UNIVERSIDAD NACIONAL TECNOLÓGICA DE LIMA SUR

FACULTAD DE INGENIERÍA Y GESTIÓN

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

“IMPLEMENTACIÓN DE MICROSERVICIOS CON KUBERNETES PARA MEJORAR LA ESCALABILIDAD EN LA ARQUITECTURA DE LA APLICACIÓN EN UNA EMPRESA DEDICADA AL COMERCIO ELECTRÓNICO”

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de

INGENIERO DE SISTEMAS

PRESENTADO POR EL BACHILLER

PAREDES CALLE, DANIEL

ASESOR

TACZA VALVERDE, IGNACIO RUBEN

Villa el Salvador

2021

Dedicatoria

El presente trabajo es dedicado a mis padres, esposa, hija y hermanos por el apoyo incondicional que me brindaron en el trascurso de mi vida.

Agradecimientos

A mi amigo Cliff La Torre por su constante apoyo en el transcurso de la elaboración de este proyecto y por ser una pieza clave para poder culminarlo.

INDICE

LISTADO DE TABLAS

LISTADO DE FIGURAS

Resumen

El proyecto detallará como el área de Arquitectura, tecnología e integración implementa un microservicios con kubernetes para mejorar la escabilidad en la arquitectura de la aplicación.

El motivo de esta implementación es por causa del modelo de arquitectura que utiliza actualmente el área ATI. Esta arquitectura presenta el problema cuando los servicios web utilizan mucho recurso o mal uso de recurso, ocasionando que los servicios web terminen indisponibles y sea necesario generarse una sesión para realizar la asignación de recursos e iniciando los servicios web. Esto genera mucho tiempo, y durante ese tiempo los servicios web no están disponibles.

Con el apoyo del marco de trabajo y el uso de las tecnologías para la integración, entrega y despliegue continuo se logrará realizar un microservicio escalabre horizontalmente en un plazo de 2 semanas.

Esta implementación servirá como práctica para la migración de todos los servicios web del área de ATI para lograr la reducción de incidentes ocasionados por el alto uso de consumo de recursos.

INTRODUCCIÓN

El área de Arquitectura, tecnología e integración de la empresa de supermercados se encarga de desarrollar servicios web monolíticos, el presente trabajo implementará una solución de cambio de arquitectura.

Actualmente el área de ATI se encarga del desarrollo de los servicios web de una manera que no permite lograr el autoescalamiento de recursos utilizados por estos servicios web. Esto ocasiona incidentes de indisponibilidad de los servicios.

Por tal motivo es necesario la implementación de microservicios con kubernetes para mejorar la escabilidad en la arquitectura de la aplicación.

Esta implementación permitirá la reducción de incidentes ocasionados por el alto uso de consumo de recursos.

CAPÍTULO I. ASPECTOS GENERALES.

1.1. Contexto (empresa).

La organización objeto del presente trabajo, es una cadena de supermercados líder del país con más de 450 tiendas y una plataforma de comercio electrónico en todo el Perú.

Esta empresa en mención cuenta con el área de Arquitectura, Tecnología e Integración (ATI) en donde se desarrolla servicios web monolíticos utilizando el modelo de arquitectura de software Integration BUS V10 y el manifiesto SOA. Estos servicios web son expuestos en la red de la empresa para que el área comercial pueda acceder a ellos.

Este modelo de arquitectura en el área ATI presenta el problema cuando los servicios web utilizan mucho recurso o mal uso de recurso, ocasionando que los servicios web terminen indisponibles y sea necesario generarse una sesión para realizar la asignación de recursos e iniciando los servicios web. Esto genera mucho tiempo, y durante ese tiempo los servicios web no están disponibles.

La arquitectura desarrollada no constituía una alternativa conveniente que sea escalable y mantenga siempre disponible los servicios web cuando se realice la asignación de recursos.

El trabajo tiene por objetivo resolver el problema mencionado utilizando Kubernetes, la plataforma más extensiva de que permite la orquestación de servicios de infraestructura. Esto es posible gracias a la escabilidad de la unidad de orquestación que es POD. Ya no sería necesario generar servicios web sino microservicios que se encontrarían en contenedores y estos estarían corriendo dentro de los POD.

1.2. Delimitación temporal y espacial del trabajo.

El trabajo de migración de un servicio web a un microservicio se ha desarrollado en 2 semanas. Inició el jueves 14 de febrero del 2019 y terminó el miércoles 27 de febrero del 2019. Se desarrolló en las instalaciones de una consultora particular a modo de outsoursing para la empresa de supermercados. El desarrollo del proyecto se llevó a cabo con un equipo de trabajo de 5 personas que estaban conformados por un Product Owner (PO) que tenía contacto directo con el área comercial, un Lider técnico, un SCRUM Master, 1 Devops y 1 test. El rol que ocupaba era DevOps que pertenecía al equipo de desarrollo y despliegue.

1.3. Objetivos.

1.3.1. Objetivo General.

Implementar los microservicios consumidos por el área comercial, en una empresa dedicada al comercio electrónico, con kubernetes para mejorar la escalabilidad en la arquitectura de la aplicación.

1.3.2. Objetivos Específicos.

1. Obtener la información funcional necesaria del área ATI para la implementación de los microservicios.
2. Mejorar la escabilidad horizontal mediante el uso de kubernestes en la implementación de los microservicios.
3. Reducir la cantidad de incidentes presentado por el alto uso de recursos de las aplicaciones.

CAPÍTULO II. MARCO TEÓRICO

2.1. Antecedentes.

2.1.1. Antecedentes internacionales.

Franco Nelson Funes, (2018). En su trabajo titulado Análisis de solución eficiente para escalabilidad de sistemas online mediante orquestación de contenedores, Argentina, Buenos Aires, concluyó que introducir contenedores en el ciclo de desarrollo y en operaciones ayudan a adoptar prácticas devops (CI/CD) por sus características de poder levantarse y bajarse muy simplemente a diferencia de las máquinas virtuales convencionales. Esto se vuelve más eficiente cuando la generación de clústeres de contenedores administra y monitorea a través de la gestión de orquestadores y vela por la salud de los mismos.

David & Armando & Marcelo (2020). En su trabajo titulado Plataforma colaborativa, elástica, de bajo costo y consumo basado en recursos de la nube, contenedores y móviles para HPC, Argentina, La Plata, concluyeron que ante diversas cargas de trabajo el sistema responde de una forma flexible y elástica. A la vez de forma concurrente las tareas de procesamiento son resueltas sin importar la cantidad, el volumen de carga y la potencia de los nodos trabajadores. También concluyeron que para resolver tareas de cómputo intensivo la arquitectura desarrollada constituye una alternativa de bajo costo y escalable.

Javier Tagarro (2019) En su trabajo titulado Diseño de una infraestructura Cloud de Blockchain mediante el uso de contenedores, España, Madrid, concluyó que, la existencia de estas tecnologías permitió que el proyecto sea menos denso, más fácil y muy económico. Gracias a estas nuevas tecnologías las operaciones de despliegue para tener un servidor funcional dando servicio a un gran número de clientes se realizan en segundos a comparación de otras tecnologías en las que estas mismas operaciones de despliegue se pueden tardar días y/o semanas en su realización.

2.1.2. Antecedentes nacionales.

Carlos Calla (2018). En su trabajo titulado Implementación de microservicios para el intercambio de datos en el canal de plataforma digital del Banco de Crédito del Perú, Perú, Lima, concluyó que al utilizar microservicios se logró una mejora de intercambio de datos entre los servicios y aplicaciones generando en los beneficiarios un impacto positivo al cumplir sus expectativas en el modelo de negocio.

2.2. Bases teóricas.

2.2.1. Servicio web Monolítico.

En un mismo programa se acoplan y sujetan todos los aspectos funcionales que pertenecen a la estructura del software. Se descompone en la capa que es expuesta al usuario final, la lógica de la aplicación y el sistema gestión de datos. Estas tres capas son ejecutadas sobre una misma máquina y externalizan las bases de datos. La flexibilidad de los servicios web monolíticos es mínima permitiendo que en cualquier pequeño punto de fallo tenga un gran impacto y sea ineficiente la realización del escalado horizontal. Todas las funcionalidades están escritas en un único lenguaje de programación. Manuel Perez C. (2019). libro: Arquitecturas basadas en microservicios.

2.2.2. Manifiesto SOA.

La arquitectura orientada a servicio es un diseño de software donde las interfaces de servicios se comunican mediante una red con lenguaje común y reutilizan sus elementos. Integra y permite la comunicación entre sí y que trabajen en conjunto los elementos separados del software implementado. Hay que entender que un servicio es una unidad independiente de las funciones del software. Este servicio puede realizar tareas específicas y contiene las integraciones necesarias de datos y código. Thomas Erl. (2016). Segunda edición del libro: Service-Oriented Architecture: Analysis and Design for Services and Microservices.

2.2.3. Microservicios.

El término arquitectónico de microservicios se define como una propuesta o enfoque del desarrollo y diseño de aplicaciones de software de una manera que estas

aplicaciones sean conjuntos de pequeños servicios despegables de forma independiente, se ejecuten en su propio proceso y se comuniquen entre sí utilizando mecanismos ligeros, que puede ser muchas veces una API de recursos HTTP. Estos pequeños servicios también se implementan de forma independiente a través de una maquinaria completamente automatizada. Casi es nulo la administración centralizada de estos servicios que se pueden escribir en lenguajes de programación y tecnologías de almacenamiento de datos diferentes. Fowler & Lewis. (2014). Artículo de martin fowler Microservices.

2.2.4. Kubernetes.

Es una plataforma open source, extensible y portable desarrollada por los ingenieros de Google originalmente, la cual orquesta la infraestructura de cómputos, redes y almacenamiento gestionando las aplicaciones en varios hosts para la administración de cargas de trabajo y servicios centrado en contenedores. Su diseño puede construir un ecosistema de componentes y herramientas logrando desplegar, escalar y administrar aplicaciones de una manera más fácil. Burns, B., Beda, J., & Hightower, K. (2018). Kubernetes. Dpunkt.

2.2.4.1. Clusteres de kubernetes.

Un clúster de Kubernetes es un grupo de nodos que permiten programar y ejecutar aplicaciones en contenedores. Presenta como mínimo un plan de control que gestiona a uno o varios nodos, mantiene el estado deseado del clúster, controla las aplicaciones y sus imágenes que se ejecutan en los contenedores. Luksa, M. (2017). Kubernetes in action. Simon and Schuster.

2.2.4.2. Nodos de kubernetes.

Es una máquina de trabajo virtual o física que al inicio se conocía como minion. El plano de control de kubernetes gestiona a los nodos. El nodo contiene los servicios necesarios para ejecutar los pod y la carga de trabajo. Sayfan, G. (2017). Mastering kubernetes. Packt Publishing Ltd.

2.2.4.3. Pod de kubernetes.

Es el elemento más pequeño de kubernetes. Es un grupo de uno o más contenedores de aplicaciones implementados en un solo nodo los cuales cuentan con almacenamiento de red compartida y especificaciones para ejecutar los contenedores. Burns, B., Beda, J., & Hightower, K. (2019). Kubernetes: up and running: dive into the future of infrastructure. O'Reilly Media.

2.2.5. Metodologías ágiles.

En febrero de 2001, tras una reunión celebrada en Utah: U.S. nace el término “ágil” aplicado al desarrollo de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto (Boehm, B. Turner, R. Balancing Agility and discipline. A guide for the Perplexed. Addison-Wesley. 2003).

Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

Tras esta reunión se creó “The Agile Alliance”, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida es fue el Manifiesto Ágil, un documento que resume la filosofía “ágil”. (Patricio Letelier, José Hilario, Metodologías ágiles en el desarrollo del software. España. Universidad Politécnica de Valencia. 2003).

2.2.6. SCRUM.

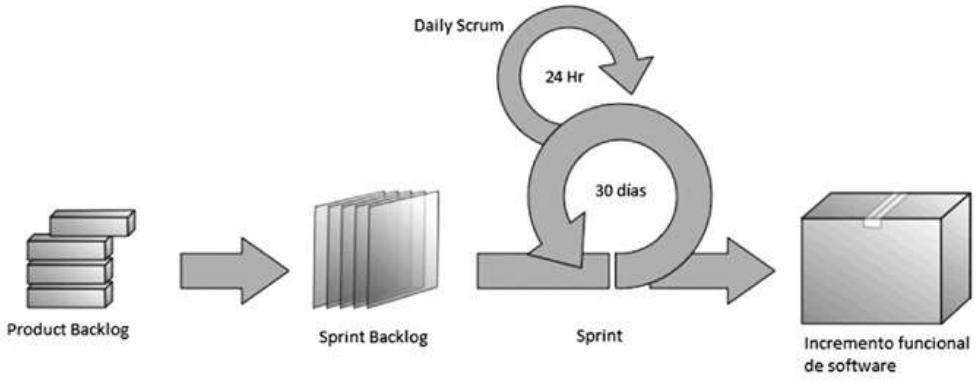
Un marco de trabajo de procesos donde los integrantes pueden atacar problemas complejos adaptativos la cual permite entregar productos posiblemente del máximo valor de forma productiva y creativamente. Scrum es:

- Ligero.
- Fácil de entender.

- Extremadamente difícil de llegar a dominar.

Scrum es usado desde principios de los años 90 para gestionar el desarrollo de productos complejos. No equivocar a Scrum como una técnica o proceso para construir productos; en vez de eso, es un marco de trabajo que está diseñado para emplear varias técnicas y procesos. Scrum muestra la eficacia relativa de la gestión del producto y las prácticas desarrollo. (Ken Schwaber y Jeff Sutherland, La guía de Scrum, 2014, página 4).

Figura 1. Scrum: Fases de un Sprint.



Fuente: (Cohn, M. User stories applied: for Agile software development. Addison Wesley, Boston, 2004.).

2.2.6.1. Roles de SCRUM.

2.2.6.1.1. SCRUM Master.

El Scrum Master es el responsable de asegurar que la teoría, prácticas y reglas de Scrum sea entendido y adoptado por el Equipo Scrum.

Para aumentar el valor creado por el Equipo Scrum, el Scrum Master ayuda a todos a modificar y entender qué interacciones con el Equipo Scrum pueden ser de ayuda y cuáles no referente a las personas externas al Equipo Scrum. (Ken Schwaber y Jeff Sutherland, La guía de Scrum, 2014, página 8).

2.2.6.1.2. Product owner.

El propietario del producto toma las decisiones del cliente. Es el responsable del valor del producto. Es el único en tomar las decisiones para simplificar la comunicación interna que considere oportuna con el cliente. Debe tener el conocimiento suficiente del producto y las atribuciones necesarias para tomar las decisiones que le corresponden. El cliente puede ser una organización grande o pequeña. (Menzinsky, López, & Palacio, Scrum Manager, 2016, pág. 33).

2.2.6.1.3. Equipo SCRUM.

El Equipo Scrum consiste en un Product Owner, el Equipo de Desarrollo y un Scrum Master. Existen los equipos Scrum autoorganizados y multifuncionales.

Mientras que el equipo autoorganizado elige la mejor manera de poner en marcha su trabajo y no son dirigidos por personas externas al equipo, el equipo multifuncional tiene todas las competencias necesarias para poner en marcha su trabajo sin depender de otras personas que no son parte del equipo.

El modelo de equipo en Scrum está diseñado para optimizar la flexibilidad, la creatividad y la productividad.

Los Equipos Scrum son los responsables de entregar productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Estas entregas incrementales de producto “Terminado” aseguran la disponibilidad de una versión potencialmente útil y funcional del producto. (Ken Schwaber y Jeff Sutherland, La guía de Scrum, 2014, página 6).

2.3. Definición de términos básicos.

2.3.1. Contenedor.

Del mismo modo que en el sector del transporte se usan contenedores físicos para aislar diferentes cargas (por ejemplo, para el transporte en buques y en trenes), las tecnologías de desarrollo de software usan cada vez más un método denominado contenerización.

Un paquete de software estándar (conocido como “contenedor”) agrupa el código de una aplicación con las bibliotecas y los archivos de configuración asociados, junto con las dependencias necesarias para que la aplicación se ejecute. Esto permite a los desarrolladores y profesionales de TI implementar aplicaciones sin problemas en todos los entornos. González Hernández, B. Seguridad en Docker.

2.3.2. ATI.

Arquitectura de tecnología e integración.

CAPÍTULO III: DESARROLLO DEL TRABAJO PROFESIONAL

3.1. Determinación y análisis del problema.

El área de ATI genera servicios web monolíticas desarrolladas con el estilo RESTful que están basadas en arquitectura REST. El área comercial hace uso de estos servicios web para acceder a información que se encuentra almacenada en tablas de la BD que pertenecen al área comercial.

El área comercial es capaz de listar, crear, leer, actualizar y borrar los registros de las tablas de la BD gracias a los métodos HTTP y URI que permite esta arquitectura.

Para la creación de estos servicios web se utilizó el modelo de arquitectura de software Integration BUS V10 y la arquitectura orientada a los servicios SOA.

3.1.1. Proceso de Implementación de las aplicaciones.

El proceso inicia con la generación, por parte del Poduct Owner, de las historias de usuarios y estableciendo la cantidad de sprints. Estás historias de usuarios son asignadas al desarrollador y el test, las cuales deberán realizarlas en la cantidad de sprint establecidos. El desarrollador se lleva las historias de creación de los elementos y despliegue del servicio web en el ambiente de calidad y productivo. Si surgiera alguna duda técnica con respecto al desarrollo de servicio web, el Leader técnico está apto para absolverla al desarrollador. El test se encarga de iniciar su historia de pruebas del servicio web una vez que esté en el ambiente de calidad. El test termina de realizar las pruebas del servicio web y la historia se da por cerrada una vez que da la aprobación para el pase del servicio web al ambiente de productivo. El desarrollador realiza la gestión del pase del servicio web al ambiente productivo. Si existiera algún bloqueante en todo el proceso del desarrollo del servicio web, el Scrum Master está apto para levantar el bloqueante.

Durante el ciclo de vida de los servicios web desde que están desplegados en producción (PRD) se han presentado un sin número de incidentes ocasionados por el

excesivo uso de recursos de S.O. Estos excesivos usos de recursos ocasionan que los servicios web terminen no disponibles para el área comercial.

3.1.2. Evaluación de los registros de incidentes.

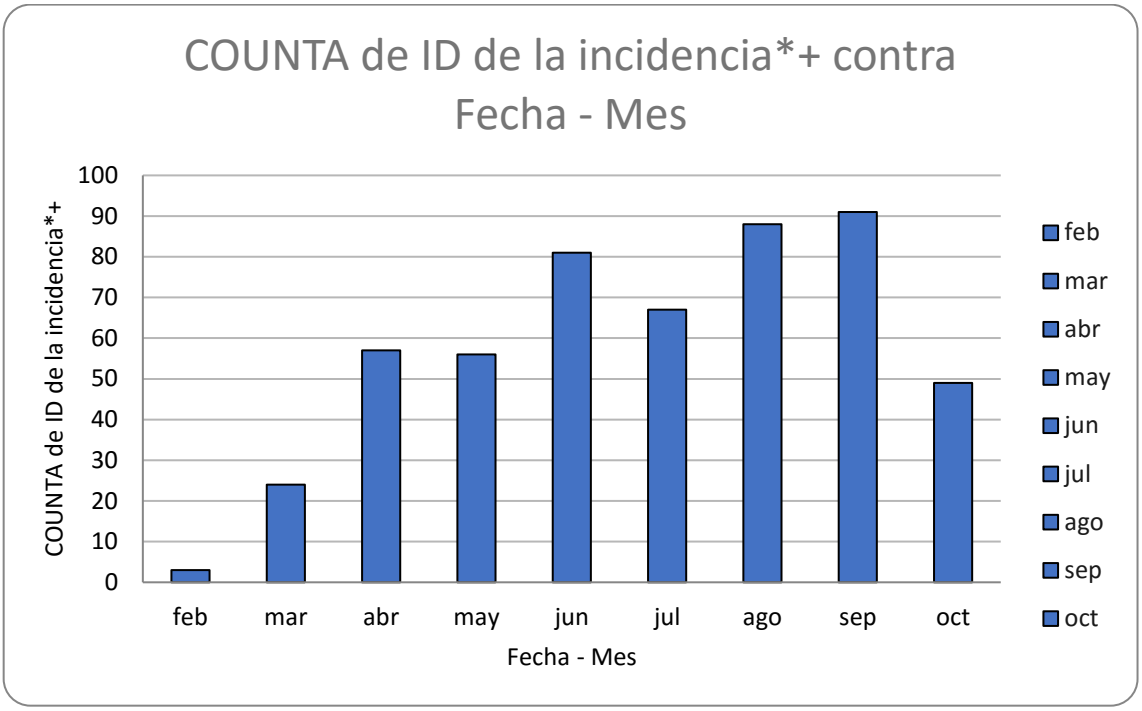
La empresa hace uso de una herramienta donde gestiona los incidentes que se presentan en el ambiente productivo. Mediante esta herramienta se pudo cuantificar las veces que los servicios web no estuvieron disponibles.

Tabla 1. Incidentes de febrero al octubre del 2018.

Fecha	COUNTA of ID de la incidencia*+
feb	3
mar	24
abr	57
may	56
jun	81
jul	67
ago	88
sep	91
oct	49
Total general	**516**

Fuente: Reporte extraído de la herramienta utilizado por la empresa.

Figura 2. Diagrama de incidentes de febrero al octubre del 2018.



Fuente (Elaboración Propia)

Es importante la escabilidad en temas de recursos y se mantengan arriba servicios web. El modelo de arquitectura orientado a servicio (SOA) utilizando el software Integration BUS V9 no permite la escabilidad y la única solución era a través de la atención de un incidente.

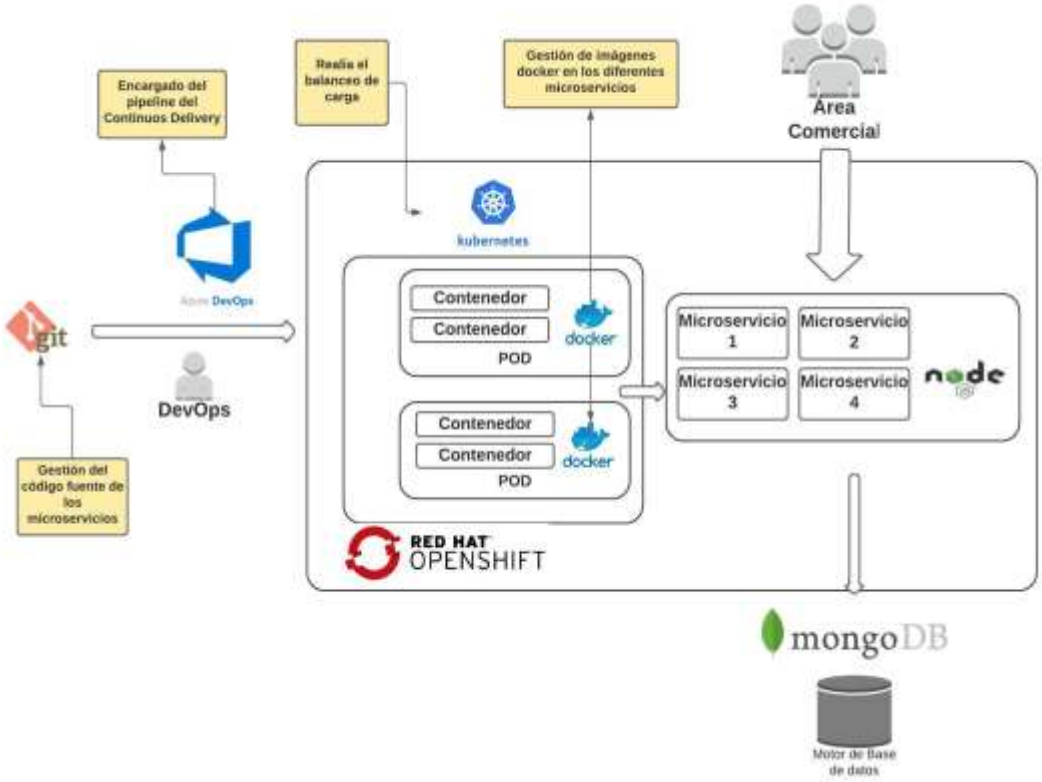
3.2. Modelo de solución propuesto.

El presente trabajo propone la utilización de la plataforma de código abierto Kubernetes como solución para la escabilidad automática de los recursos del S.O.

Para poder utilizar kubernetes es necesario migrar los servicios web monolíticas generadas con arquitectura orientada a los servicios (SOA) por la arquitectura de microservicios. Esta arquitectura está optimizada para DevOps e integración continua/entrega continua (CI/CD) y permite que servicios pequeños se desplieguen con frecuencia.

La empresa adquirió unos clústeres de kubernetes y mediante la aplicación Rancher gestionará de una forma centralizada los clústeres.

Figura 3. Arquitectura con kubernetes.



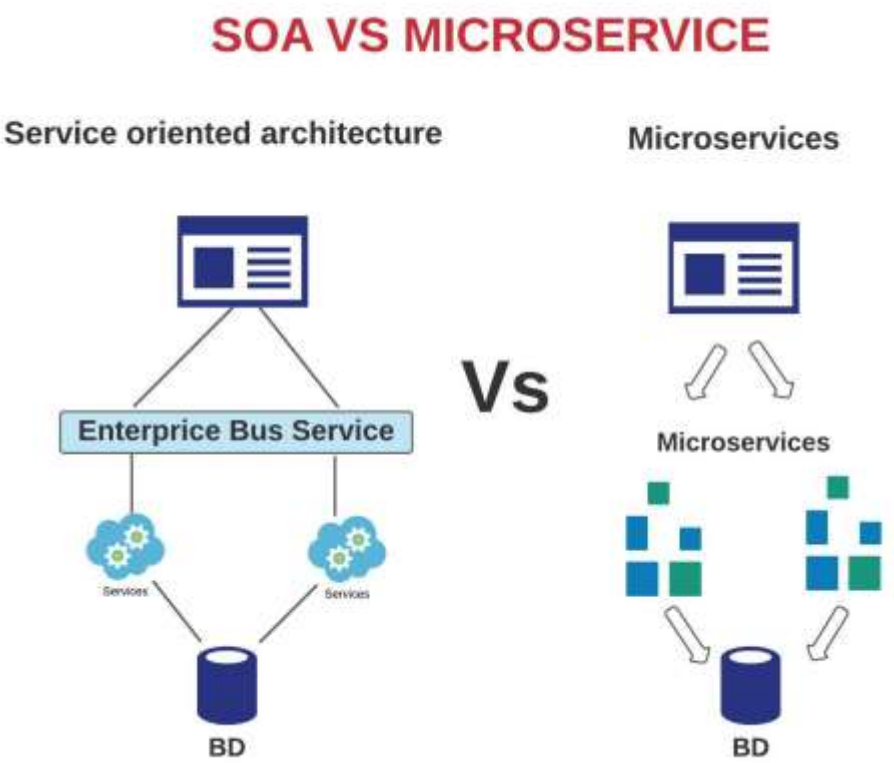
Fuente (Elaboración Propia)

Kubernetes cuenta con políticas scaling automáticas y es la plataforma más extensiva para orquestación de servicios de infraestructura. Kubernetes trabaja con una entidad de scheduling y de orquestación de nombre POD.

Kubernetes tiene entidades y recursos que hacen posible correr muchas réplicas del POD según la demanda de los recursos utilizados por los microservicios y asegurarse que se encuentren disponibles. El POD puede tener uno o varios contenedores corriendo uno al lado del otro en el mismo kernel. Los microservicios se empaquetan y ejecutan dentro de los contenedores. Los contenedores están conformados de

diferentes tecnologías, que se potencian las unas con las otras y trabajan en conjunto. Esto permite satisfacer las demandas de recursos de los microservicios.

Figura 4. Diferencia entre SOA vs Microservice.



Fuente (Elaboración Propia)

La implementación de microservicios en contenedores de docker y administrados por el orquestador Kubernetes permitirá:

- La eliminación de asignación de recursos del S.O. a los microservicios de forma manual.

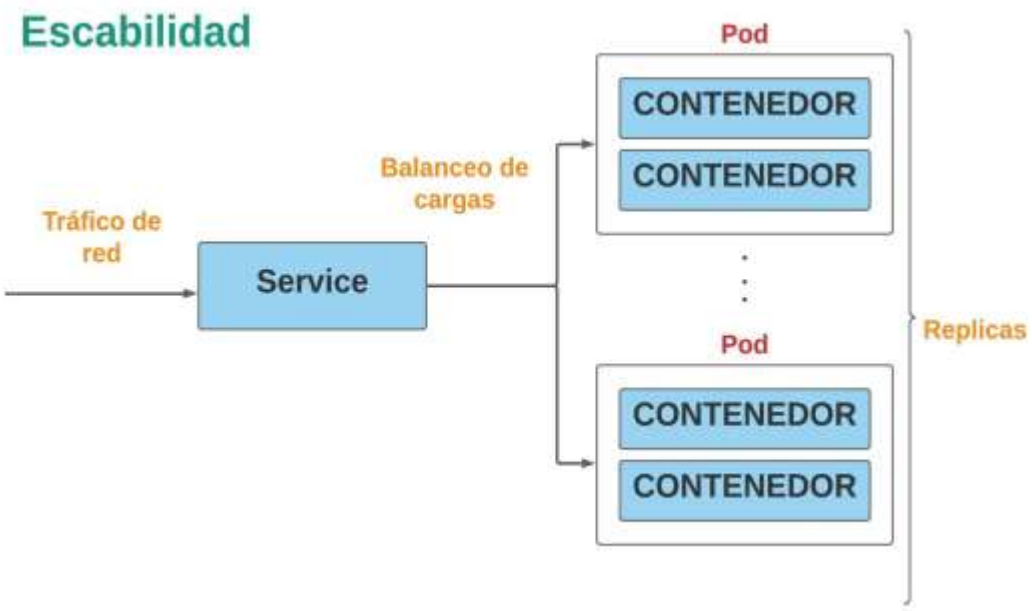
- Mayor disponibilidad de los microservicios en el ambiente productivo.

3.2.1. Escabilidad con Kubernetes.

3.2.1.1. Escabilidad por alta carga de tráfico.

Kubernetes permite realizar replicas en los pod. Se crean copias del pod cuando se realiza las réplicas. Esto sucede cuando hay demasiado tráfico y el pod no es capaz de atenderlo. Una vez ya replicados los pod, la carga del tráfico se distribuye entre ellos.

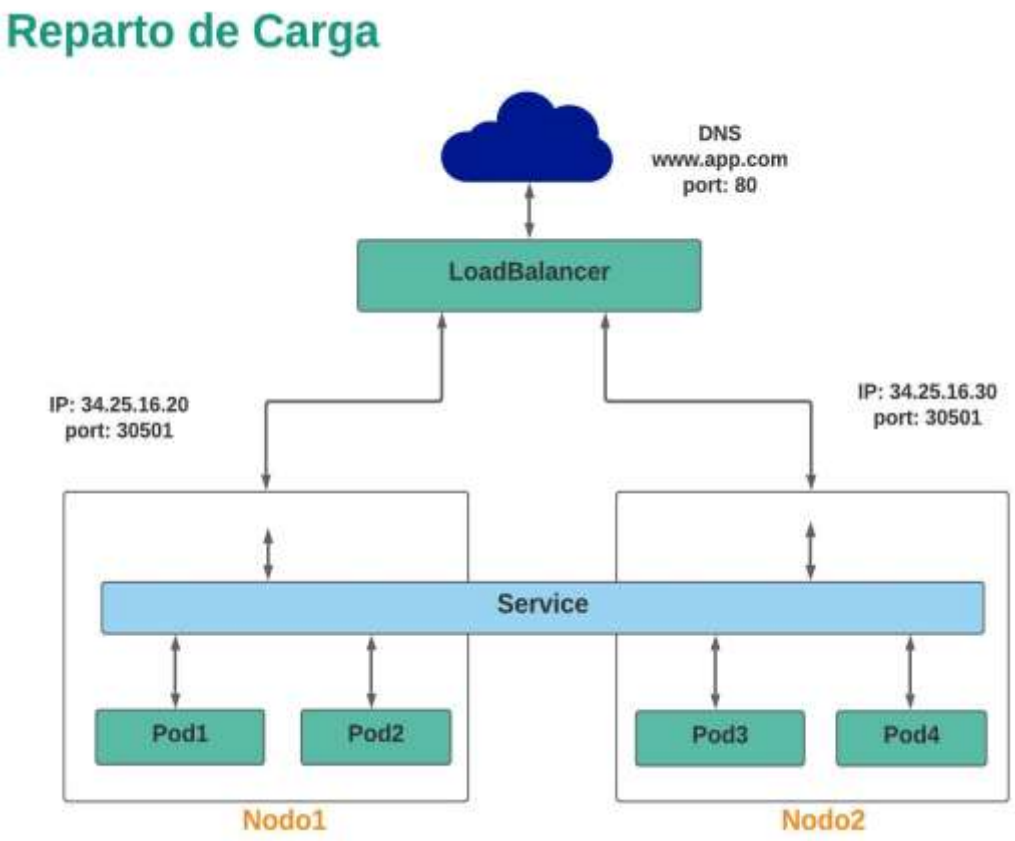
Figura 5. Escabilidad con kubernetes.



Fuente (Elaboración Propia)

Este reparto de carga es posible gracias al IPVS modo proxy que maneja. Este IPVS redirige el tráfico con poca latencia sin embargo con un rendimiento mucho mejor al tráfico de red y al sincronizar reglas de proxy. Esto permite equilibrar el tráfico a los pod.

Figura 6. Reparto de Carga en kubernetes.



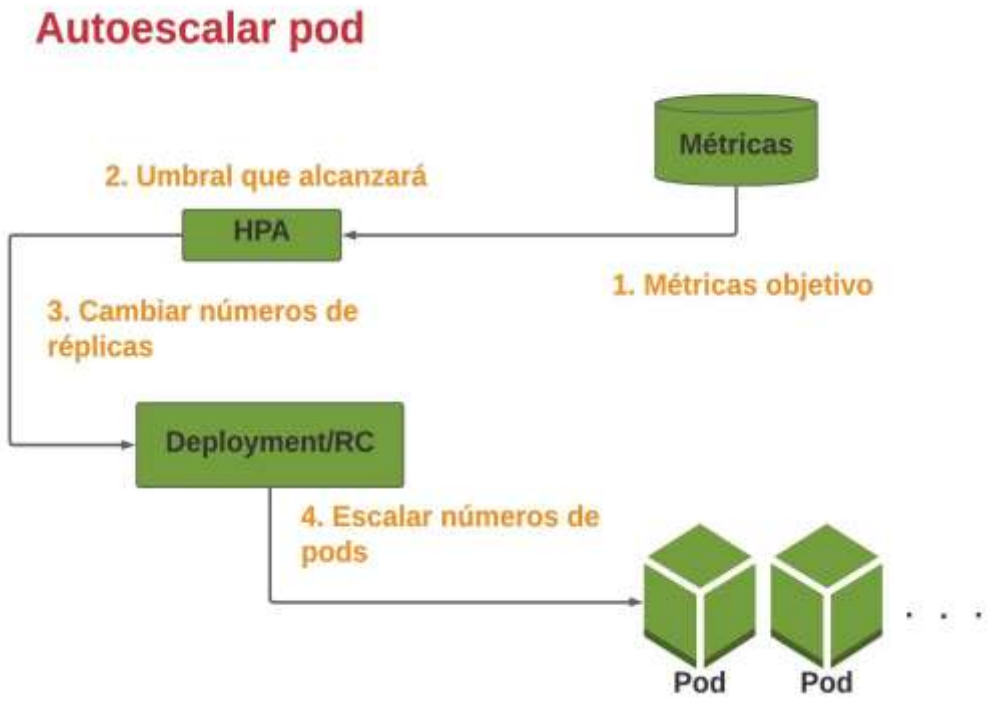
Fuente (Elaboración Propia)

3.2.1.2. Tipos de escabilidad automáticas.

3.2.1.2.1. Autoescalado horizontal del POD.

Se tiene que definir el objetivo de las métricas de carga. Es posible cambiar el número de réplicas de forma automática en base a estos objetivos definidos. Esto permite que si se sobrepasa los objetivos de las métricas se aumente de forma automáticas las réplicas del pod y si está por debajo de los objetivos de las métricas se reduce la cantidad de pod. Real Ixcayau, E. O. (2021). Propuesta para actualización curricular en el área de software incorporando talleres de Docker y Kubernetes (Doctoral dissertation, Universidad de San Carlos de Guatemala).

Figura 7. Autoescalar de manera horizontal.

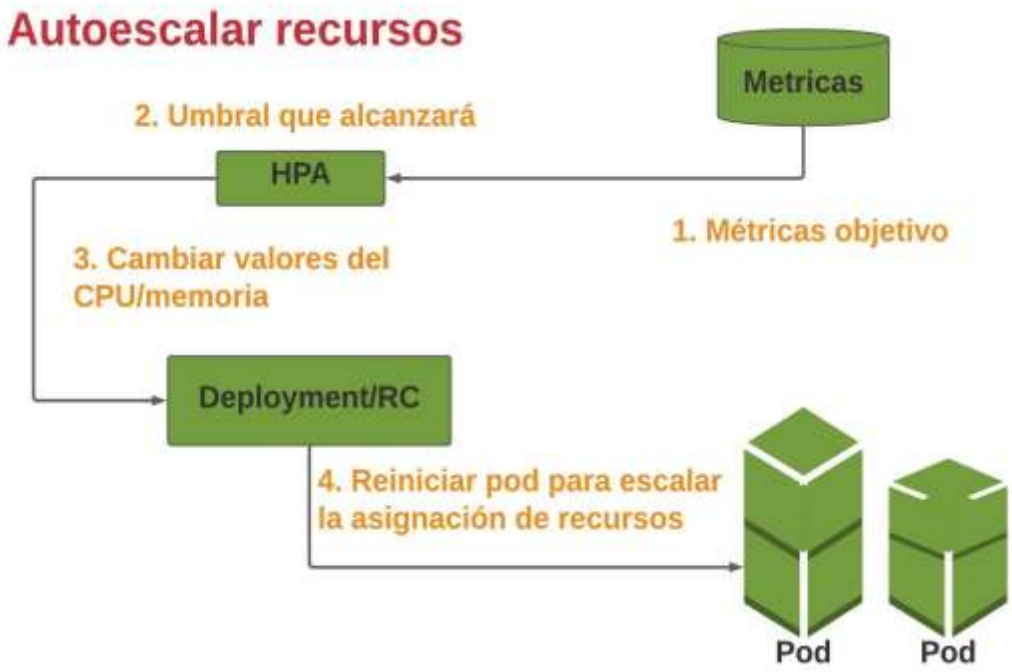


Fuente (Elaboración Propia)

3.2.1.2.2. Autoescalado vertical del POD.

Se tiene que definir los límites máximos de recursos que serían los objetivos de las métricas de carga. Es posible ajustar los recursos del pod de forma automática en base a estos objetivos definidos dependiendo de los recursos usados realmente. Es necesario reiniciar el pod para poder aplicar los nuevos recursos. Cervieri Carrau, V. V. (2019). Cloud autoscaling performance evaluation with real-world like loads and threshold tuning (Doctoral dissertation, Universitat Politècnica de València).

Figura 8. Autoescalar recursos de manera vertical.



Fuente (Elaboración Propia)

3.2.2. Proceso de Implementación.

3.2.2.1. Inicio de proyecto.

El área comercial entrega al área ATI un documento funcional del microservicio. Este documento es entregado a cada uno de los integrantes del equipo.

3.2.2.1.1. Planificación y estimación.

Todos los integrantes del equipo se involucran en esta etapa para planificar, estimar y crear los artefactos tomando en cuenta las técnicas y juegos de estimación.

3.2.2.1.2. Product Backlog.

El producto Owner elabora en su totalidad la lista de requerimientos en la cual asigna una prioridad (Alta, media o baja). Esta lista de requerimientos permite tener un orden inicial de todos los requerimientos.

Tabla 2. Product Backlog.

N°	Historia de Usuario	Prioridad
1	Creación de elementos en el IDE Visual Code Studio para el microservicio	Alta
2	Creación y ejecución de elementos para la integración y despliegue continnuo en la plataforma de Microsoft Azure Devops para el microservicio	Alta
3	Pruebas en el ambiente de calidad de los microservicios	Alta
4	Pase a producción del microservicio y validación de escabilidad en los clústeres de producción	Alta

Fuente (Elaboración Propia)

3.2.2.1.3. Historias de usuario.

EL product owner realiza la elaboración, redacción de todas las historias de usuario. El equipo de desarrollo realiza la estimación del valor de cada historia mediante el uso de las Cartas para estimación de póquer (Fibonacci).

Historia 1: Creación de elementos en el IDE Visual Code Studio para el microservicio.

Tabla 3. Historia de usuario 001.

HISTORIA DE USUARIO					
Número:	001		**Usuario:**	DevOps	
Nombre:	Creación de elementos en el IDE Visual Code Studio para el microservicio				
Prioridad:	ALTA	**Valor:**	13	**Sprint:**	1
Descripción:	Es necesario crear las líneas de código por el cual estará diseñado el microservicio y pasos para el despliegue del mismo en el cluster de kubernetes				
Precondición:	Ninguna				

Requerimientos:	Mediante el uso del lenguaje de programación node,js se crea con el IDE Visual Studio Code la estructura del microservicio que consiste en exponer los enpoint de los servicios web y el método para realizar acciones ETL con los registros de la bd mongodb. También se crea los archivos para el despliegue del microservicio en los clústeres de kubernetes.
Observación:	Ninguna.

Fuente (Elaboración Propia)

Historia 2: Creación y ejecución de elementos para la integración y despliegue continuo en la plataforma de Microsoft Azure Devops para el microservicio.

Tabla 4. Historia de usuario 002.

HISTORIA DE USUARIO					
Número:	002		**Usuario:**	DevOps	
Nombre:	Creación y ejecución de elementos para la integración y despliegue continuo en la plataforma de Microsoft Azure Devops para el microservicio				
Prioridad:	ALTA	**Valor:**	21	**Sprint:**	1
Descripción:	Son los elementos que interpretaran las acciones detalladas en las líneas de código y ejecutarán el despliegue del microservicio en el cluster de kubernetes				
Precondición:	Haber cerrado la historia de usuario 001				
Requerimientos:	Creación de los elementos en azure devops Subida del los elemento generados en la historia previa despliegue continuo del microservicio en los ambientes calidad y producción				
Observación:	Ninguna				

Fuente (Elaboración Propia)

Historia 3: Pruebas en el ambiente de calidad de los microservicios.

Tabla 5. Historia de usuario 003.

HISTORIA DE USUARIO					
Número:	003		**Usuario:**	test	
Nombre:	Pruebas en el ambiente de calidad de los microservicios				
Prioridad:	ALTA	**Valor:**	13	**Sprint:**	1
Descripción:	El test tiene la responsabilidad de realizar todas las pruebas necesarias en el tiempo establecido para validar que se cumpla la escabilidad del microservicio				
Precondición:	Haber cerrado la historia de usuario 002				
Requerimientos:	A través del postman realiza el consumo de los endpoint expuestos por los microservicios. Validar utilizando la aplicación del rancher que los pod del microservicio escalen cada vez que hay demasiado tráfico ocasionadas por las pruebas de estrés				
Observación:	Ninguna				

Fuente (Elaboración Propia)

Historia 4: Pase a producción del microservicio y validación de escabilidad en los clústeres de producción.

Tabla 6. Historia de usuario 004.

HISTORIA DE USUARIO					
Número:	004		**Usuario:**	DevOps	
Nombre:	Pase a producción del microservicio y validación de escabilidad en los clusteres de producción				
Prioridad:	ALTA	**Valor:**	21	**Sprint:**	1

Descripción:	Una vez que el test dé la aprobación de las pruebas en calidad se procede a solicitar la aprobación al comité de pases de producción. Luego dar seguimiento de la escabilidad del microservicio en producción
Precondición:	Haber cerrado la historia de usuario 003
Requerimientos:	Gestión para la aprobación del pase a producción Activar el release de prd para que se realice el despliegue del microservicio en el clúster de producción Seguimiento de la escabilidad del microservicio en producción
Observación:	Ninguna

Fuente (Elaboración Propia)

Esta etapa queda concluida con el Product Backlog.

Tabla 7. Product Backlog.

N°	Historia de Usuario	Prioridad	Valor
1	Creación de elementos en el IDE Visual Code Studio para el microservicio	Alta	13
2	Creación y ejecución de elementos para la integración y despliegue continnuo en la plataforma de Microsoft Azure Devops para el microservicio	Alta	21
3	Pruebas en el ambiente de calidad de los microservicios	Alta	13
4	Pase a producción del microservicio y validación de escabilidad en los clusteres de producción	Alta	21

Fuente (Elaboración Propia)

3.2.2.1.4. Spring Planning.

Se planificó y definió todos los sprint a entregar y los puntos resaltantes del PBI (Product Backlog Item). Estas acciones se realizaron con todos los miembros del equipo en una sesión.

Tabla 8. Sprint 1.

N° Sprint	PBI	Historia de Usuario	Tarea	Tiempo (Días)
1	1	Creación de elementos en el IDE Visual Code Studio para el microservicio	Se crea el código para la creación del microservicio.	2
			Se crea el archivo docker que será utilizado en el pipeline para realizar la integración continua con docker	1
			Se crea los archivos yml que también será utilizado en el pipeline para para desplegar el microservicio en los clústeres de kubernetes de los ambientes.	1
			Se crea el repositorio MICROSERVICIOS-01	1
			Se crea el pipeline de integración continua. En el pipeline se relaciona con el archivo docker y los archivos yml.	1
	2	Creación y ejecución de elementos para la integración y despliegue continnuo en la plataforma de Microsoft Azure Devops para el microservicio	Se crea los artefactos mediante la ejecución del pipeline	1
			Se crea el relese utilizando los artefactos creados previamente. En el reléase se detalla los ambientes donde estará desplegado el microservicio	1
			Se crea los artefactos mediante la ejecución del pipeline	1
			Subida del los elemento generados en el IDE y despliegue continuo del microservicio en los ambientes calidad y producción	1
Total				10

Fuente (Elaboración Propia)

Tabla 9. Sprint 2.

N° Sprint	PBI	Historia de Usuario	Tarea	Tiempo (Días)
2	3	Pruebas en el ambiente de calidad de los microservicios	Realizar el consumo de los endpoint expuestos por los microservicios a través del postman	3
			Validar utilizando la aplicación del rancher que los pod del microservicio escalen cada vez que hay demasiado tráfico ocasionadas por las pruebas de estrés	2
	4	Pase a producción del microservicio y validación de escabilidad en los clusteres de producción	Gestión para la aprobación del pase a producción	1
			Activar el release de prd para que se realice el despliegue del microservicio en el cluster de producción	1
			Seguimiento de la escabilidad del microservicio en producción	3
Total				10

Fuente (Elaboración Propia)

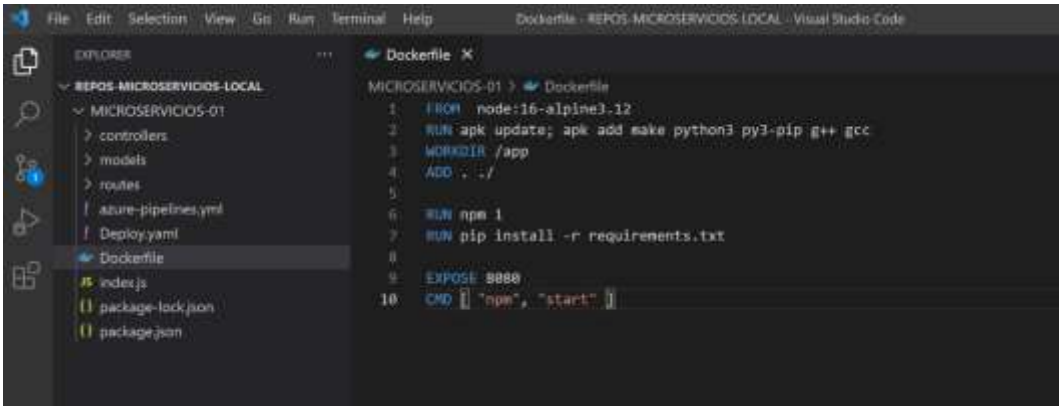
3.2.2.1.5. Implementación.

3.2.2.1.5.1. Desarrollo de Sprint 1.

3.2.2.1.5.1.1. Creación de elementos en el IDE Visual Code Studio para el microservicio.

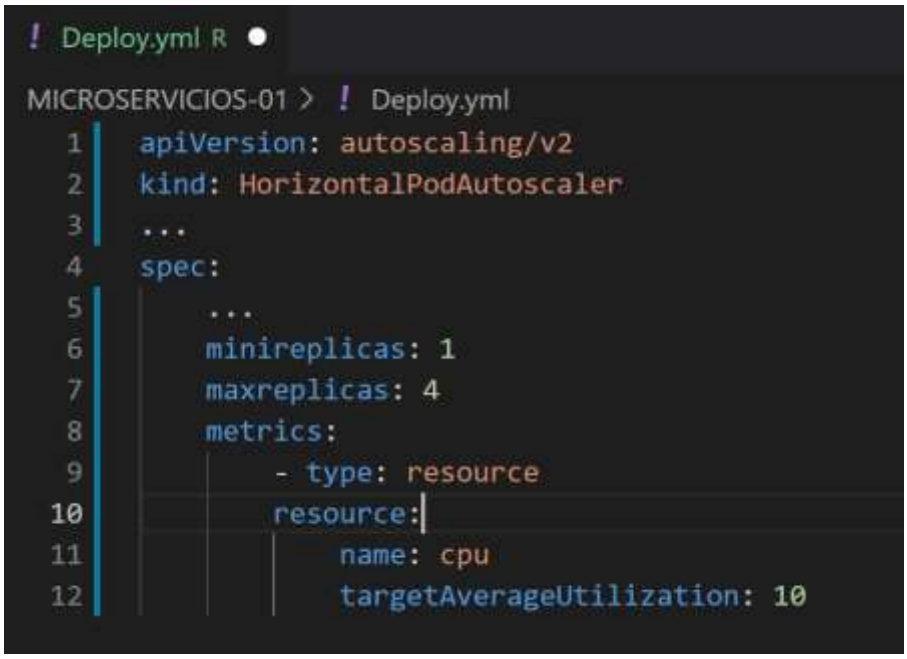
- Se crea el código para la creación del microservicio.
- Se crea el archivo docker que será utilizado en el pipeline para realizar la integración continua con docker.
- Se crea los archivos yml que también será utilizado en el pipeline para para desplegar el microservicio en los clústeres de kubernetes de los ambientes.

Figura 9. Dockerfile.



Fuente (Elaboración Propia)

Figura 10. Deploy.yml.

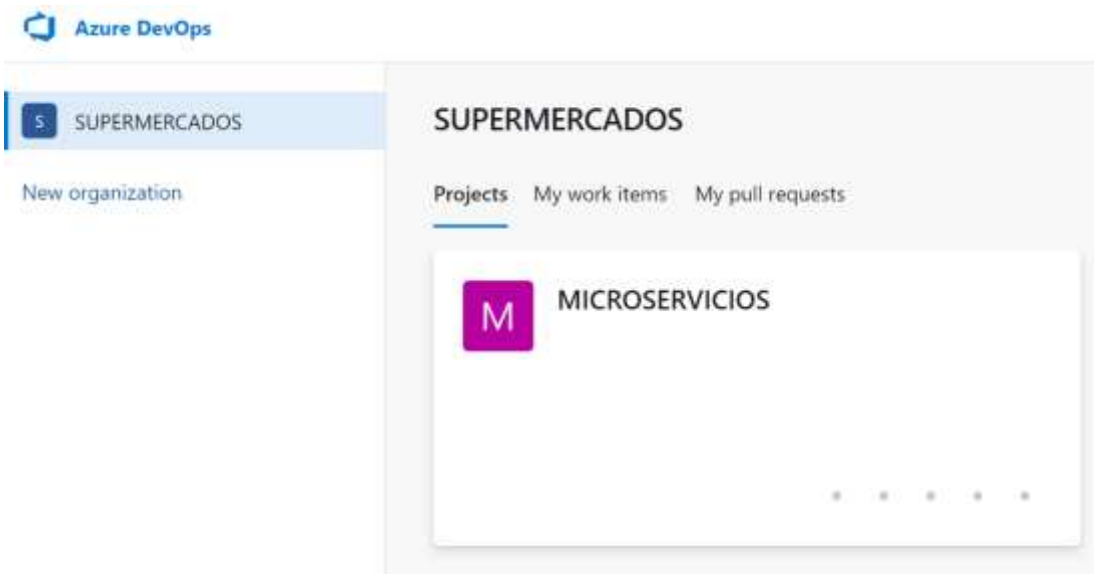


Fuente (Elaboración Propia)

3.2.2.1.5.1.2. Creación y ejecución de elementos para la integración y despliegue continuo en la plataforma de Microsoft Azure Devops para el microservicio.

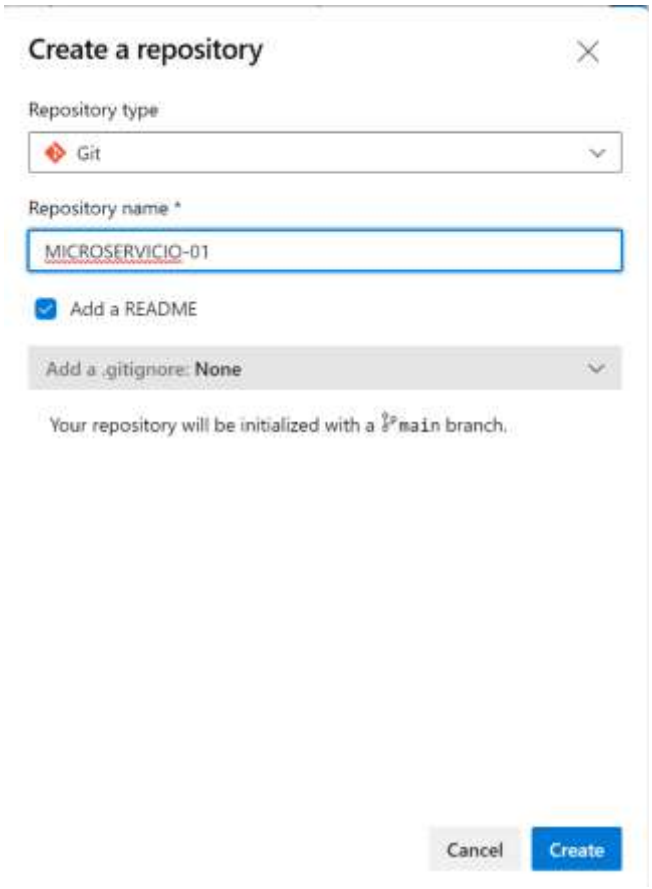
En la organización SUPERMERCADOS existe un proyecto creado llamado MICROSERVICIOS. En este proyecto se crea el repositorio MICROSERVICIOS-01.

Figura 11. Proyecto en azure devops.



Fuente (Elaboración Propia)

Figura 12. Proyecto en azure devops.



Fuente (Elaboración Propia)

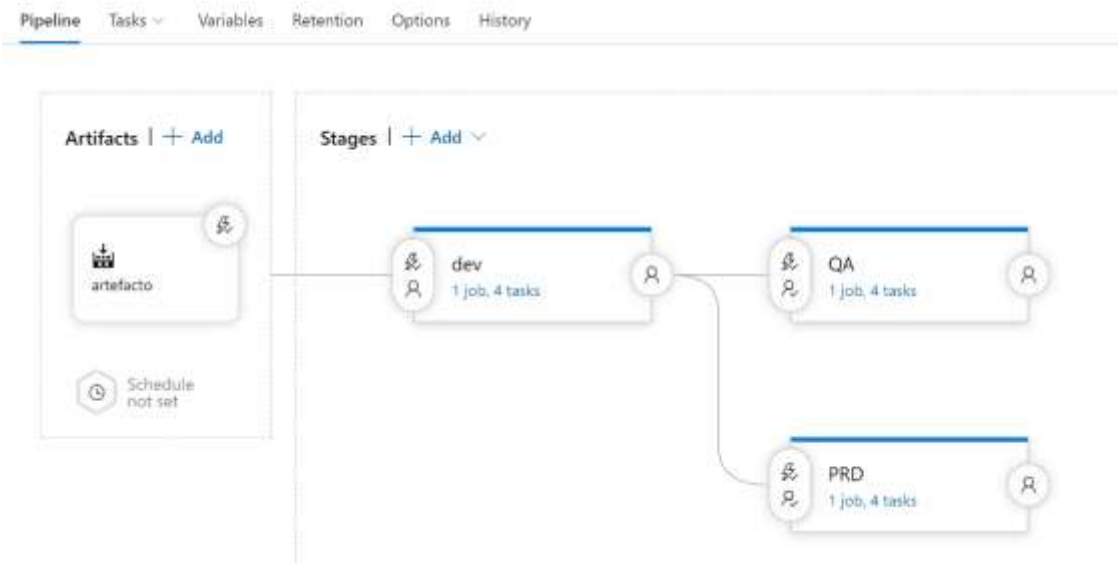
- Se crea el pipeline de integración continua. En el pipeline se relaciona con el archivo docker y los archivos yml.
- Se crea los artefactos mediante la ejecución del pipeline.

Figura 13. Pipeline en azure devops.

Fuente (Elaboración Propia)

- Se crea el release utilizando los artefactos creados previamente. En el reléase se detalla los ambientes donde estará desplegado el microservicio.

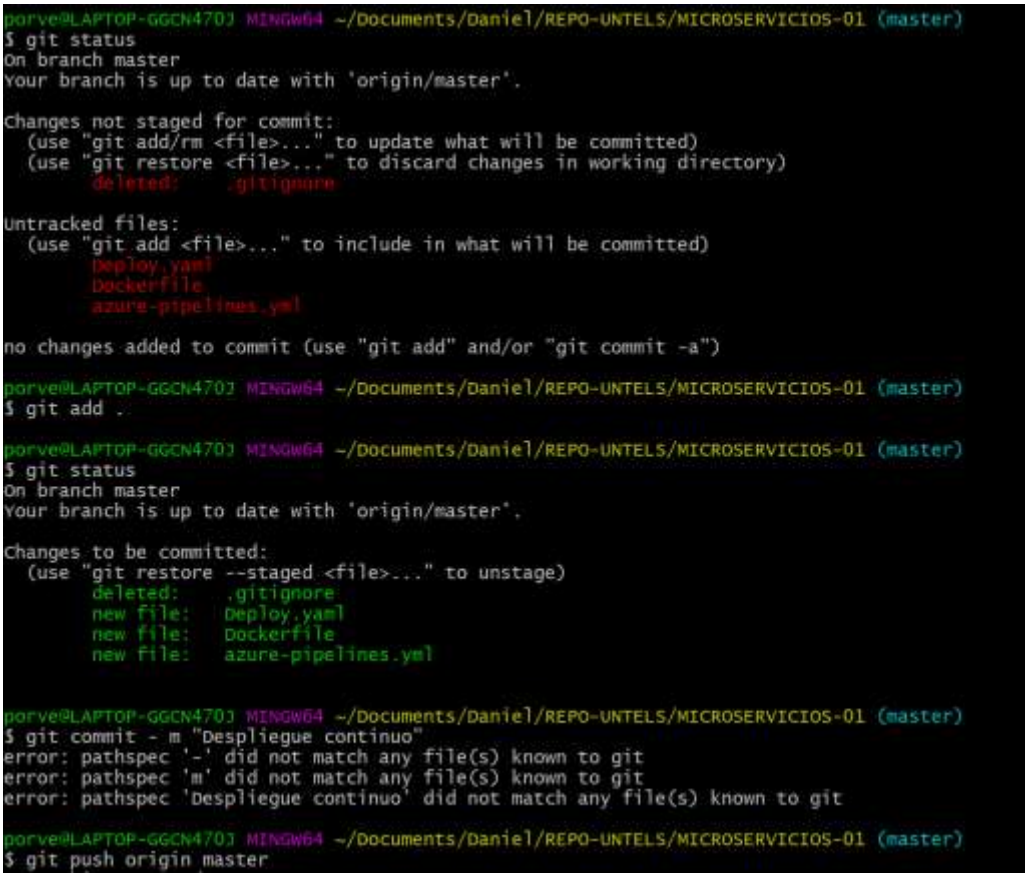
Figura 14. Release en azure devops.



Fuente (Elaboración Propia)

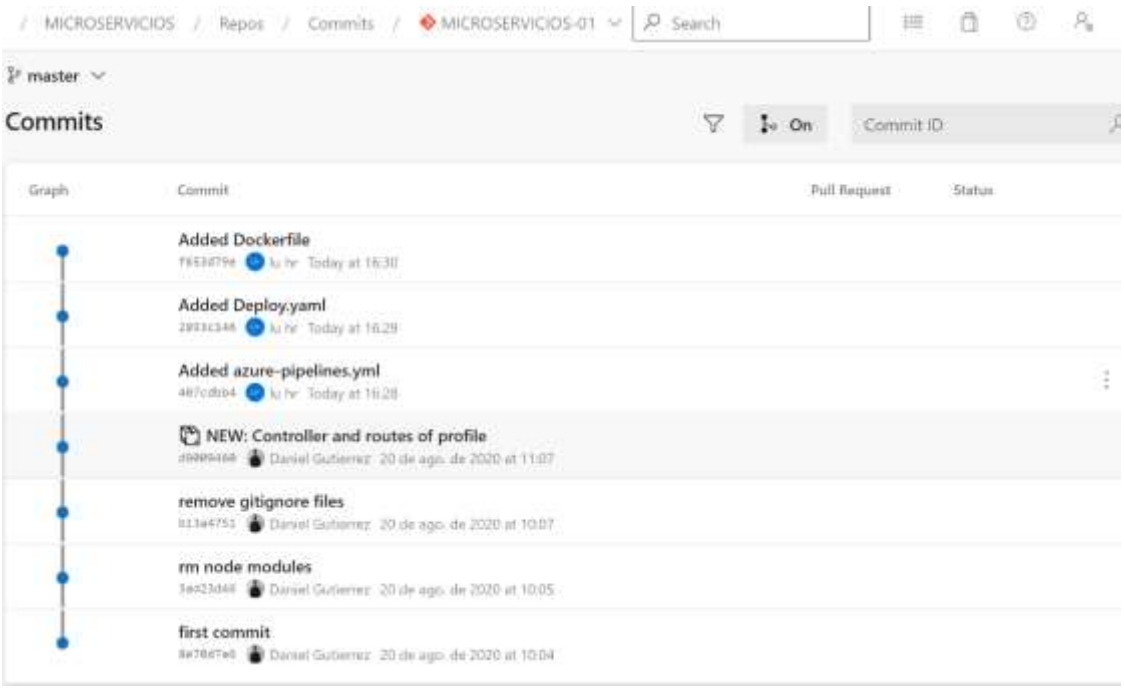
Luego mediante la práctica de integración continua y la herramienta git, el desarrollador entrega los cambios del desarrollo del microservicio, creados con lel IDE Visual Studio Code, en la rama master del repositorio central.

Figura 15. Comandos git



Fuente (Elaboración Propia)

Figura 16. Commit en azure devops.



Fuente (Elaboración Propia)

Se debe generar el compilado mediante integración continua (CI). Gracias a la práctica del despliegue continuo por cada push que el desarrollador ejecuta se complia el microservicio en el ambiente de desarrollo y calidad de forma automática permitiendo al desarrollador y al test, cada uno en su respectivo ambiente asignado, realizar pruebas unitarias, pruebas análisis de código o pruebas de cobertura.

3.2.2.1.5.2. Desarrollo de Sprint 2.

3.2.2.1.5.2.1. Pruebas en el ambiente de calidad de los microservicios.

A través del postman realiza el consumo de los endpoint expuestos por los microservicios.

Validar utilizando la aplicación del rancher que los pod del microservicio escalen cada vez que hay demasiado tráfico ocasionadas por las pruebas de estrés

Una vez que el test termina de realizar las pruebas en el ambiente de calidad y da su aprobación de las pruebas, para realizar la entrega continua del microservicio terminado en producción se tiene que esperar la previa aprobación que se realiza por el comité de cambios.

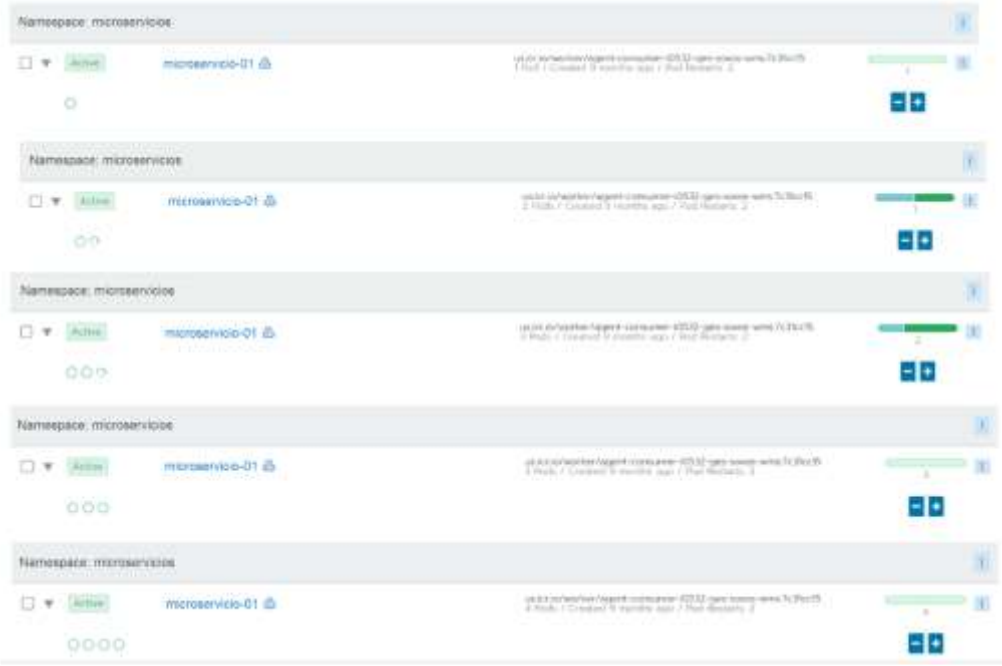
3.2.2.1.5.2.2. Pase a producción del microservicio y validación de escabilidad en los clusteres de producción.

- Gestión para la aprobación del pase a producción.

- Activar el release de producción para que se realice el despliegue del microservicio en el cluster de producción.

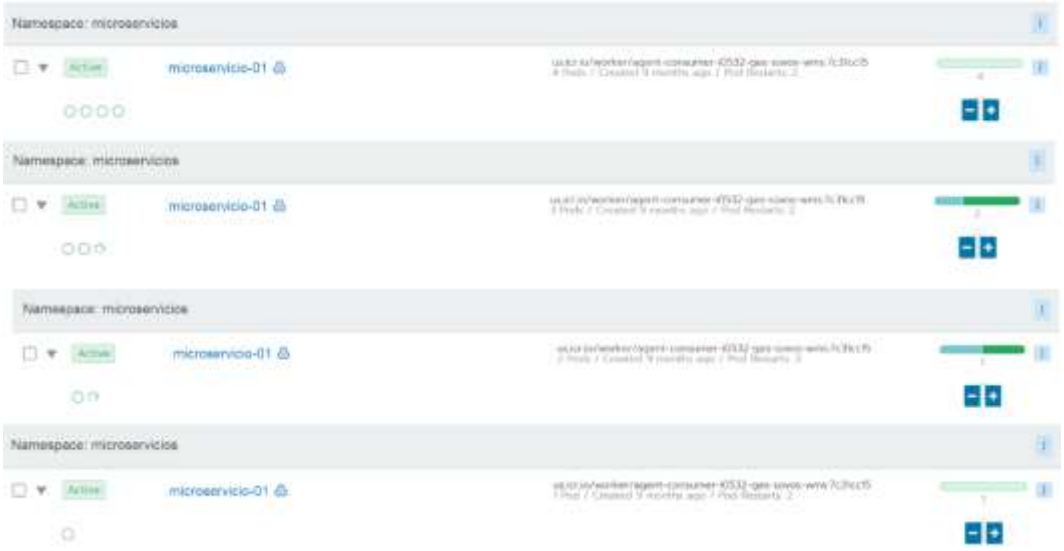
- Seguimiento de la escabilidad del microservicio en producción.

Figura 17. Autoescalamiento horizontal del pod en el Rancher (Aumenta)



Fuente (Elaboración Propia)

Figura 18. Autoescalamiento horizontal del pod en el Rancher (Disminuye)



Fuente (Elaboración Propia)

3.3. Resultados.

Mediante el uso del lenguaje de programación node,js se crea con el IDE Visual Studio Code la estructura del microservicio que consiste en exponer los enpoint de los servicios web y el método para realizar acciones ETL con los registros de la bd mongodb. También se crea los archivos para el despliegue del microservicio en los clústeres de kubernetes.

Con los elementos de la plataforma azure devops se realiza la ejecución del despliegue continuo de una manera automática y se logra desplegar el microservicio en los clústeres de desarrollo y calidad.

Luego de pruebas de estrés en el ambiente de calidad se logra comprobar que el microservicio realiza el autoescalamiento de forma horizontal.

Una vez pasado el microservicio al ambiente productivo se logra evidenciar la disminución de incidentes ocasionados por el excesivo uso de recursos.

CONCLUSIONES

La única manera posible de implementar un microservicio con kubernetes para mejorar la escabilidad en la arquitectura de la aplicación es mediante prácticas devops que consiste en integración, entrega y despliegue continuo. Hay muchas herramientas que permiten realizar estas prácticas sin embargo para este proyecto se utilizó la plataforma azure devops porque Microsoft era el partner tecnológico de la empresa donde se llevó a cabo el proyecto.

La integración de las tecnologías de azure devops con kubernetes para realizar el microservicio y desplegarlo en un clúster resultó ser una excelente opción ya que permitió simplificar las tareas y generar un producto en menos de dos semanas en el ambiente productivo.

RECOMENDACIONES

Se recomienda realizar un estudio, trabajo o investigación sobre la técnica de la ingeniería de software llamada refactorización que sirve para reestructurar un código fuente sin cambiar su comportamiento ni funcionalidad. Como parte del proceso de desarrollo de software es muy importante realizarla ya que permite la eliminación de código fuente defectuoso que muchas veces provoca el mal uso de recursos en las aplicaciones.

Para términos académicos donde estudiantes desean realizar microservicios con una herramienta que utilice las prácticas DevOps y sea gratuita se recomienda el servidor de automatización Jenkis.

REFERENCIAS BIBLIOGRÁFICAS

Manuel Perez C. (2019). libro: Arquitecturas basadas en microservicios.

Thomas Erl. (2016). Segunda edición del libro: Service-Oriented Architecture: Analysis and Design for Services and Microservices.

Fowler & Lewis. (2014). Artículo de martin fowler Microservices

Burns, B., Beda, J., & Hightower, K. (2018). Kubernetes. Dpunkt.

Luksa, M. (2017). Kubernetes in action. Simon and Schuster.

Sayfan, G. (2017). Mastering kubernetes. Packt Publishing Ltd.

Burns, B., Beda, J., & Hightower, K. (2019). Kubernetes: up and running: dive into the future of infrastructure. O'Reilly Media.

Jordán Marcos, D. (2020). Estudio sobre las metodologías ágiles y metodologías tradicionales para gestión de proyectos de software.

González Hernández, B. Seguridad en Docker.

Ken Schwaber y Jeff Sutherland, La guía de Scrum, 2014.

Cohn, M. User stories applied: for Agile software development. Addison Wesley, Boston, 2004.

Menzinsky, López, & Palacio, Scrum Manager, 2016.

Real Ixcayau, E. O. (2021). Propuesta para actualización curricular en el área de software incorporando talleres de Docker y Kubernetes (Doctoral dissertation, Universidad de San Carlos de Guatemala).

Cervieri Carrau, V. V. (2019). Cloud autoscaling performance evaluation with real-world like loads and threshold tuning (Doctoral dissertation, Universitat Politècnica de València).

ANEXOS

ANEXO 1

Encuesta realizada por red hat a las empresas