

Son encargados de ejecutar los pods que son asignados por el nodo máster, se llevan a cabo todos los componentes y servicios necesarios para desplegar las aplicaciones y balancear el tráfico entre servicios. Para ello se ejecutan los siguientes procesos:

- Kubelet: gestiona y comunica el uso de recursos, el estado en el que encuentra los nodos y pods que se estén ejecutando. (Cabrera, 2015).
- Kube-proxy: provee servicios de red y permite la comunicación interna y externa del clúster. Cada nodo ejecuta un proxy y un balanceador de cargar. (Cabrera, 2015).

1.3.2 OBJETOS KUBERNETES

Los objetos de Kubernetes son entidades dentro de la plataforma y están compuestas de varios objetos que Kubernetes utiliza para representar el estado del clúster como pods, servicios, deployments, etc.

1.3.2.1 PODS

Es la unidad más pequeña de un clúster, pueden existir varias replicas ya que el pod está compuesto por un conjunto de uno o varios contenedores y sus componentes se despliegan en un mismo host compartiendo recursos como red, almacenamiento y otros recursos más. Al crear un pod se le asigna un identificador único y se planifica en nodos donde permanece hasta su finalización. (Carrasco, 2020).

Los pods son efímeros ya que al ser destruidos se pierde toda la información. De tal modo que si se desarrolla aplicaciones persistentes se debe usar volúmenes para persistir datos de los pods.

1.3.2.2 DEPLOYMENT

Los deployments ayudan no solo a desplegar aplicaciones, estos se caracterizan como objetos de alto nivel. Nos permite definir funciones como el control de replicas, la escalabilidad de pods, despliegue automatizado y entre otras funciones más. Al describir el estado deseado en un objeto Deployment, el controlador se encarga de cambiar el estado actual al estado deseado. Un Deployment administra las actualizaciones de un conjunto de pods en las que se puede escalar y actualizar con límites de recursos optimizados.

1.3.2.3 SERVICE

Un servicio se utiliza para agrupar un conjunto de pods que trabajan juntos para proporcionar un servicio concreto. Mediante un servicio se puede acceder a las aplicaciones y al mismo tiempo ofrecen una red virtual mediante la cual podemos conectar usando un único nombre del sistema de nombres de dominios (DNS). El contenedor CoreDns del Control Plane, permite gestionar los nombres de los servicios. Un servicio garantiza que el tráfico de red se puede dirigir a un conjunto actual de pods que están designados para la carga de trabajo.

1.3.2.3.1 TIPOS DE SERVICIOS

Para exponer un Deployment se configuran mediante un servicio. Kubernetes presenta 4 tipos básicos de servicios de red cada uno con un diferenciador:

- ClusterIp: permite el acceso interno entre distintos servicios, solo se puede exponer aplicaciones como servicios de IP de clúster de la red privada. Los pods pueden alcanzar el servicio mediante el número de puerto de la aplicación que se despliegue.
- NodePort: este crea un puerto con una numeración alta en cada nodo, este puerto es abierto en la IP de cada nodo. Por defecto el puerto generado está en el rango de 30000 hasta 32767 y una dirección IP de clúster interna al servicio. Con esta configuración permite acceder al servicio desde una zona externa al clúster mediante IP del nodo utilizando el puerto del servicio.
- LoadBalancer: expone un servicio externamente mediante un balanceador de carga externo al clúster, este tipo solo esta soportado en servicios de nube pública. Crea un servicio NodePort y un ClusterIp indicándole al balanceador que se comunique con el nodePort

correspondiente, se puede asignar un puerto aleatorio o se puede dejar el puerto 80 que esta por defecto.

- ExternalName: añade un registro DNS CNAME al CoreDns de kubernetes. Esta configuración sirve para proporcionar un nombre DNS personalizado de un servicio externo al clúster. (Carrasco, 2020).

1.3.2.4 VOLÚMENES

Los archivos que están dentro de un contenedor son efímeros, por esta razón se hace uso de los volúmenes. Un volumen en Docker hacer referencia a un directorio en disco o en otro contendor, este provee controladores de volúmenes, pero la funcionalidad es limitada. Los volúmenes proporcionan un mecanismo de complemento para conectar contenedores que sean efímeros con almacenamiento de datos persistentes en otros lugares.

1.3.2.5 NAMESPACES

El Namespace es un clúster pequeño dentro de su clúster físico de Kubernetes. Se puede tener varios namespaces dentro de un único clúster de Kubernetes y todos están aislados lógicamente del uno al otro. Existen cuatro Namespaces que son los siguientes:

- Default: es el nombre por defecto que se da para aquellos que no especifican ningún espacio de nombre para lanzar aplicaciones.
- Kube-system: es el espacio para aquellos objetos creados por el propio sistema de Kubernetes.
- Kube-public: espacio reservado de nombres que se crea de manera automática y es legible por todos los usuarios.

- Kube-node-lease: administra objetos de asignación de arrendamiento con tiempo asociado a cada uno de los nodos del clúster. (Google LLC, 2021)

1.3.2.6 REPLICASET

La ReplicaSet es una función de Kubernetes que son usados para garantizar la disponibilidad de un número específico de réplicas de pods. El funcionamiento de una ReplicaSet se define mediante campos, incluyendo un selector que muestra como identificar a los pods que pueden adquirir un número de replicas, esto nos asegura que el pod siempre esté funcionando y totalmente disponible. Sus principales características son: el servicio siempre está disponible, es tolerante a errores y presenta escalabilidad totalmente dinámica.

1.3.2.7 DAEMONSET

Garantiza que todos los nodos del clúster ejecuten un número de copias de un pod y este pueda ser escalable. El Daemonset se puede usar para ejecutar almacenamiento en un clúster en cada nodo, también realiza un proceso de recolección de logs y ejecución de un proceso de monitorización.

1.3.3 K8S

K8S es una abreviación que se obtiene al remplazar las ocho letras “ubernete” por el número, fue creada por Google, escrita en lenguaje de programación Go, código abierto y probado en producción para manejar sistemas grandes. Es un orquestador a gran escala que administra aplicaciones en contenedores. Los desarrolladores de DevOps utilizan para implementar, escalar, mantener, programar y operar automáticamente varios contenedores de aplicaciones en grupos de nodos. Para realizar una orquestación el sistema se subdivide en distintos componentes, capaces

de proveer las distintas funcionalidades, ejemplo la migración de microservicios a Kubernetes. (Programador Clic, 2020).

1.3.4 K3S

Es una versión ligera de Kubernetes la cual no es accesible para crear un entorno de producción. Es fácil de instalar, la memoria se reduce a la mitad de todos los archivos binarios, su peso es menos de 40 MB.

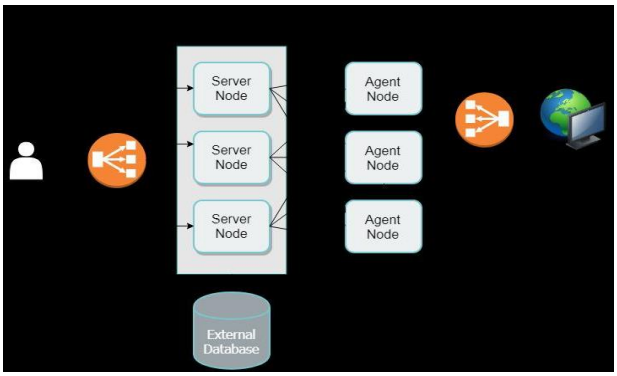


Fig. 5 Arquitectura K3s de alta disponibilidad (Programador Clic, 2020).

Un solo clúster de servidor puede cumplir con varios requisitos de uso, pero para entornos necesarios, se requiere el funcionamiento normal a largo plazo del cluster K3s e incluye dos o más nodos de servidor.

Características principales de K3s:

- Función alfa
- Complemento incorporado del proveedor de la buen
- Almacenamiento incorporado

Los escenarios principales de uso de K3s son:

- Edge
- Iot
- CI
- ARM

1.3.3 INGRESS

Ingress se utiliza para exponer rutas HTTP y HTTPS desde el exterior a servicios dentro del Clúster. Esta herramienta operar la séptima capa de red, para usar Ingress se necesita de un controlador. Este controlador es simplemente un Pod que se ejecutan en el Cluster y tiene como función asegurar el tráfico que ingres se administre de la manera que se haya definido.

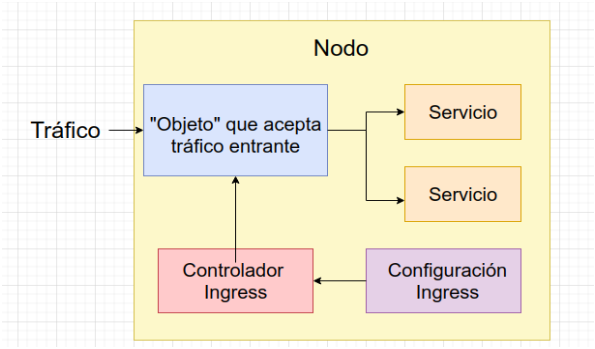


Fig. 6 Diagrama de funcionamiento de Ingress (Juan Pujol, 2020)

1.3.4 HERRAMIENTAS PARA PRUEBA DE CARGA Y PRUEBA DE STRESS

PRUEBAS DE CARGA

Las pruebas de carga hacen referencia a la carga que puede soportar un sistema a través de una carga determinada y poder verificar las causas que pueden condicionar su rendimiento. Estas pruebas de rendimiento son un tipo de pruebas no funcionales y forman parte del proceso de desarrollo de software midiendo el rendimiento de una aplicación o sitios web.

PRUEBAS DE STRESS

Las pruebas de Stress hacen referencia a realizar sobrecargas a un sistema sobrepasando los límites de sus características para verificar como y cuando fallará. El objetivo principal es encontrar el punto de quiebre. Se puede colocar una gran carga de base de datos, peticiones continuas al sistema o almacenar información sobrecargando la capacidad de memoria del sistema.

1.3.4.1 LOADVIEW

Esta herramienta proporciona servicios de pruebas de rendimiento bajo demanda, estas pruebas son ejecutadas en cuestión de minutos a través de un panel. LoadView realiza pruebas de infraestructura de internet, conexiones, conmutadores, etc. Esta herramienta proporciona varias funcionalidades para configurar y ejecutar pruebas de carga incluidos las siguientes tareas como: HTTP/s, página web, SOAP Web API y colecciones de cartero.

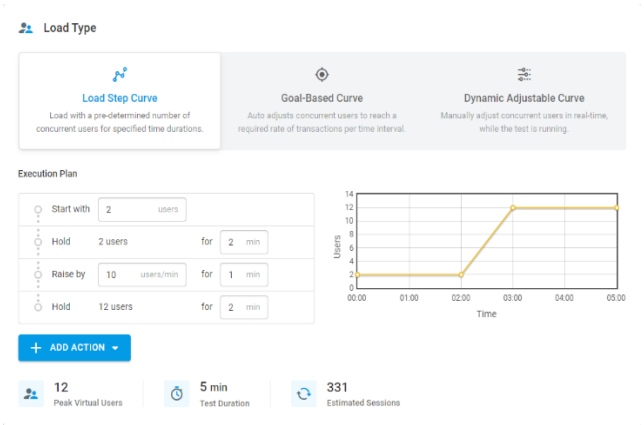


Fig. 7 Panel de resultados de rendimiento (Dotcom-monitor, 2020).

1.3.4.2 JMETER

Es una herramienta que mide el rendimiento de código abierto basado en Java de Apache Foundation. Esta se encarga de sacar las métricas de rendimiento para aplicaciones web,

aplicaciones FTP, servidores back-end, objetos java, Servicios web de tipo SOAP / REST, etc. JMeter tiene como función realizar pruebas funcionales, pruebas de rendimiento, pruebas de regresión y pruebas de estrés. Envía solicitudes a los servidores web simulando el comportamiento del navegador.

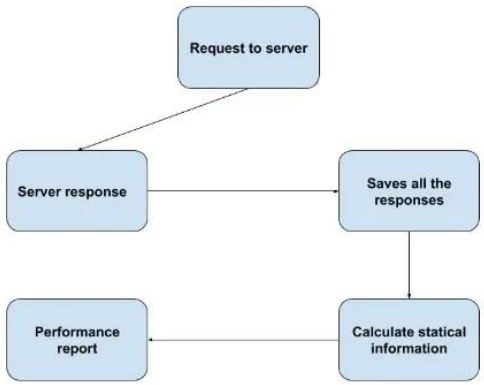


Fig. 8 Flujo de trabajo (Education-wiki, 2022).

1.4 ARQUITECTURAS DE SOFTWARE

Garantiza que todos los nodos del clúster ejecuten un número de copias de un pod y este pueda ser escalable. El Daemonset se puede usar para ejecutar almacenamiento en un clúster en cada nodo, también realiza un proceso de recolección de logs y ejecución de un proceso de monitorización.

1.5 GRAPHQL

1.5.1 INTRODUCCION

GraphQL es una especificación que engloba dos elementos principales:

1. **Lenguaje de consultas**: Permite a los clientes especificar qué datos necesita.
2. **Entorno de ejecución**: Permite la implementación de un servicio web que responda a las mismas consultas GraphQL a través de la especificación de un esquema tipado, donde se

enlistan los datos que el servicio web puede entregar y las operaciones para responder a las solicitudes de los clientes.

GraphQL es una especificación de implementación, no es un librería o marco de trabajo, que permite desarrollar servicios web, con la ventaja de que además de exponer los datos y las reglas para consultar provee de un lenguaje de consulta para consultar información.

```
query{
  getLibros{
    nombre
  }
}
```

Fig. 9 Ejemplos de una consulta con el lenguaje GraphQL (Elaboración propia)

El leguaje de GraphQL es muy expresivo porque permite declarar que información es la que se requiere, de la figura anterior se requiere los libros, pero únicamente su nombre, a diferencia de otras alternativas donde la respuesta para una consulta se encuentra definida y no pueden ser modificada por el cliente.

1.5.2 LENGUAJE DE CONSULTAS GRAPHQL

1.5.2.1 INTRODUCCION

GraphQL es un lenguaje de consultas fuertemente tipado para las APIs para ejecutar las consultas del lado del servidor, utiliza una definición de un sistema de tipos para los datos. El usuario mediante una única petición puede obtener los datos necesarios con una consulta personalizada.

Esta tecnología nació en el año 2012 como una alternativa para optimizar la aplicación móvil de Facebook, pero fue solo hasta el año 2015 que fue lanzada públicamente proporcionado una sintaxis flexible para describir los requisitos de datos e interacciones a la hora de crear aplicativos.

En la fig.11 surge la interrogante de cómo es que GraphQL entiende la consulta (query) y brinda una respuesta adecuada, el entorno de ejecución juega un papel muy importante juntamente con

un lenguaje de programación backend y librerías propias para GraphQL que permiten definir un esquema que se detallará a continuación.

1.5.2.2 ESTRUCTURA

1.5.2.2.1 ESQUEMA (SCHEMA)

El esquema permite especificar qué datos se expondrán, sus tipos, sus relaciones, su formato y la forma para dar respuesta a las consultas.

```
type Libro {
  id: ID!
  nombre: String!
  autor: String
}

type Query {
  getLibros(pagina: Int, limite: Int = 1): [Libro]
  getLibro(id: ID!): Libro
}

type Mutation {
  insertarLibro(input: LibroInput): Libro
  actualizarLibro(id: ID!, input: LibroInput): Libro
  eliminarLibro(id: ID!): Alerta
}
```

Fig. 10 Ejemplo de un esquema GraphQL (Elaboración propia)

Además de la definición de tipos que se tiene dentro del esquema, este necesita de una serie de resolvers que son operaciones que dan respuestas a las consultas realizadas, para ello utilizando información de un base de datos, un arreglo o de un servicio web.

- **IMPORTANCIA**

El esquema es el centro de la implementación del servidor de GraphQL sin el mismo no se puede construir la API.

- **CONVENCIONES**

El lenguaje es flexible, no impone pautas, es recomendable seguir unas convenciones.

1. Campos: Utilizar la estructura camelCase : miCampo
2. Tipos: Utilizar la estructura PascalCase: MiTipo

3. Enums: Utilizar la estructura PascalCase para el nombre:MiEnum, para sus valores en ALL_CAPS (VALOR_ENUM1,VALOR_ENUM2)

- **FORMAS DE REPRESENTACION**

Una de las mayores ventajas que proporciona GraphQL es que brinda una definición clara de los datos disponibles en su API a través del esquema. El esquema permite que GraphQL tenga una variedad de herramienta útiles para los desarrolladores, incluidos exploradores de consultas de autocompletado como **GraphiQL** y **GraphQL playground**, integraciones de un entono de desarrollo integrado (IDE), herramientas de validación de consultas, etc. Pero para poder usar primero debe adquirir el esquema de la API y convertirlo en formato apropiado para la herramienta. (Stubailo, 2021), se puede representar utilizando:

1. EL lenguaje de definición de esquemas GraphQL o SDL (uso de buildSchema).
2. El Objeto GraphQLSchema.

1.5.2.2.2 CAMPOS

Los **campos** de cada objeto se definen por un tipo especial de dato.

Son los que se solicitan mediante una estructura de petición en la cual se especifican campos contenidos dentro de un objeto.

```
{
  getLibros {
    id
    nombre
    autor
  }
}
```

Fig. 11 Campos de una consulta GraphQL (Elaboración propia)

1.5.2.2.3 SCHEMA ROOT TYPE

El esquema contiene un tipo de raíz, que engloba todos los tipos en un simple tipo que tiene:

- Query
- Mutation
- Subscription

1.5.2.2.4 FRAGMENT

Pieza lógica que se puede compartir entre múltiples consultas y mutaciones.

1.5.3.2.5 TIPOS DE DATOS

Los **tipos de datos** que pueden ser:

1. Tipos de escalares
2. Tipos de objetos
3. Tipos de Enumeración

- **TIPOS DE ESCALARES (SCALAR TYPES)**

Son datos primitivos que pueden almacenar un solo valor (Int, Float, String, Boolean, ID).

SINTAXIS DE UN ESCALAR

[nombreDeLaPropiedad]: [tipoDeDato]

STRING	INT	BOOLEAN
nombre: String	edad: Int	activo: Boolean

Tabla 1 Sintaxis de un escalar type (Elaboración propia)

- **TIPOS DE OBJETO (OBJECTS TYPES)**

Objetos personalizados, debido a que un escalar por sí mismo no es suficiente para abstraer algunos modelos de datos que tienen complejidad.

Se utiliza la palabra reservada **type**

SINTAXIS DE UN OBJECT TYPE

```
type Libro {
  id: ID!
  nombre: String!
  autor: String
}
```

Fig. 12 Sintaxis de un object type (Elaboración propia)

- **TIPOS DE ENUMERACION (ENUM TYPES)**

Son tipos de datos similares a un tipo escalar, muy utilizados para trabajar con un listado de valores predefinidos.

Se utiliza la palabra reservada **enum.**

SINTAXIS DE UN ENUM

```
type Libro {
  id: ID!
  nombre: String!
  autor: String
  categoria: Categoria
}

enum Categoria {
  VIDA_DE_SANTOS
  NOVIAZGO
  MATRIMONIO
}
```

Fig. 13 Sintaxis de un enum type (Elaboración propia)

1.5.2.2.6 MODIFICADORES DE TIPOS (MODIFIER TYPE)

Se emplean para modificar el comportamiento de un campo, se tiene 2 tipos:

a) !: Indica que el valor es obligatorio

b) []: Lista de valores con un elemento o más.

1.5.2.2.7 INTERFACES

Un interfaz es una definición abstracta de atributos comunes, es útil para crear objetos de diferentes tipos.

```
interface Persona {
  nombre: String!
  email: String
  edad: Int!
}

type Autor implements Persona {
  nombre: String!
  email: String
  edad: Int!
  libro: String
}
```

Fig. 14 Ejemplo de una interface Persona (Elaboración propia)

1.5.2.2.8 ARGUMENTOS

Son información (clave – valor) que se relaciona con un campo de consulta, permite filtrar datos en una consulta.

```
query{
  getLibro(id:"1"){
    id
    nombre
    autor
  }
}
```

Fig. 15 Argumento en una consulta GraphQL (Elaboración propia)

1.5.2.2.9 VARIABLES

Dato de entrada para una consulta, son valores dinámicos que se puede asignar a los argumentos.

ESTRUCTURA

$ id – el símbolo $ indica que id es una variable.

```
mutation ($input: LibroInput) {
  insertarLibro(input: $input) {
    id
    nombre
    categoria
  }
}
```

QUERY VARIABLES

```
{
  "input": {
    "nombre": "Iglesia Católica Dulce Hogar",
    "autor": "Wilson Tamayo"
  }
}
```

Fig. 16 Variables en una consulta GraphQL (Elaboración propia)

1.5.2.2.10 TIPO DE ENTRADA (INPUT TYPE)

Se emplea la palabra reservada **input** que permite trasmitir argumentos complejos, permitiendo establecer un nombre común para los mismos. (Porcello & Banks, 2018).

```
input LibroInput {
  nombre: String!
  autor: String
  categoria: Categoria = NOVIAZGO
}
```

Fig. 17 Tipo de entrada input type (Elaboración propia)

1.5.2.2.11 TIPOS DE RAIZ (ROOT TYPES)

Son puntos de entrada que sirven para comunicar al cliente con el servidor a través de ellos. GraphQL puede gestionar la información mediante la función conocida como CRUD que es el acrónimo de “Crear, Leer, Actualizar y Borrar”, que especifica tres tipos de operaciones:

a) **Query** = Lectura de datos, punto de entrada para cada solicitud del lado del cliente.

b) **Mutation** = Permite manipular la información (Crear, actualizar, eliminar), devuelven un valor igual que la consulta, por ello se puede consultar datos en función del valor de retorno de una mutación.

c) **Subscription** = Permite realizar actualizaciones en tiempo real. (Porcello & Banks, 2018).

```
type Query {
  getLibros(pagina: Int, limite: Int = 1): [Libro]
  getLibro(id: ID!): Libro
}

type Mutation {
  insertarLibro(input: LibroInput): Libro
  actualizarLibro(id: ID!, input: LibroInput): Libro
  eliminarLibro(id: ID!): Alerta
}
```

Fig. 18 Type Query y Type Mutation (Elaboración propia)

1.5.2.2.12 COMENTARIOS

GraphQL permite documentar las APIs añadiendo características en el esquema, para ello admite Markdown.

Tipos:

a) 1 línea => “”

b) Mas de 1 línea => “““”””

```
type Query {
  "Consulta para recuperar el listado de los libros"
  getLibros(pagina: Int, limite: Int = 1): [Libro]
  "Consulta para recupar los detalles de un libro específico, necesario proporcionar el ID"
  getLibro(id: ID!): Libro
}
```

Fig. 19 Comentario de 1 línea (Elaboración propia)

1.5.2.4 GRAPHIQL

Es una herramienta para el navegador para poder escribir, validar y probar consultas GraphQL. GraphiQL es posible porque GraphQL se pensó para ofrecer una experiencia optimizada de desarrollo. (Gatsby, 2022).

Componentes:

- 1) Sección para escribir Queries y Mutations.
- 2) **QUERY VARIABLES** sección para el paso de variables.
- 3) Visualizar los resultados que devuelve el servidor en formato JSON.

- 4) Documentación de la **API.**

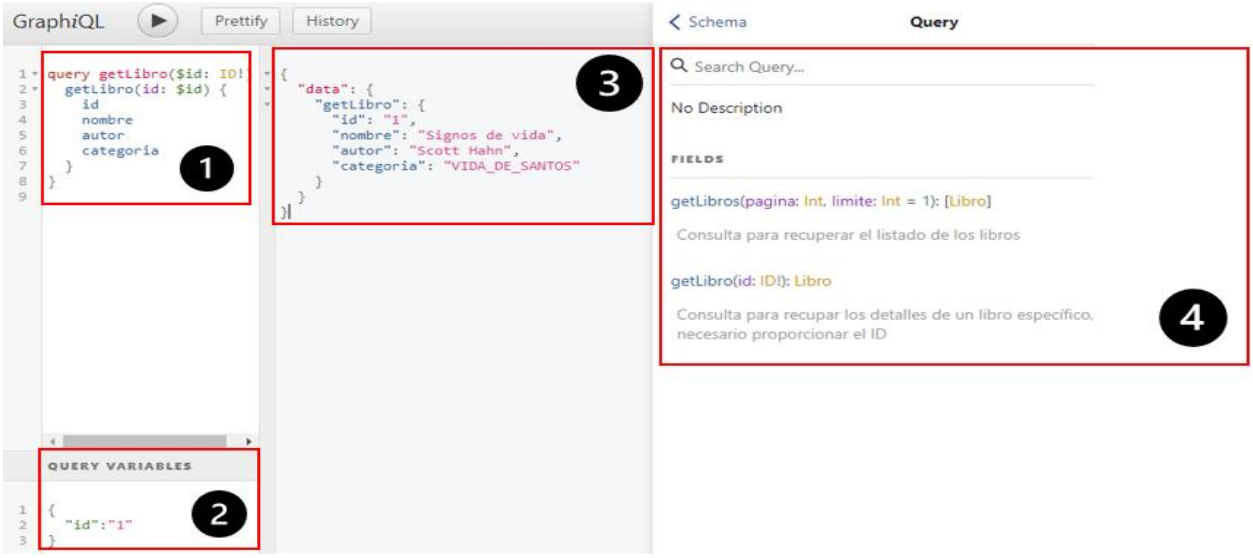


Fig. 20 Componentes de GraphiQL (Elaboración propia)

1.5.2.5 GRAPHQL PLAYGROUND

Es un IDE gráfico, interactivo para el navegador basado en graphiql creado por Prisma y mantenido por la graphql Foundation.

Según (Valdez, 2020) GraphQL Playground utiliza componentes de graphiql debajo y tiene características adiciones:

- Documentación interactiva
- Recarga automática del esquema(schema)
- Soporte para suscripciones (subscriptions)
- Historia de consultas(queries)
- Configuración de http headers
- Tabs (pestañas)
- IDE configurable.

Componentes:

Fig. 21 Componentes de GraphQL Playground (Elaboración propia)

1.5.2.6 APOLLO GRAPHQL

La plataforma Apollo GraphQL ayuda a crear, consultar, administrar, escalar y brinda herramientas para poder montar en su misma plataforma el servicio GraphQL que utilizada algunas herramientas que contribuyen al open source, además ayudan a solventar los problemas de resolvers personalizados y la modularización. (Apollo-GraphQL,2022).

1.5.2.6.1 APOLLO-SERVER

Alternativa para no usar Express, permite hacer más rápido y directo el servicio. (Apollo-GraphQL,2022).

1.5.2.6.2 GRAPHQL-TOOLS

Brindan una estructura sobre cómo construir un esquema GraphQL y resolvers en JavaScript. (The-Guild, 2022).

1.5.2.6.3 APOLLO-CLIENT

El cliente de Apollo tiene una herramienta que facilita el desarrollo Frontend al actuar como un cliente HTTP que se conecta a la API y brinda almacenamiento en cache, manejo de errores e incluso la capacidad de administrar el estado. (Apollo-GraphQL,2022).

Existen tanto para el lado del servidor y cliente

1.5.2.7 VALIDACION

Se emplea un conjunto de reglas de validación para verificar que el contenido de una consulta sea correcto, para ello empleando un sistema de tipos, se puede tener los siguientes errores

- **CAMPO INEXISTENTE**

Los campos que se definen en una consulta deben existir en la definición del esquema de lo contrario se mostrará un error.

- **SIN CAMPOS PARA DEVOLVER**

Sucede cuando de un tipo de objeto no se selecciona ningún campo.

1.5.2.8 EJECUCION – EXECUTION

Para la ejecución de la operación una parte vital es la validación de los campos, para mostrar los resultados de la petición el servidor ejecuta estas operaciones por medio de resolvers. (GraphQLFundation,2021).

- **RESOLUTORES (RESOLVERS)**

Funciones para obtener los datos de los campos definidos en el esquema, cada función tiene su función de correspondencia.

La declaración del resolver en sí es diferente en cada lenguaje, debido a que representa la lógica para la adquisición de los datos.

Se puede también emplear resolvers asíncronos cuando la carga de datos se hace desde una base o una API. (Rodríguez,2020).

- **RESPUESTA**

La respuesta es mostrada en un formato clave-valor que tiene un formato JSON. (GraphQLFundation,2021).

1.6 GRAPHL VS REST

Según (Valdez, 2020) se tiene las siguientes diferencias:

REST	GRAPHQL
Una URL tiene acceso a un recurso por ello se deben hacer consultas a varios endpoints (url) para obtener la información que se está buscando.	Una URL brinda acceso a toda la información.
No se puede elegir que se va a recibir en el JSON de la respuesta, se descarga información innecesaria.	Los esquemas (schemas) sirven como un contrato entre el cliente y el servidor, con ello se puede escoger que campos se requieren.
Versionado existen múltiples versiones ocasionado problemas de incompatibilidad con versiones anteriores a la de los clientes.	No existe versionado.
No auto documenta, para documentar se debe utilizar herramientas externas como SWAGER o POSTAM.	Auto documentado aprovechando su naturaleza fuertemente tipada al definir los tipos en el esquema.
Almacenamiento en caché, ya que esta implementado mediante HTTP.	No tiene almacenamiento en caché, dejando la responsabilidad al cliente, que es solventado con Apollo Client.
	El Frontend y el Backend pueden trabajar completamente independiente.

Tabla 2 Diferencias entre REST y GraphQL

1.7 SISTEMAS RECOMENDADORES

Es una herramienta importante que ayuda a los usuarios a conocer la información que necesita de acuerdo con su interés. Entre los ámbitos más extendidos se encuentran las recomendaciones de productos de tiendas online, videos, música, libros también se encuentra recomendaciones de

perfiles de las redes sociales. El funcionamiento de un sistema recomendador analizar y procesar información de todo el historial del usuario ya sea en compras, en contenidos y en algunas opciones que el usuario realice para al final predecir que producto puede ser interesante para el usuarios o empresa en la que se use un sistema recomendador. Existen dos tipos de sistema recomendadores que son: los filtros colaborativos y los filtros basados en contenido. (Cleverdata, 2021).

1.7.1 FILTROS COLABORATIVOS

También se le conoce como filtro de usuario basado en la lógica de las características del usuario donde esta se aplica a los sistemas de recomendación para optimizar su funcionamiento. Los filtros colaborativos funcionan de forma especial para sacar predicciones automáticas sobre el interés. Con la interacción del usuario y los datos que generan ayudan a crear una especie de inteligencia colectiva que ayuda a incrementar la calidad y ser optimo en las recomendaciones. Existen diferentes métodos que se usan para encontrar usuarios parecidos y el método que se usará estará basado en la función de correlación de Pearson. (Cleverdata, 2021).

Las ventajas que presenta el filtro colaborativo son:

- Tiene presente el puntaje de otros usuarios.
- No tiene la necesidad de extraer o realizar estudios de información del elemento recomendado.
- Son adaptables al interés del usuario.

En Python se puede usar una versión de filtrado colaborativo con la librería de Pandas y varias librerías para funciones matemáticas.

1.7.1.1 FUNCIÓN DE CORRELACIÓN DE PEARSON

El coeficiente de correlación de Pearson es un índice que mide el grado de relación entre distintas variables siempre y cuando sean cuantitativas. Cuando la asociación entre elementos no es lineal el coeficiente no se encuentra representado adecuadamente. El rango de valores que puede tomar es de +1 a -1, el valor de 0 indica que no existe asociación entre variables. Las características de correlación de Pearson que se deben cumplir son las siguientes:

- En la escala de medida debe ser una escala de intervalo.
- Las variables deben de estar distribuidas.
- La asociación debe ser lineal.
- No debe haber valores distintos en los datos.

Para realizar el cálculo del coeficiente de relación se tiene la siguiente fórmula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})\,(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

La relación entre variables son las siguientes:

- Si el coeficiente es igual a 1, esto significa que las variables tienen correlación positiva correcta.
- Si el coeficiente es igual a -1, esto significa que las variables tienen una correlación negativa correcta.
- Cuando el valor de coeficiencia esté más cerca de 1 la correlación entre variables es mayor, en caso de que el valor este más cercano a 0 las variables no tienen correlación lineal no es tan fuerte.

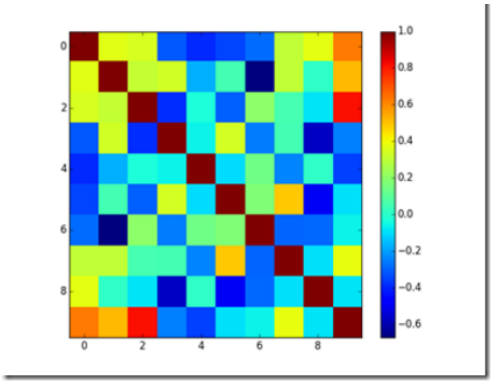


Fig. 22 Ejemplo de matriz de correlación (Carvajal, 2018).

1.7.1.2 VECINOS MÁS CERCANOS

En términos de clasificación, la técnica de los k vecinos más próximos es considerada como simple, popular y eficiente. Sus características más relevantes se basan en que no son paramétrico, es decir, no toma en cuenta ningún dato con respecto a la distribución de la información y no realiza conjeturas sobre ella. Además, la fase de entrenamiento no se visualiza con facilidad pues no demuestra generalizaciones. En esta sección se disponen los k grupos que se manifiestan por un atributo de los elementos que son parte de ellos. (Carvajal, 2018).

El algoritmo de los vecinos más cercanos funciona de la siguiente manera:

- Calcula la distancia entre el ítem a clasificar.
- Selecciona los “K” los elementos más cercanos.
- Realizar una votación de mayoría entre k puntos.

Este método busca en las observaciones más cercanas a la que trata de predecir y clasifica el punto de interés basado en la gran cantidad de datos que le rodean. El algoritmo presenta dos características:

- Supervisado: quiere decir que se tiene etiquetado nuestro conjunto de datos de entrenamiento con resultado o clase dada a una fila de datos
- Basado en instancia: el algoritmo aprende o memoriza las instancias de entrenamiento que son usadas como base de conocimiento para la fase de predicción.

1.7.2 FILTROS BASADOS EN CONTENIDO

Los filtros basados en contenido tienen como base de la predicción, es decir, utiliza las características del artículo como la marca, precio, calificaciones, tamaño, etc. para hacer las recomendaciones. Este trabaja con los datos que el usuario proporciona ya sea explícita o implícitamente, cada vez que el usuario proporciona más entradas sobre recomendaciones, el motor se vuelve más preciso. Estos recomendadores utilizan métricas de similitud través de algoritmitos de clasificación, clustering o análisis de texto. (Núñes, 2012).

Las ventajas que tiene los sistemas basados en contenido son:

- Recomendación por contenido y no necesita datos u opiniones de otros usuarios.
- El sistema genera explicaciones sobre la recomendación.
- En modelado está presente en las características y no necesitan proveerlas de otros usuarios.

1.8 HERRAMIENTAS TECNOLÓGICAS

El presente trabajo busca la Implementación de un sistema recomendador para un comercio electrónico utilizando GraphQL como middleware para el consumo de microservicios en infraestructura basadas en código, a continuación, se detalla las herramientas a emplearse.

1.8.1 MEAN STACK

El término "**MEAN STACK**" se define como un conjunto de tecnologías basadas en **JavaScript**, que se emplean para el desarrollo de aplicaciones y páginas webs complejas.

MEAN es el acrónimo formado por las tecnologías que los integran: **MONGODB, EXPRESS, ANGULAR Y NODE.JS** a continuación se explicará cada una de ellas. (Alarcón, 2015).

1.8.1.1 NODE.JS

Entorno de ejecución de aplicaciones multiplataformas y de código abierto (Open Source) creado en el motor de JavaScript V8 de Chrome que brinda rendimiento y escalabilidad.

Al ser de código abierto permite crear aplicaciones web sin tener que pagar por ello, con muy alto rendimiento, sin importar el sistema operativo, pero únicamente utilizando como lenguaje de programación JavaScript. (Alarcón, 2015).

1.8.1.2 EXPRESS

Node.js facilita crear la lógica de las aplicaciones debido que contiene un módulo de protocolo HTTP, pero es muy complejo y costoso tenerlo que hacerlo a bajo nivel.

Para solventar este problema nación **Express**, permite crear aplicaciones web de una forma más fácil, este framework está escrito en JavaScript, su objetivo principal es ofrecer soporte para las necesidades más primordiales (gestión de peticiones y respuestas, rutas, vistas, etc.) (Alarcón, 2015).

1.8.1.3 ANGULAR

Es un framework de código abierto creado y soportado por Google que facilita la creación y programación de aplicaciones web de una sola página, las conocidas webs **SPA** (Single Page Application). (Alarcón, 2015).

Utiliza el patrón **MVC** (Modelo, Vista, Controlador) asegurando un desarrollo rápido.

1.8.1.4 MONGODB

Tradicionalmente se ha utilizado bases de datos relacionales que emplea SQL (Structured Query Language) un lenguaje estructurado para gestionar la información, sin embargo, los tipos de información que requieren las aplicaciones WEB demandan más información, menor coherencia y sobe todo escalabilidad, como solución surgen las bases de datos **NoSQL.**

MongoDB es un gestor de datos **NoSQL** que nación a finales del año 2007, pero solo fue hasta 2009 que decidieron liberarla como Open Source, es una base de datos basada en documentos en formato JSON. (Alarcón, 2015).

JSON no forma parte de **MEAN,** pero es una pieza fundamental que significa **JavaScript Simple Object Notation,** que permite representar objetos en forma de código JavaScript, que es un formato ligero y muy fácil de leer y procesar, allí su importancia.

JSON es el pegamento de todas las capas porque es el formato para transferir los datos entre todos los niveles de una aplicación: navegador, servidor web y servidor de datos. (Alarcón, 2015).

1.8.1.5 OTROS ELEMENTOS

Metalenguajes: Hace referencia a lenguajes encima de otros lenguajes para trabajar fácilmente por ejemplo **Sass** o **Less** para escribir hojas de estilos más potentes y flexibles. De igual manera ocurre con JavaScript, metalenguajes como **TypeScript** que ofrece capacidades que no brinda JavaScript nativamente. (Alarcón, 2015).

Npm (Gestor de librerías).

Al desarrollar con Node.js se usa el gestor oficial para este entorno conocido como **NPM** (Node Package Manager.).

Nodemailer

Módulo para el envío automático de emails.

Python

Python es un lenguaje de programación orientado a objetos para el desarrollo rápido de aplicaciones. Un beneficio importante es la librería estándar como el intérprete están disponibles gratuitamente.

Flask

Flask es un micro framework para Python.

Postman

Es una aplicación que permite hacer pruebas a las API REST. Se puede escribir, monitorizar y simularla pruebas. Esta herramienta tiene muchas funcionalidades para gestionar el ciclo de vida de una API.

JWT

Es un estándar abierto basado en JSON, sirve para crear tokens de acceso que brindan la opción de poder propagar identidad y privilegios.

Se compone de 3 partes:

- un encabezado o header: Identifica el algoritmo usado para generar la firma.
- un contenido o payload: Contiene la información de los privilegios, datos personales, etc.
- y una firma o signature que se compone:
    - key: La palabra secreta para comprobar su validez.
    - unsignedToken
    - asignature

1.8.2 HERRAMIENTAS DE TESTEO

1.8.2.1 KARMA

Es un test-runner desarrollado por el equipo de Angular, es un módulo que permite automatizar tareas, es el encargado de ejecutar las pruebas. (Pérez, 2020).

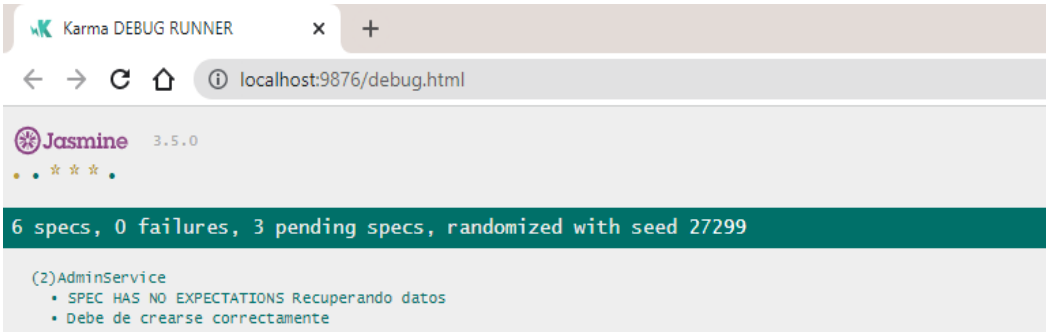


Fig. 23 Karma en ejecución (Elaboración propia)

1.8.2.2 JASMINE

Es un framework para testing integrando en angular, permite hacer pruebas unitarias y pruebas de integración, tecnología que no requiere un **DOM** (Document Object Model – Modelo de objetos de Documento) para hacer las pruebas y sus sintaxis es muy fácil de entender. (Pérez, 2020).

```
import { TestBed } from '@angular/core/testing';
import { AdminService } from './admin.service';
import { ApolloTestingController, ApolloTestingModule } from 'apollo-angular/testing';

describe('(2)AdminService', () => {
  let service: AdminService;
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [ApolloTestingModule],
    });
    service = TestBed.inject(AdminService);
  });

  it('Debe de crearse correctamente', () => {
    expect(service).toBeTruthy();
  });
```

Fig. 24 Sintaxis de Jasmine (Elaboración propia)

1.8.3 STRIPE

Stripe es una infraestructura de pagos para internet que proporciona una pasarela de pago que se puede integrar con un sitio web, los pagos a través de esta plataformas se pueden realizar con tarjetas de crédito o tarjetas de débito con rapidez y seguridad lo que lo diferencia de otras

plataformas es que los datos de la tarjeta de crédito o débito que se ingresan a través de un formulario se envían a los servidores de Stripe, lo que garantiza que el proceso sea seguro debido a que dichos datos sensibles no se guardan en la base local del servidor. Con Stripe no hace falta salir de la página para realizar el pago.

1.9 METODOLOGÍAS DE DESARROLLO DE SOFTWARE

1.9.1 INTRODUCCIÓN

Scrum es un marco de trabajo dentro de las metodologías ágiles para el desarrollo y mantenimiento de productos complejos que tiene un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo. (Schwaber & Sutherland, 2013).

1.9.2 MARCO DE TRABAJO SCRUM

La metodología Scrum se divide en las siguientes fases:

- El qué y el quién: dentro de los miembros se identifica los roles y las responsabilidades de estos.
- El cuándo y el dónde: representan el sprint.
- El cómo y el porqué: hace referencia al equipo Scrum y las herramientas que van a utilizar.

Roles SCRUM qué y el quién

Tiene tres roles:

- El Dueño /*Product Owner(PO)* es el responsable de filtrar todas las cosas que debería tener el proyecto para de esta manera poder priorizar las tareas en el backlog (requerimientos), que nos indica que es lo que se debe hacer y que es lo que no se debe hacer, para que de esta formar el equipo de desarrollo pueda entregar algo funcional.
- El Scrum Master es una persona experta en Scrum, su función principal es que el equipo trabaje según las guías Scrum, es decir facilita el trabajo.

- El Equipo de desarrollo /*Development Team* es el equipo de profesionales (Programadores de Backend, Frontend, etc.) que se encargan de crear el producto.

Dentro de esta metodología el sprint es su característica principal pudiendo tener un tiempo mínimo de una semana y un máximo de 4 semanas, tiene como objetivo entregar algo funcional y cuenta con los siguientes eventos.

- Planeación del Sprint (Sprint Planning)
- Reunión del equipo de Scrum (Daily Scrum Meeting): reuniones con un bloque de tiempo de 15 minutos para que el equipo de desarrollo sincronice sus actividades.
- Revisión del Sprint (Sprint Review): el equipo Scrum y lo interesados se reúnen para mostrar lo que se ha completado el desarrollo del software.
- Retrospectiva de Sprint (Sprint Retrospective): el equipo Scrum se reúne con la finalidad inspeccionarse a sí mismos y poder crear un plan de mejoras. (Schwaber & Sutherland, 2013).

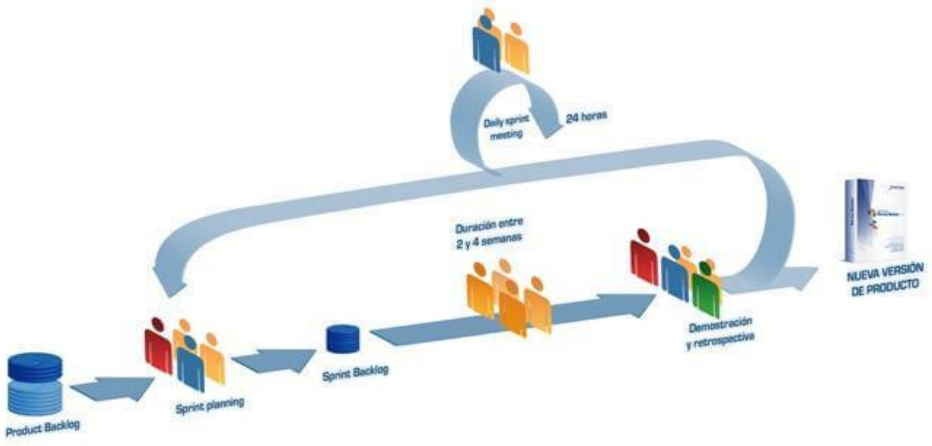


Fig. 25 Roles de Scrum (Softeng)

CAPÍTULO 2 ANÁLISIS, REQUERIMIENTOS Y PLANTEAMIENTO DEL PROBLEMA

2.1 PROBLEMA DE ESTUDIO

(Banchoff, 2019) presenta "Infraestructura como Código" con el objetivo de visualizar la aplicabilidad e impacto de la infraestructura como código en la calidad de los proyectos de innovación en donde surge la interrogante de: ¿Cómo ser más productivo? a partir de ello ¿Cómo empezar a crear una aplicación? y ¿De qué forma la misma va a consumir los microservicios? para la infraestructura dentro de un ambiente de desarrollo sería mediante un proveedor de nube como Azure, AWS Cloud, Digital Ocean, etc. pero ¿Qué pasaría si se quiere replicar en un ambiente de producción? Es por ello, que se presenta la infraestructura basada en código para brindarnos una solución horizontal.

Por otro lado, para consumir los microservicios la forma normal sería el uso de RESTful, pero este servicio requiere conectarse a varios Endpoints, según (Garrido, 2018) REST ha sido un caso claro de éxito, sin embargo, su concepción CRUD conlleva limitaciones debido a su inflexibilidad pues se debe utiliza diferentes URI's para leer o escribir un único recurso. A partir de lo anterior, se va construyendo la información que se requiere proyectar en la pantalla, este proceso se puede convertir en un proceso tedioso de mantener.

Para el año 2012, la aplicación de móvil de Facebook presentaba un bajo rendimiento en la presentación de las vistas esto debido a que al manejar grandes cantidades de datos que generaban excesivas conexiones que posterior se traducían en un rendimiento muy ineficaz tanto del lado del cliente como a nivel del servidor. Para solventar dicha problemática se reestructuró el proceso para obtener la información mediante una única petición, de esta manera apareció GraphQL, pero solo hasta el año 2015 Facebook lanzaría la versión open source (código abierto) de GraphQL.

(Rodríguez,2020). Así pues, GraphQL proporciona soluciones al brindar un mejor rendimiento y flexibilidad.

Finalmente, en un comercio electrónico un sistema de recomendación ayuda con sugerencias y recomendaciones que permiten al usuario tener una mejor experiencia al momento de comprar artículos, la funcionalidad del sistema recomendador se basa en el filtrado que permite brindar mejores resultados dentro de la búsqueda, para ello se aplica diferentes técnicas de inteligencia artificial

2.2 JUSTIFICACIÓN

En Ecuador al revisar las infraestructuras basadas en código y GraphQL, la información encontrada es casi nula. En consecuencia, las infraestructuras basadas en código son herramientas que permiten construir, modificar y llevar un control de la infraestructura. Por ejemplo, al crear máquinas virtuales o servidores de base de datos u otros recursos en la nube y por algún motivo surgen errores se corre el riesgo de perder todo el trabajo. En definitiva, las infraestructuras basadas en código brindan la posibilidad de tener una infraestructura versionada debido a su flexibilidad y alto rendimiento.

API-REST ha mostrado sus limitaciones debido a su tipo de arquitectura lo que afecta de manera directa en la optimización y flexibilidad, debido a que su consumo del lado del cliente se vuelve algo tedioso ya que se debe realizar muchas peticiones para obtener la información personalizada (varios endpoints), por ello una API-GRAPHQL es una alternativa a la optimización de consulta de datos que puede adaptarse a las necesidades actuales. (Rodríguez,2020).

GraphQL es una nueva forma de ver las API’s, ya que ahora en la parte del Frontend o del cliente (App móvil) se tiene un lenguaje de consultas, pero también es importante en el lado del servidor que tiene un entorno de ejecución para entregar sus consultas, para ello se conecta a la base de

datos y devuelve la información. GraphQL es independiente del lenguaje de programación (Python, java, etc.) y la base de datos (SQL, NoSQL), inclusive puede ser una RESTful ya que se puede escribir por encima.

Un sistema recomendador basado en técnicas de inteligencia artificial permite sugerir artículos similares que previamente el usuario compro o sus gustos, lo que conlleva una mejor experiencia dentro del e-commerce. (Utrera, Cuevas, 2017).

2.3 ELICITACIÓN DE REQUERIMIENTOS

2.3.1 REQUERIMIENTOS

La recopilación de los requerimientos es una tarea muy importante dentro del marco para realizar cualquier proyecto, es una etapa que debe pasar el proyecto antes de realizar su ejecución ya que esto ayuda a la hora de evaluar el éxito del proyecto, por ello deben ser correctos, precisos y no ambiguos.

Según (Sommerville, 2005) los requerimientos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requerimientos reflejan las necesidades de los clientes frente a un sistema que ayude a resolver algún problema.

A continuación, se presenta la elicitación de requerimientos.

2.3.2 REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales de un sistema describen lo que el sistema debe hacer, es decir se definen los recursos específicos que el sistema debe proporcionar, se toma del documento de requerimientos del usuario, por ello su implicación hace referencia a la interacción con el usuario. (Sommerville, 2005).

Es decir, los requerimientos funcionales definen las funciones de la aplicación y los servicios que proporciona.

A continuación, se detallan los siguientes requerimientos:

2.3.2.1 INFRAESTRUCTURA COMO CODIGO

2.3.2.1.1 REQUERIMIENTOS FUNCIONALES

Identificador: RF01	**Nombre:** Aprovisionamiento de servidores Terraform.	**FUNCIONAL**
Descripción: Despliegue de toda la infraestructura.		**Categoría (Visible/No visible):** No Visible
Objetivo: Unificar el desarrollo de software a una sola aplicación.		
Información de entrada: • Proveedores de nube.		**Información de salida:**
Condición de aceptación: Ninguna		
Condición previa: Tener un servicio de cloud computing.		
Pos condición: Aprovisionamiento de servidores en el estado deseado.		
Prioridad: Alta		

Tabla 3 RF Aprovisionamiento de servidores Terraform (Elaboración propia)

Identificador: RF02	**Nombre:** Despliegue de Kubernetes.	**FUNCIONAL**
Descripción: Acceso a cada servidor desde el exterior.		**Categoría (Visible/No visible):** No Visible
Objetivo: Orquestar la gestión de múltiples desarrollos simultáneamente.		
Información de entrada: • Servicios		**Información de salida:**
Condición de aceptación: No existe.		
Condición previa: Disponer de imágenes de aplicaciones desarrolladas en cualquier lenguaje de programación.		
Pos condición: Mantener activa las aplicaciones produciendo escalabilidad.		
Prioridad: Alta		

Tabla 4 RF Despliegue de Kubernetes (Elaboración propia)

Identificador: RF03	**Nombre: S**ervicio de base de datos con mongoBD en Kubernetes.	**FUNCIONAL**

Descripción: Servicio de base de datos para acceder desde el exterior.	**Categoría (Visible/No visible):** No Visible
Objetivo: Desplegar una base de datos con mongoDB	
Información de entrada: • Imagen de mongoDB	**Información de salida:**
Condición de aceptación: No existe.	
Condición previa: Disponer de una conexión a mongoDB	
Pos condición: Mantener activa a la base de datos con volúmenes para realizar respaldos.	
Prioridad: Alta	

Tabla 5 RF Servicio de base de datos con mongoBD en Kubernetes (Elaboración propia)

Identificador: RF04	**Nombre:** Servicio del Backend	**FUNCIONAL**
Descripción: Servicio del Backend donde procesa información a través del Frontend.	**Categoría (Visible/No visible):** No Visible	
Objetivo: Acceder a la información que solicita a través de la aplicación y devolver al usuario final.		
Información de entrada: • Imagen del Backend en nodejs	**Información de salida:**	
Condición de aceptación: No existe.		
Condición previa: Imagen de la aplicación con todas las librerías.		
Pos condición: Devolver la información al usuario.		
Prioridad: Alta		

Tabla 6 RF Servicio del Backend (Elaboración propia)

Identificador: RF05	**Nombre:** Servicio del Frontend	**FUNCIONAL**
Descripción: Servicio del Frontend donde el sitio web interactúa con los usuarios.	**Categoría (Visible/No visible):** No Visible	
Objetivo: Recibir información del usuario y realizar peticiones al Backend.		
Información de entrada: • Imagen del Frontend realizada en Angular	**Información de salida:**	
Condición de aceptación: No existe.		
Condición previa: Imagen de la aplicación con sus respectivas librerías.		
Pos condición: Muestra el diseño de la página web.		
Prioridad: Alta		

Tabla 7 RF Servicio del Frontend (Elaboración propia)

Identificador: RF06	**Nombre:** Servicio del sistema recomendador.	**FUNCIONAL**
Descripción: Recomendar categorías que sean del interés del usuario.	**Categoría (Visible/No visible):** No Visible	
Objetivo: Recibir información del usuario a través de filtros colaborativos.		
Información de entrada: • Imagen del Sistema recomendador con Python y Flask.	**Información de salida:**	
Condición de aceptación: No existe.		
Condición previa: Imagen del sistema recomendador con sus librerías.		
Pos condición: Muestra recomendaciones de categorías para el usuario.		
Prioridad: Alta		

Tabla 8 RF Servicio del sistema recomendador (Elaboración propia)

Identificador: RF07	**Nombre:** Repositorio de imágenes en DockerHub	**FUNCIONAL**
Descripción: Acceso de las imágenes con sus respectivas versiones.	**Categoría (Visible/No visible):** No Visible	
Objetivo: Registrar o encapsular aplicaciones para desplegar en producción.		
Información de entrada: • Aplicaciones.	**Información de salida:**	
Condición de aceptación: No existe.		
Condición previa: Disponer de aplicaciones para subir a DockerHub.		
Pos condición: Mantener versiones de imágenes.		
Prioridad: Alta		

Tabla 9 RF Repositorio de imágenes en DockerHub (Elaboración propia)

2.3.2.1.2 REQUERIMIENTOS NO FUNCIONALES

Identificación del requerimiento.	RNF01
Clase del RNF:	Servidor
Descripción del requerimiento	El servicio de la base de datos debe tener únicamente sola una réplica.
Importancia:	Evitar problemas de integridad de datos.

Tabla 10 RNF01 (Elaboración propia)

Identificación del requerimiento.	RNF02
Clase del RNF:	Servidor
Descripción del requerimiento	El servicio de la base de datos al iniciar debe tener volúmenes para tener Backups.
Importancia:	Tener respaldos de información.

Tabla 11 RNF02 (Elaboración propia)

Identificación del requerimiento.	RNF03
Clase del RNF:	Servidor
Descripción del requerimiento	Un clúster de Kubernetes debe tener 2 nodos, el nodo máster y el nodo worker, la cual deben ser auto escalable.
Importancia:	Evitar caída del clúster y tener alta disponibilidad.

Tabla 12 RNF03 (Elaboración propia)

Identificación del requerimiento.	RNF04
Clase del RNF:	Servidor
Descripción del requerimiento	El servicio del Backend debe tener un balanceador de carga.
Importancia:	Evitar saturación al realizar peticiones.

Tabla 13 RNF04 (Elaboración propia)

Identificación del requerimiento.	RNF05
Clase del RNF:	Servidor
Descripción del requerimiento	Agregar protocolo de comunicación https tanto en el servidor y en el cliente.
Importancia:	Proteger las conexiones contra intervención de terceros.

Tabla 14 RNF05 (Elaboración propia)

Identificación del requerimiento.	RNF06
Clase del RNF:	Servidor
Descripción del requerimiento	El tiempo de respuesta del Frontend no debe ser mayor a 2 segundos.
Importancia:	Mostrar una página optima en dar respuesta al usuario.

Tabla 15 RNF06 (Elaboración propia)

Identificación del requerimiento.	RNF07
Clase del RNF:	Servidor
Descripción del requerimiento	Las pruebas de rendimiento se ejecutan con la herramienta JMeter.
Importancia:	Monitorear el servidor para agilizar los mantenimientos.

Tabla 16 RNF07 (Elaboración propia)

2.3.2.2 COMERCIO ELECTRÓNICO

2.3.2.2.1 REQUERIMIENTOS FUNCIONALES DEL ADMINISTRADOR

- **PEDIDOS**

Identificador: RF01	**Nombre:** Procesar pedidos nuevos	**FUNCIONAL**
Descripción: El administrador podrá gestionar los estados de los pedidos (Completar, Cancelar.)	**Categoría (Visible/No visible):** Visible	
Objetivo: Asegurar el proceso de los pedidos de la tienda.		
Información de entrada: • Estado del pedido.	**Información de salida:**	
Condición de aceptación: Ninguna		
Condición previa: Previamente debe haber pedidos registrados.		
Pos condición: El sistema deberá guardar el estado del pedido.		
Prioridad: Alta		

Tabla 17 RF Procesar pedidos nuevos (Elaboración propia)

Identificador: RF02	**Nombre:** Observar los detalles de los pedidos.	**FUNCIONAL**
Descripción: El administrador de la tienda podrá observar los pedidos con sus respectivos detalles dependiendo de su estado.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar mostrar los detalles de los pedidos.		
Información de entrada: • Filtro del pedido por estado (Procesando, Completo, Cancelado).	**Información de salida:** Lista de los pedidos existentes. • ID • Usuario • Teléfono • Dirección de envío • Fecha • Descripción • Iva • Total	

	• Factura • Estado
Condición de aceptación: Pedidos existentes.	
Condición previa: Anteriormente deben haberse registrado pedidos.	
Pos condición: El sistema deberá mostrar el listado de los pedidos registrados.	
Prioridad: Alta	

Tabla 18 RF Observar los detalles de los pedidos (Elaboración propia)

- **USUARIOS**

Identificador: RF03	**Nombre:** Ingresar usuarios nuevos.	**FUNCIONAL**
Descripción: El administrador de la tienda podrá ingresar nuevos usuarios a los que podrá otorgar permisos.		**Categoría (Visible/No visible):** Visible
Objetivo: Garantizar el ingreso de nuevos usuarios.		
Información de entrada: • Nombre • Apellido • Correo • Rol		**Información de salida:**
Condición de aceptación: Ninguna		
Condición previa: Identificarse como administrador.		
Pos condición: El sistema deberá guardar los usuarios registrados.		
Prioridad: Alta		

Tabla 19 RF Ingresar usuarios nuevos (Elaboración propia)

Identificador: RF04	**Nombre:** Observar los detalles de los usuarios.	**FUNCIONAL**
Descripción: El administrador de la tienda podrá observar todos los usuarios con sus detalles dependiendo de su estado.		**Categoría (Visible/No visible):** Visible
Objetivo: Garantizar mostrar los detalles de los usuarios.		
Información de entrada: • Filtro del usuario por estado (Todos, Activos, Inactivos).		**Información de salida:** Lista de los usuarios registrados. • ID • Nombres • Apellidos • Correo electrónico • Permiso • Estado
Condición de aceptación: Usuarios existentes.		
Condición previa: Previamente debe haber usuarios registrados.		
Pos condición: El sistema deberá mostrar el listado de los usuarios registrados.		
Prioridad: Alta		

Tabla 20 RF Observar los detalles de los usuarios (Elaboración propia)

Identificador: RF05	**Nombre:** Actualizar datos de los usuarios.	**FUNCIONAL**
Descripción: El administrador podrá actualizar los datos de los usuarios.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar actualizar los datos de un usuario.		
Información de entrada: • Nombre • Apellido • Correo • Rol	**Información de salida:**	
Condición de aceptación: Usuarios existentes.		
Condición previa: Previamente debe haber usuarios registrados.		
Pos condición: El sistema deberá mostrar el listado de los usuarios registrados.		
Prioridad: Alta		

Tabla 21 RF Actualizar datos de usuario (Elaboración propia)

Identificador: RF06	**Nombre:** Deshabilitar cuentas de los usuarios.	**FUNCIONAL**
Descripción: El administrador de la tienda podrá deshabilitar una cuenta de usuario.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar deshabilitar una cuenta de usuario.		
Información de entrada: • ID usuario.	**Información de salida:**	
Condición de aceptación: Usuarios existentes.		
Condición previa: Previamente debe haber usuarios registrados.		
Pos condición: El sistema no deberá mostrar el usuario deshabilitado en los usuarios activos.		
Prioridad: Alta		

Tabla 22RF Deshabilitar cuenta de usuario (Elaboración propia)

Identificador: RF07	**Nombre:** Habilitar cuentas de los usuarios.	**FUNCIONAL**
Descripción: El administrador de la tienda podrá habilitar una cuenta de usuario.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar habilitar una cuenta de usuario.		
Información de entrada: • ID usuario.	**Información de salida:**	
Condición de aceptación: Usuarios existentes.		
Condición previa: Previamente debe haber usuarios registrados.		
Pos condición: El sistema deberá mostrar el usuario habilitado en los usuarios activos.		
Prioridad: Alta		

Tabla 23 RF Habilitar cuenta (Elaboración propia)

- **PRODUCTOS**

Identificador: RF08	**Nombre:** Ingresar nuevos productos	**FUNCIONAL**

Descripción: El administrador de la tienda podrá ingresar nuevos productos.	**Categoría (Visible/No visible):** Visible
Objetivo: Garantizar el ingreso de nuevos productos.	
Información de entrada: • Nombre • Autor • Url imagen principal • Fecha de lanzamiento • Urls captura • Stock • Precio • Categoría	**Información de salida:**
Condición de aceptación: Ninguna	
Condición previa: Identificarse como administrador.	
Pos condición: El sistema deberá guardar los productos ingresados.	
Prioridad: Alta	

Tabla 24 RF Ingresar nuevos productos (Elaboración propia)

Identificador: RF09	**Nombre:** Observar los detalles de los productos	**FUNCIONAL**
Descripción: El administrador de la tienda podrá visualizar los productos con sus respectivos detalles.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar mostrar los detalles de los productos.		
Información de entrada:	**Información de salida:** Lista de los productos existentes. • ID • Nombre • Autor • Stock • Precio • Estado	
Condición de aceptación: Producto existente.		
Condición previa: Previamente debe haber productos registrados.		
Pos condición: El sistema deberá mostrar el listado de los productos ingresados.		
Prioridad: Alta		

Tabla 25 RF Observar los detalles de un producto.

Identificador: RF10	**Nombre:** Actualizar datos de los productos.	**FUNCIONAL**
Descripción: El administrador de la tienda podrá actualizar los datos de los productos.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar actualizar los datos de un producto.		
Información de entrada: • Nombre • Autor • Url imagen principal	**Información de salida:**	

• Fecha de lanzamiento • Urls captura • Stock • Precio • Categoría	
Condición de aceptación: Productos existentes.	
Condición previa: Previamente debe haber productos.	
Pos condición: El sistema deberá mostrar el listado de los productos.	
Prioridad: Alta	

Tabla 26 RF Actualizar producto (Elaboración propia)

- **CATEGORÍA**

Identificador: RF12	**Nombre:** Observar los detalles de las categorías.	**FUNCIONAL**
Descripción: El administrador de la tienda podrá visualizar todas las categorías con sus detalles.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar un despliegue y detalles de las categorías.		
Información de entrada:	**Información de salida:** Lista de los productos existentes. • ID • Nombre • Estado	
Condición de aceptación: Categoría existente.		
Condición previa: Previamente debe haber categorías registradas.		
Pos condición: El sistema deberá mostrar el listado de las categorías registradas.		
Prioridad: Alta		

Tabla 27 RF Visualizar detalles categoría (Elaboración propia)

2.3.2.2.2 REQUERIMIENTOS FUNCIONALES DEL USUARIO

Identificador: RF01	**Nombre:** Iniciar sesión.	**FUNCIONAL**
Descripción: El usuario debe iniciar sesión en el sistema para realizar alguna compra.	**Categoría (Visible/No visible):** Visible	
Objetivo: Permitir el acceso del usuario al sistema.		
Información de entrada: • Correo electrónico • Contraseña.	**Información de salida:**	
Condición de aceptación: Ninguna.		
Condición previa: Previamente el usuario debe haberse registrado.		
Pos condición: Acceso a su cuenta.		
Prioridad: Alta		

Tabla 28 Iniciar sesión (Elaboración propia

Identificador: RF02	**Nombre:** Cerrar sesión.	**FUNCIONAL**
Descripción: El usuario debe poder cerrar sesión para garantizar la privacidad de los datos de su cuenta.		**Categoría (Visible/No visible):** Visible
Objetivo: Permitir al usuario cerrar la sesión.		
Información de entrada:		**Información de salida:**
Condición de aceptación: Ninguna.		
Condición previa: Previamente el usuario debe haber iniciado sesión.		
Pos condición: Denegar el acceso a su cuenta.		
Prioridad: Alta		

Tabla 29 RF Cerrar sesión (Elaboración propia)

Identificador: RF03	**Nombre:** Registrarse el usuario en la tienda.	**FUNCIONAL**
Descripción: El usuario deberá ingresar sus datos personales en el formulario de registro.		**Categoría (Visible/No visible):** Visible
Objetivo: Guardar la información de los usuarios.		
Información de entrada: • Nombres • Apellidos • Fecha de nacimiento • Correo electrónico • Contraseña.		**Información de salida:**
Condición de aceptación: Ninguna.		
Condición previa: Ninguna.		
Pos condición: Nuevo usuario registrado.		
Prioridad: Alta		

Tabla 30 RF Registrar usuario (Elaboración propia)

Identificador: RF04	**Nombre:** Comprar productos.	**FUNCIONAL**
Descripción: El usuario podrá comprar los productos que se venden en la tienda, debe permitir añadir y eliminar productos de su carrito de compras.		**Categoría (Visible/No visible):** Visible
Objetivo: Garantizar que el usuario realice compras.		
Información de entrada: El sistema debe solicitar: • Producto seleccionado • Cantidad • Nombres • Apellidos • Correo Electrónico • Teléfono		**Información de salida:** • Productos comprados • Valor total

• Provincia • Ciudad • Dirección de entrega • información de la tarjeta.	
Condición de aceptación: Diversidad de productos.	
Condición previa: Productos registrados.	
Pos condición: Compra exitosa.	
Prioridad: Alta	

Tabla 31 RF Comprar productos (Elaboración propia)

Identificador: RF05	**Nombre:** Ingresar, actualizar y observar detalles de la cuenta.	**FUNCIONAL**
Descripción: El usuario podrá ingresar, actualizar los datos ingresados.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar la disponibilidad de la información del usuario registrado.		
Información de entrada: • Datos personales • Actualización de datos personales.	**Información de salida:**	
Condición de aceptación: Información de la cuenta.		
Condición previa: Cumplir con el registro.		
Pos condición: Datos actualizados exitosamente.		
Prioridad: Alta		

Tabla 32 RF Ingresar, actualizar y observar detalles de la cuenta (Elaboración propia)

Identificador: RF06	**Nombre:** Cancelar pedidos realizados.	**FUNCIONAL**
Descripción: El usuario podrá cancelar los pedidos realizados.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar que los pedidos de los usuarios puedan ser cancelados.		
Información de entrada: • ID pedido.	**Información de salida:**	
Condición de aceptación: Ninguna.		
Condición previa: Tener realizada una compra.		
Pos condición: Realizar una devolución exitosa.		
Prioridad: Alta		

Tabla 33 RF Cancelar pedidos realizados (Elaboración propia)

Identificador: RF07	**Nombre:** Consultar los pedidos realizados.	**FUNCIONAL**
Descripción: El usuario podrá visualizar los detales de los pedidos que ha realizado.	**Categoría (Visible/No visible):** Visible	
Objetivo: Desde el panel de información de la cuenta el usuario pueda ver la información detallada de los pedidos realizados.		
Información de entrada: • Ninguno.	**Información de salida:** Lista de los pedidos realizados.	

	• ID • Usuario • Teléfono • Dirección de envío • Fecha • Descripción • Iva • Total • Factura • Estado
Condición de aceptación: Ninguna.	
Condición previa: Usuario debe haber realizado una compra	
Pos condición: Ninguna.	
Prioridad: Alta	

Tabla 34 RF Consultar pedidos realizados (Elaboración propia)

Identificador: RF08	**Nombre:** Restablecer contraseña.	**FUNCIONAL**
Descripción: El usuario podrá restablecer su contraseña.	**Categoría (Visible/No visible):** Visible	
Objetivo: Garantizar que el usuario pueda recuperar su cuenta en caso de olvidar la contraseña.		
Información de entrada: • Correo electrónico.	**Información de salida:**	
Condición de aceptación: Ninguna.		
Condición previa: Usuario debe haberse registrado.		
Pos condición: Acceso a su cuenta.		
Prioridad: Alta		

Tabla 35 RF Restablecer contraseña (Elaboración propia)

2.3.2.2.3 DOCUMENTACIÓN CASO USO DEL ADMINISTRADOR

Los casos de usos explican de forma gráfica lo que se debe resolver y los actores involucrados.

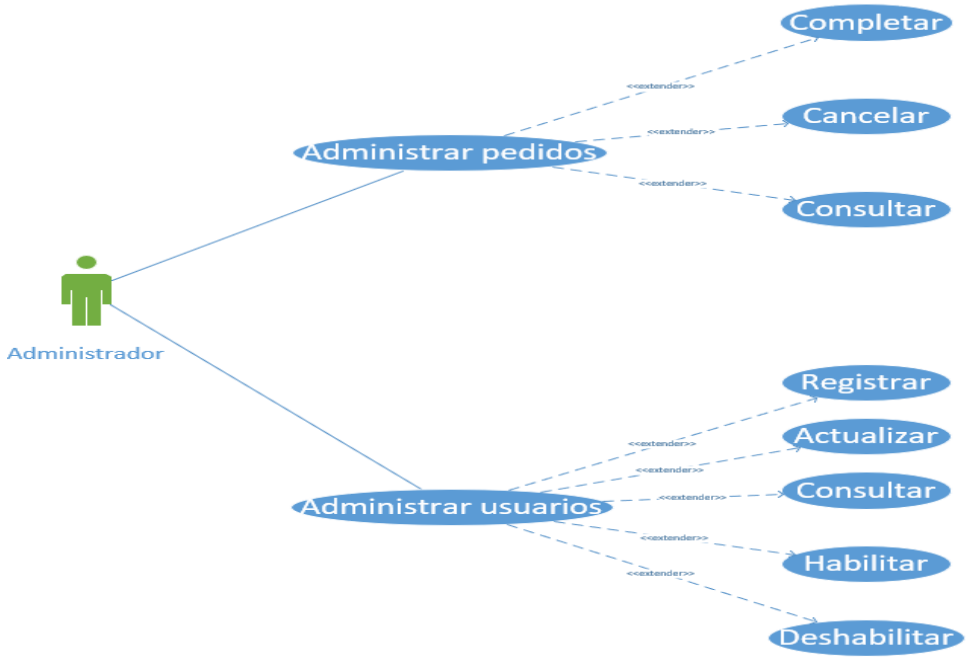


Fig. 26 Modelo de Caso de Uso: Administrador (Elaboración propia)

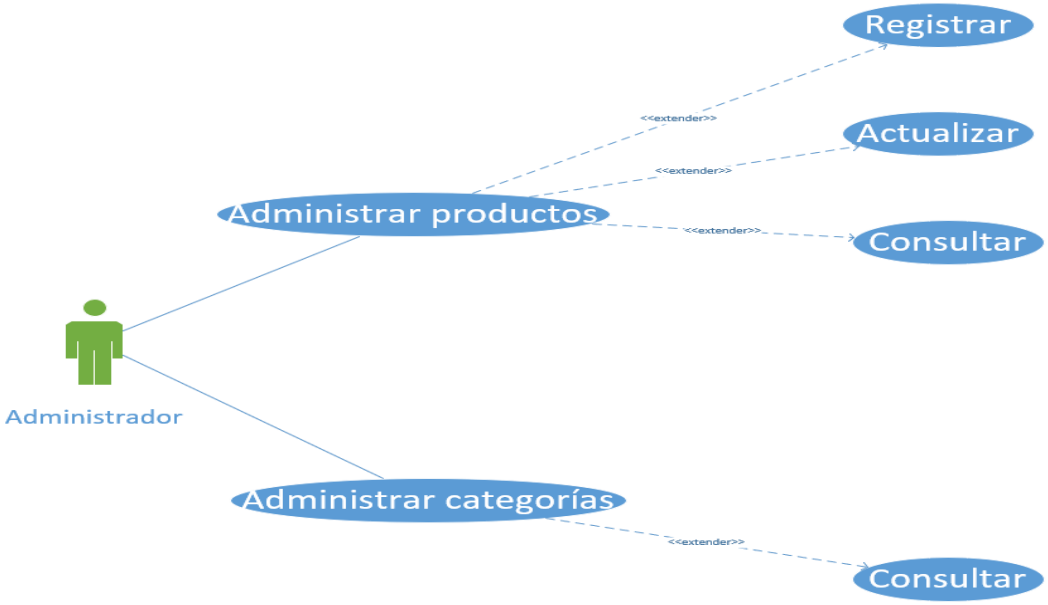


Fig. 27 Modelo de Caso de Uso: Administrador (Elaboración propia)

2.3.2.2.4 DOCUMENTACIÓN CASO USO DEL USUARIO

Fig. 28 Modelo de Caso de Uso: Usuario (Elaboración propia)

2.3.2.3 SISTEMA RECOMENDADOR

- Permitir predicciones de productos basadas es su perfil de compras(categorías).
- A la hora de generar recomendaciones el sistema debe usar:
    - Filtro colaborativo
    - Usar la similitud (Pearson).
- El sistema dispondrá de una interfaz gráfica.
- El sistema debe ser integrado con el comercio electrónico.

2.3.3.2 COMERCIO ELÉCTRONICO

- **Disponibilidad**: Garantiza el acceso a la cuenta de los usuarios para realizar compras.
- **Rendimiento**: Los tiempos de carga deben ser cortos**.**
- **Mantenibilidad**: Fácil de solucionar errores y poder agregar nuevas funcionalidades.
- **Confidencialidad**: Proteger la información de accesos no autorizados.
- **Estabilidad**: Tener el menor número de fallas posibles

- **Usabilidad**: Proporcionar una interfaz de fácil manejo.

2.3.3.3 SISTEMA RECOMENDADOR

- **Disponibilidad**: Garantiza el acceso a las recomendaciones de los usuarios al realizar compras.

CAPÍTULO 3 DISEÑO DE LA ARQUITECTURA

3.1 DEFINICION DE LA ARQUITECTURA

Hoy en día varias empresas realizan buenas prácticas partiendo de arquitecturas tradicionales, en este contexto, una arquitectura elaborada con Terraform se combina y construye de manera segura y eficiente la infraestructura. Esta arquitectura presenta una infraestructura de mayor calidad con mejores capacidades de administración maximizando la eficiencia y evitando el error humano. Dentro de esta arquitectura en Terraform tenemos una plataforma de Kubernetes la cual administra cargas de trabajo y servicios.

Se recomienda crear tres nodos mínimos para desplegar servicios en producción para el Clúster para eliminar problemas de sobre carga o rendimiento. En el clúster se define la arquitectura, esta cuenta con un servicio de base de datos mongoDB juntamente con un volumen para evitar integridad de datos en caso de que el Pod presenta fallas por sobre carga. La aplicación Backend desarrollada con Nodejs y la aplicación Frontend desarrollada con Angular, presentan una conexión interna con servidor y con la base de datos, estas se conectan a partir del servicio ClusterIp. También se tiene un Pod que contiene el sistema recomendador realizado en Python con Flask. La conexión entre el cliente y el servidor también se usa el servicio ClusterIp. Ahora bien, para exponer al usuario final se usa un objeto de tipo Ingress el cual conecta al dominio y subdominios, donde serán expuestos a través del servicio LoadBalancer, este enviará todas las peticiones al Ingress.

Se utiliza un objeto de ClusterIssuer el mismo se encarga de especificar la autoridad de certificación para que los servicios contengan protocolos Https.

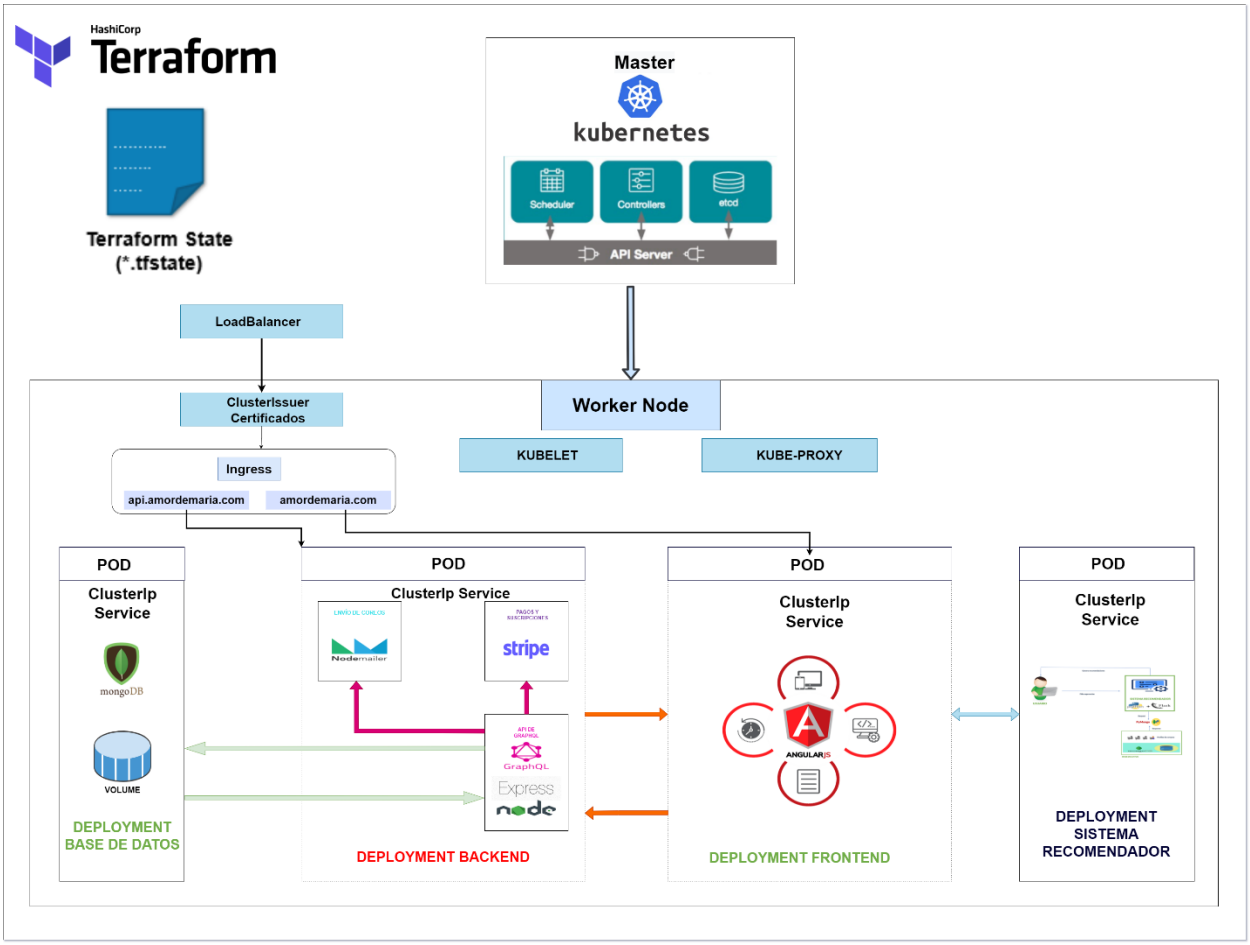


Fig. 29 Arquitectura General (Elaboración propia).

3.2 ARQUITECTURA Y COMPONENTES INFRAESTRUCTURA

En primera instancia se tiene la arquitectura de Terraform para la infraestructura como código con archivos de configuración declarativos que se usan para crear, administrar y actualizar recursos de infraestructura automatizando la administración de recursos en un flujo de trabajo. En el archivo de configuración se tiene los plugins y la conexión hacia el Provider, donde apunta el id del servicio en la nube. Los Provisioners se usan para definir acciones específicas en la maquina remota con el fin de preparar servidores de infraestructura. Luego se tiene el Init, este inicializa el directorio de los archivos de configuración. Además, se usa el comando Plan para crear un plan de ejecución y

alcanzar el estado deseado de la infraestructura y, finalmente se tiene el comando Apply para realizar cambios y desplegar toda la infraestructura en el servicio de Azure con Kubernetes.

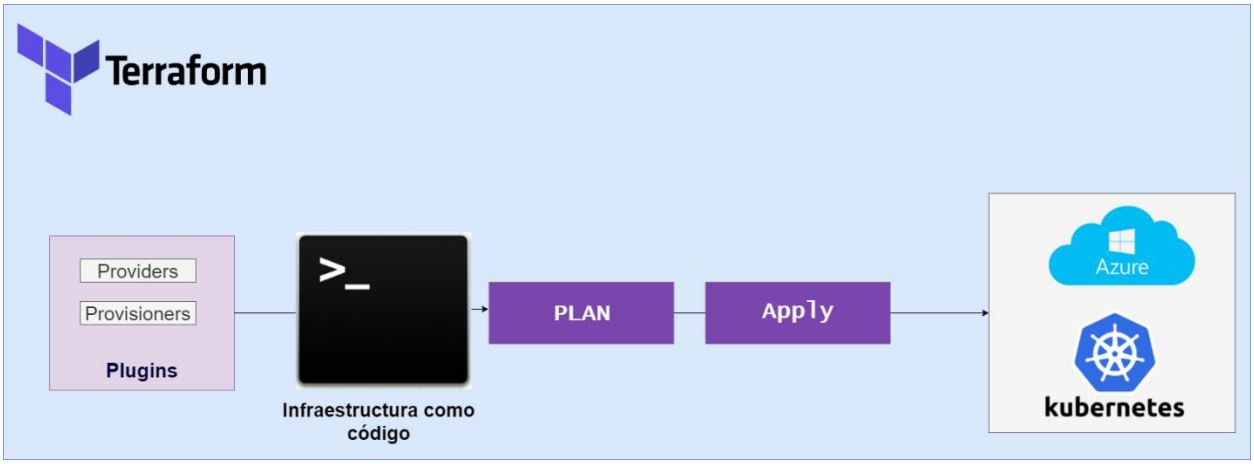


Fig. 30 Arquitectura Terraform (Elaboración propia).

3.2.1 DEFINICIÓN DE PROVIDERS

Se debe especificar el proveedor de nube con el que deseamos conectar. En este caso se está conectado al proveedor de Azure, el cual está mantenido por HashiCrop. Ver anexo 18.

3.2.2 ARCHIVO MAIN.TF

Después de realizar las configuraciones tenemos el archivo main.tf, aquí se va a agregar las características de configuración para el módulo. En esta parte damos un nombre al recurso, nombre del nodo y espacio del disco. Ver anexo 19.

3.2.3 ARQUITECTURA DE KUBERNETES

En la arquitectura se presenta un balanceador de carga apuntando al registro DNS, en este caso se tiene dos subdominios. El balanceador de carga abarca todas las peticiones y luego los envía al Ingress este actúa como filtro teniendo en cuenta la referencia y el subdominio al cual se envió una petición y reenviando al servicio que corresponde.

Se definieron cuatro Deployments, el primero contiene la base de datos mongoDB, el segundo abarca el Backend, el tercero comprende el Frontend y el ultimo encierra el sistema recomendador.

En el Deployment se va a definir todas las características que se desea para la aplicación como el número de replicas para balancear la carga y evitar caídas de servicios. Para el Deployment de la base de datos se debe configurar con una réplica para evitar problemas de integridad de datos, los puertos que se exponen para la base de datos son los 27017, estos servicios son de tipo ClusterIp, es decir que son visibles únicamente para el clúster.

En cada Pod se definió manifiestos declarativos de Kuberentes, una ventaja de usar esta tecnología es que al momento de sacar versiones de aplicaciones se pueden actualizar fácilmente sin tener que eliminar toda la aplicación pues el despliegue es automático.

Cada parte de la arquitectura presenta un rol importante ya que todas están relacionas entre si presentando al usuario final una aplicación optima.

Componentes de la arquitectura:

- Nodo máster: tiene el entorno de ejecución para el control plane, el cual se encarga de gestionar el estado del clúster.
- Nodo worker: es el encargado de ejecutar las aplicaciones en pods.
- Pod: es el lugar en donde se encuentra los contendores de sus respectivas imágenes para el despliegue.
- Deployment de base de datos: contiene la imagen del mongoDB con el estado deseado que se migrará en primera instancia.
- Deployment del Backend, Frontend y sistema recomendador: contiene el api en Nodejs, aplicación en Angular y el sistema recomendador realizado con Python y Flask. Estas imágenes fueron creadas con Docker.
- Balanceador de carga: envía peticiones al Ingress, utiliza los puertos 443 y 80 para recibir peticiones.

- ClusterIp: permite la conexión entre los cuatro pods.
- Ingress: abarca los dominios y subdominios estos apuntan a cada Deployment con el servicio ClusterIp.
- ClusterIssuer: emite certificados ssl para el servidor y el cliente manteniendo una conexión segura.

3.2.3.1 BASE DE DATOS MANIFIESTO

Cada Deployment tiene un archivo de manifiesto el cual incluye instrucciones en un archivo yaml donde están las especificaciones de cada aplicación. Estas instrucciones contienen configuraciones sobre el despliegue de la aplicación al nodo.

En este manifiesto especificamos todas las características que deseamos para la base de datos, una de las principales recomendaciones es tener un único pod siempre activo.

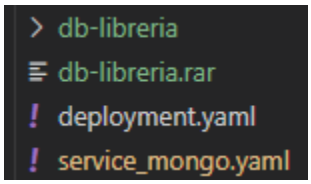


Fig. 31 Manifiesto para el mongoDB (Elaboración propia).

Estos archivos contienen la configuración para el despliegue de la base de datos mongoDB, se tiene configurado un volumen donde se almacenará datos de mongoDB evitando integridad de datos. En el Anexo 1 se muestra el deployment de la base de datos.

Dentro del Deployment se define el apiVersion, en este caso se utiliza la versión apps/v1, ya que es un objeto de tipo Deployment. Además, se agregó etiquetas para que su uso sea más fácil de obtener servicios, Pods, etc. Se define el campo selector el cual identifica los Pods que se debe gestionar o administrar. También se tiene el campo de replicas, el cual se define una sola replica para evitar la integridad de datos. En la parte del spec se especifica el contenedor que se desea, en este caso de utiliza la imagen oficial de mongo definiendo con la versión, aquí se utiliza la versión

3.6.3. en el campo env se agrega las credenciales para la conexión a la base de datos. En sección resources se detalla las características de cpu y memoria. En el campo de ports se especifica el puerto del contenedor de mongoDB, para esto se usa el puerto 27017. La configuración del servicio se puede ver en el Anexo 2.

Se define el servicio de tipo ClusterIP especificando el puerto, en este caso el puerto 27017, para hacer referencia al Deployment se usa el nombre del selector: mongo. En el apiVersion se define como v1 porque es un servicio de tipo ClusterIP que se puede acceder a través del puerto 27017.

3.2.3.2 BACKEND MANIFIESTO

Para la definición del manifiesto del Backend se usa una imagen construida en Docker, esta imagen tiene la aplicación en NodeJs con todas sus librerías. Dicha ilustración está en el Docker Hub. Para el despliegue se tiene un Deployment con sus respectivas configuraciones. Puede ver las configuraciones en el Anexo 3.

El Deployment es similar a la base de datos, aquí no se hace uso de variables de entorno, volúmenes. Este archivo especifica que se va a tener dos replicas para tener una distribución de carga correcta, también se utiliza los comandos para hacer el despliegue de la imagen creada en Docker. Para este Deployment se usa el puerto 2002 para NodeJs. Archivo de configuración en el Anexo 4.

En este archivo se define el servicio el cual es de tipo ClusterIP, también se expone el mismo puerto que se usó en el Deployment.

3.2.3.3 FRONTEND MANIFIESTO

En la definición del archivo para el Frontend se tiene la imagen de la aplicación desarrollada con Angular y con todas las librerías que necesita. Dentro de la configuración del Deployment se

agrega el comando para desplegar la aplicación que está en el Docker Hub. Archivo de configuración en el Anexo 5.

El puerto que se agrega para la aplicación desarrollada en Angular es el 4200. Este interactuará con el usuario final y envía peticiones al Backend y muestra la información deseada por el usuario. Archivo de configuración en el Anexo 6.

Las configuraciones son idénticas a los anteriores servicios, en este archivo se modifica el puerto que en este caso es el puerto 4200 que se utilizó en el Deployment, también se debe apuntar al nombre, en este caso se agrega el nombre angular.

3.2.3.4 API SISTEMA RECOMENDADOR MANIFIESTO

En este archivo se tiene la imagen creada del sistema recomendador, este sistema está desarrollado en el lenguaje de Python con Flask. Para que se ejecute la imagen se agregó el comando de ejecución para Python y el resto de las configuraciones son iguales a los otros archivos manifiestos. Archivo de configuración en el Anexo 7.

El puerto que se utiliza es el 4000, tiene comunicación con el Frontend. Este recibe y envía datos para el usuario en el Frontend a través de filtros colaborativos. Archivo de configuración en el Anexo 8.

El archivo manifiesto del servicio se apunta al puerto 400, se tiene configuraciones similares al resto de archivos manifiestos.

Una vez definida todos los archivos manifiestos con todas las especificaciones se debe ejecutar los comandos para desplegar todos los manifiestos configurados para su respectiva tarea. Si toda la estructura está correctamente, los archivos manifiestos se ejecutarán de la misma manera sin problemas.

```
jarevalop1@Azure:~/sistema_recomendador$ kubectl get deployments --all-namespaces=true
NAMESPACE     NAME                READY   UP-TO-DATE   AVAILABLE   AGE
default       angular             1/1     1            1           4m47s
default       mongo               1/1     1            1           6m28s
default       node                0/1     1            0           5m23s
default       recomendador        1/1     1            1           3m58s
kube-system   coredns             2/2     2            2           95m
kube-system   coredns-autoscaler  1/1     1            1           95m
kube-system   metrics-server      1/1     1            1           95m
kube-system   tunnelfront         1/1     1            1           95m
jarevalop1@Azure:~/sistema_recomendador$ kubectl get pods
NAME                               READY   STATUS             RESTARTS   AGE
angular-9d4977886-xszxg            1/1     Running            0          6m23s
mongo-56fc6f4f58-bkh29             1/1     Running            0          8m4s
node-8687fc548-ghfxn               0/1     ImagePullBackOff   0          6m59s
recomendador-b79fdb9d9-h6hhb       1/1     Running            0          5m34s
```

Fig. 32 Despliegue de todos los servicios en Kubernetes (Elaboración propia).

Podemos observar los despliegues de cada deployment con sus réplicas y cada uno con su relación.

3.3 DISEÑO Y ARQUITECTURA COMERCIO ELECTRONICO

En esta sección se va a ver las decisiones que se han tomado para el comercio electrónico en el aspecto de diseño, se describirá la arquitectura y la relación entre cada uno de los componentes que lo integran.

3.3.1 ARQUITECTURA

En lo referente a la arquitectura se ha decidido separar el backend (servidor) y frontend (interfaz gráfica) con el fin de proporcionar un desarrollo más rápido y un estado independiente que brinda comodidad a la hora de programar cada uno de los componentes y permitir que sea mucho más escalable y flexible a la hora de cambiar tecnologías.

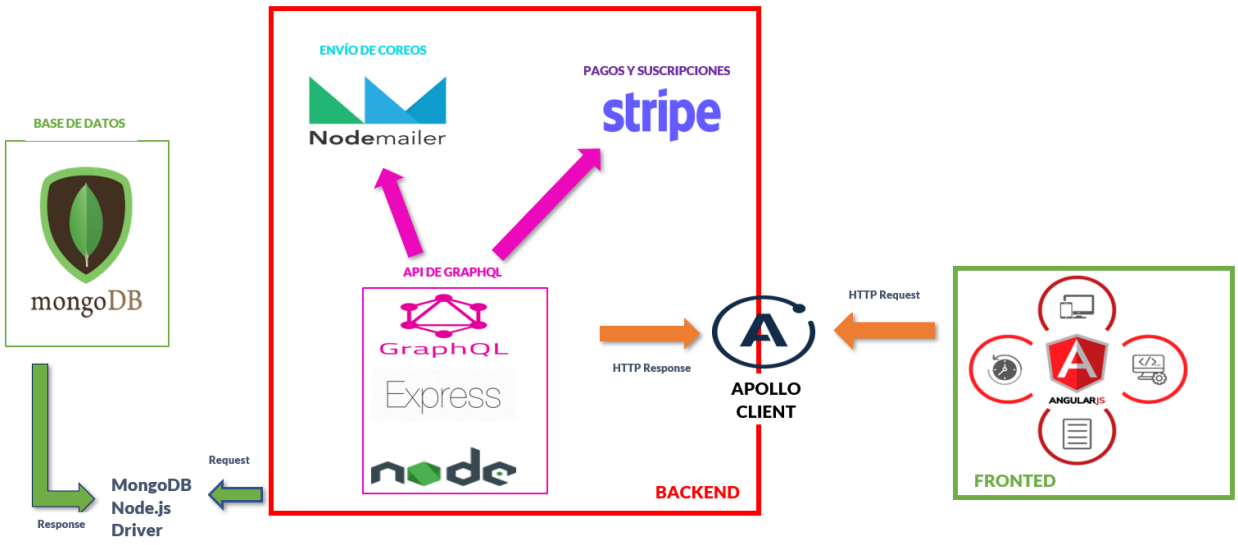


Fig. 33 Arquitectura Backend – Fronted (Elaboración propia)

Backend

La API de GraphQL se ha desarrollado con Node, usando Express como framework para manejar las peticiones (aplicar middlewares, etc.) juntamente con TypeScript haciendo uso de Apollo Server Express para trabajar con GraphQL y el paquete de MongoDB NodeJS Driver que permite crear una conexión con la base de datos.

Para poder enviar correos se usa Nodemailer y para procesar los pagos se hace uso de Stripe.

Frontend

Toda la implementación del cliente se realiza con Angular que se caracteriza por permitir utilizar templates declarativos, inyección de dependencias y uno de sus pilares fundamentales es crear componentes reutilizables, emplea el modelo **MVC** (Modelo, Vista, Controlador), el framework puede trabajar con JavaScript puro, pero un añadido muy especial es el uso de TypeScript que potencia las capacidades de este, brindado así la opción de poder crear aplicaciones web robustas. Es modular, por lo que su eje principal es un core y módulos que permiten acceder a más características cuando sea necesario.

3.4 DISEÑO Y ARQUITECTURA SISTEMA RECOMENDADOR

3.4.1 ARQUITECTURA

Fig. 34 Arquitectura del sistema recomendador (Elaboración propia)

La API del sistema recomendador se ha desarrollado con **Flask**, que es un muy similar a Node.js permite el desarrollo completo, rápido y ligero, utilizando Python como lenguaje de programación no incluye un **ORM (**Objet Relational Mapping**)** dejando al programador tomar las decisiones de diseño, se empleó el paquete de **PyMongo** que permite crear una conexión con la base de datos para recuperar los datos de los perfiles de compras de los usuarios y también se utilizó otras librerías para desarrollar el sistema recomendador y posteriormente brindar las recomendaciones al usuario.

CAPÍTULO 4 IMPLEMENTACIÓN DE LA ARQUITECTURA

4.1 PROVEEDORES CLÚSTER KUBERNETES EN LA NUBE

Para realizar el despliegue de la infraestructura se utilizó un proveedor en la nube como es la de Azure. Vamos a detallar los planes de Microsoft Azure en la siguiente tabla:

Instancia	Núcleos	RAM	Almacenamiento temporal	Precio por uso
A0	1	0,75 GB	20 GB	$ 0,02/hora
A1	1	1,75 GB	225 GB	$ 0,08/hora
A2	2	3,50 GB	490 GB	$ 0,16/hora
A3	4	7,00 GB	1.000 GB	$ 0,32/hora
A4	8	14,00 GB	2.040 GB	$ 0,64/hora

Tabla 36 Tabla de características de Azure (Elaboración propia).

Al realizar el despliegue de la infraestructura en Azure, se utilizó la cuenta estudiantil el cual tiene una suscripción gratis con plan básico para hacer uso de los servicios de Azure. En este sentido, se utilizó el plan básico que Azure brinda en cuentas de suscripción. Si se desea ampliar características debemos cambiar el plan de Azure para tener servicios avanzados con enormes infraestructuras. El uso de Azure es adecuado para cargas de trabajo de desarrollo, servidores de compilación, sitios web, repositorios de código, etc.

Existen más proveedores que se pueden utilizar, para ello se debe revisar los planes verificando que las características del servicio cumplan con las necesidades para desplegar infraestructura.

4.2 IMPLEMENTACION DE LA ARQUITECTURA DE TERRAFORM

Definimos una arquitectura en Terraform para llevar a cabo el despliegue. Esta herramienta soporta varios proveedores de infraestructura en la nube, los servicios más populares son: Google Cloud, Amazon Web Service, Azure y Digital Ocean. Para realizar el despliegue con Terraform y Azure se necesita el identificar o token que el proveer tiene. En este caso vamos a usar el identificador del servicio de Azure.

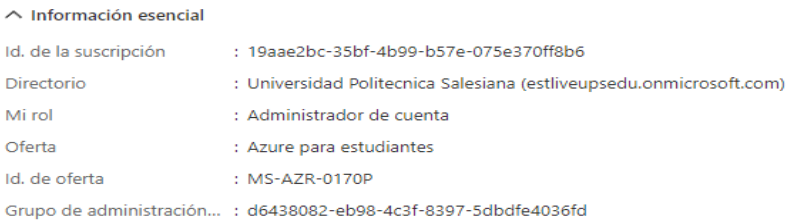


Fig. 35 Información general de Azure (Elaboración propia).

Con esta información general podemos conectarnos al servicio y realizar configuraciones y despliegues. Para conectar Terraform debemos definir en un archivo las configuraciones con las credenciales del servicio de Azure.

```
provider "azurerm"{
    subscription_id = "19aae2bc-35bf-4b99-b57e-075e370ff8b6"
    client_id       = "cbe76e0c-48e1-4152-af6a-7cabc9440d7b"
    client_secret   = "cha~rvZz4_ryPxx-ES_t57snRPE9YQGR_e"
}
```

Fig. 36 Definición de credenciales del servicio de Azure (Elaboración propia).

Al aplicar el comando APPLY, todas las configuraciones realizadas en Terraform se ejecutan y conectan automáticamente al servicio de Azure. En las configuraciones también se configuran el nombre del clúster, se asigna la versión y también se puede definir cuantos nodos deseamos desplegar, en este caso se despliegan 2 nodos.

```
resource "azurerm_kubernetes_cluster" "tesis-ecommerce" {
  name                = "tesis-aks"
  location            = azurerm_resource_group.tesis-ecommerce.location
  resource_group_name = azurerm_resource_group.tesis-ecommerce.name
  dns_prefix          = "tesis-ecommerce"

  default_node_pool {
    name            = "container"
    node_count      = 2
    vm_size         = "Standard_D2_v2"
    os_disk_size_gb = 30
  }
```

Fig. 37 Configuración de características (Elaboración propia).

Se agrega características de auto escalado para que en caso de que los nodos fallen, inmediatamente se active otro nodo manteniendo activos los servicios. Luego de realizar todas las configuraciones necesarias se procede a aplicar los comandos de despliegue.

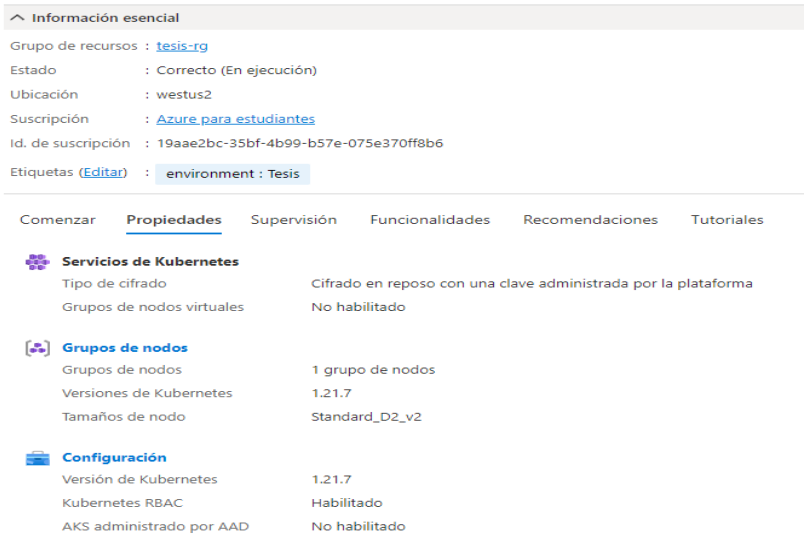


Fig. 38 Información del clúster de Kubernetes (Elaboración propia).

Si todas las configuraciones cumplen la estructura y no salen del límite que brinda el servicio, el archivo main.tf se ejecutará sin problemas, en caso de que existan errores no se desplegará y lanzará errores de compilación en la consola.

4.3 IMPLEMENTACION DE LA ARQUITECTURA BACKEND CON DOCKER Y KUBERNETES

Para realizar el despliegue del Backend, lo primero que se realizó fue crear la imagen en Docker. Para ello se definió las características de la aplicación, se desplegó con Nodejs. El archivo de configuración de Docker se agregó el puerto y el servidor en el que se va a levantar.

```
FROM node:14
# create root application folder
WORKDIR /app
# copy configs to /app folder
COPY package*.json ./
COPY tsconfig.json ./
# copy source code to /app/src folder
COPY ./src /app/src
```

```
RUN npm install
RUN npm run build
EXPOSE 2002
CMD [ "node", "./dist/main.js" ]
```

Tabla 37 Archivo Dockerfile para el Backend (Elaboración propia).

Para crear el docker-compose debemos tener le repositorio creado el Docker Hub para luego direccionar desde el archivo de configuración al repositorio.

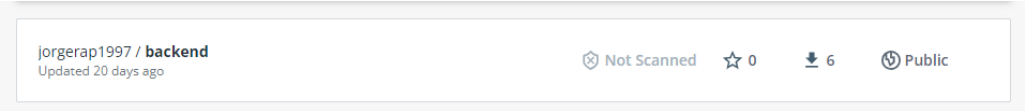


Fig. 39 Repositorio del Backend en Docker Hub (Elaboración propia).

Luego se crea el docker-compose y agregamos el nombre del repositorio del Docker Hub. También debemos definir los comandos de ejecución y el puerto en el que la aplicación se ejecutara.

```
version: "3.6"
services:
  node:
    image: jorgerap1997/backend:latest
    restart: always
    command: npm run start
    ports:
      - "2002:2002"
```

Tabla 38 Definición del archivo docker-compose (Elaboración propia).

Finalmente se ejecuta los archivos y la imagen se crea correctamente. Al final subimos la versión al Docker Hub.

Luego se procede a crear el deploymente y el service como de detalla en el capítulo 3 sobre los archivos manifiestos. Estos archivos en la sección de las imágenes tienen direccionado al repositorio de Docker Hub. Al ejecutar el archivo deployment y el service se tiene el despliegue del Backend.

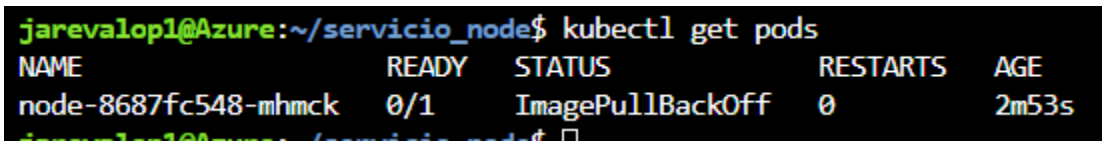


Fig. 40 Backend en corriendo en ambiente de producción (Elaboración propia).

4.4 IMPLEMENTACION DE LA ARQUITECTURA FRONTEND CON DOCKER Y KUBERNETES

Para desplegar el Frontend se creó la imagen en Docker con todas sus librerías. El Frontend se desarrolló con angular. Para ello se definió el archivo Dockerfile. Dentro del archivo están las características y librerías que necesitan el proyecto para desplegarse sin problemas.

```
FROM node:14-alpine
# 14.18-alpine
RUN mkdir -p /app
WORKDIR /app
COPY package*.json /app
RUN npm install
COPY . /app
```

Tabla 39 Archivo Dockerfile para el Backend (Elaboración propia).

Luego se debe crear el repositorio en el Docker Hub, este repositorio se usa para direccionar desde los services de Kubernetes.

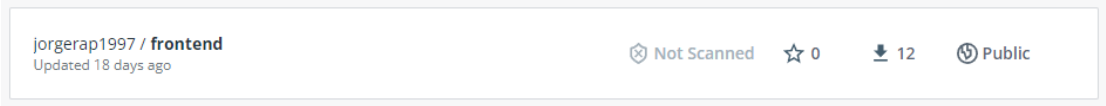


Fig. 41 Repositorio del Frontend en Docker hub (Elaboración propia).

Luego se procede a crear el Tambié-compose, de la misma manera se debe redireccionar al nombre del repositorio que se creó para el Frontend.

```
version: "3.6"
services:
  node:
    image: jorgerap1997/frontend:latest
    restart: always
    command: npm start
    ports:
      - "4200:4200"
```

Tabla 40 Definición del archivo Tambié-compose para el Frontend (Elaboración propia).

Al final ejecutamos los archivos de Docker y además subimos las imágenes al repositorio de Docker Hub.

De la misma manera en Kubernetes se tiene creado Deployment y el Service, en donde se ejecutan los archivos manifiestos para tener el servicio activo.

```
jarevalop1@Azure:~/angular-frontend$ kubectl get pods
NAME                       READY   STATUS              RESTARTS   AGE
angular-9d4977886-5lt9k    0/1     ContainerCreating   0          14s
```

Fig. 42 Frontend en corriendo en ambiente de producción (Elaboración propia).

4.5 IMPLEMENTACION DE LA ARQUITECTURA SISTEMA RECOMENDADOR CON DOCKER Y KUBERNETES

Para el sistema recomendador también se creó una imagen en Docker. Esta imagen fue creada con Python. Dentro del proyecto se tiene el archivo requirements.txt, el cual contiene todas las librerías necesarias. Se define el archivo Dockerfile con Python.

```
FROM python:3.8
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY ./requirements.txt /code/requirements.txt
RUN python -m pip install --upgrade pip
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
COPY ./app /code/app
```

Tabla 41 Archivo Dockerfile para el sistema recomendador (Elaboración propia).

También se ha creado un repositorio para el sistema recomendador. Aquí tendremos todas las actualizaciones que se realicen a futuro.

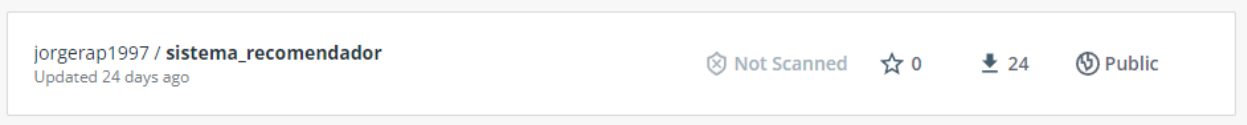


Fig. 43 Repositorio del sistema recomendador en Docker hub (Elaboración propia)

Al final se ejecuta en Kubernetes el Deployment y el Service. Cuando se despliega el sistema recomendador tiene conexión con el Frontend ya que el sistema recomendador actúa enviando y recibiendo peticiones por parte del usuario final. En el capítulo 3 se explica más a detalle cómo se está comunicando con el Frontend.

```
jarevalop1@Azure:~/sistema_recomendador$ kubectl get pods recomendador-b79fdb9d9-vknn5
NAME                              READY   STATUS              RESTARTS   AGE
recomendador-b79fdb9d9-vknn5   0/1     ContainerCreating   0          37s
jarevalop1@Azure:~/sistema_recomendador$
```

Fig. 44 Sistema recomendador en ambiente de producción (Elaboración propia).

4.6 IMPLEMENTACION DE LA ARQUITECTURA BACKEND

Estructura de la aplicación.

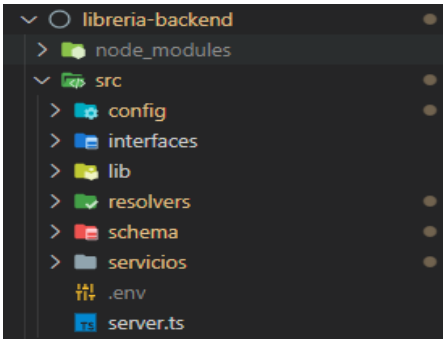


Fig. 45 Estructura de la aplicación backend (Elaboración propia)

- **config:** Directorio que contiene las constantes que se usan en el proyecto, la configuración de las variables de entorno y la cadena de conexión con Nodemailer
- **interfaces:** Directorio que contiene todas las interfaces, que se las define como un contrato donde se especifica las cosas que debe tener para implementar.
- **lib:** Directorio que contiene los archivos de configuración a la base de datos, autenticación y a la API de Stripe.
- **resolvers:** Directorio que contiene los archivos que dan solución a lo definido en el esquema.
- **schema:** Contiene todas las definiciones de los diferentes tipos de raíz (Query, Mutation, Suscription)
- **servicios:** Incluye cada uno de los servicios que permiten que las diferentes operaciones que usan dentro de la aplicación sean reutilizables.

PROCESO.

Creación del Servidor Node Express con los ajustes necesarios.

Para configurar el servidor se emplea Express que brinda la posibilidad de utilizar un middleware que se trata de una función que tiene accesos tanto al objeto de petición como al de respuesta.

El orden en el que se carga el middleware es importante.

En la implementación se incluye las variables de entorno que permiten identificar el entorno de trabajo (desarrollo / producción) y de acuerdo con ello cargar la configuración correspondiente, se realiza la configuración de Apollo Server, al que se pasa 2 parámetros el **schema** donde está definido la estructura de la API y como segundo parámetro **introspection** con su valor en true para poder visualizar el **schema** en GraphQL Playground, posterior a ello se aplica el middleware de la aplicación.

Schema GraphQL – Definiendo las primeras especificaciones de tipos

El definir un schema es importante porque con ello definimos el contrato de la API.

Ver anexo 9

Primero se ha definido el tipo de raíz Query para obtener la lista de usuario, los corchetes hacen referencia a una lista, si se requiere que el contenido de la lista no sea nulo se emplea [!], para que la lista no devuelva nulo se emplea [!].

Definir un resolver.

Tanto el nombre que se definió en el esquema con el que se define en el resolver debe tener el mismo nombre.

Definir el tipo de raíz a la que se va a dar solución

Para dar solución al Query se pasan 4 parámetros por lo general se emplean los 3 primeros.

1) **root:** Información de raíz, por defecto indefinido en queries, mutations y subscriptions, al trabajar con los resolvers de tipos se tiene la información del padre.
2) **args**: Los argumentos, si en el esquema en el tipo de definición se agrega parámetros estos deben incluirse en este apartado.
3) **context:** Un objeto que va a compartir información en todos los resolvers, esto se trae de Apollo Server con información de: token, fuente de datos, instancia de la base de datos, instancia de la configuración de Stripe, etc.
4) **info:** Se emplea para consultar información de la operación que se está ejecutando de manera detallada.

Luego entre las llaves se debe dar solución a lo que se ha definido en el tipo de definición en el schema.

Ver anexo 10

Servidor – Conexión a la base de datos MongoDB

El Stack que se emplea permite una comunicación muy sencilla entre el cliente y el servidor al utilizar JSON (JavaScript Object Notation), para consultar la información se puede realizar a través de consola o empleado clientes como Robo 3T o MongoDBCompass.

A las tablas se las conocen con el nombre de documentos y a los registros como colecciones.

Servidor – Añadir conexión de base de datos al contexto de Apollo Servidor

El primer paso es añadir mediante la inicialización del servidor y luego al contexto para finalmente agregar a la configuración de Apollo Server, lo que permite hacer uso de la instancia mediante el context.

Ver anexo 11.

Autenticación JWT – Creando la clase y función para firmar tokens.

Ver anexo 12.

El primer parámetro dentro de la función sign:

1.**Payload**: Privilegios, información del usuario, etc. Es un parámetro obligatorio

2.**Palabra secreta**: Obligatoria

3.**Opcional**: Se emplea para especificar el tiempo de caducidad.

Envío automático de correos con Nodemailer.

Se utiliza para enviar correos de:

- Activación de cuenta
- Recuperación de contraseña
- Confirmación de pedidos.

Como seguridad se ha añadido una contraseña de aplicación que brinda la ventaja de no tener que usar la contraseña de la cuenta de Gmail, si tal vez se roban la contraseña simplemente se desactiva la aplicación.

Para enviar un correo se requiere

- Definir dentro del esquema la definición del tipo para enviar el correo.
- Añadir el tipo de objetos que se necesita.
- Dar solución en los resolvers.

API Stripe Pagos Únicos

En este apartado solo se emplea para simular los pagos debido que para dar de alta la cuenta y usarla en producción se necesita tener una empresa formada.

PROCESO

- Formulario para recopilar la información de pago de usuario.

- Enviar información a Stripe para crear Token.
- Enviar la información de pago al servidor con el Token.
- El servidor usa el Token para almacenar el id del cliente de Stripe.

Para procesar los pagos Stripe usa el formato cero decimales, para el dólar la unidad es 100 que se debe multiplicar por el valor a pagar y este valor se tiene que dividir para 100, que da como resultado lo que se debe pagar en Stripe.

El valor minino es 0.50 dólares y el máximo 999,999.99

4.7 IMPLEMENTACION DE LA ARQUITECTURA FRONTEND

Estructura de la aplicación.

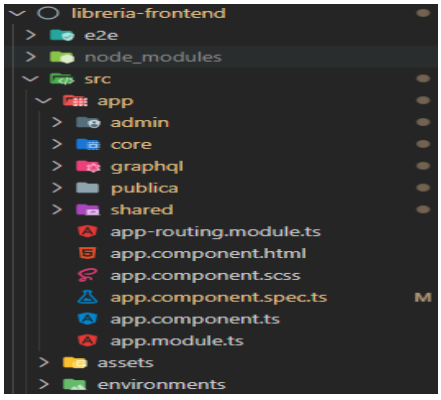


Fig. 46 Estructura de la aplicación frontend (Elaboración propia)

- **admin:** Directorio que contiene toda la estructura para realizar las acciones como administrador de la tienda.
- **core:** Directorio que contiene componentes especiales como: la definición de constantes, la definición de interfaces y lo más importante las guards para la protección de rutas y los servicios que se compartirán dentro de la aplicación.
- **graphql:** Contiene cada uno de los módulos, operaciones (fragments,mutation,query, subscription) y servicios para trabajar con GraphQL.

- **publica:** Directorio que contiene toda la estructura para la parte publica de la tienda que se divide en:
    - **core:** Se tiene componentes especiales como el carrito de compras, header, navbar, footer y servicios (Stripe, carrito de compras) que se comparten en la parte pública.
    - **paginas:** Cada una de las páginas que se mostraran al cliente donde se tiene: contacto, formularios (registro, inicio de sesión, pago, etc)**,** inicio, nosotros, perfil, productos, tienda.
- **shared:** Directorio que contiene componentes que se compartirán a nivel de toda la aplicación.

Cliente REST

Angular posee un objeto HTTP propio para poder realizar peticiones a un servicio REST, para ello se debe usar inyección de dependencias.

Servicios

Permiten modularizar la aplicación, debido a que es un elemento que permite reutilizar código y que tenga una única responsabilidad.

Integración de Stripe con Angular

Para ello se necesita configurar los valores para la conexión a la API de GraphQL, para ello Angular provee una carpeta **enviroment** en la que se tiene dos archivos:

- Configuración local
- Configuración para producción.

En cada uno de ellos se debe especificar los valores que se necesitan dependiendo del proyecto, para la aplicación del proyecto se tiene la conexión a la API GraphQL, al websocket que permite realizar las suscripciones y la conexión a la API de Stripe.

Los valores anteriores se deben integrar en el fichero de configuración del Módulo de Stripe para hacer su uso en la aplicación de angular.

Integración de Angular con el Sistema Recomendador.

Para ello se crea un servicio empleando el método **POST** para poder acceder a la API

Ver anexo 13.

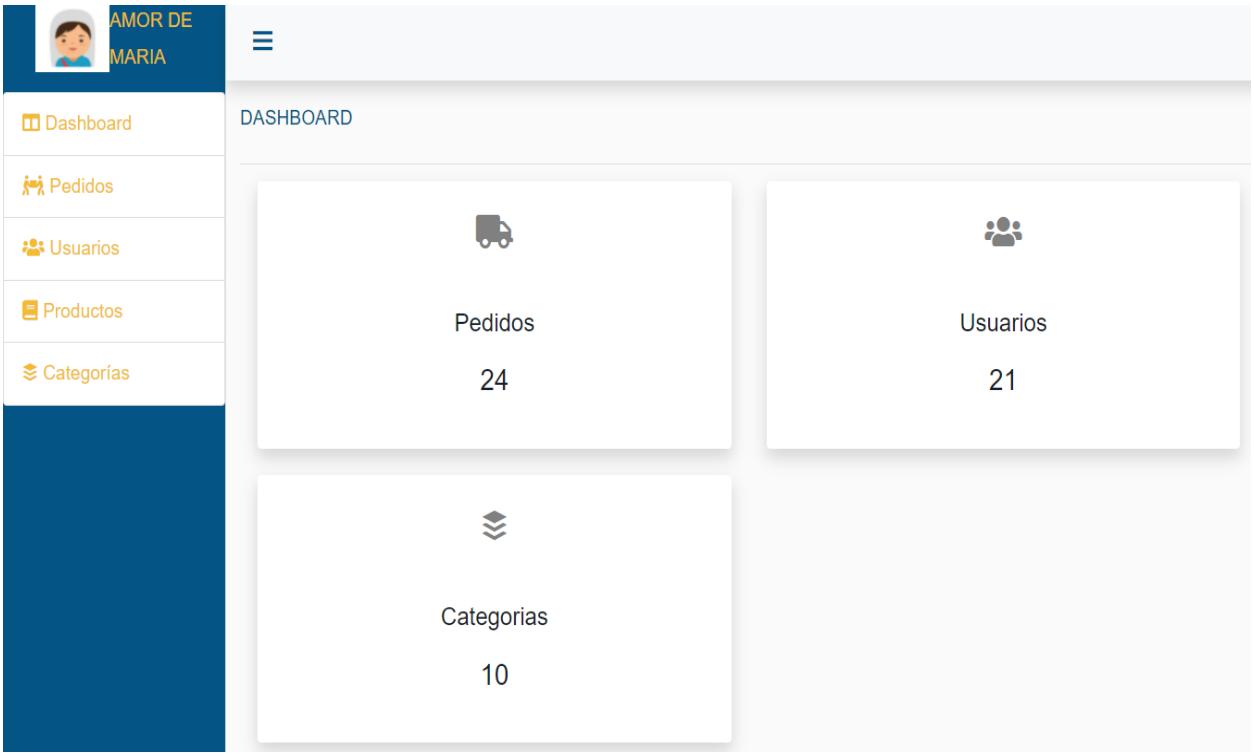


Fig. 47 Panel de administración del comercio electrónico (Elaboración propia)

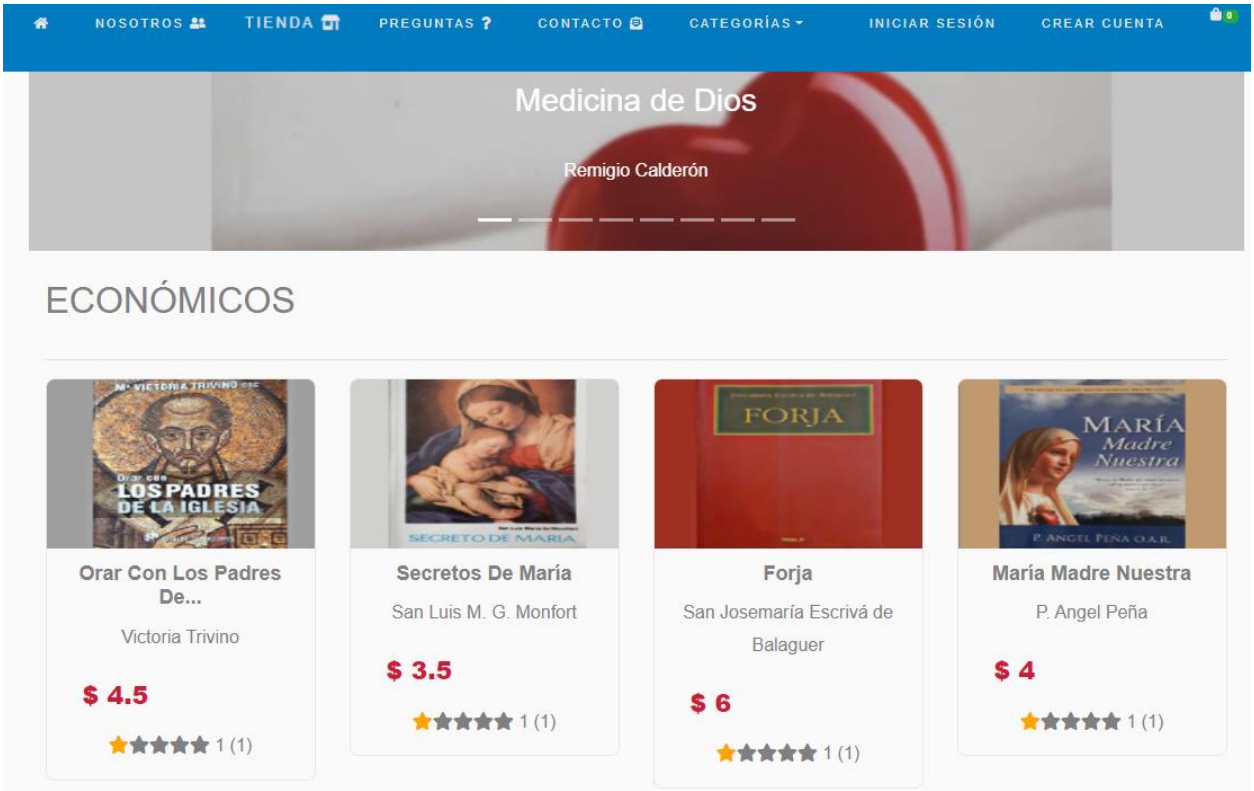


Fig. 48 Inicio comercio electrónico (Elaboración propia)

4.8 IMPLEMENTACION DE LA ARQUITECTURA SISTEMA RECOMENDADOR

Para el desarrollo del sistema recomendador se utiliza un entorno virtual Python donde se instala el Framework y partir de esto se procederá a construir la aplicación, un entorno virtual permite tener un usuario sin privilegios donde se instalan los diferentes módulos que se usan cuando el entorno es activado.

Estructura de la aplicación.

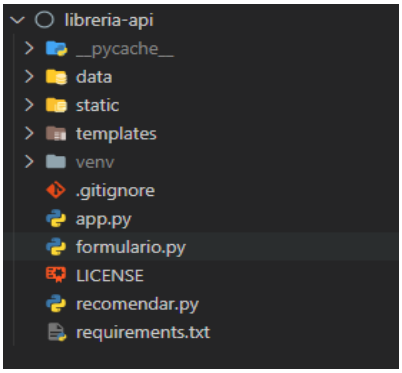


Fig. 49 Estructura de la aplicación Sistema Recomendador (Elaboración propia)

- **data**: En este directorio se encuentras los **datasets** de las compras de los usuarios y los productos de la tienda, estos valores se recuperan directamente de la base de datos.
- **static**: En este directorio se encuentra el archivo de estilos para la vista.
- **templates**: En este directorio se encuentran los archivos de las vistas correspondientes a: un archivo base, un formulario para buscar y uno para mostrar los resultados de las recomendaciones.
- **app.py:** Archivo donde se encuentra creado y configurado el servidor.
- **formulario.py:** Lógica para el funcionamiento del formulario.
- **recomedar.py:** Lógica que contiene todo el proceso para realizar la recomendación

Para las recomendaciones el sistema usa la técnica de **Filtro Colaborativo** que su base principal es utilizar otros usuarios como base para recomendar productos al usuario de entrada, para encontrar la similitud se emplea la función de **Correlación de Pearson**.

Proceso.

- Elegir un usuario con libros comprados (pedidos completados.)
- Basado en la categoría de los libros comprados, encuentra a los primeros n vecinos
- Obtener el registro de los libros comprados por el usuario para cada vecino.
- Calcular un puntaje de similitud.
- Recomendar los registros con la puntuación más alta.

4.9 IMPLEMENTACIÓN DE PRUEBA DE CARGA.

Para realizar pruebas con respecto al Backend vamos a basarnos en un requerimiento el mismo se verifica el límite que alcance en peticiones al servidor del Backend. Asimismo, se verificará el número de peticiones que el servidor responde cuando el cliente envié solicitudes al servidor. Para

ello vamos a utilizar un Endpoint el cual será de tipo POST. Además, se tiene un objeto de tipo Json con los parámetros que necesita el Endpoint. Ejemplo del query variable de tipo Json:

```
{
"correo":"amordemaria.com",
"password":"123"
}
```

Se utiliza la herramienta JMeter para realizar la prueba de peticiones. Al momento que se realiza la petición, se tendrá como respuesta un Token y los datos que se registró. Para verificar debe agregar el query y las query variables.

```
query iniciarSesion($correo:String!, $password:String!){
  login(correo:$correo, password:$password) {
    estado
    mensaje
          token
          usuario{
      id
      nombre
      apellido
      rol
    }
  }
}
```

Tabla 42 Query para el inicio de sesión (Elaboración propia).

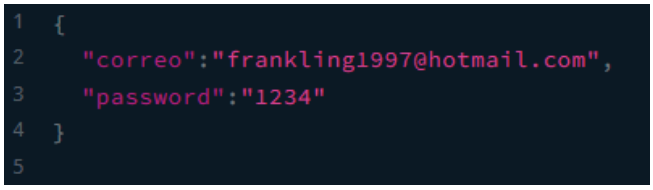


Fig. 50 Variables que recibe el Query (Elaboración propia)

Tenemos el correo y la contraseña como parámetro. Esta petición consulta en la base y devuelve el Token de acceso.

```
{
  "data": {
    "login": {
      "estado": true,
      "mensaje": "Bienvenido. Iniciaste sesión correctamente.",
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3VhcmlvIjp7Il9pZCI6IjYxZmVjN2JlZThhNTJmMDAxM2UzYmNj
ZCIsIm5vbWJyZSI6IkZyYW5rbGluIiwiYXBlbGxpZG8iOiJHdWFsbHBhIiwiY29ycmVvIjoiZnJhbmtsaW5nMTk5N0Bob3
RtYWlsLmNvbSIsInJvbCI6IkNMSUVOVEUiLCJpZCI6IjQiLCJhY3Rpdm8iOnRydWV9LCJpYXQiOjE2NDQwODc0NTUsImV4
cCI6MTY0NDA5MTA1NX0.RvAvdsPOW4f34haupFRLKbdaiTUD_RIvjGck02U2-XA",
      "usuario": {
        "id": "4",
        "nombre": "Franklin",
        "apellido": "Guallpa",
        "rol": "CLIENTE"
      }
    }
  }
}
```


Fig. 51 Respuesta del servidor (Elaboración propia)

Existen varios métodos que realizan procesos con respuesta de un token. Las pruebas se pueden realizar a mano, pero esto tomaría mucho tiempo en tener resultados ya que existen varios métodos que se utilizaron para el proyecto. Para tener resultados de pruebas vamos a interactuar con el registro de usuarios verificaremos el tiempo de respuesta, errores al realizar peticiones y otras desventajas que se encuentre en el servidor. Usando la herramienta JMeter se va a analizar todos los resultados en tiempo de respuesta del Backend.

4.10 IMPLEMENTACIÓN DE PRUEBA DE STRESS

En las pruebas de Stress vamos a realizar directamente al servidor verificando el número de replicas cuando exista sobrecarga en el servidor. Para verificar como se replica el servidor se va a realizar las pruebas con el servidor del Backend.

Luego, creamos un service de tipo LoadBalancer para el Backend para realizar pruebas externas. Esto me permitirá acceder desde el exterior mediante la Ip del servicio y el puerto en la que este levantado, en este caso el 2002.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: node
  labels:
    app.kubernetes.io/name: node
    app.kubernetes.io/component: backend
spec:
  type: LoadBalancer
  ports:
  - port: 2002
    targetPort: 2002
  selector:
    app.kubernetes.io/name: node
    app.kubernetes.io/component: backend
```

Fig. 52 Servicio para la prueba de Stress. (Elaboración propia).

Se ejecutarán peticiones verificando el límite que soporta el Clúster.

4.11 RESULTADOS

Toda la arquitectura se desarrolló cumpliendo los objetivos específicos al inicio del proyecto. Los resultados se obtuvieron en base a los requerimientos no funcionales. Tomando en cuenta los requerimientos se escogió los más importantes para realizar pruebas y evaluar el rendimiento en producción. A continuación, se mostrará los resultados que se obtuvieron utilizando la herramienta JMeter.

Posteriormente, se realizó las pruebas con peticiones al Backend. En esta primera prueba se llevó a cabo con la conexión a internet a una velocidad lenta.

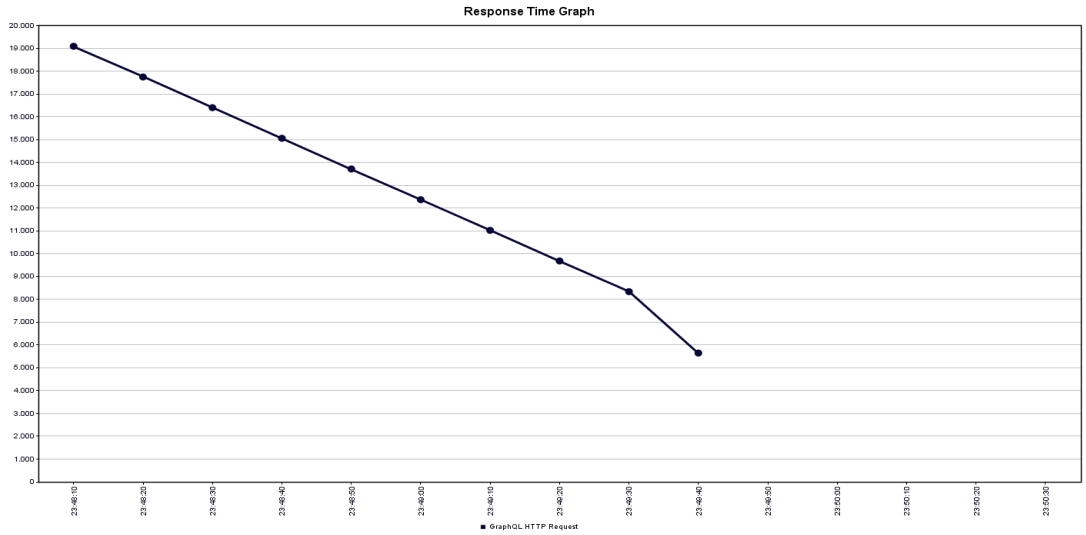


Fig. 53 Grafica de Test de carga con conexión baja (Elaboración propia)

Etiqueta	# Muestras	Media ↑	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent
GraphQL HTTP ...	9035	5306	271	49523	7647,51	13,22%	18,1/sec	26,54	
Total	9035	5306	271	49523	7647,51	13,22%	18,1/sec	26,54	

Fig. 54 Test de carga con conexión baja (Elaboración propia).

En los resultados se puede ver que el error es bien alto ya que en resultados generales muestra el 13,22 %, se realizó 1000 peticiones con la calidad del internet en un estado bajo.

También, se cambió la configuración de la conexión a internet, en esta ocasión se cambió el estado a nivel alta en conexión y los resultados fueron óptimos.

Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec
GraphQL HTTP R...	4000	5504	4063	7753	21000	21010	416	136277	5,53%	29,3/sec	21,53	16,00
Total	4000	5504	4063	7753	21000	21010	416	136277	5,53%	29,3/sec	21,53	16,00

Fig. 55 Test de carga con conexión alta (Elaboración propia)

Los resultados son óptimos, se puede ver que el error es de 5,45 %, de la misma manera se realizó con 4000 muestras. Los tiempos que están en milisegundos no superan el segundo. Se logra observar resultados óptimos con un buen rendimiento en el servidor.

Podemos ver las características de cada registro en su petición.

Muestra #	Tiempo de co...	Nombre del hilo	Etiqueta	Tiempo ...	Estado	Bytes	Sent Bytes	Latency	Connect Time(...
1	16:04:18.071	LoginUsuarios ...	GraphQL HTTP...	272		1322	591	272	135
2	16:04:18.069	LoginUsuarios ...	GraphQL HTTP...	290		1322	591	290	141
3	16:04:18.069	LoginUsuarios ...	GraphQL HTTP...	290		1322	591	290	140
4	16:04:18.069	LoginUsuarios ...	GraphQL HTTP...	300		1322	591	300	144
5	16:04:18.069	LoginUsuarios ...	GraphQL HTTP...	300		1322	591	300	141
6	16:04:18.071	LoginUsuarios ...	GraphQL HTTP...	292		1322	591	292	141
7	16:04:18.069	LoginUsuarios ...	GraphQL HTTP...	300		1322	591	300	144
8	16:04:18.070	LoginUsuarios ...	GraphQL HTTP...	300		1322	591	300	144
9	16:04:18.075	LoginUsuarios ...	GraphQL HTTP...	302		1322	591	302	140
10	16:04:18.071	LoginUsuarios ...	GraphQL HTTP...	307		1322	591	307	144
11	16:04:18.077	LoginUsuarios ...	GraphQL HTTP...	307		1322	591	307	147
12	16:04:18.074	LoginUsuarios ...	GraphQL HTTP...	311		1322	591	311	150
13	16:04:18.080	LoginUsuarios ...	GraphQL HTTP...	308		1322	591	308	144
14	16:04:18.080	LoginUsuarios ...	GraphQL HTTP...	309		1322	591	309	144
15	16:04:18.070	LoginUsuarios ...	GraphQL HTTP...	322		1322	591	322	154
16	16:04:18.081	LoginUsuarios ...	GraphQL HTTP...	312		1322	591	312	143
17	16:04:18.084	LoginUsuarios ...	GraphQL HTTP...	315		1322	591	315	140
18	16:04:18.073	LoginUsuarios ...	GraphQL HTTP...	327		1322	591	327	151
19	16:04:18.084	LoginUsuarios ...	GraphQL HTTP...	327		1322	591	327	140
20	16:04:18.077	LoginUsuarios ...	GraphQL HTTP...	336		1322	591	336	147
21	16:04:18.090	LoginUsuarios ...	GraphQL HTTP...	325		1322	591	325	134
22	16:04:18.077	LoginUsuarios ...	GraphQL HTTP...	340		1322	591	340	147
23	16:04:18.083	LoginUsuarios ...	GraphQL HTTP...	336		1322	591	336	141
24	16:04:18.089	LoginUsuarios ...	GraphQL HTTP...	340		1322	591	340	144

Fig. 56 Resultados de 1000 muestras (Elaboración propia)

Se observa el tiempo de respuesta del servidor, la respuesta no pasa del segundo. La ilustración muestra los resultados de las 24 muestras. Podemos ver resultados óptimos por parte del servidor. A continuación, se muestra los resultados de manera gráfica en JMeter.

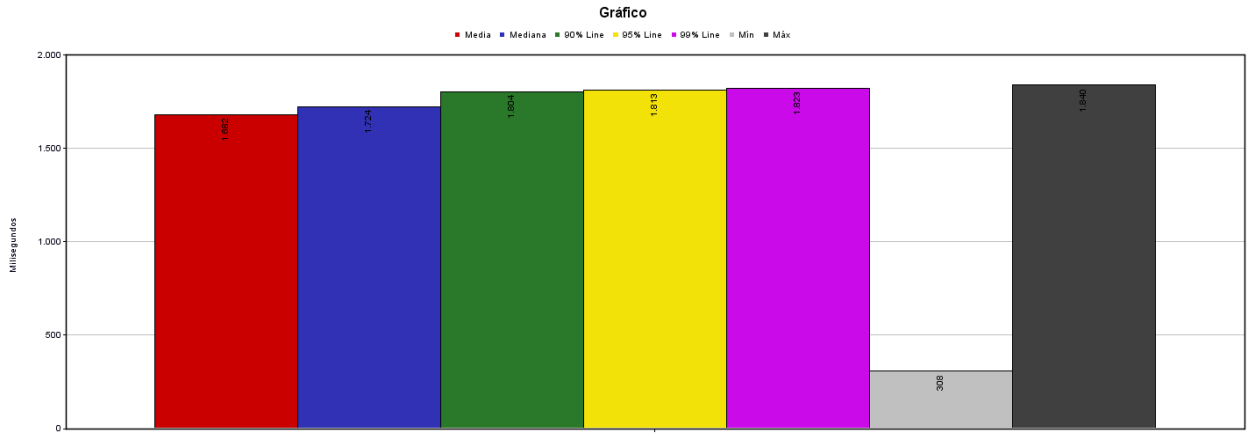


Fig. 57 Gráfica de resultados en JMeter (Elaboración propia).

En la gráfica podemos ver los tiempos de respuesta que son óptimos. Además, se realizaron el testeo con 1000 muestras y se puede verificar que la última petición en tiempo de respuesta es de 1840 ms dando como resultado un excelente tiempo de respuesta. En cuanto a la desviación se puede observar que el tiempo es de 1622 ms. Centrándonos directamente al servidor, verificar que el tiempo aproximado es de 1804 en un minuto, en cuanto a la media la aproximación es de 1724 ms, las peticiones no sobrepasan los 17 ms.

Verificando los resultados durante el intervalo de tiempo logramos que el tiempo de respuesta que tiene variación durante las peticiones hacia el servidor.

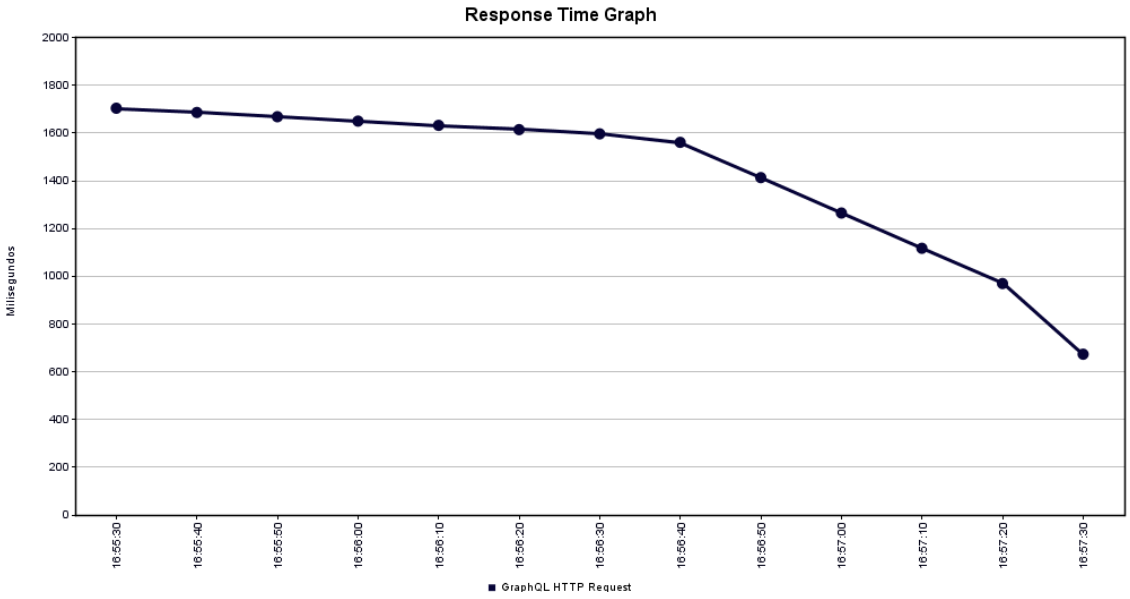


Fig. 58 Gráfica del tiempo de respuesta (Elaboración propia).

La arquitectura del sistema recomendador se desarrolló cumpliendo los objetivos específicos al inicio del proyecto. Los resultados se obtuvieron en base a los requerimientos funcionales. Tomando en cuenta los requerimientos se escogió el más importantes para realizar pruebas. A continuación, se mostrará los resultados que se obtuvieron utilizando las herramientas de Karma y Jasmine.

La prueba escrita en Jasmine permite validar si el servicio se está creando correctamente y para verificar si realmente se está conectando al servicio y se está realizando las recomendaciones se necesita un espía **httpClient** juntamente con un ambiente para simular las compras.

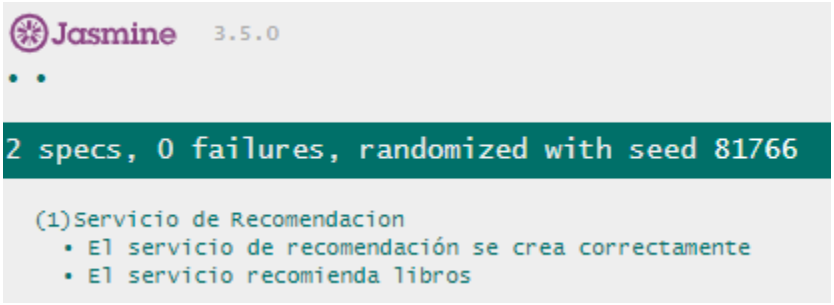


Fig. 59 Test para validar el sistema recomendador dentro del aplicativo del comercio electrónico

METRICAS DE EVALUACION DEL SISTEMA RECOMENDADOR
RECALL

Permite ver que elementos que le gustan a un usuario fueron recomendados.

FORMULA

$$Recall = \frac{tp}{tp + fn}$$

tp = Cantidad de elementos recomendados

fn = Total de elementos que le gustan a un usuario

$$Recall = \frac{3}{3 + 2}$$

$$Recall = 0.6$$

Si el recall es mayor mejor son las recomendaciones.

CAPÍTULO 5 CONCLUSIONES, RECOMENDACIONES Y TRABAJOS FUTUROS

5.1 CONCLUSIONES

- Revisar artículos y documentación acerca de GraphQL y sistemas de recomendación permitió definir la base principal de la arquitectura tecnológica. De tal manera, se revisó cada una de las ventajas y desventajas, los retos que implicaba y con ello poder hacerle frente a los obstáculos y problemas que se fueron presentando al largo del desarrollo del proyecto. [Primer objetivo].
- El Stack MEAN permite desarrollar una potente aplicación Web pues utiliza con único lenguaje JavaScript lo que permite a los desarrollares tener un solo lenguaje en las diferentes partes del proyecto, además el uso de MongoDB hace que la aplicación en cuanto a la carga de datos e interacción sea un proceso muy ágil debido al uso de JSON y finalmente brinda facilidad para mejorar e integrar nuevas funcionalidades tanto con Angular en el frontend como Express en el backend de una manera sencilla y facilitando su despliegue. [Tercer objetivo].
- Stripe es una excelente alternativa al trabajar con pasarelas de pagos para un comercio electrónico además que es una que menos cobra en cuanto a comisiones, el problema que se presenta es que todavía no está operativa en Ecuador, pero está en continuo crecimiento ahora mismo se encuentra en 46 países, lo mejor de Stripe es su excelente documentación y su forma de implementación en los diferentes lenguajes asimismo cada uno de los ejemplos que se muestran en la documentación son realmente fáciles de entender. [Tercer objetivo].
- El desarrollo del sistema recomendador se realizó con Flask, un framework minimalista tiene una curva de aprendizaje muy baja que se complementa con una gran cantidad de

librerías que le brindan muchas funcionalidades. Además, esta tecnología usa Python como Lenguaje de programación que brinda una gran profundidad en el análisis de los datos para desarrollar aplicaciones en el campo de la Inteligencia Artificial; y su incorporación representa una forma novedosa en un mercado digital y ha sido muy interesante trabajar con Filtros Colaborativos. [Cuarto objetivo].

- La infraestructura como código podemos automatizar muchos procesos y tareas con Terraform. Se definió una arquitectura la cual se desliga automáticamente la base de datos, el Backend, el Frontend, sistema recomendador. En suma, se conoció el alcance y el límite que posee esta herramienta. Asimismo, se puede conectar con varios proveedores en la nube, en este caso se utilizó Azure. Por consiguiente, se automatizó el aprovisionamiento y la creación de la infraestructura del proyecto mejorando un ambiente para producción. Con la infraestructura se verificó que se tiene menos tiempo y esfuerzo en administrar los servicios, también se tiene menos riesgo de romper un ambiente luego de un deployment. [Segundo Objetivo]
- La administración de la arquitectura no es totalmente autónoma ya que se deben crear contendores con otra herramienta, en este caso se utilizó Docker para crear imágenes de cada parte de la aplicación así manteniendo una infraestructura para cada contendor y luego realizar el despliegue con un orquestador que es Kubernetes. [Segundo Objetivo]
- Se utilizó Kubernetes para orquestar toda la infraestructura dividiendo en servidores cada parte de la aplicación, el cual un nodo de Kubernetes contiene cada imagen de la aplicación. Estas imágenes fueron escritas en archivos manifiestos con configuraciones para cada arquitectura el cual estos están en Pods y cada pod es óptimo y escalable para evitar caídas o perdidas de datos. [Sexto Objetivo]

5.2 RECOMENDACIONES

- Tener los temas claros sobre la infraestructura, esto agilizará las configuraciones para obtener buenas prácticas y evitar fallos en desligues. Primero se debe realizar pruebas de manera local pues para usar un servicio en la nube tiene un precio a pagar y cada uso de recursos aumenta el costo.
- Tener conocimiento claro para crear imágenes de Docker ya que facilitará el despliegue las aplicaciones en contenedores con características definidas para la aplicación.
- Revisar toda la documentación sobre Kubernetes para tener buenas prácticas en uso de los nodos y llevar buenas replicas sin perder datos, saber cómo declarar archivos manifiestos con sus respectivas configuraciones con certificados para portar del protocolo HTTPs así manteniendo conexiones seguras y evitando accesos por terceros.
- Utilizar plantillas para un desarrollo más rápido en el apartado del Frontend.
- Desarrollar e implementar el aplicativo de manera local para realizar pruebas y validaciones antes de su subida a producción.
- Usar software de control de versiones para comparar, fusionar o poder restaurar versiones de la aplicación que se esté desarrollando y además tener una copia del código ante cualquier imprevisto que pueda suceder.
- Verificar la compatibilidad de las tecnologías, muy importante resaltar que la versión 2 de Apollo Client es compatible con versiones anterior a Angular 11.

5.3 TRABAJOS FUTUROS

El Proyecto fue desarrollado con tecnologías que están en continuo crecimiento, por lo que la arquitectura se puede mejor. A continuación, se mencionan algunos posibles trabajos futuros:

- Modificar el Stack MEAN a MERN, en lugar de Angular usar React ya que tanto GraphQL Y React son desarrollados y mantenidos por Facebook lo que puede brindar grandes ventajas en el desarrollo.
- Implementar GraphQL con Relay herramienta similar a Apollo Client desarrollada por Facebook su objetivo es ofrecer interacciones instantáneas de respuesta a la interfaz de usuario.
- Implementar Stripe en un entorno de Producción y comparar con otras alternativas dentro del Mercado.

En cuanto a la implementación de la infraestructura como código se puede realizar mejoras en las arquitecturas utilizando otras herramientas generando optimización en tiempos y evitando grandes configuraciones para despliegue de servidores.

A continuación, se menciona algunos posibles trabajos futuros:

- Integración de las imágenes en Github para realizar actualizaciones de versiones de manera directa y ya no realizar le proceso de configuración para Docker Hub.
- Integrar la configuración de las redes definidas por el software el cual permite versionar la configuración y automatizar los cambios. AWS VPC es una implementación de SDN.
- Implementar infraestructura multinube estandarizada a gran escala teniendo un flujo de trabajo para todas las nubes.

REFERENCIAS

Rodríguez, K. (2020). Desarrollo de un envoltorio del api-rest de mendeley con GraphQL. [archivo PDF]. Recuperado de: http://repositorio.utn.edu.ec/bitstream/123456789/10292/2/04%20ISC%20546%20TRABAJO%20GRADO.pdf

Porcello, E. Banks. A. (2018), Learning GraphQL: Declarative Data Fetching for Modern Web Apps. [archivo E-BOOK]. Recuperado de: https://vdoc.pub/download/learning-graphql-declarative-data-fetching-for-modern-web-apps-17qtlfm47tc8

Ghebremicael, E. S., (2017), Transformation of REST API to GraphQL for Open TOSCA. [archivo PDF]. Recuperado de: https://elib.uni-stuttgart.de/handle/11682/9369

Schwaber, K. Sutherland, J. (2013). "La Guía definitiva de Scrum: Las Reglas de Juego" (1era ed.) [archivo PDF]. Recuperado de: https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf

Banchoff, M. (2019). Infraestructura como Código Caso de estudio: Cientópolis. [archivo PDF]. Recuperado de: http://sedici.unlp.edu.ar/bitstream/handle/10915/86142/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Garrido, A. (2018). Integrando fuentes heterogéneas de datos en Web con GraphQL (1era ed.) [archivo PDF]. Recuperado de: https://dehesa.unex.es/bitstream/10662/7973/1/TFGUEX_2018_Garrido_Roman.pdf

Utrera, E. Cuevas, A. (2017). Sistemas de recomendación semánticos: Una revisión del Estado del Arte. [archivo PDF]. Recuperado de: https://www.researchgate.net/publication/319058324_Sistemas_de_recomendacion_semanticos_Una_revision_del_Estado_del_Arte

Sommerville, I. (2005). Ingeniería del software. [archivo PDF]. Recuperado de: http://zeus.inf.ucv.cl/~bcrawford/AULA_ICI_3242/Ingenieria%20del%20Software%207ma.%20Ed.%20-%20Ian%20Sommerville.pdf

Coalla, J. (2018). Proyecto de Fin de Grado Lets's Get Fitness – Desarrollo de una aplicación web con MEAN Stack. [archivo PDF]. Recuperado de: https://oa.upm.es/50969/1/TFG_%20JOSE_LUIS_COALLA_CENCERRADO.pdf

María, C. (2015). Gestión de contenedores Docker. [archivo PDF]. Recuperado de: https://dit.gonzalonazareno.org/gestiona/proyectos/2015-16/proyectoDK.pdf

Edward. Núñes, (2012). Sistemas de recomendación de contenidos para libros inteligentes. [archivo PDF]. Recuperado de: http://di002.edv.uniovi.es/~cueva/investigacion/tesis/Tesis-Edward.pdf

Stefani. Carvajal., (2018). Prototipo de un sistema de recomendación de libros para la biblioteca. [archivo PDF]. Recuperado de: https://bibliotecadigital.univalle.edu.co/bitstream/handle/10893/17465/0586632.pdf?sequence=1&isAllowed=y

The GraphQL-Fundation. (2022). Introduction to GraphQL. GraphQL. https://graphql.org/learn/

Apollo-GraphQL. (2021). Write query resolvers Learn how a GraphQL query fetches data. Apollo Docs. https://www.apollographql.com/docs/tutorial/resolvers/

Stubailo, S. (19 de mayo de 2021). Three ways to represent your GraphQL schema. Apollo Blog.

https://www.apollographql.com/blog/backend/schema-design/three-ways-to-represent-your-graphql-schema/

Gatsby. (2022). Introducing GrapiQL. Documentation.

https://www.gatsbyjs.com/docs/how-to/querying-data/running-queries-with-graphiql/

Valdez, H. (22 de mayo de 2020). Graphlql playground review completa. Holiviel Valdez.

https://holivielvaldez.dev/es/blog/graphlql-playground-review-completa/#:~:text=El%20graphql%20playground%20es%20un,%2C%20documentos%20interactivos%20y%20colaboraci%C3%B3n

Apollo-GraphQL. (2022). Write query resolvers Learn how a GraphQL query fetches data. Apollo Docs. https://www.apollographql.com/docs/tutorial/resolvers/

Apollo-GraphQL. (2022). Introduction to Apollo Server. Apollo Docs. https://www.apollographql.com/docs/apollo-server/

The-Guild. (2022). Introduction. GraphQL Tools. A set of utilities for faster GraphQL development.

https://www.graphql-tools.com/docs/introduction

Alarcón, M. (19 de enero de 2015). Qué es el stack MEAN y cómo escoger el mejor para ti. Campus MVP.

https://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-para-ti.aspx

Pérez, A. (20 de abril de 2020). Cómo usar testing en Angular con Jasmine y Karma. Digital55. https://www.digital55.com/desarrollo-tecnologia/como-usar-testing-angular-jasmine-karma/

Softeng (s.f).Proceso y Roles de Scrum. Softeng Your competitive advantaje. https://ichi.pro/es/graphql-una-historia-de-exito-para-paypal-checkout-57737096788819

ICHI.PRO. (s.f). GraphQL: Una historia de éxito para Paypal Checkout. ICHI.PRO. https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html

Mario, Saffirio. (23 de julio de 2018). Tecnologías de la Información y Procesos de Negocios. Contenedores | Containers. *https://msaffirio.com/2018/07/23/contenedores-containers/*

Yanagishita. (21 de junio de 2017). Arquitectura docker. *https://www.cnblogs.com/13224ACMer/p/7062324.html*

Víctor, Cuervo. (2 de diciembre de 2019). Arquitectura Docker *https://www.arquitectoit.com/docker/arquitectura-docker/*

Devops, Latam. (13 de septiembre de 2019). ¿Qué es Docker? ¿Cómo funciona? *https://devopslatam.com/que-es-docker-como-funciona/*

Ximena, Rodríguez. (24 de julio de 2019). Qué es DockerFile. *https://openwebinars.net/blog/que-es-dockerfile/*

Maxtor. (3 de agosto de 2019). Balanceo de carga (Swarm y docker-compose) *https://seguridadzero.com/balanceo-de-carga-swarm-y-docker-compose*

RedHat. (15 de enero de 2020). ¿Qué es un clúster de Kubernetes? *https://www.redhat.com/es/topics/containers/what-is-a-kubernetes-cluster*

Ramon, Carrasco. (2020). Conceptos básicos de Kubernetes. *https://www.ramoncarrasco.es/es/content/es/kb/175/conceptos-basicos-de-kubernetes*

Google, LLC. (23 de abril de 2021). Administre un clúster de múltiples usuarios de GKE con espacios de nombres. *https://www.cloudskillsboost.google/focuses/14861?locale=es&parent=catalog&qlcampaign=7*

7-terr32-818

Programador, Clic. (2020). Introducción a Kubernetes-K3S ligero. *https://programmerclick.com/article/15581285410/*

Ekaterina, Novoseltseva. (3 de noviembre de 2020). Infraestructura Como Código: Beneficios Y Herramientas. *https://apiumhub.com/es/tech-blog-barcelona/infraestructura-como-codigo-beneficios-y-herramientas/*

Ignacio, Sánches. (3 de agosto de 2017). Infraestructura como Código (II): Terraform.

https://enmilocalfunciona.io/infraestructura-como-codigo-con-terraform-2/

Frankier, Flores. (29 de agosto de 2021). Qué es Infraestructura como Código
https://ifgeekthen.nttdata.com/es/infraestructura-como-codigo

Cleverdata. (2021). Sistemas de recomendacion de contenido con Machine Learning.
https://cleverdata.io/sistemas-recomendacion-machine-learning/

Juan, Pujol. (30 de mayo de 2020). Ingress en Kubernetes Desmitificado: ¿Qué lo diferencia de un NodePort o un LoadBalancer?
https://desarrollofront.medium.com/ingress-en-kubernetes-desmitificado-qu%C3%A9-lo-diferencia-de-un-nodeport-o-un-loadbalancer-b0cf060a6f8a

Dotcom-monitor. (2020). Pruebas de rendimiento en línea (carga y estrés) con LoadView
https://www.dotcom-monitor.com/wiki/es/knowledge-base/solucion-de-prueba-de-carga/

Education-wiki. (2022). ¿Qué es el JMeter?
https://es.education-wiki.com/9105507-what-is-jmeter

Hopla-software. (2022). Casos de éxito. Infraestructura
https://es.education-wiki.com/9105507-what-is-jmeter

Stefano, Benco. (15 de junio de 2020). Caso de éxito: Henneo, inteligencia artificial aplicada a la recomendación de noticias
https://www.hiberus.com/crecemos-contigo/caso-de-exito-henneo-inteligencia-artificial-aplicada-a-la-recomendacion-de-noticias/

ANEXOS

Anexo 1 Deployment de mongoDB (Elaboración propia)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo
  labels:
    app.kubernetes.io/name: mongo
    app.kubernetes.io/component: backend
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: mongo
      app.kubernetes.io/component: backend
  replicas: 1
  template:
    metadata:
      labels:
        app.kubernetes.io/name: mongo
        app.kubernetes.io/component: backend
    spec:
      containers:
        - name: mongo
          image: mongo:3.6.3
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              value: admin
            - name: MONGO_INITDB_ROOT_PASSWORD
              value: cuenca
          args:
            - --bind_ip
            - 0.0.0.0
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          ports:
            - containerPort: 27017
          volumeMounts:
            - name: mongodb
              mountPath: /var/lib/mongodb/
          volumes:
            - name: mongo
              persistentVolumeClaim:
              claimName: mongodb-pvc
```

Anexo 2 Servicio de mongoDB (Elaboración propia)

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: mongo
  labels:
    app.kubernetes.io/name: mongo
    app.kubernetes.io/component: backend
spec:
  type: ClusterIP
  ports:
  - port: 27017
    targetPort: 27017
  selector:
    app.kubernetes.io/name: mongo
    app.kubernetes.io/component: backend
```

Anexo 3 Deployment del Backend (Elaboración propia)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node
  labels:
    app.kubernetes.io/name: node
    app.kubernetes.io/component: backend
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: node
      app.kubernetes.io/component: backend
  replicas: 2
  template:
    metadata:
      labels:
        app.kubernetes.io/name: node
        app.kubernetes.io/component: backend
    spec:
      containers:
        - name: node
          image: jorgerap1997/server:latest
          command: ["/bin/sh", "-c", "sleep 40 && npm run start"]
          args:
            - --bind_ip
            - 0.0.0.0
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
```

```
        ports:
          - containerPort: 2002
```

Anexo 4 Servicio del Backend (Elaboración propia)

```
apiVersion: v1
kind: Service
metadata:
  name: node
  labels:
    app.kubernetes.io/name: node
    app.kubernetes.io/component: backend
spec:
  type: ClusterIP
  ports:
  - port: 2002
    targetPort: 2002
  selector:
    app.kubernetes.io/name: node
    app.kubernetes.io/component: backend
```

Anexo 5 Deployment del Frontend (Elaboración propia)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: angular
  labels:
    app.kubernetes.io/name: angular
    app.kubernetes.io/component: frontend
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: angular
      app.kubernetes.io/component: frontend
  replicas: 1
  template:
    metadata:
      labels:
        app.kubernetes.io/name: angular
        app.kubernetes.io/component: frontend
    spec:
      containers:
        - name: angular
          image: jorgerap1997/frontend:latest
          command: ["/bin/sh", "-c", "sleep 40 && npm start"]
          args:
```

```
          - --bind_ip
          - 0.0.0.0
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
          - containerPort: 4200
```

Anexo 6 Servicio del Frontend (Elaboración propia)

```
apiVersion: v1
kind: Service
metadata:
  name: angular
  labels:
    app.kubernetes.io/name: angular
    app.kubernetes.io/component: frontend
spec:
  type: ClusterIP
  ports:
  - port: 4200
    targetPort: 4200
  selector:
    app.kubernetes.io/name: angular
    app.kubernetes.io/component: frontend
```

Anexo 7 Deployment del sistema recomendado (Elaboración propia)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: recomendador
  labels:
    app.kubernetes.io/name: recomendador
    app.kubernetes.io/component: backend
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: recomendador
      app.kubernetes.io/component: backend
  replicas: 1
  template:
    metadata:
      labels:
        app.kubernetes.io/name: recomendador
        app.kubernetes.io/component: backend
```

```
    spec:
      containers:
        - name: recomendador
          image: jorgerap1997/sistema_recomendador:latest
          command: ["/bin/sh", "-c", "sleep 40 && python app/app.py"]
          args:
            - --bind_ip
            - 0.0.0.0
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          ports:
            - containerPort: 4000
```

Anexo 8 Servicio del sistema recomendador (Elaboración propia)

```
apiVersion: v1
kind: Service
metadata:
  name: recomendador
  labels:
    app.kubernetes.io/name: recomendador
    app.kubernetes.io/component: backend
spec:
  type: ClusterIP
  ports:
  - port: 4000
    targetPort: 4000
  selector:
    app.kubernetes.io/name: recomendador
    app.kubernetes.io/component: backend
```

Anexo 9 Definir el esquema

```
type Query{
    "Lista de usuarios registrados en la Base de Datos (Clientes/Administradores)"
    usuarios:[Usuario!]!
}
type Usuario{
    "Identificador único"
    id: ID!
    "Nombre/s"
    nombre: String!
    "Apellido/s"
    apellido: String!
    "Correo Electrónico"
```

```
    correo:String!
    "Contraseña asociada a la cuenta"
    contrasena: String !
    "Fecha de registro en la Base de datos"
    fechaRegistro: String !
    "Fecha de nacimiento"
    fechaNacimiento: String !
}
```

Anexo 10 Definir un resolver

```
import { IResolvers } from 'graphql-tools';
const resolversQuery: IResolvers ={
    Query:{
        usuarios(root,args,context,info){
            return [
                {
                    id:1,
                    nombre:'Marco',
                    apellido: 'Loja',
                    correo:'mloja@hotmail.com',
                    contrasena:'',
                    fechaRegistro: '',
                    fechaNacimiento: ''
                }
            ];
        }
    }
};
export default resolversQuery;
```

Anexo 11 Añadir conexión de base de datos al contexto de Apollo Server

```
const dasededatos = new BasedeDatos();
    const db = await dasededatos.init();
     const context = { db} ;
    const server = new ApolloServer({
        schema,
        introspection:true,
        context
    });
```

Anexo 12 Autenticación JWT – Creando la clase y función para firmar tokens

```
import { FECHA_EXPIRACION, MENSAJES, SECRET_JWT_KEY } from
'../config/constants';
```

```
import jwt from 'jsonwebtoken';
import { IJwt } from '../interfaces/jwt.interface';
class JWT{
    private secretJWTKey = SECRET_JWT_KEY  as string;
    firmar(datos : IJwt , expiresIn: number = FECHA_EXPIRACION.H24){
        return jwt.sign(
            {usuario: datos.usuario},
            this.secretJWTKey,
            {expiresIn}
        );
    }
}
export default JWT;
```

Anexo 13 Integración de Angular con el Sistema Recomendador.

```
private apiUrl = 'http://localhost:4000/apirecomendar';
  recomendacionesUsuario(data): Observable<any> {

    const headers = new HttpHeaders()
      .set('content-type', 'application/json')
      .set('Access-Control-Allow-Origin', '*');
    return this.http.post<any[]>(this.apiUrl, data, { 'headers': headers });
  }
```

Anexo 14 Sintaxis de Terraform (Ignacio Sánchez, 2017).

```
resource "azurem_virtual_network" "demo" {
  name = "demo-virtual-network"
  address_space = ["10.0.0.0/16"]
  location = "${var.azurem_location}"
  resource_group_name = "${azurem_resourcer_group.demo.name}"

}
resource "azurem_subnet" "demo" {
  name = "demo-subnet"
  resource_group_name = "${azurem_resourcer_group.demo.name}"
  virtual_network_name = "${azurem_virtual_network.demo.name}"
  address_prefix = "100.0.1.0/24"

}
```

Anexo 15 Principales instrucciones de dockerfile.

FROM Nos indica para especificar la imagen base sobre la que se construirá la aplicación dentro del contenedor como, por ejemplo, Ubuntu, Python, Ngnix.	Sintaxis: FROM ubuntu:18.04
RUN Nos permite ejecutar cualquier comando sobre una imagen base; por ejemplo, instalar paquetes o librerías (apt-get, yum install, etc).	Sintaxis: RUN apt-get update
ENV Establece variables de entorno para el contendor. Ejemplo: DEBIAN_FRONTEND noninteractive. Este componente permite instalar varios archivos .deb sin tener que interactuar con ellos.	Sintaxis: ENV NODE_ENV production
ADD Con esta instrucción se puede copiar las URL, archivos y directorios desde una ubicación especifica y a su vez los agrega al sistema de ficheros del contenedor	Sintaxis: ADD recursos/jdk-7u79-linux-x64.tar.gz /usr/local/tar/
EXPOSE La instrucción nos permite declarar el puerto en el que se escucha el servicio de la imagen.	Sintaxis: EXPOSE 8080
CMD Mediante esta instrucción se puede definir una serie de comandos que se ejecutarán una sola vez cuando se inicie el contenedor.	Sintaxis: CMD ["node", "./dist/main.js"]
ENTRYPOINT Permite indicar argumentos de entrada para la imagen.	Sintaxis: ENTRYPOINT [“/bin/echo”, “Hello”]
VOLUME Esta opción crea un punto de montaje en el sistema de archivos de la imagen que se va a usar para encajar un volumen externo.	Sintaxis: VOLUME /mivol
WORKDIR Configuración del directorio de trabajo sobre el que se van a aplicar las instrucciones.	Sintaxis WORKDIR /app
COPY Es la expresión para copiar archivos y directorios al contenedor, similar a ADD.	Sintaxis COPY package*.json./

Anexo 16 Sintaxis docker-compose.yaml (Maxtor, 2019).

```
version: "3"
services:
  web:
        build: .
        ports:
          - "8080:80"
        networks:
          - webnet
        redis:
          image: "redis:alpine"
networks:
  webnet:
```

Anexo 17 Principales apartados.

Version Es la versión del motor, esta indica las instrucciones para interpretar el fichero. Para poder usar stacks es necesario tener mínimo la versión 3.	Ejemplo version: '3'
Services Se refiere a la configuración de la cantidad de servicios contenerizados que se van a ejecutar.	Ejemplo services: frontend: image: vuejs-app backend: image: springboot-app db: image: postgres
Build Se usa para indicar que el servicio se basará en la construcción del dockerfile con referencia a la ruta.	Ejemplo build: ./directorio
Context Aquí se define la ruta en la que se encuentran ubicados los ficheros necesarios para el contexto de la imagen.	Ejemplo build: context: ./directorio
Args En esta sección se configuran los argumentos con un solo valor, estos se pueden interpretar como resultados de los valores de ambiente en la máquina que se está ejecutando el compose.	Ejemplo build: args: - version=1 - user=jsitech
Image Indica el nombre de la imagen con el que el contenedor se creará.	Ejemplo: image: jsitech/shodan image: ubuntu:14.04 image: alpine
Command	Ejemplo:

Permite configurar el comando y se ejecuta al finalizar la carga del contenedor.	command: npm run start
Environment Aquí se agrega variables de entorno en caso de que el contenedor lo necesite.	Ejemplo: environment: MYSQL_DATABASE=jsitech MYSQL_USER=jsitech MYSQL_PASSWORD=jsitech
Ports Permite mapear los puertos que van a enrutar desde el exterior al interior del contenedor.	Ejemplo: ports: - "2002:2002"
Volumes En esta sección se puede hacer que el directorio actual se mapee directamente con el lugar donde se ha creado la aplicación.	Ejemplo: volumes: - /opt/data:/var/lib/mysql
Networks Permite indicar la lista de redes que se conectará con los distintos servicios.	Ejemplo: networks: - Red Prueba - Red producción

Anexo 18 Archivo de configuración del Provider (Elaboración propia)

```
provider "azurerm" {
  features {}
}

terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=2.66.0"
    }
  }
}
```

Anexo 19 Archivo de configuración main.tf (Elaboración propia).

```
resource "azurerm_resource_group" "tesis-ecommerce" {
  name     = "tesis-rg"
  location = "West US 2"

  tags = {
    environment = "Tesis"
  }
}
```

```
resource "azurerm_kubernetes_cluster" "tesis-ecommerce" {
  name                = "tesis-aks"
  location            = azurerm_resource_group.tesis-ecommerce.location
  resource_group_name = azurerm_resource_group.tesis-ecommerce.name
  dns_prefix          = "tesis-ecommerce"

  default_node_pool {
    name            = "container"
    node_count      = 1
    vm_size         = "Standard_D2_v2"
    os_disk_size_gb = 30
  }

  service_principal {
    client_id     = var.appId
    client_secret = var.password
  }

  role_based_access_control {
    enabled = true
  }

  tags = {
    environment = "Tesis"
  }
}
```