UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA:

INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:

Ingeniero de Sistemas

TEMA:

DISEÑO DE UN PROTOTIPO DE UNA ARQUITECTURA BASADA EN MICROSERVICIOS PARA LA INTEGRACIÓN DE APLICACIONES WEB ALTAMENTE TRANSACCIONALES. CASO: ENTIDADES FINANCIERAS

AUTOR:

DIEGO FERNANDO CHICAIZA RIOS

TUTOR:

FRANKLIN EDMUNDO HURTADO LARREA

Quito, marzo del 2020

CESIÓN DE DERECHOS DE AUTOR

Yo, Diego Fernando Chicaiza Rios, con documento de identificación 1715103212, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud que soy autor del trabajo de titulación intitulado: "DISEÑO DE UN PROTOTIPO DE UNA ARQUITECTURA BASADA EN MICROSERVICIOS PARA LA INTEGRACIÓN DE APLICACIONES WEB ALTAMENTE TRANSACCIONALES. CASO: ENTIDADES FINANCIERAS" mismo que 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 aplicación a lo determinado en la Ley de Propiedad Intelectual, en mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato digital a la biblioteca de la Universidad Politécnica Salesiana.

Diego Fernando Chicaiza Rios

1715103212

Quito, marzo del 2020

DECLARACIÓN DE COAUTORÍA DEL DOCENTE TUTOR

Yo, declaro que bajo mi dirección y asesoría fue desarrollado el trabajo de titulación "DISEÑO DE UN PROTOTIPO DE UNA ARQUITECTURA BASADA EN MICROSERVICIOS PARA LA INTEGRACIÓN DE APLICACIONES WEB ALTAMENTE TRANSACCIONALES. CASO: ENTIDADES FINANCIERAS", realizado por Diego Fernando Chicaiza Rios, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana para ser considerado como trabajo final de titulación.

Quito, marzo del 2020

Ing. Franklin Edmundo Hurtado Larrea

1713382016

DEDICATORIA

La presente tesis dedico a Dios que ha sido mi guía y mi fortaleza para seguir adelante, ha sido mi inspiración para conseguir cosas importantes en mi carrera profesional, a mi familia en especial a mi madre que ha sido un pilar fundamental en mi formación académica, a mi esposa e hijos que son el motor de mi vida y mi inspiración para mejorar cada día.

AGRADECIMIENTO

Agradezco a la Universidad Politécnica Salesiana por brindarme la formación académica, por los valores inculcados en todos estos años de estudio para llegar a ser un profesional en la carrera técnica de Ingeniería en Sistemas Informáticos.

A mi familia que me ha apoyado en todas las etapas de mi vida profesional, siendo un pilar fundamental en mis logros académicos que pueda conseguir.

A mis padres, con su esfuerzo y guía me han sabido llevar por los caminos del bien, en especial a mi madre que siempre me brindo un apoyo incondicional para que sea un profesional.

A mi Tutor Franklin Hurtado, que con su experiencia y conocimiento durante todo el periodo de las tutorías y en la fase académica me guio en la parte técnica y humana hasta obtener el resultado esperado.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Desde el principio, la ingeniería de software buscó desarrollar, crear y mantener aplicaciones informáticas que satisfagan las necesidades de las organizaciones y los requisitos técnicos de los consumidores. Por lo tanto, la arquitectura de microservicios nació como un enfoque para mejorar las arquitecturas tradicionales (monolíticas), enfocadas en la estabilidad, la confiabilidad, un alto grado de tolerancia a fallas y una escala simple.

“El proyecto técnico pretende dar lineamientos para la construcción de una arquitectura basada en microservicios, el diseño puede ser implementado utilizando cualquier lenguaje de programación, en escenarios simples y complejos”, esto dependerá del análisis técnico que realice al inicio del proyecto.

El diseño de esta arquitectura está orientada a aplicaciones altamente transaccionales para el caso instituciones financieras que tienen una arquitectura compleja se busca dar pautas para mejorar su diseño.

Palabras claves: instituciones financieras, microservicios, arquitectura monolítica

ABSTRACT

Currently, financially in both the public and private sectors, software development is carried out to meet the automation needs of internal processes. This development corresponds to the trends imposed by the platform, by the programming language or by the experience in the development of the domain, which becomes the implementation of traditional or monolithic construction systems. From the beginning, software engineering sought to develop, create and maintain computer applications that meet the needs of organizations and the technical requirements of consumers. Therefore, the micro service architecture was born as an approach to improve traditional architectures, focused on stability, reliability, a high degree of fault tolerance and a simple scale.

Keywords: financial institution, financial services, micro services, monolithic architecture

INTRODUCCIÓN

Antecedentes

"El segmento financiero siempre está eligiendo la tecnología como medio para desarrollar sus actividades, hasta el punto de impulsar la economía, los avances tecnológicos como:", inteligencia artificial, el cloud y big data, han provocado un cambio en el comportamiento del consumidor y los modelos de negocio, dado lugar a la cuarta revolución industrial (Álvarez, 2018), la digitalización es posible gracias a las funcionalidades que brindan los canales digitales, banca web, banca móvil, los cuales en la actualidad están posicionados como los más completos e innovadores del mercado (Udagawa, 2016).

Por otro lado, la naturaleza virtual de los servicios ofrecidos en el mundo financiero significa que el crecimiento en el uso de los servicios de banca por Internet tiene un impacto significativo en el sector bancario.

A fines de la década del 60 ya existía hardware informático que mecanizaba ciertas aplicaciones específicas del sector financiero pero, como lo señala (Fanjul Suárez & Valdunciel Bustos, 2008) la conexión entre la banca y la tecnología comenzó en la década del 70, cuando se introdujo el teleproceso en el sector bancario, lo que permite el procesamiento de datos ingresados en terminales conectadas a computadoras centrales con alta capacidad y fiabilidad para procesar grandes volúmenes de información.

Está claro que el comienzo de la banca sin sucursales o la telebanca que de acuerdo con el Banco Central Europeo (BCE), se refiere a la prestación de servicios personal bancario sin contacto entre los empleados del banco y sus clientes. Este banco también se conocerá como banca electrónica o digital, ya que está respaldado principalmente por los avances tecnológicos (Fanjul Suárez & Valdunciel Bustos, 2008).

La consolidación de las aplicaciones en línea y el aumento en el intercambio de información entre entidades y clientes conducen al desarrollo de sistemas de transferencia electrónica de datos (EDI), basados en la expansión de redes cerradas que garantizan la seguridad de las transacciones lo cual constituye uno de los principales fundamentos de la banca electrónica (SERES, 2017).

El sistema bancario español ha sido pionero en este proceso de transformación tecnológica, participando directamente en los primeros desarrollos de terminales de banca electrónica y en la implementación del procesamiento remoto de datos. España se destaca en Europa por proporcionar soluciones tecnológicas avanzadas y sofisticadas; además, como señalan Lozano y Sebastián (2006) la posterior creación del Sistema Nacional de Compensación Electrónica (SNCE) ha permitido que el sistema financiero español sea considerado uno de los más eficientes.

La fase de desarrollo de nuevos canales de servicio al cliente comenzó en la década de 1990, se ofrecen diferentes alternativas a cada cliente para usar según sus necesidades y circunstancias específicas. Así, en 1992 apareció el sistema de banca telefónica que se convirtió en otro canal para sucursales y cajeros automáticos, esta herramienta ofrece más flexibilidad en los horarios para que la gestión de operaciones se pueda realizar de inmediato, pero como no hay presencia física de los empleados, esto no convence a los clientes (Emilio Ontiveros Baeza, 2012).

El canal que ha tenido el mayor impacto en el sector financiero es Internet que se unió al banco a mediados de la década del 90, ofreciendo flexibilidad y accesibilidad en respuesta al usuario actual que quiere trabajar en cualquier momento, en cualquier lugar y en tiempo real (Emilio Ontiveros Baeza, 2012).

Desde 2002 las instituciones financieras han iniciado un cambio estratégico para rentabilizar al internet y están comenzando a considerar este canal como una forma de diferenciarse con

nuevas oportunidades de negocios. Una estrategia más activa comienza con la mejora del rendimiento y con un suministro más diversificado de productos y servicios (transacciones interbancarias, créditos, suscripciones hipotecarias, etc.) (Emilio Ontiveros Baeza, 2012).

En 2004, la calidad de las conexiones a Internet mejoró, los precios de las computadoras cayeron y el rendimiento de las computadoras aumentó, creando un ambiente ideal para el crecimiento de la banca por Internet. Gracias al gran desarrollo tecnológico, ha aparecido una nueva estructura web, web 2.0 o de red social, más rápida, interactiva y visual, en la que el usuario es el personaje principal y puede generar contenido. Este es un gran cambio con respecto a los sitios web anteriores (Web 1.0), con páginas estáticas programadas en HTML.

Problema

En los sistemas altamente transaccionales existen mecanismos para soportar la carga de peticiones realizadas a una aplicación web, pero no es la solución buscar por medio de aumento de hardware tener disponible el sistema, las entidades financieras que sus transacciones tienen a crecer deben buscar soluciones tecnológicas integrales para ganar competitividad y ofrecer un buen servicio al cliente, es por tal motivo que se plantea un diseño de una arquitectura basada en microservicios para responder a peticiones solicitadas por el usuario en los tiempos requeridos evitando los famosos cuellos de botella, hasta caídas de sistema.

Actualmente, una aplicación altamente transaccional tiene que soportar mucha carga de peticiones, según un estudio realizado por diario el comercio, el 73% de las transacciones que se realizaron en instituciones financieras en el 2017 ocupó los canales electrónicos. “El Banco Central del Ecuador informó que la tasa de crecimiento del uso de pagos digitales durante los últimos cinco años fue del 16%, pero entre 2016 y 2017 aumentó en 30% (Tapia, 2018)”.

En Banco Pichincha también crece el número de clientes que prefieren canales electrónicos (Tapia, 2018). En este banco, el último año se redujeron en 4.000.000 las transacciones desde oficinas (Tapia, 2018), Aumentando el uso de los canales digitales actualmente es de más del 50% de las transacciones bancarias se realizan en línea (Tapia, 2018).

El tema planteado servirá como guía para diseñar de una arquitectura bien definida, que cumpla con las necesidades tecnológicas, requerimientos funcionales y no funcionales de una empresa (Entidades financieras).

Justificación

El presente trabajo técnico propone una nueva arquitectura de software basada en microservicios que permita dar lineamientos para la construcción de sistemas informáticos en capa media (middleware), con énfasis en la escalabilidad, disponibilidad, rendimiento y fácil mantenimiento.

La importancia que tiene una arquitectura en microservicios radica en los beneficios que aporta en aspectos tecnológicos, el monitoreo ayuda a validar la salud de la aplicación, siguiendo los lineamientos expuestos en este trabajo técnico tendremos registros de log estructurados que son fáciles de interpretar, las herramientas de monitoreo podrán integrarse a estos repositorios para leer esta información, y presentar en sus tableros de monitoreo, esto ayuda a tener un control de la aplicación en tiempo real, recibir alertas tempranas de posibles fallos en nuestro sistema (Gupta, 2015).

Una arquitectura basada en microservicios granulares, es un facilitador para las prácticas de integración. También ayuda a entregar más rápido las nuevas funciones en la aplicación, ya que la composición fina de las funciones le permite ejecutar y probar microservicios en forma aislada y desarrollarlos de manera autónoma, manteniendo claros las interfaces entre ellos. Siempre que no cambie las interfaces o los contratos, puede cambiar la implementación interna de cualquier microservicio o agregar nuevas funcionalidades sin romper otros microservicios (Newman, 2015).

Adicionalmente se puede mencionar que los microservicios se pueden escalar de forma independiente. En lugar de tener una aplicación única monolítica, que se debe escalar como una unidad, se pueden escalar individualmente microservicios específicos. De esta forma, puede escalar sólo el área funcional que necesita más potencia de procesamiento o ancho de banda de

red para soportar la demanda, en lugar de ampliar otras áreas de la aplicación que no necesitan escalarse. Eso significa ahorros de costos porque se aprovechan mejor los recursos y se necesita menos hardware (Namit Tanasseri, 2017).

Las aplicaciones empresariales pueden ser complejas por la lógica de negocio embebida en sus pantallas, los cambios solicitados deben ser bien analizados para no generar impacto a los proveedores y consumidores del servicio, al tener un monolito se afecta la funcionalidad en las aplicaciones, encareciendo el valor de mantenimiento (Richardson, 2019).

El principio de la transformación digital es la facilidad y la rapidez al cambio sin afectar rendimiento de una petición solicitada o proceso involucrado, por todo lo expuesto es que nace la arquitectura de microservicio que está relacionado con la agilidad por ende con metodologías ágiles.

Objetivo General

Diseñar e implementar un prototipo de arquitectura basada en microservicios para la integración de aplicaciones web orientadas a entidades financieras altamente transaccionales.

Objetivos Específicos

Diseñar una arquitectura escalable aplicando principios de diseño de microservicios en aplicaciones altamente transaccionales.

Implementar un escenario de pruebas para la ejecución de los microservicios basado en la arquitectura propuesta.

Desarrollar tres interfaces estandarizadas para la construcción de microservicios utilizando el marco de trabajo que será implementado.

Demostrar las ventajas que ofrece esta arquitectura respecto al rendimiento, despliegue de los microservicios y la facilidad de realizar cambios que solicita el cliente.

Garantizar la disponibilidad de los microservicios para que las aplicaciones puedan consumirlos y presentar la información solicitada por el cliente.

Con el planteamiento de este modelo se pretende mejorar el nivel transaccional de la aplicación con bajos costos de mantenimiento.

Marco metodológico

Análisis y Justificación de la Metodología

Para la ejecución de las historias de usuario definidas en el análisis de requerimientos se proceden a crear los siguientes sprints.

El sprint 0, tiene relación con los componentes transversales tales como diseño y construcción de marco de trabajo, estos permitirán ganar velocidad en los desarrollos de los microservicios tendrán embebidos librerías que permitirán la reutilización de código, es importante planificar tiempo para realizar estas tareas porque son componentes bases en nuestra arquitectura que ayudaran a estandarizar el desarrollo, la duración de este sprint será de 4 semanas, de acuerdo a sprint.org. A partir del sprint 1, 2, 3, 4 se entregará funcionalidad al negocio con las liberaciones que se realizará en cada uno de ellos, la duración de cada uno de ellos será de dos semanas.

Definición de sprints

Para agregar valor al negocio es necesario crear sprints según el marco de trabajo utilizado, en nuestro caso se plantea los siguientes:

Sprint 0: En este periodo de trabajó el equipo realizará el análisis, diseño de la arquitectura planteada relacionada con la historia de usuario HU001, y la construcción del marco de trabajo componente transversal que tendrá el mecanismo de log y la estructura de los microservicios simples y compuestos, este requerimiento está relacionado con la historia de usuario HU002, se da como finalizado cuando todas sus tareas estén en el estado Hecho de acuerdo a los criterios de aceptación que se plantea en las historias de usuario.

Sprint 1: En este periodo de trabajo el equipo realizará el diseño y la construcción de la historia de usuario HU002, la cual solicita la creación de un microservicio que permita visualizar los datos demográficos de un cliente este sprint estará finalizado cuando las tareas creadas estén en estado hecho.

Sprint 2: En este periodo de trabajo el equipo realizará el diseño y la construcción de la historia de usuario HU003, esta solicita la creación de un microservicio que permita visualizar los datos personales de un cliente, este sprint estará finalizado cuando las tareas creadas estén en estado hecho.

Sprint 3: En este periodo de trabajo el equipo realizará el diseño y la construcción de la historia de usuario HU004, esta solicita la creación de un microservicio que permita visualizar los productos que tiene un cliente, este sprint estará finalizado cuando las tareas creadas estén en estado hecho.

Sprint 4: En este periodo de trabajo el equipo realizará los escenarios de pruebas y ejecución de estos, que hace referencia a la historia de usuario HU005, este sprint estará finalizado cuando las tareas creadas estén en estado hecho.

En el desarrollo de cada sprint se empleará el siguiente marco de trabajo:

Reunión para la planificación del sprint. En ella, se divide el tiempo de duración del sprint, así como el objetivo y entregable del mismo. Además, el equipo de desarrollo deberá saber cómo realizarlo.

Scrum semanal. Se realiza la sincronización de actividades para elaborar el plan de cada semana de acuerdo al marco de trabajo establecido por el tutor del proyecto técnico.

El trabajo de desarrollo durante el sprint. Se asegurará que los objetivos se están cumpliendo, que no se producen cambios que alteran el objetivo del sprint y se mantiene una retroalimentación constante con el cliente o dueño del proyecto. En este caso el tutor del proyecto técnico. Revisión del sprint, reunión con el cliente o dueño del proyecto, en la que se estudia y revisa el Product Backlog del sprint. Se definen los aspectos a cambiar, en caso de ser necesario.

Planeación y estimación

Consiste en procesos relacionados con la planificación y las tareas de estimación, incluye los siguientes procesos:

- Creación de historias de usuario.
- Creación de tareas.
- Estimación de tareas.
- Creación de Sprint Backlog.

En esta fase el equipo procede a realizar la estimación de lo solicitado en las historias de usuario, previa validación de product owner para realizar esta actividad se realiza lo siguiente:

Cada participante debe tener instalado en su móvil la aplicación scrum poker, luego se debe ejecutar los siguientes pasos.

El product owner, presenta la historia de usuario para ser estimada. Y se despejan las dudas que los participantes puedan tener. Todos los participantes proceden a realizar su estimación en forma secreta, sin influenciar al resto del equipo, no se debe dejar ver la carta con el valor que muestra la aplicación luego de haberla elegida. Una vez que todos los integrantes han estimado, se presentan los valores establecidos en la aplicación se procede a discutir las razones de los valores estimados.

Scrum es un marco de trabajo que está orientado a un grupo de personas que tienen asignado su rol, para este caso práctico todos los roles que propone esta metodología serán desempeñados por el alumno, excepto el rol de product owner que desempeñará el tutor del proyecto técnico las estimaciones fueron realizadas en base al juicio experto, es la principal premisa en que se basa la estimación ágil es el conocimiento, la experiencia del equipo.

Se evalúa el número de horas según la complejidad emitidas por los participantes, esta tarea se ejecuta hasta llegar a un consenso con todo el equipo el tiempo estimado es el que se toma en cuenta para el desarrollo de la tarea.

Estimación diseño e implementación de prototipo de una arquitectura basada en microservicio para aplicaciones altamente transaccionales.

En la Figura 1, se visualiza las estimaciones realizadas de todos los componentes que más adelante serán priorizados de acuerdo con los sprints definidos.

Estimación de tareas del proyecto técnico

Figura 1. Elaboración de tareas

Elaborado por: Diego Chicaiza

Creación de Sprint backlog.

Para la creación del sprint backlog es necesario tener claro los requerimientos y el tiempo en el cual se debe entregar las tareas, esto es importante en esta metodología porque se tiene la visibilidad de las tareas a ser priorizadas que son generadas a través de las historias de usuario.

Sprint 0 Backlog.

En la Figura 2, se observa las tareas del product backlog priorizadas para este sprint 0.

Tareas product backlog

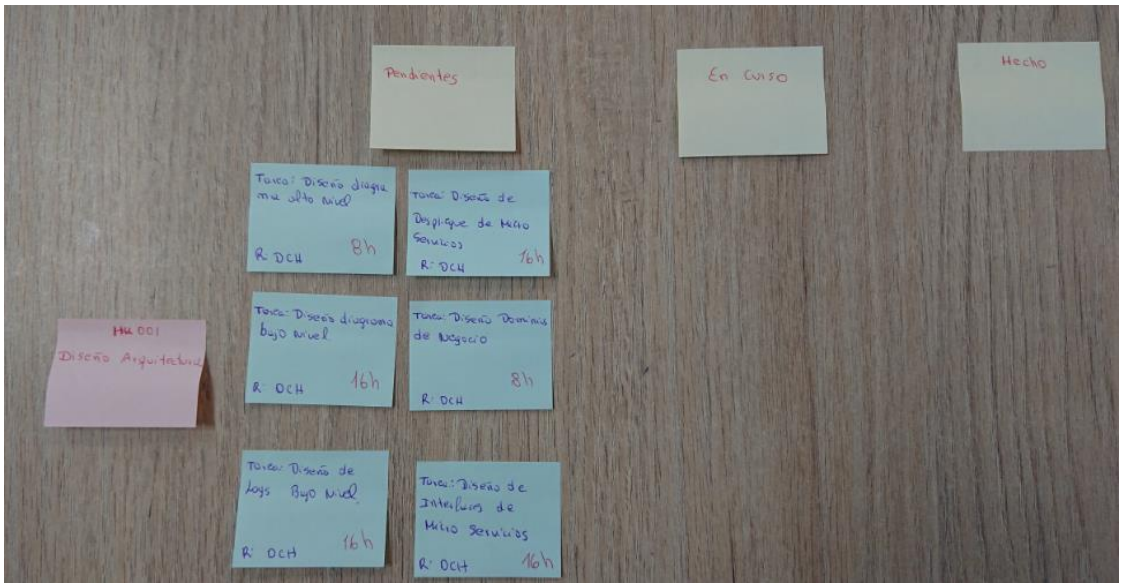

Figura 2. Elaboración de tareas product backlog

Elaborado por: Diego Chicaiza

De acuerdo a lo definido estas son las tareas que se ejecutaran en este sprint hacen referencia a las historias de usuario HU001, luego inician las tareas de la HU002, las tareas empiezan en el estado pendiente de acuerdo a la priorización se irán moviendo a los siguientes estados.

En la Figura 3, se observa como las tareas van cambiando de estado de acuerdo con la priorización de ejecución de tareas.

Estado de tareas del sprint en ejecución

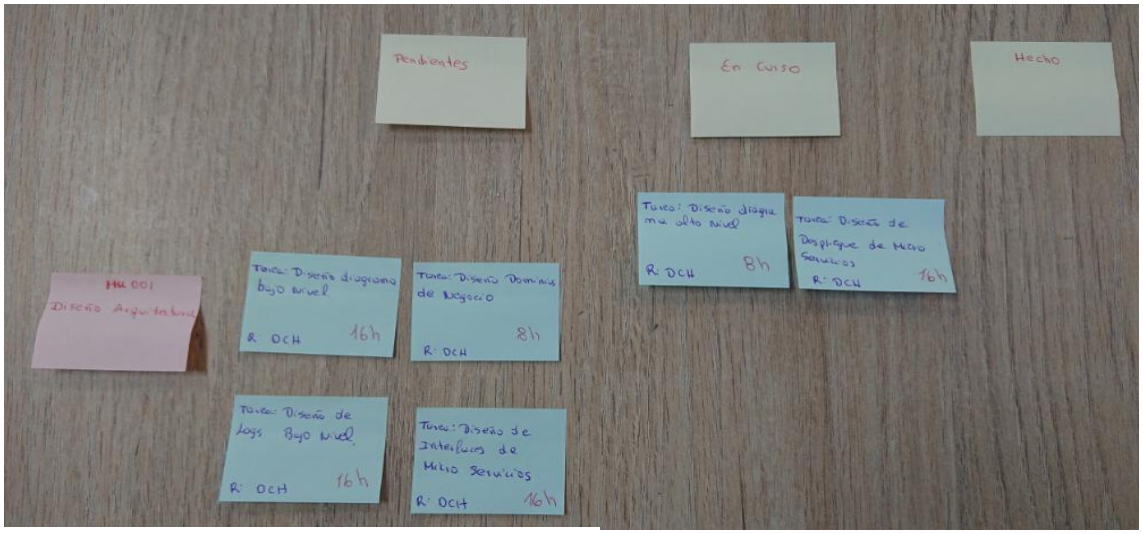

Figura 3. Descripción del estado de tareas

Elaborado por: Diego Chicaiza

En la Figura 4, se puede observar las tareas y su iteración que deben cumplir en este sprint hasta que lleguen al estado hecho que significa que fue cumplida 100% la tarea asignada.

Tareas estado hecho, finalizaron de acuerdo a la estimación

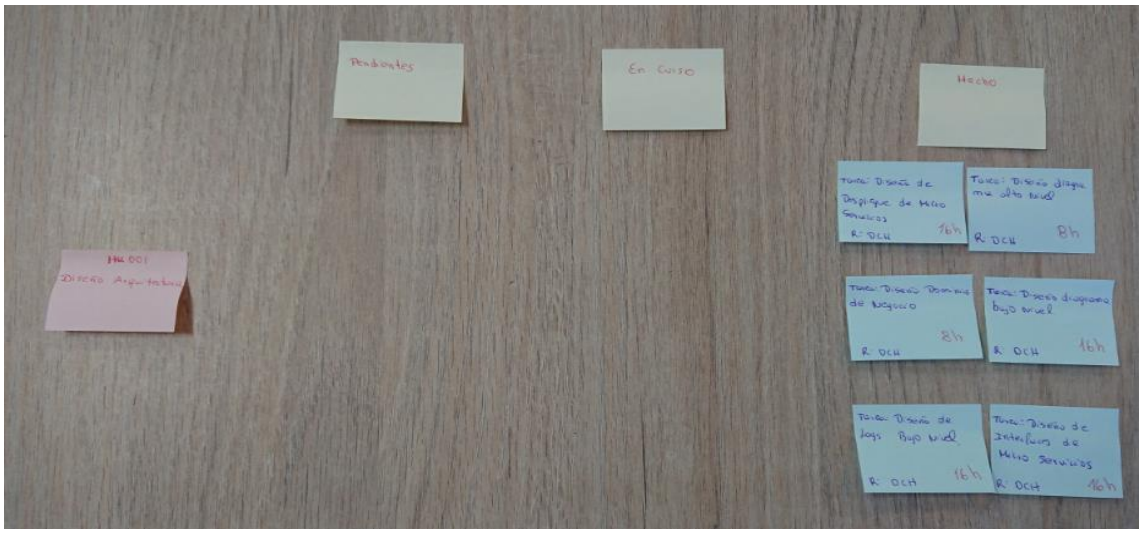

Figura 4. Elaboración de tareas de hecho

Elaborado por: Diego Chicaiza

Este proceso se realizará para todos los sprints hasta llegar a finalizar todas sus tareas, en el scrum board deben estar las tareas en estado hecho.

En la Figura 5, se observa las tareas finalizadas de todos los sprint ejecutados.

Tareas finalizadas según las estimaciones realizadas

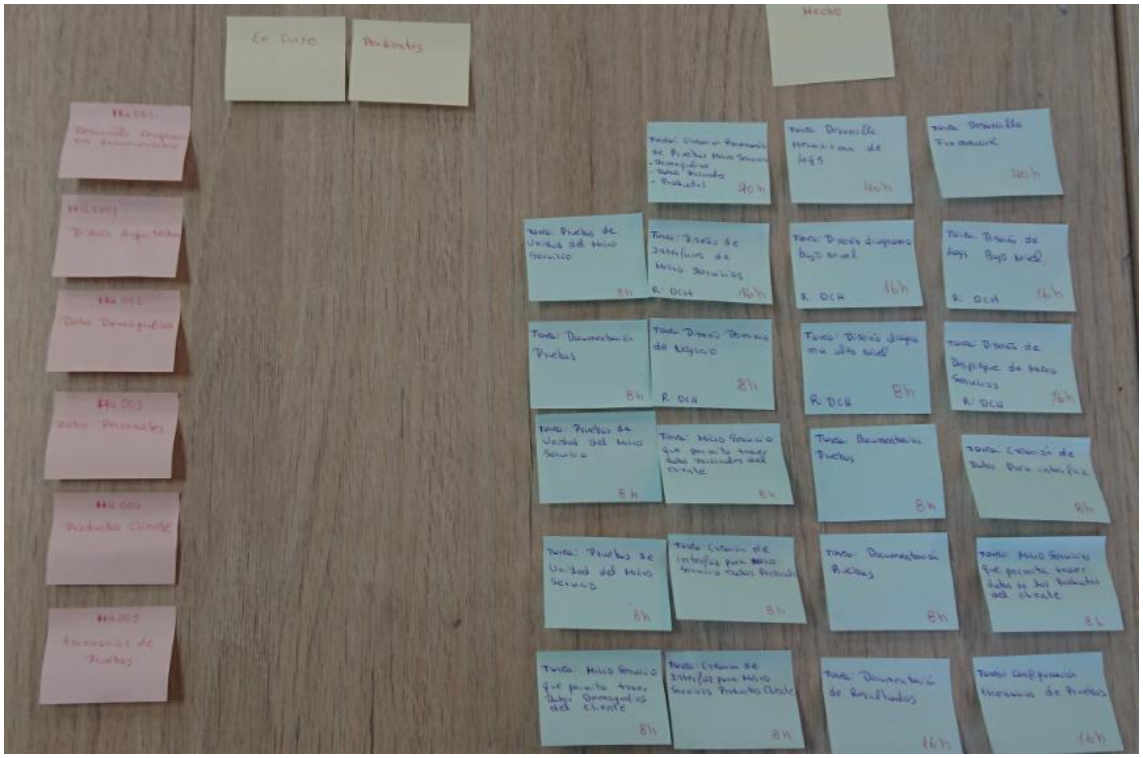

Figura 5. Descripción de tareas finalizadas

Elaborado por: Diego Chicaiza

Este método de estimación fue de mucha ayuda porque parte de todas las tareas que tiene el proyecto asociada a cada sprint, la manera visual que se presenta en los tableros da visibilidad del real avance de las tareas, alertando de marera temprana si se presentan impedimentos para llegar al objetivo, tareas en estado hecho.

CAPÍTULO 1

1. MARCO TEÓRICO

1.1. Ingeniería de Software

(Pressman, 2015), definen el desarrollo de software como: el uso de un enfoque sistemático, disciplinado y cuantitativo para el desarrollo de sistemas informáticos, operación y mantenimiento de software; en otras palabras, el uso de la tecnología en software.

1.1.1. Modelos de Proceso de Software

La simplificación del modelo de desarrollo de software debe ser única, en general, las actividades de modelado, sus relaciones, resultados y partes interesadas que la conforman, así como todos los modelos permite la simplificación y la abstracción de la realidad. Existen dos principales enfoques que son según (Pressman, 2015):

- El tradicional o lineal, del cual derivan los modelos cascada, iterativo y por componentes.
- El ágil, de un modelo iterativo-incremental

1.2. Metodologías de Desarrollo Ágil

Tradicionalmente, el proceso en cascada se ha utilizado en la implementación de proyectos de software, comenzando con la definición de requisitos, la planificación completa del proyecto, el diseño de software y finalmente la creación y prueba de productos. De esta manera, se creó una gran cantidad de software, pero con el tiempo, las empresas han seguido enfrentando los mismos problemas asociados con el proceso de desarrollo en cascada. La historia del desarrollo

ágil comenzó cuando se encontró otra forma de resolver estos problemas, comenzando con el manifiesto de desarrollo ágil de software (Patricio Letelier, 2006):

- Personas e interacciones sobre procesos y herramientas.
- El software funciona por encima de la documentación.
- La colaboración del cliente por encima de la negociación del contrato.
- La respuesta al cambio por encima del seguimiento de un plan.

La siguiente tabla resume los aspectos más relevantes de las metodologías de desarrollo tradicionales en comparación con las metodologías de desarrollo ágil:

Tabla 1. Comparativa metodología tradicional vs metodología ágil

Tradicionales	Ágiles
Predictivos	Adaptativos
Orientados a procesos	Orientados a personas
Proceso rígido	Proceso flexible
Se concibe como un proyecto	Un proyecto es subdividido en varios proyectos más pequeños
Poca comunicación con el cliente	Comunicación constante con el cliente
Entrega de software al finalizar el desarrollo	Entregas constantes de software
Documentación extensa	Poca documentación

Nota: Comparación de metodologías tradicional vs ágil (Torres, 2003)

Según el trabajo realizado por Montoya, Uribe, & Rodríguez (2013), las principales razones para adoptar enfoques ágiles están relacionadas con: cambio rápido, la necesidad de resultados rápidos y requisitos emergentes; y por otro lado, estas metodologías tienen muchas ventajas,

entre las cuales se encuentran: una excelente adaptación a los cambios y al dinamismo; se enfocan en personas y valores, no en procesos y planes; mitigar riesgos demostrando valores y funcionalidad en el proceso de desarrollo; proporcionar un tiempo de comercialización más rápido; aumentar la productividad (reducir la documentación) y, si el proyecto no es factible, su falla ocurre más o menos rápidamente y sin consecuencias particulares.

En la Figura 6, se observa la iteración del flujo de trabajo en la ejecución de una tarea.

Marco de trabajo Scrum

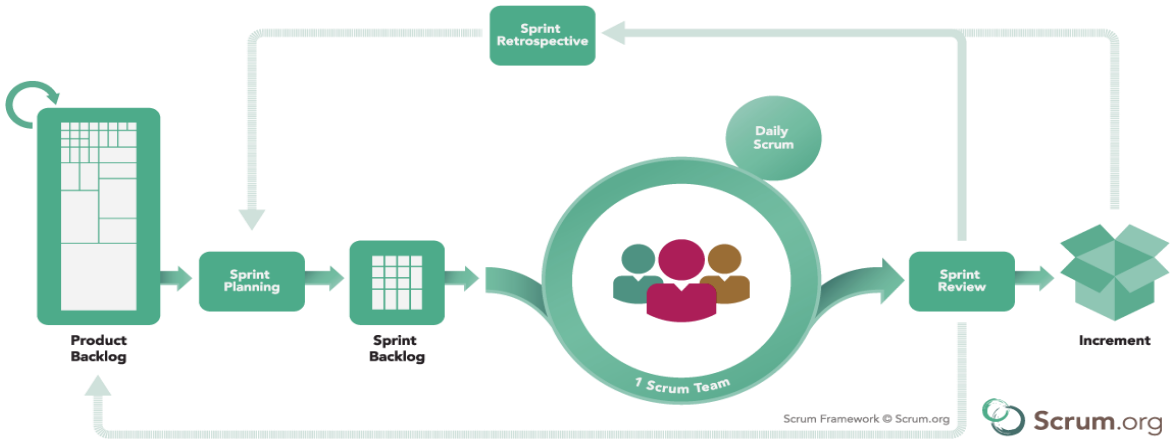

Figura 6. Flujo del Marco de trabajo

Fuente: (Scrum.org, 2016)

Scrum es el macro de trabajo para administrar proyectos de desarrollo de software, que se basa en la definición de roles, eventos, artefactos y reglas para compartir estos elementos.

Según Sabella (2016), la estructura está conformada de la siguiente manera:

1.2.1. Roles Principales

Product owner: se asegura de que scrum team trabaje desde una perspectiva de negocio comercial. Escriba y priorice las historias de usuarios que forman parte del product backlog.

Scrum master (o Facilitador): responsable de eliminar impedimentos hace que el equipo alcance la meta del sprint. Scrum master no es un líder ya que el equipo está organizado para seguir las reglas.

Scrum team: responsable de proporcionar un producto compuesto por 1-5 personas con las habilidades transversales necesarias para la función (análisis, diseño, desarrollo, prueba, documentación, etc.).

1.2.2. Documentos

Product backlog: es un documento de alto nivel utilizado en todo el proyecto, incluye todos los requisitos de un proyecto con descripciones generales de funciones deseables y prioritarias.

Sprint backlog: es un subconjunto de tareas que se desarrollarán durante el próximo sprint.

Burn down/up chart: es un gráfico que mide el número de requisitos de la cartera de pedidos, lo que le permite ver el progreso del proyecto.

1.2.3. Proceso

Historias de usuario: En este proceso, se crean historias de usuarios y los criterios asociados. Las historias de usuario generalmente están escritas por el propietario del producto y tienen la intención de proporcionar una presentación clara de los requisitos del cliente y su total comprensión por todas las partes interesadas.

Requerimientos de historia de usuario

- Son una manera abreviada de hacer requerimientos

- Una historia de usuario es una promesa a tener en una conversación
- El objetivo es reemplazar documentos por conversaciones
- Evolución gradual: qué, tamaño, prioridad, diseño, etc.

Creación de tareas: En este proceso, las historias de usuarios aprobadas por el usuario se dividen en actividades específicas y se combinan en listas de actividades. A menudo task planning meeting se desarrolla con este propósito.

Estimación de tareas: En este proceso el equipo scrum conjuntamente con las tareas de estimación, evalúa el esfuerzo requerido para completar cada actividad en la lista de actividades. El resultado de este proceso es una lista de actividades planificadas.

Creación sprint backlog Durante este proceso el equipo scrum tiene un sprint planning meeting donde se crea un sprint backlog que incluye todas las tareas que deben realizarse en el sprint.

1.3. Aplicación monolítica

En una aplicación monolítica toda la lógica se realiza en un único servidor de aplicaciones o en una sola aplicación que lo hace todo. Las aplicaciones monolíticas típicas se crean en grandes grupos múltiples, lo que requiere una cuidadosa coordinación de cada cambio. Las aplicaciones monolíticas también se tienen en cuenta cuando hay varios servicios API que proporcionan lógica empresarial, con una sola capa de presentación que es una aplicación web grande, es decir, la aplicación hace todo. En ambos casos, la arquitectura de microservicios puede proporcionar una alternativa.

En la Figura 7, se observa un caso de monolítico, el gran equipo de desarrolladores sube sus cambios a un único repositorio de código fuente.

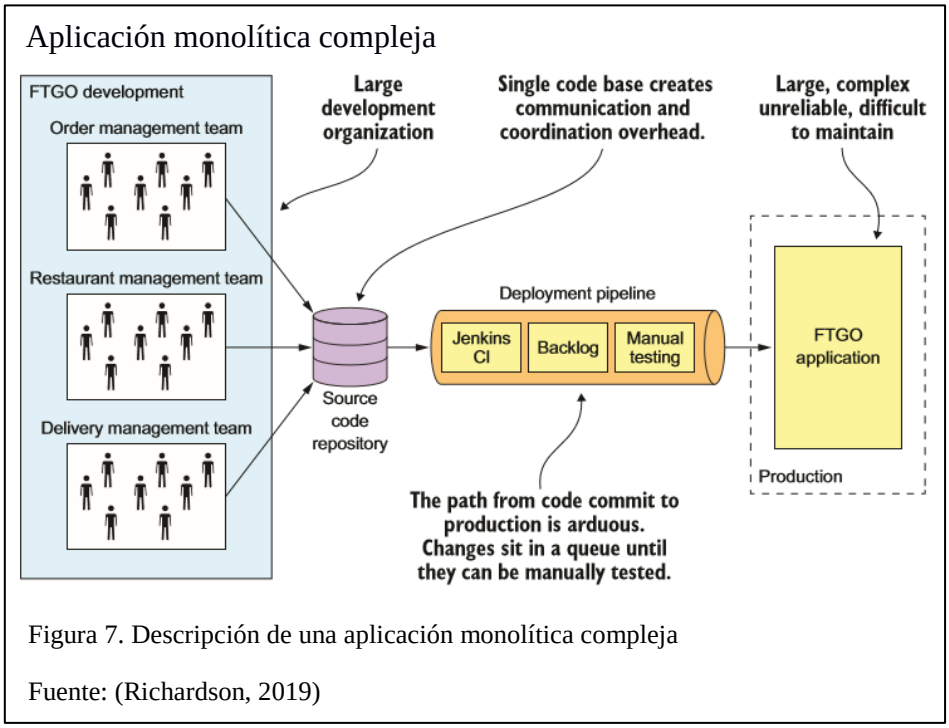

Figura 7. Descripción de una aplicación monolítica compleja

Fuente: (Richardson, 2019)

La imagen anterior muestra tres equipos independientes que trabajan en áreas superpuestas de una aplicación, lo que siempre hace difícil identificar quién está trabajando en la aplicación, lo que afecta la calidad del código y por lo tanto la calidad de las aplicaciones y accesibilidad. Al mismo tiempo, se hace más difícil para los equipos de desarrollo individuales realizar cambios sin afectar a otros equipos que tienen cambios inconsistentes

1.3.1. Arquitectura de software monolítica

Este servicio está destinado a ser parte de la funcionalidad de un sistema que busca optimizar para el máximo servicio al cliente, en el que cada servicio es diferente, ya que está optimizado independientemente de los clientes. En este sentido, una aplicación web monolítica representa una aplicación con una única base de código que proporciona una variedad de servicios, utilizando diferentes interfaces, como páginas HTML encerradas en un solo caché (memoria).

1.3.2. Inconvenientes en arquitecturas monolíticas

Para comprender mejor por qué la arquitectura monolítica no siempre es la mejor opción, se han resaltado las siguientes desventajas (Newman, 2015):

- Si el sistema requiere mantenimiento o funcionalidad y necesita ser extendido, el cambio afecta a todas las funciones que conducen al uso de recursos innecesarios.
- Si una aplicación contiene un repositorio de código que se integra con la misma base de datos, este será el único punto de falla para todo el sistema en caso de un error.
- La tecnología es frágil, el software es frágil, es difícil actualizarlo, por lo que es propenso a errores.
- Si necesita reutilizar código eficiente, puede haber restricciones con otros equipos de desarrollo para una buena comunicación.

1.4. Microservicios

Es una pequeña serie de aplicaciones de métodos de desarrollo de aplicaciones, cada una operando con sus propios procesos y mecanismos de comunicación simples, a menudo utilizando un recurso de interfaz de programación personalizada (API). El protocolo de transferencia de hipertexto (HTTP). Estos servicios se basan en capacidades comerciales y son una distribución totalmente automatizada, según (Posta, 2016).

El término microservicios no es relativamente nuevo, este estilo arquitectural fue dicho por Martin Fowler en un taller de arquitectos de software como una descripción del nuevo campo que los participantes estaban explorando. No existe una definición en concreto para microservicio, sin embargo, una aproximación que la realiza (Newman, 2015) lo define como: “Pequeños servicios autónomos que trabajan juntos”.

La arquitectura de dominio es importante: la modularización en microservicios para diferentes dominios es importante, ya que determina cómo se dividen los grupos de microservicios en un nodo de integración. Los problemas a este nivel también afectan a la organización. Solo una arquitectura de dominio sólida puede garantizar el desarrollo independiente de microservicios (Wolff, 2017).

En la Figura 8, se observa la integración de los microservicios con sus componentes, de acuerdo a las definiciones solicitadas por el equipo de trabajo.

Patrón básico de arquitectura de microservicios

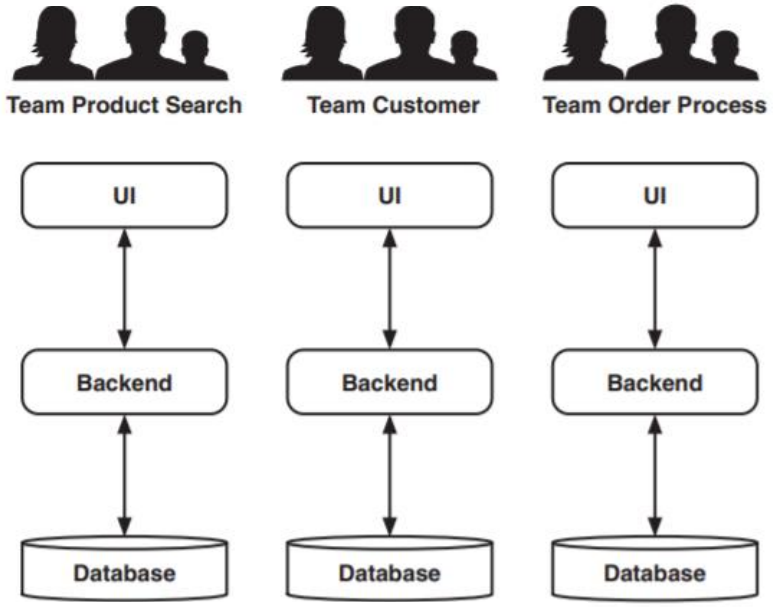

Figura 8. Descripción del patrón de microservicios

Fuente: *(Wolff, 2017)*

1.4.1. Beneficios

Un enfoque de microservicios provee de una serie de beneficios tales como:

Intensa modularización: La arquitectura de microservicio consiste en dividir una aplicación o sistema en unidades más pequeñas, la modulación facilita la automatización y proporciona medios precisos de abstracción.

Intercambiabilidad: Los microservicios se pueden reemplazar fácilmente por módulos de sistemas monolíticos. Simplemente reemplazar microservicios reduce el costo de malas decisiones, si el microservicio está construido con una tecnología o enfoque, se puede sobrescribir si es necesario.

Desarrollo sostenible: El desarrollo de software sostenible está vinculado a la modularidad intensiva y fácil sustitución o intercambio debido a que los microservicios no están conectados directamente a la tecnología, debido al desarrollo evolutivo, lo que permite el uso de nuevas tecnologías adaptadas a cada problema.

Entrega continua: una característica de la modularidad es que cada microservicio se puede distribuir de forma independiente a los ambientes productivos y no productivos.

Libre elección de tecnologías: a diferencia de una aplicación monolítica, todos deben estar de acuerdo con el microservicio, en un lenguaje de programación, infraestructura o biblioteca, incluso en una versión particular de estas plataformas, con la adición de servicios independientes que tienen la ventaja de una variedad de plataformas.

Time-to-market: los microservicios reducen el tiempo de comercialización, lo que le permite hacer cambios solo a aquellos que necesitan un cambio, y cada equipo responsable de desarrollar uno o más microservicios puede desarrollar e implementar características sin estar en desacuerdo con el tiempo restante. Equipo que permite la operación en paralelo y nuevas funciones lo antes posible.

1.4.2. Arquitectura de software basada en microservicios

(Newman, 2015), define la arquitectura de microservicios como "un nuevo estilo arquitectónico compatible entre sí pero con servicios separados". Este tipo de arquitectura intenta desmantelar el software complejo en aplicaciones independientes más pequeñas que las integran en protocolos ligeros que pueden modificarse sin afectar la partición general del sistema. Según (Torre, 2018), el punto importante es que son adecuados para todas las situaciones.

La Figura 9, se observa que un microservicio puede ser desarrollado en cualquier tecnología y su despliegue es independiente, cada microservicio es desplegado en un contenedor que se integra a su repositorio de información base de datos.

Figura 7. Ejemplo arquitectura orientada a microservicios

Fuente: (Torre, 2018)

1.4.3. SOA y Microservicios

Según (Wolff, 2017), en términos de microservicios, siempre hay un vínculo con una arquitectura puramente de servicio, pero los enfoques son diferentes, incluso si eso no significa que uno sea

mejor que el otro, todo dependerá del propósito de su uso: SOA (arquitectura orientada a servicios) se basa principalmente en la orquestación de sus servicios, mientras que la arquitectura basada en microservicios está orientada, por así decirlo, a la coreografía de los servicios. Sin embargo, los microservicios son el resultado de una mayor flexibilidad y escalado horizontal en términos de rendimiento general de la aplicación.

SOA se enfoca en organizaciones donde hay grupos de desarrollo del lado del servidor y grupos de desarrollo separados para interfaces de usuario, mientras que, en el enfoque de microservicios, el grupo debe hacer todo lo posible para simplificar la interacción y, por lo tanto, acelere la implementación de funciones. Sin embargo, desde un punto de vista conceptual, arquitectónico y organizativo, los dos enfoques tienen consecuencias diferentes. En la tabla se presentan algunas diferencias entre SOA y Microservicios.

Tabla *2.* Diferencias entre SOA y Microservicios

	SOA	Microservices
Alcance	Arquitectura para toda la empresa	Arquitectura para cada proyecto
Flexibilidad	Flexibibles en orquestación	Flexibilidad, despliegue rápido, desarrollo ágil e independiente
Organización	Los servicios son implementados por equipos del mismo proyecto	Los microservicios son implementados por diferentes equipos incluso de otro proyecto

Despliegue	Despliegue monolítico de varios servicios	Un microservicio puede implementarse individualmente
Interfaz de usuario	Pantalla única para todos los servicios.	El servicio contiene la interfaz de usuario por cada microservicio

Elaborado por: Diego Chicaiza

1.4.4. Ventajas de utilización de microservicios

Hay muchas opciones para elegir una arquitectura de microservicio, a continuación, se ofrece una breve descripción de los motivos principales:

Escalamiento: cada microservicio puede ampliarse independientemente de los otros servicios. Esto elimina la necesidad de actualizar todo el sistema cuando se utilizan muchas funciones, a menudo será un gran alivio.

Resiliencia: Los microservicios se pueden reemplazar fácilmente, por lo que la recuperación ante desastres es muy simple. El nuevo microservicio no debe usar ninguna parte de la base de código o la tecnología de microservicio anterior. Los microservicios más pequeños facilitan la actualización y, si las decisiones tecnológicas o de enfoque se limitan a microservicios, se pueden sobrescribir si es necesario.

Modularización: El microservicio es un poderoso concepto de modularidad. Siempre que el sistema se cree a partir de varios componentes de software, surgirán fácilmente dependencias innecesarias si alguien se refiere a una clase o función en la que no se puede utilizar, en poco tiempo, se crean muchas dependencias en las que el sistema ya no es compatible o desarrollado.

Velocidad: Los microservicios ofrecen un buen acceso al mercado. Si los grandes equipos del sistema responden a uno o más microservicios y las funciones solo requieren modificaciones a ese microservicio, cada equipo puede desarrollar y ejecutar a la producción sin coordinar el tiempo con otros grupos. Por lo tanto, muchos equipos pueden trabajar con muchas funciones en paralelo (Newman, 2015).

Flexibilidad: No existen restricciones sobre el uso de la tecnología en el desarrollo de microservicios, ciertas tecnologías se pueden utilizar para realizar ciertas funciones, por ejemplo, para una base de datos, Esto minimiza el riesgo potencial y permite que varios microservicios tomen decisiones tecnológicas independientes.

1.4.5. Contexto de utilización para microservicios

Hay tres formas principales de usar microservicios:

- Los microservicios son ideales para aplicaciones de alta demanda porque uno de sus objetivos principales es la alta disponibilidad. Para sistemas de baja transacción, la arquitectura monolítica se adapta fácilmente a las necesidades del cliente
- Con respecto a los microservicios, debe tenerse en cuenta que un monolito debe tener una estructura específica; La transición a un nuevo tipo de alcance no puede ser imposible sin puntos de partida específicos
- El microservicio es ideal para sistemas grandes, lo que significa que son más grandes que el tamaño de referencia porque tienen problemas especiales con la edición. En otras palabras, su exacerbación plantea nuevos problemas.

1.5. Comparativa de arquitecturas

Después de describir arquitecturas monolíticas y microservicios para concluir este capítulo, comenzamos comparándolos para obtener una visión objetiva de la naturaleza posible o imposible de la migración de la arquitectura. Finalmente, el objetivo de la evaluación de la arquitectura de software es determinar si cumple con los requisitos de calidad y si la arquitectura seleccionada es adecuada para el proyecto.

Hay muchas formas de evaluar la arquitectura del software, tales como: modelo, escenario, experiencia y matemática. Algunos de ellos son: métodos de análisis de arquitectura de software basados en SAAM; Método de comparación de la arquitectura del software SACAM.

El objetivo es alcanzar un enfoque comparativo entre las dos arquitecturas, continuaremos utilizando los aspectos importantes de SACAM (Método de Comparación de Arquitectura de Software), desarrollado por el Instituto de Desarrollo de Software de la Universidad Carnegie Mellon. Este enfoque fue diseñado para justificar el proceso de selección de una arquitectura al comparar la capacidad de dos o más candidatos con el sistema deseado.

Tabla 3. Arquitecturas Monolíticas vs. Microservicios

Categoría	Arquitectura Monolítica	Arquitectura de Microservicios
Código	Código único para toda la aplicación.	Fragmentos de código. Cada microservicio tiene su propio código.
Comprensibilidad	Confuso y difícil de mantener.	Mayor facilidad de interpretación, más fácil de mantener.

Despliegue	Implementaciones complejas con ventanas de mantenimiento por su complegidad.	Despliegue sencillo ya que cada microservicio se puede implementar de forma individual, con un tiempo de fuera de servicio mínimo.
Idioma	Totalmente desarrollado en un solo lenguaje de programación.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.
Escalamiento	Requiere escalar la aplicación completa, aunque los cuellos de botella estén localizados.	Le permite escalar microservicios que ocasionen cuellos de botella sin escalar la aplicación completa.

Fuente: (Torres, 2003)

1.6. Contenerización de aplicaciones

Un contenedor únicamente tiene lo necesario para ejecutar la aplicación, mientras una máquina virtual tiene todo el sistema operativo ocupa muchos recursos memoria procesador es más lento al arrancar, no se puede tener muchas maquinas arrancadas, mientras que contenedores podemos tener cientos siempre y cuando dispongamos suficiente hardware.

En la Figura 10, se visualiza la comparativa entre una máquina virtual y un contenedor, la diferencia es que una máquina virtual necesita un sistema operativo para arrancar sus procesos haciéndola más lenta, los contenedores no necesitan un sistema operativo para ejecutar sus procesos haciéndolos más livianos y rápidos.

Utilización de recursos de máquinas virtuales y contenedores

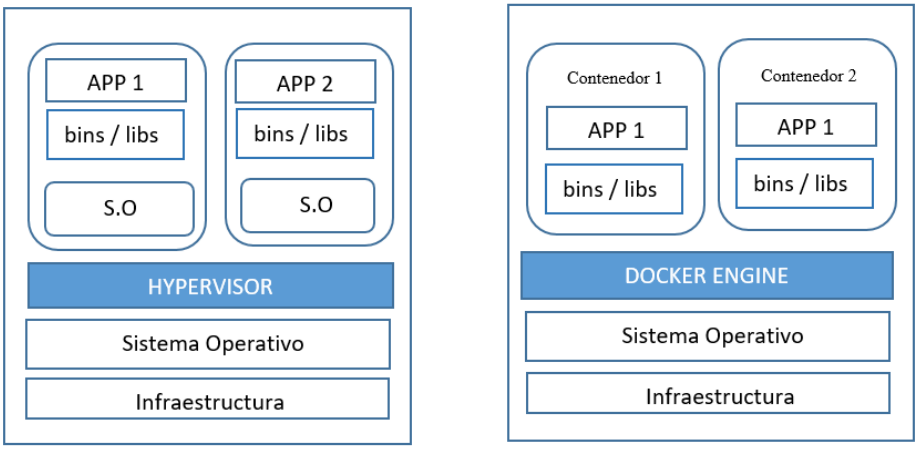

Figura 80. Comparación de utilización de recursos de máquinas virtuales y contenedores

Elaborado por: Diego Chicaiza

La palabra “contenedor” se define como "un objeto de almacenamiento / protección o transferencia de algo, la idea de los contenedores de software es similar. Estas son imágenes independientes e inmutables, la mayoría de las cuales solo están disponibles a través de la API.

Los contenedores no tienen núcleos del sistema operativo, lo que los hace más rápidos y flexibles que las máquinas virtuales. El contenedor puede manejar cada aplicación en múltiples procesos de kernel con múltiples conjuntos aislados, lo que permite que el aislamiento mantenga todo el estado del contenedor y luego se reinicie en un estado completamente transparente en un sistema host particular.

1.7. Despliegue continuo

Según (Grubor, 2017), el despliegue continuo es el proceso de integrar una aplicación en la producción al momento del registro, empaque, prueba e inspección. Las organizaciones suelen utilizar herramientas de integración continua como Jenkins, Hudson, Bamboo y otras para automatizar la implementación de aplicaciones. Estas herramientas pueden proporcionar una distribución segura de la aplicación (pruebas de producción y proceso de validación) con

condiciones de prueba apropiadas, lo que permite una validación correcta mediante procesos de prueba automatizados. Actualmente, estas herramientas facilitan enormemente la automatización de la implementación y distribución de los componentes involucrados en este proceso.

1.8. Monitoreo

El seguimiento de los microservicios se vuelve más difícil a medida que aumenta su número se debe utilizar mecanismos de notificación para determinar el funcionamiento de cualquier parte de la plataforma, según (Posta, 2016), el monitoreo es un aspecto importante en una arquitectura de microservicios, que ayuda a supervisar y permite visualizar los tiempos de espera de cada uno de ellos protegiéndonos contra la latencia inesperada.

Debido a los procesos de ejecución para implementar microservicios, como la integración continua, el monitoreo y el registro también son necesarios, por lo tanto, es necesario saber dónde almacenar estas métricas. Para resolver este problema, se utilizan bases de datos no relacionales como Graphite, Prometheus, Riac y elastic search con herramientas para visualizar estos volúmenes de datos en forma de gráficos, como kibana.

Según (Hunter, 2017), el registro del log es una parte importante para mantener su microservicio operativo. Los registros contienen información valiosa de la trazabilidad de los componentes invocados, pueden ayudar a armar los saltos realizados de las invocaciones internas realizadas a componentes involucrados en el proceso de ejecución del microservicio, en el mundo de microservicios altamente transaccionales y redundantes, necesita agregar todos esos registros a un repositorio común.

1.8.1. Kibana

Según (Gupta, 2015) kibana es una herramienta que forma parte de la suite ELK, que consiste en elasticsearch, logstash y kibana. Está construido y desarrollado por Elastic. Kibana es una plataforma de visualización, permite ver en tiempo el real el comportamiento de los microservicios según la configuración realizada, se basa en elasticsearch y aprovecha sus funcionalidades.

En la Figura 11, podemos visualizar la relación de los componentes ELK, logstash se utiliza para insertar datos directamente en elasticsearch, estos datos no se limitan a los de registro, sino que pueden incluir cualquier tipo de datos, elasticsearch almacena datos que vienen como entrada de logstash y kibana usa los datos almacenados en elasticsearch para mostrar en sus tableros la data monitoreada.

Componentes ELK

Figura 91. Descripción de Componentes ELK

Fuente: *(Gupta, 2015)*

1.8.2. Elasticsearch

Es un motor de analítica y análisis distribuido se basa en bases de datos no relacionales en su diseño tiene colecciones y documentos que hacen referencia a las tablas y registros en una base relacional, su estructura de datos es json.

1.8.3. Logstash

Logstash es una herramienta que pertenece a la suite ELK se usa para la administración de logs. Con el uso de esta herramienta se puede recolectar, analizar y guardar los logs para futuras búsquedas de acuerdo a lo solicitado por el negocio.

1.9. Herramientas de Diseño y Desarrollo

Para el diseño e implementación del tema planteado se utiliza una mezcla de productos open source y herramientas académicas que nos bridan las grandes empresas de tecnología.

1.9.1. Smartdraw

Smartdraw, es un software que te permite crear gráficos de alta calidad profesional, como: mapas mentales, organigramas, diagramas de proyectos y otros elementos visuales de negocios (Lowe, 2009). Esta herramienta tiene dos versiones, una edición en línea y una edición descargable, para el diseño se utiliza la versión en línea.

1.9.2. IBM Integration

Este IDE de desarrollo permite trabajar con todos los nodos de integración utiliza todas las características habilitadas, no existe un límite para el número de recursos que puede crear y mantener. IBM Integration para desarrolladores (Developer Edition) está disponible en los sistemas operativos Linux x86-64 y Windows x86-64, se proporciona únicamente para fines de desarrollo, prueba y evaluación (IBM, 2019).

1.9.3. JMeter

JMeter es una herramienta para realizar pruebas de rendimiento. Es un proyecto de código abierto, se puede ejecutar en "modo distribuido" en la nube, generando la carga de miles de usuarios. Su licencia es de Apache, que no tiene ninguna restricción de uso, JMeter tiene un formato estándar para escribir la prueba de rendimiento. La mayoría de las herramientas comerciales son compatibles (Sai Matam, 2017).

CAPÍTULO 2

2. ANÁLISIS Y DISEÑO

2.1. Análisis del problema

A pesar de los esfuerzos de las instituciones financieras privadas y públicas en Ecuador, las tasas bancarias son actualmente bajas. En los últimos 10 años, el nivel de actividad bancaria ha aumentado ligeramente, pero teniendo en cuenta el índice del 25% del banco en 2005, en 2010, solo aumentó al 37%. Estos porcentajes son globales, en el sentido de que incluyen el índice de bancos públicos, bancos privados, cooperativas, sociedades y compañías financieras (ASOBANCA, Directorio de la Asociación de Bancos, 2016).

El acceso a los niveles de América Latina no es muy impresionante, por ejemplo, el acceso a los servicios bancarios para Ecuador es del 35%, que es similar al nivel promedio entre países de la región, que es 36%. Chile es el primero con 60%, Panamá con 46% y Brasil con 43% (ASOBANCA, La importancia de la profundización financiera y bancarización en el ecuador, 2010).

Según los datos actuales proporcionados por Pedro Solines, director de bancos y compañías de seguros, en 2012 el índice en Ecuador era del 50,12% (según un estudio de la Asociación de Bancos Privados de Ecuador) y en 2013, se ubicó alrededor del 52%. Estas son cifras que se mejoran aún más por los diferentes planes para permitir que los ciudadanos se conecten al sistema bancario.

En la Figura 12, se observa el crecimiento del sector bancario en nuestro país, cada año existe un aumento significativo de clientes.

Índice de Bancarización

Figura 102. Índice de Bancarización en el Ecuador 2012

Fuente: ABPE

La clave es la capacidad de los ciudadanos de ganar confianza al demostrar alternativas cruciales a la banca en línea, garantizar que tengan acceso a dinero seguro y que la plataforma este siempre disponible.

En los últimos años, gracias al rápido desarrollo de las tecnologías de la información y la comunicación, Internet se ha establecido como un canal para proporcionar servicios financieros. En Ecuador, muchas organizaciones bancarias han introducido portales transaccionales en internet, siendo un canal adicional para dar facilidades al usuario, según los estudios antes descritos el cliente cada vez más utiliza este medio para realizar operaciones financieras como se puede ver en la Figura 12. Los sistemas ya no soportan el crecimiento transaccional generando intermitencias hasta fuera de servicio de sus portales transaccionales.

2.2. Análisis de especificaciones técnicas funcionales

En una arquitectura de microservicios uno de los principios es la agilidad y la independencia, es por tal motivo debemos elegir una metodología adecuada, para este trabajo de titulación emplearemos el marco de trabajo Scrum.

2.2.1. Análisis de especificación de requisitos

En base a las reuniones periódicas que se mantiene con el cliente, para el caso tutor de tesis se crear las siguientes historias de usuario.

Creación de historias de usuario

Para realizar la implementación del prototipo planteado se procede a crear historias de usuario que permitirá saber claramente lo que el cliente necesita (entidades financieras), este rol lo desempeñara el tutor de tesis, en las reuniones periódicas con el cliente se obtiene claramente los requisitos.

Historia de usuario HU 001 diseño de arquitectura

De acuerdo a las reuniones realizadas con el product owner, es necesario crear una arquitectura basada en los principios de la teoría de microservicios.

Criterios de aceptación

- Tener un diagrama de alto nivel donde pueda explicar el funcionamiento a nivel general del prototipo planteado.
- Tener un diagrama de bajo nivel donde explique el funcionamiento a más detalle de los componentes que serán creados y su relación de la arquitectura.

- Todo componte a ser desarrollado debe estar relacionado con un dominio de negocio por tal motivo se debe plantear un del diseño de los dominios de negocio.
- Todos los componentes transversales como librerías, mecanismo de log deben tener su diseño arquitectónico diagrama bajo nivel.
- Generar diagrama de despliegue esto tendrá relación con la teórica de microservicios es decir que independizara los componentes a ser desplegados.

Historia de usuario HU002 datos demográficos

De acuerdo a las reuniones realizadas con el product owner, se debe crear un microservicio para extraer datos demográficos del cliente y proporcionar esta información a las aplicaciones que la necesitan.

Criterios de aceptación

- El microservicio debe guardar su registro en el log (registros que dejan los microservicios en el repositorio) de operación para poder validar su trazabilidad en el caso de requerirlo.
- El microservicio debe responder en el tiempo definido tiempo de respuesta menor al segundo.
- El microservicio debe tener una estructura bien definida header (cabecera del microservicio) estructura fija y body (cuerpo del microservicio) estructura variable de acuerdo con los parámetros definidos.
- Pruebas unitarias de ejecución del microservicio.

Historia de usuario HU003 datos personales

De acuerdo a las reuniones realizadas con el product owner, es necesario crear un microservicios para extraer información de los datos personales del cliente.

Criterios de aceptación

- El microservicio debe guardar su registro en el log de operación para poder validar su trazabilidad en el caso de requerirlo.
- El microservicio debe responder en el tiempo definido tiempo de respuesta menor al segundo.
- El microservicio debe tener una estructura bien definida header (cabecera del microservicio) estructura fija y body (cuerpo del microservicio) estructura variable de acuerdo a los parámetros definidos.
- Pruebas unitarias de ejecución del microservicio.

Historia de usuario HU004 productos

De acuerdo a las reuniones realizadas con el product owner, se debe crear un microservicio que permita extraer información de los productos que posee un cliente.

Criterios de aceptación

- El microservicio debe guardar su registro en el log de operación para poder validar su trazabilidad en el caso de requerirlo.
- El microservicio debe responder en el tiempo definido tiempo de respuesta menor al segundo.

- El microservicio debe tener una estructura bien definida header (cabecera del microservicio) estructura fija y body (cuerpo del microservicio) estructura variable de acuerdo a los parámetros definidos.
- Pruebas unitarias de ejecución del microservicio.

Historia de usuario HU005 escenarios

De acuerdo con las reuniones realizadas con el product owner, se debe realizar pruebas de estrés en el prototipo desarrollado, los microservicios deben soportar la carga transaccional establecida en el plan de pruebas.

Criterios de aceptación.

- Escenario de pruebas a ser ejecutado
- Resultados de las pruebas, demostrar como soporta el prototipo desarrollado en base a la arquitectura planteada.
- Ejecución de un microservicio al momento que se realiza las pruebas de estrés para demostrar el tiempo de respuesta que no degrade, adjuntar evidencias.
- Las invocaciones realizadas deben ser registradas en el log de operación.

2.3. Escenarios

Entre el desarrollo de la arquitectura final y la identificación de la arquitectura base existe un alto grado de incertidumbre y número de cambios a considerar, por lo tanto, se requiere técnicas que permitan diferenciar entre las aspiraciones de los interesados y lo que realmente pueda desarrollarse, una de estas técnicas son los escenarios de negocio.

El escenario de negocio utilizaremos para identificar los requerimientos de negocio más relevantes y para que los interesados establezcan un consenso de estos, los escenarios de negocio definen:

- Procesos de negocio y conjunto de aplicaciones
- El ambiente tecnológico y de negocio
- El componente informático y humano
- El resultado deseado de una adecuada ejecución del escenario

Los escenarios de negocio no prevén lo que pasará, sino que describen los cambios posibles a futuro son un instrumento para la planeación estratégica y para llevarlos a cabo se debe hacer un profundo análisis de variables cuantitativas y cualitativas según (TOGAF, 2014) con esto se valida el escenario del negocio con la realidad propuesta.

Un escenario de negocio es una descripción detallada de un problema el cual se realizan términos de negocio y de arquitectura, esto permite identificar requerimientos particulares en relación con su contexto, también nos ayuda a tener claro la solución que se plantea (TOGAF, 2014).

Otro beneficio de un escenario de negocio es alinear las tecnologías de la información con el negocio esto significa seleccionar y diseñar recursos de tecnologías de información propio de un negocio en particular para conocer la oportunidades y soluciones que estos generan.

Para establecer un buen escenario de negocio se debe tener las siguientes características (TOGAF, 2014):

Específico: donde se define que se debe hacer en el negocio.

Medible: proporciona las métricas para el éxito del negocio.

Factible: valida la definición del problema y plantea la solución.

Realista: muestra que el problema pueda ser resuelto teniendo en cuenta las limitaciones de los recursos.

2.3.1. Construcción escenario

Para la construcción del escenario de pruebas se toma en cuenta el siguiente esquema.

El problema se refiere a la identificación y documentación del problema del escenario planteado, va relacionado con el ambiente que es la suma de todos los factores internos y externos que afectan al negocio, se debe identificar y conocer cuando ocurren.

Actores informáticos se debe identificar todos estos actores informáticos que hacen parte del escenario.

Los sistemas informáticos tradicionales (sistemas de agencia), han tenido éxito en soportar las cargas transaccionales, pero esto no es suficiente cada día la atención al cliente se va deteriorando, ocasionando grandes filas de espera, por tal motivo es necesario que las entidades bancarias cuenten con aplicaciones basadas en la web.

Estas aplicaciones tienen que estar diseñadas para soportar un gran número de operaciones concurrentes este factor está empujando el rendimiento de la aplicación por encima de sus límites, lo que amenaza la disponibilidad del sistema web.

La banca web es uno de los canales digitales de mayor demanda en la actualidad, por este canal se realizan transacciones financieras, las gerencias apalancan este medio para atraer más clientes, dándoles las facilidades para que puedan realizar sus consultas de saldos, movimientos, pagos de servicios básicos, transferencia de dinero, entre otras opciones.

Por tal motivo es obligatorio garantizar que el sistema informático esté disponible 24/7, 24 horas, 7 días de la semana. Para garantizar la disponibilidad del sistema debemos realizar un escenario de pruebas de acuerdo a la transaccionabilidad, aumentando a este valor un porcentaje de un 25% a la carga real esperada, otro factor importante que debemos tomar en cuenta al realizar el escenario de pruebas es el tiempo en el cual el sistema soportara una cantidad de usuarios, es recomendable proyectar cuantos usuarios tendremos a dos años y realizar el escenario tomando en cuenta estas variables.

2.3.2. Pruebas de carga escenario normal.

La prueba de carga consiste en someter al microservicio a una carga normal tomando en cuenta los parámetros iniciales propuestos.

Alcance

Se aplicará la prueba de carga a los siguientes microservicios.

Nombrado de microservicios de acuerdo a la definición, en la sección. Diseño nombrado de microservicios, Figura 20.

MSCClientesConsultaDemograficos0001

MSCClientesConsultaBasicos0002

MSCProductosConsulta0001

Objetivos

Validar que los tiempos cumplan con lo establecido tiempo de respuesta.

Confirmar el porcentaje de efectividad de los microservicios.

Tabla 4. Pasos escenario normal

Entrada	Actividad	Validación
Input (datos de negocio) de los microservicios a ser ejecutados con su respectiva URL, y puerto	Aplicar la carga definida en los criterios de rendimiento, configurar los escenarios para tener la carga efectiva (TPS), de acuerdo con lo definido	Estabilidad y tiempos de respuesta óptimos

Elaborado por: Diego Chicaiza

2.3.3. Prueba de estrés

La prueba de estrés consiste en subir gradualmente la carga partiendo desde la carga normal definida, hasta que los microservicios dejen de responder de forma óptima evidenciando de esta manera hasta cuánto se puede soportar.

Alcance

Se aplicará la prueba de carga a los siguientes microservicios.

Nombrado de microservicios de acuerdo a la definición, en la sección. Diseño nombrado de microservicios, Figura 20.

MSCClientesConsultaDemograficos0001

MSCClientesConsultaBasicos0002

MSCProductosConsulta0001

Objetivos

Determinar la carga efectiva tps y una carga referencial de los microservicios pueden soportar el llamado de peticiones desde las aplicaciones.

Tabla 5. Pasos prueba de estrés

Entrada	Actividad	Validación
Input (datos de negocio) de los microservicios a ser ejecutados con su respectiva URL, y puerto	Aplicar la carga referencial e ir subiendo la carga hasta poder saber hasta cuándo puede soportar un microservicio	Estabilidad y tiempos de respuesta óptimos

Elaborado por: Diego Chicaiza

2.4. Diseño

2.4.1. Diseño de arquitectura

En la arquitectura de microservicios, los servicios son pequeños, independientes y están conectados de manera flexible. Cada servicio es un código base independiente, que puede ser administrado por un pequeño equipo de desarrolladores.

Diseño diagrama conceptual alto nivel: este diagrama presenta el diseño de la arquitectura de la solución, este insumo ayuda a las gerencias de las entidades financieras tener una visión de los componentes y el rol que cumplirán en el diseño. En la Figura 13, se puede observar la relación que tienen los componentes de la arquitectura, desde la aplicación hasta los repositorios de información (base de datos) nuestro análisis y diseño está enfocado en los microservicios.

Diagrama conceptual alto nivel

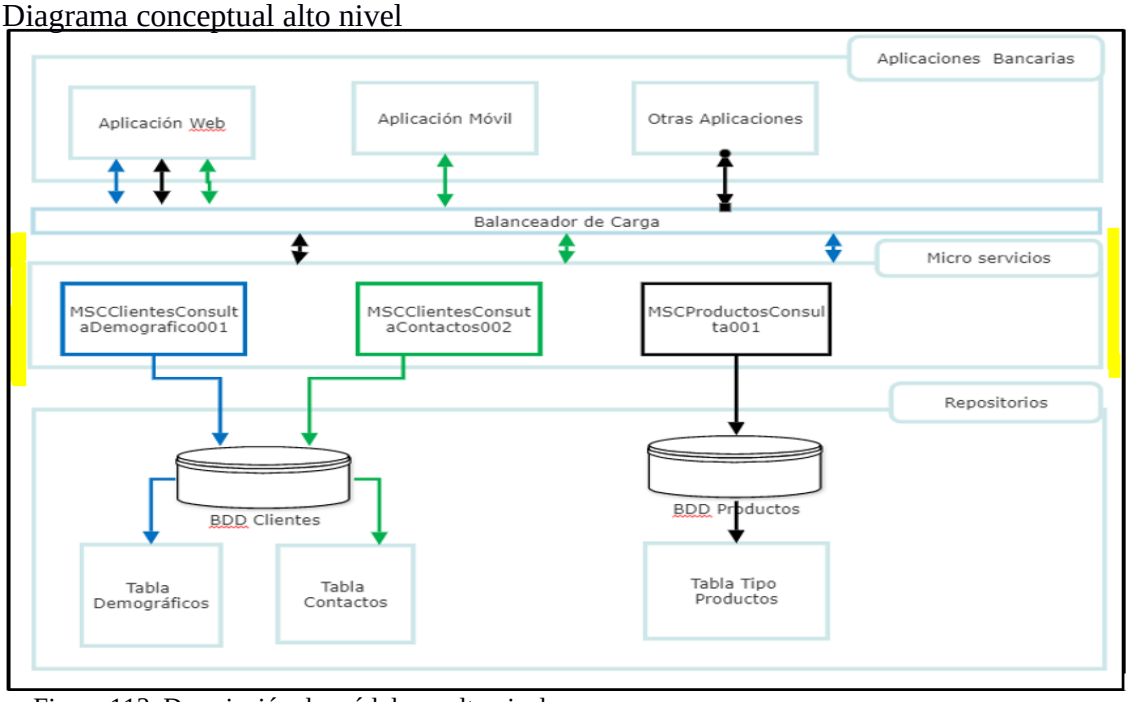

Figura 113. Descripción de módulos a alto nivel

Elaborado por: Diego Chicaiza

De acuerdo al alcance definido para el proyecto técnico, el gráfico resalta los componentes de nuestro estudio.

Descripción de componentes

- **Aplicaciones bancarias:** en el gráfico representa todas las aplicaciones que invocaran los microservicios esperando una respuesta de acuerdo a la petición realizada.
- **Balanceador de carga:** componente de red que permite distribuir las peticiones que provienen de la aplicación hacia los nodos donde se alojan los microservicios.
- **Microservicios:** en el gráfico se puede visualizar los microservicios que se crearan para la demostración del prototipo siguiendo los principios de la teoría de microservicios
- **Repositorios:** los repositorios hacen referencia a todos los backend, como base de datos son aquellos donde se aloja información del cliente de acuerdo a las necesidades del negocio.

Diseño de diagrama conceptual bajo nivel: Después de analizar y evaluar las mejores prácticas para crear microservicios, se produce este resultado, que desempeñará un papel clave en el diseño. Este será un prerrequisito para empezar con la construcción del escenario planteado aquí demostraremos como se debe construir la arquitectura de cada componente, marco de trabajo de microservicios, mecanismo de log, despliegue.

En la Figura 14, se puede observar cómo se integrarán los microservicios con los componentes reutilizables (log de aplicación) y sus repositorios de información.

Diagrama conceptual bajo nivel

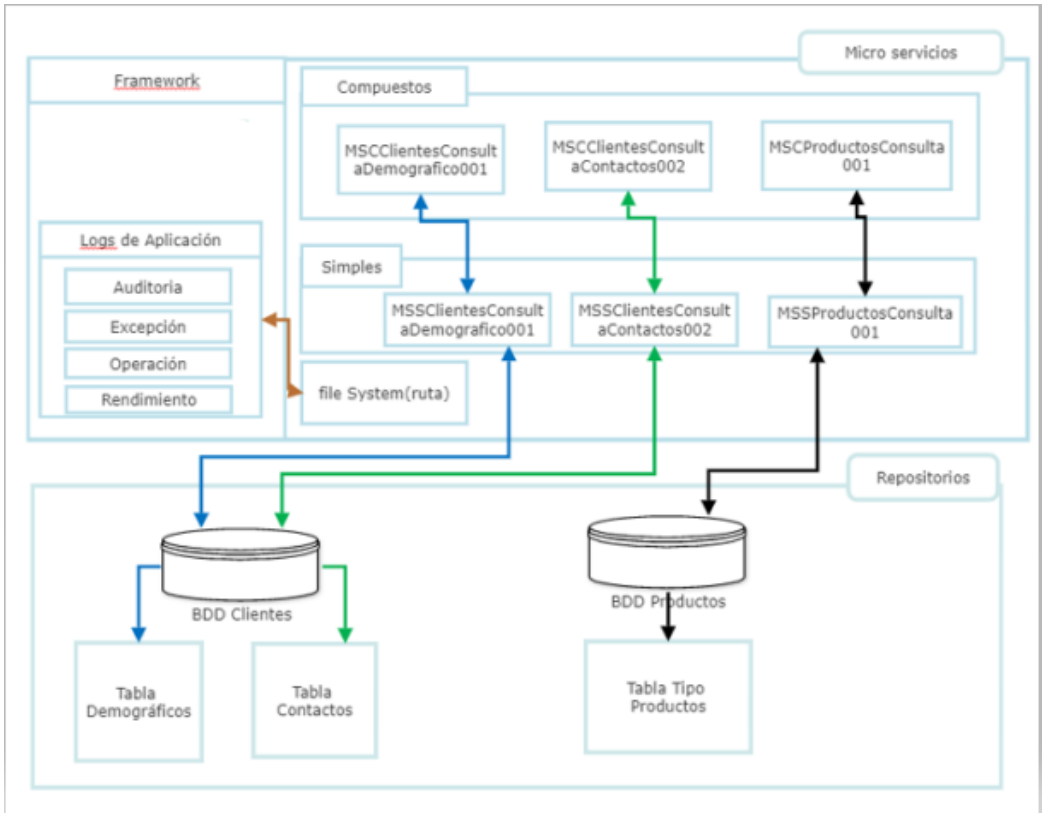

Figura 124. Descripción de módulos a bajo nivel

Elaborado por: Diego Chicaiza

Descripción de Componentes

Marco de trabajo

Basado en el desarrollo de software el marco de trabajo es una estructura de soporte conceptual y tecnológica específica, generalmente con artefactos específicos o módulos de software que pueden servir como base para organizar, desarrollar software y pueden incluir soporte para programas de biblioteca que contribuyen a ello, así como diseñar e integrar diferentes componentes del proyecto de desarrollo.

Según el concepto presentado, el marco de trabajo creado para implementar este prototipo permitirá que los microservicios se desarrollen más rápido y de manera más eficiente, de acuerdo con los conceptos de la teoría de microservicios, permitirá la reutilización de mecanismos transversales como los registros de log de aplicación, la creación de microservicios basados en el header (cabecera del microservicio), body (cuerpo del microservicio) y tag error (conjunto de campos fijos) definido, los mensajes que ingresaran a través de los microservicios viajaran por este marco de trabajo hasta obtener los resultados esperados.

Microservicios

- **Microservicios compuestos:** en el diagrama podemos interpretar que son aquellos que manejan la composición de microservicios simples o en ciertos casos invocación a microservicios compuestos son los encargados de agregar funcionalidad técnica como cache, encriptación u otra clase de eventos o simplemente pasando la información recuperada de la invocación realizada, estos microservicios poseen la interfaz mediante la cual consumirán sus datos de acuerdo a la lógica de negocio.
- **Microservicios simples:** en nuestro diagrama podemos interpretar que son aquellos que maneja la recuperación de datos se integran con los componentes del backend, cumplen una única función. Cada uno debe ser independiente y autónomo.
- **Log de aplicación:** uno de los componentes importantes en el diseño de una arquitectura basada en microservicios es el mecanismo de log estos componentes tendrán el registro de todas las transacciones.
- **File system:** son las rutas en donde se alojarán los archivos del registro del log de la aplicación.

- **Repositorios:** son componentes backend, como base de datos para nuestro caso se tiene dos bases de datos que contendrán un esquema y dos tablas las cuales interactúan con los microservicios simples, estas bases están diseñadas de acuerdo al dominio de negocio, clientes y productos.

2.4.2. Diseño Dominios de Negocio

Durante la fase de diseño, es importante determinar el dominio al que pertenecerá el microservicio es necesario realizar un análisis técnico funcional para determinar adecuadamente la relación entre el microservicio y el dominio. Es importante entender el requerimiento funcional que el cliente ha solicitado para la creación de microservicios de esta manera se podrá relacionar al dominio de negocio.

En el prototipo, se crearán los siguientes microservicios de acuerdo con el historial de usuarios definido por el cliente.

- HU002: se requiere que la aplicación presente datos demográficos de un cliente.
- HU003: se requiere que la aplicación presente datos personales de un cliente.
- HU004: se requiere que la aplicación presente los productos del cliente.

De acuerdo a los requerimientos expuestos y tomando como referencia BIAN (red de arquitectura para la industria bancaria), se realiza el análisis para determinar con que dominios serán relacionados los microservicios a implementar.

BIAN, para definir un dominio propone diferentes dominios funcionales que abarcan dominios de negocio que están relacionados con la lógica bancaria.

En la Figura 15, se puede observar los dominios de negocios propuestos por la red de arquitectura para la industria bancaria, respecto a clientes y productos.

Figura 135. Descripción de requerimientos BIAN

Fuente: *(Wire., 2011)*

En una arquitectura de microservicios orientados a la banca se debe considerar lo expuesto por lo tanto se define que todo diseño debe tener un dominio funcional que va relacionado con un dominio de negocio, para el caso se define lo siguiente:

Tabla 6. Definición dominio de negocio

Cantidad	Dominio Funcional	Dominio de Negocio
1	Datos de Referencia	Productos
2	Ventas y Servicios	Clientes

Elaborado por: Diego Chicaiza

- Esta definición se basa en el análisis antes expuesto los microservicios deben estar bajo los dominios definidos.
- Microservicio que realiza consultas de datos demográficos y datos básicos del cliente deben estar en el dominio de clientes.
- Microservicio que realiza consulta de productos como cuentas, préstamos, inversiones bancarias debe estar en el dominio de productos.

Diagrama conceptual dominios

En la Figura 16, se puede visualizar la definición de los dominios de acuerdo a la funcionalidad solicitada historias de usuario, se ata al esquema correspondiente de acuerdo al dominio.

Diagrama conceptual de dominios

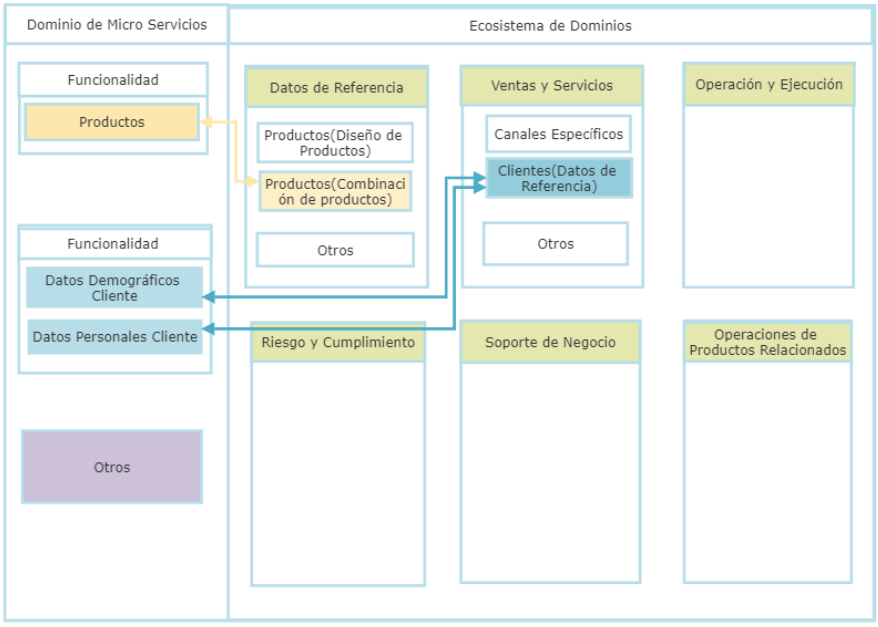

Figura 146. Diagrama de dominios participantes

Elaborado por: Diego Chicaiza

De acuerdo con la funcionalidad se debe analizar cada uno de los dominios definidos por BIAN para relacionar un microservicio a un dominio de negocio.

2.4.3. Diseño marco de trabajo

Para reducir el tiempo de desarrollo, se crea un marco de trabajo que permita estandarizar los desarrollos, con estructuras de interfaces definidas, y mecanismos transversales como log implementados, esto permite una implementación más rápida de microservicios de acuerdo con los lineamientos establecidos en el marco teórico.

- **Componente procesar aplicación:** debe ser llamado cuando se construya la interfaz del microservicio contiene la estructura definida del header y el tag error.
- **Componente registrar Log:** permitirá guardar los registros de una transacción en el repositorio file system definido.

En la Figura 17, se puede observar como el marco de trabajo ofrece funcionalidad transversal (log de aplicación, estructura genérica de cabecera, error de la interfaz) a los microservicios a través de sus librerías.

Estructura del marco de trabajo

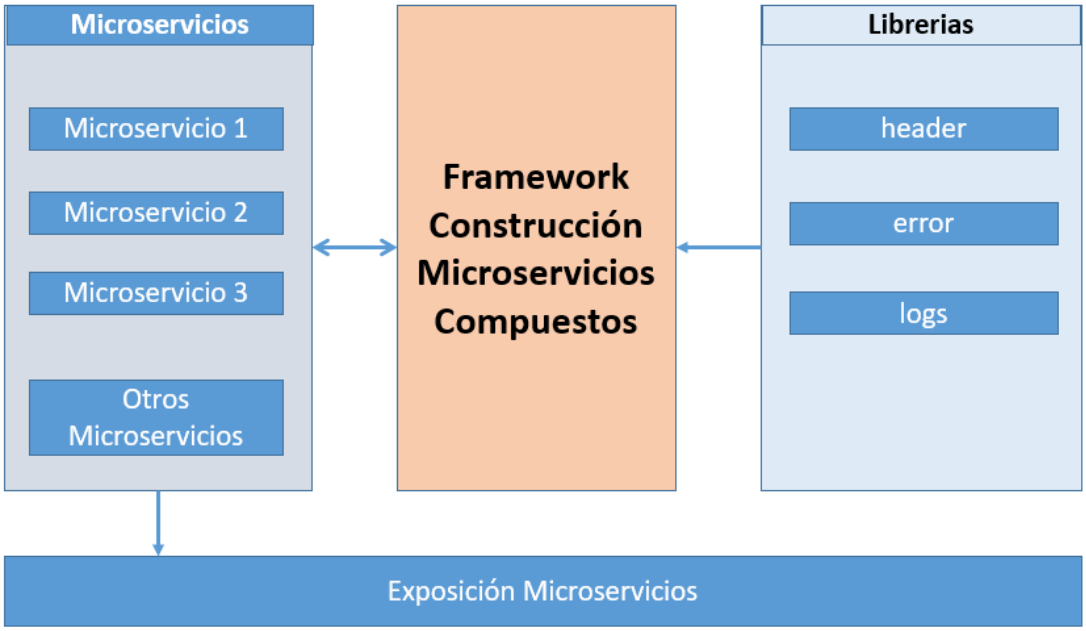

Figura 157. Diseño de la estructura del marco de trabajo

Elaborado por: Diego Chicaiza

2.4.4. Diseño log de aplicación

Uno de los componentes importantes en el diseño de una arquitectura basada en microservicios es el mecanismo de log, se tendrá los siguientes:

- **Log de Operación:** permite tener métricas relacionadas con el valor de las transacciones que pueden ser útiles para un análisis de negocio.
- **Log de Rendimiento**: estos registros se utilizan para el análisis de rendimiento avanzado de las aplicaciones.

- **Log de Auditoría:** Permite la rastreabilidad de las transacciones e identificar en cualquier momento quien realizo cada operación, cuando fue realizada y desde donde se realizó, esto es muy importante para las personas de auditoria.
- **Log de Excepción**: Permite identificar excepciones producidos por el aplicativo o componentes intermedios de interactúen con nuestro microservicio.

En la Figura 18, podemos visualizar la arquitectura planteada del mecanismo de log, los microservicios simples y compuestos utilizara el módulo procesar logs para dejar el registro en los repositorios (archivos).

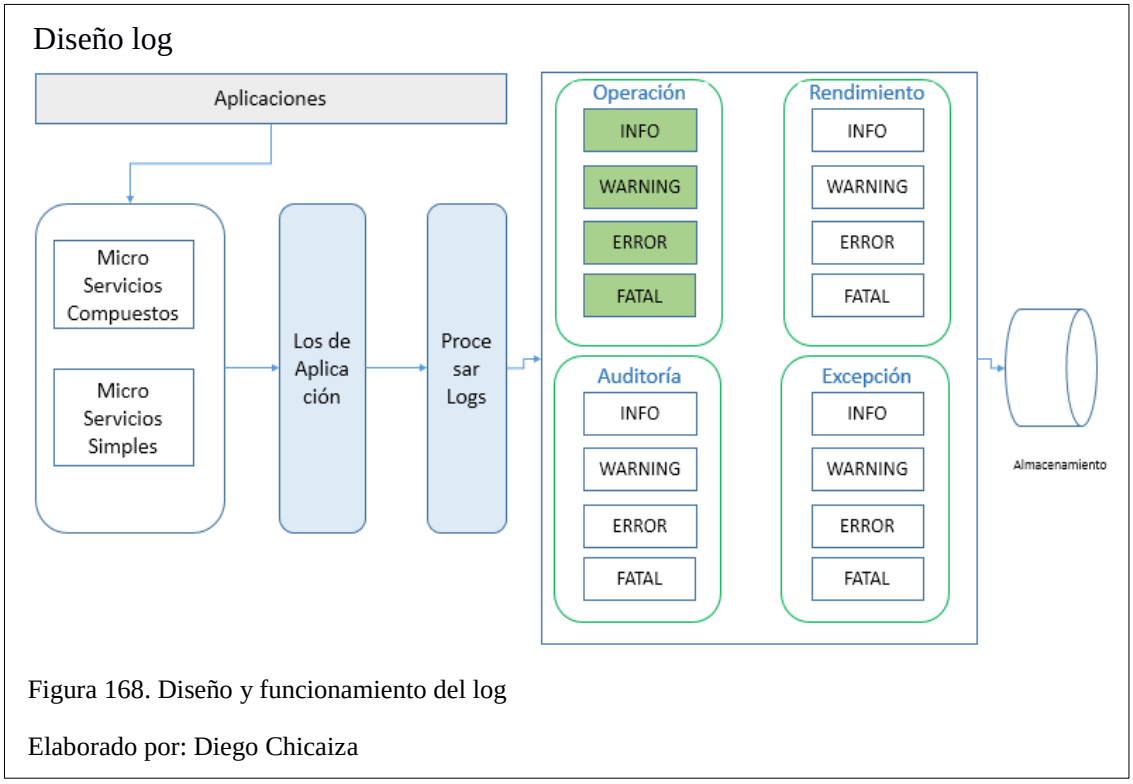

Figura 168. Diseño y funcionamiento del log

Elaborado por: Diego Chicaiza

Cada log tendrá una estructura bien definida, esto es muy importante para la lectura e interpretación de su información. En la arquitectura planteada es necesario ofrecer a las entidades financieras tableros de información del estado de las transacciones es decir cuántas transacciones están ingresando, cuantas exitosas o fallidas se ejecutaron en el sistema, al tener un log entendible y fácil de interpretar por las herramientas existentes ejemplo Kibana, es fácil

ofrecer al negocio monitoreo en tiempo real, las entidades financieras siempre buscan mejorar la atención al cliente es por eso que es necesario tener una visión clara del comportamiento de sus sistema para buscar mejoras.

La arquitectura de log basada en la teoría de microservicios ayuda a detectar problemas de manera más rápida, al tener muchos microservicios simples que cumplen una acción el registro del log es más sencillo y entendible.

2.4.5. Diseño nombrado de microservicios

En la construcción de microservicios se debe definir en nombre que adoptara cada componente de acuerdo la arquitectura planteada se define lo siguiente:

Diseño Lógico + Dominio de negocio + Método descriptivo + Secuencial

Diseño lógico, son las divisiones de componentes de la sección microservicios planteados en el diagrama de arquitectura de bajo nivel, de acuerdo con lo expuesto se tiene:

MSC: Microservicio Compuesto

MSS: Microservicio Simple

Dominio de negocio, está definido de acuerdo con los dominios de negocio BIAN, de acuerdo con el análisis realizado se tiene:

Clientes

Productos

Método descriptivo, hace relación a la funcionalidad de negocio para la que va a ser creado el microservicio.

Secuencial, número generado para la secuencia de la creación del microservicio.

De acuerdo con lo expuesto los nombres son los siguientes:

MSCClientesConsultaDemograficos0001

MSCClientesConsultaContactos0002

MSCProductosConsulta0001

Como se puede ver en la Figura 19, para el nombrado de microservicios es mandatorio la definición de los componentes expuestos en el diseño todos ellos tienen relación de acuerdo a su lógica técnica funcional.

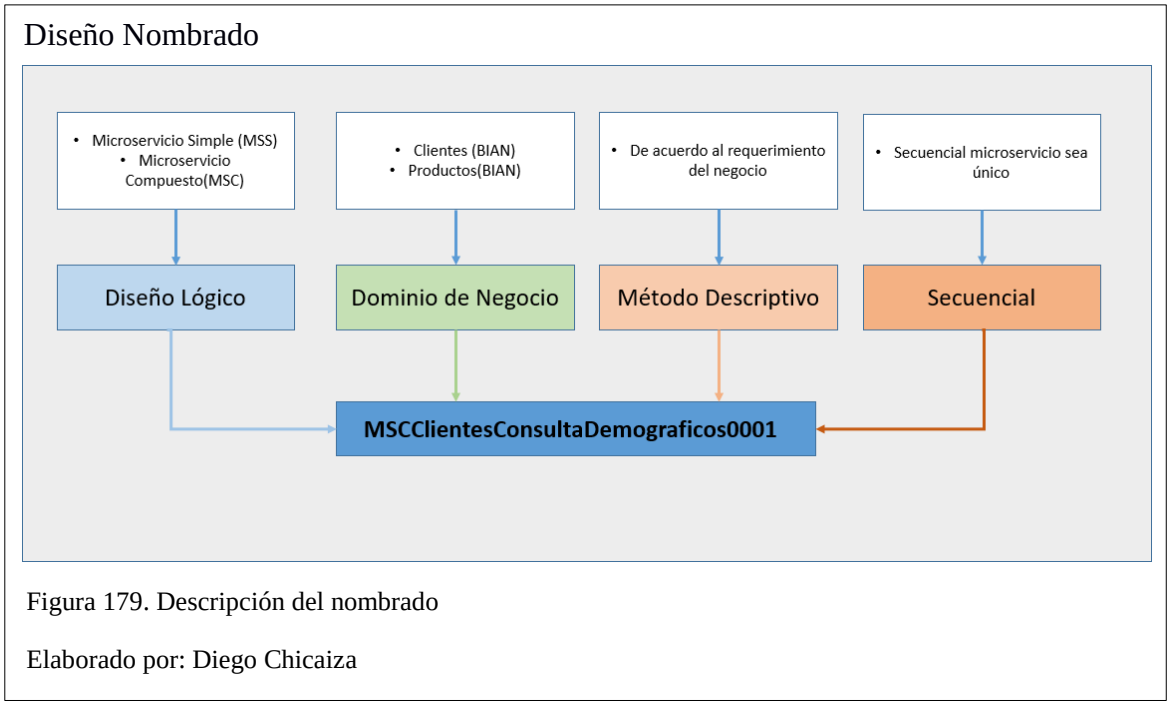

Figura 179. Descripción del nombrado

Elaborado por: Diego Chicaiza

2.4.6. Definición de interfaz de un microservicio

Las nuevas situaciones que enfrentan los sistemas informáticos es la trazabilidad de las peticiones. En las arquitecturas clásicas una petición que llegaba a la aplicación era resuelta funcionalmente dentro de la propia aplicación, salvo alguna interacción con base de datos o con algún servicio para funcionalidades de autenticación o similares. Pero en las arquitecturas de

microservicios basados en el principio de responsabilidad única, nos encontramos un ecosistema compuesto por varias aplicaciones.

La solución a este problema pasa por asociar un identificador a cada petición que el usuario realice a nuestro microservicio, este identificador se incluirá también en la trazabilidad de log que vaya generando dicha petición en la aplicación y en caso de invocar a otro microservicio también se propagará, de forma que tendrá el mismo valor en microservicios diferentes, permitiéndonos seguir la funcionalidad ejecutada por cada petición realizada por el usuario.

Por lo antes expuesto se plantea crear la siguiente estructura de interfaces que debe tener un microservicio.

Header: Campos obligatorios que permiten la interoperabilidad entre los microservicios, de acuerdo a las definiciones antes expuestas, el header es la base para componentes transversales como logs y monitoreo.

Header In: Se refiere a todos los campos definidos para la cabecera de entrada de un microservicio.

Header Out: Se refiere a todos los campos definidos para la cabecera de salida de un microservicio.

Body In: Campos requeridos para implementar las capacidades, crear, consultar, actualizar, eliminar, en los microservicios.

Body Out: Campos resultantes de las capacidades, crear, consultar, actualizar, eliminar, en los microservicios.

Error: Campos que detallan la severidad del mensaje de respuesta.

Definición campos Header In / Out

- Canal: código que es definido por la aplicación
- Guid: código que permite identificar a una transacción como irrepetible desde el punto de vista de negocio.
- Estación: código de agencia o departamento de donde se realizó la petición esto es enviado desde la aplicación.
- Usuario: identifica con que usuario ejecutaron la petición de la transacción.
- IP: Dirección pública del cliente o del servidor de la aplicación de negocio.

Definición campos Error

- Código: Código de respuesta de cada componente
- Mensaje: Mensaje técnico de respuesta
- Identificador: Código de petición de microservicio.
- Severidad: Severidad de la respuesta puede ser INFO, WARN, ERROR, FATAL.

Diseño de Interfaz

Los microservicios deben tener la siguiente interfaz.

En la Figura 20, se puede observar la estructura de la interfaz de los microservicios compuestos la interfaz de entrada está compuesta por un Header In (cabecera de entrada) y un Body In (cuerpo de entrada) y una interfaz de salida que tiene un Header Out (cabecera de salida) un Body Out (cuerpo de salida) y un tag error.

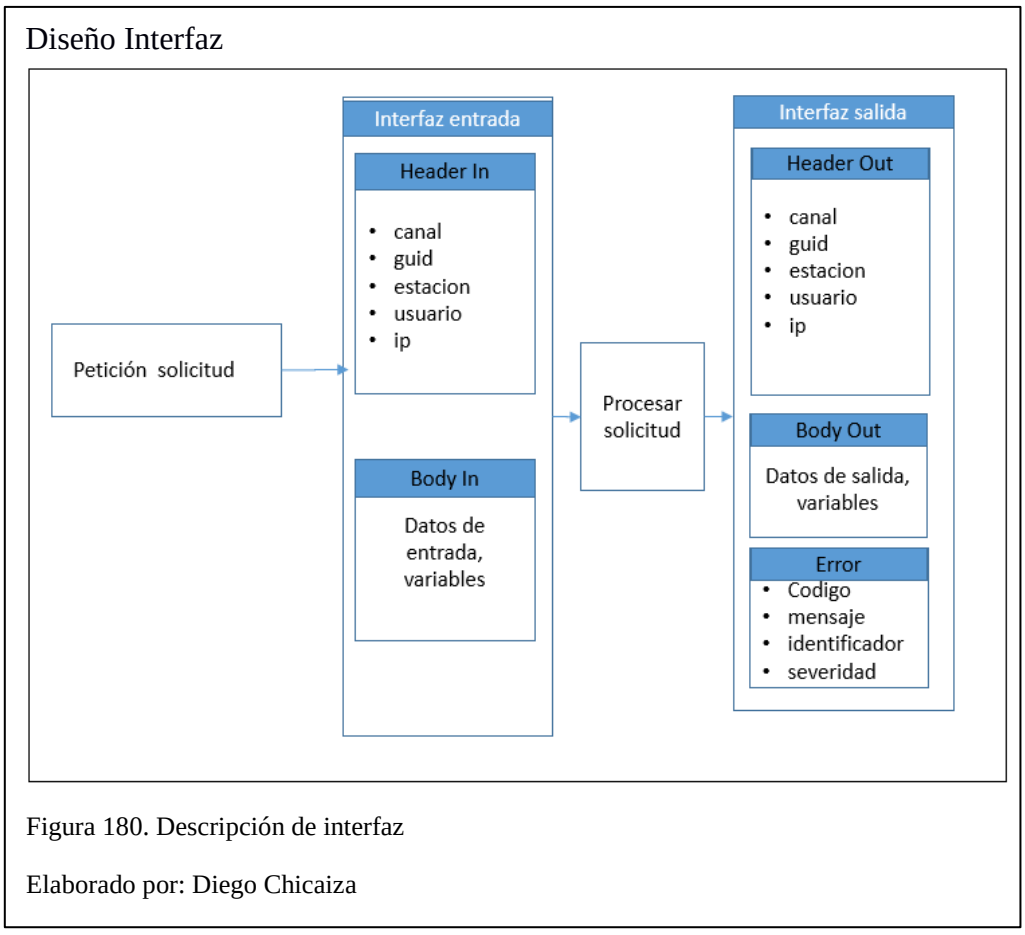

Figura 180. Descripción de interfaz

Elaborado por: Diego Chicaiza

2.4.7. Diseño de Despliegue.

En el diseño de una arquitectura de microservicios un concepto fundamental es el despliegue independiente, por tal motivo se plantea el siguiente diseño.

Se simulará que tenemos dos servidores con los nodos de integración que nos proporciona este ID (integration Toolkit) de desarrollo cada nodo tendrá la siguiente estructura.

Integration Server: este servidor de integración será creado (la creación de este componente se realiza en el nodo de integración del IDE de desarrollo) de acuerdo a los dominios de negocio definido, para nuestra demostración se crea dos tipos de servidores de integración.

IServerClientes, IServerProductos

De acuerdo a la carga transaccional se crea dos tipos de servidores de integración esto permite aislar funcionalidad de los microservicios, generando dependencia entre los componentes desplegados en la plataforma.

Servidor de integración agrupado: aquí se desplegarán todos los microservicios que no son transaccionales compartirán recursos, pero estarán relacionados de acuerdo al dominio de negocio.

Servidor de integración transaccional: aquí se desplegará un solo microservicio de acuerdo a la carga transaccional, o si es un microservicio transversal que sea llamado por otros componentes de nuestra arquitectura.

Los microservicios transaccionales deben estar aislados ejecutándose con sus propios recursos hardware y software, aislar es una buena práctica para mitigar posibles fallos, si es necesario se puede crear otro servidor de integración con el microservicio asociado de esa manera podremos garantizar la disponibilidad del microservicio.

En la Figura 21, se observa el diseño propuesto para el despliegue de microservicios, se tiene dos nodos los cuales alojan servidores de integración agrupados y transaccionales, tanto el nodo1 como el nodo 2 tendrán la misma cantidad de microservicios por el balanceo de peticiones.

Diseño Nodos de despliegue

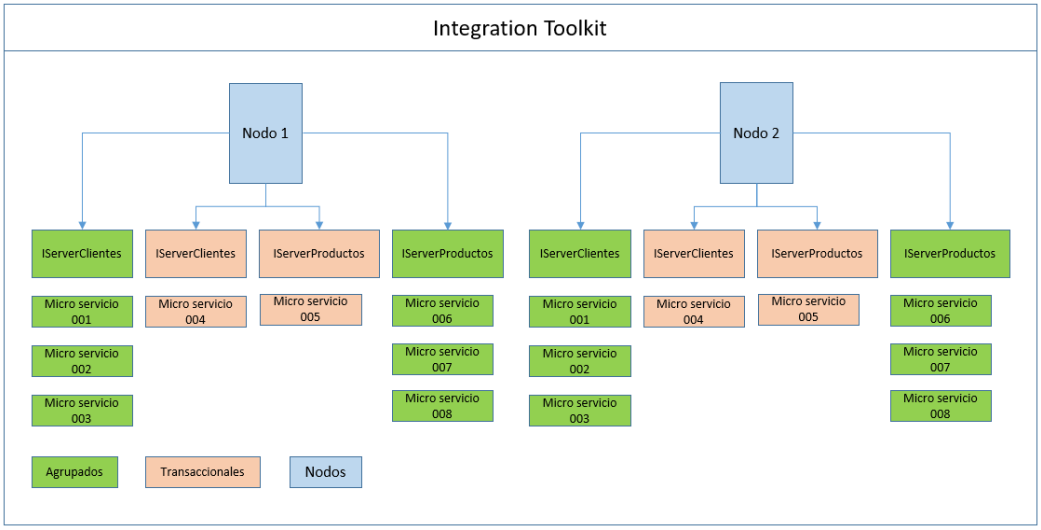

Figura 191. Descripción de los nodos de despliegue

Elaborado por: Diego Chicaiza

Los microservicios deben ser desplegados en el esquema propuesto después de realizar un análisis de su carga transaccional o el impacto técnico que puede tener en nuestra arquitectura.

CAPÍTULO 3

3. CONSTRUCCIÓN Y PRUEBAS

Para la demostración de las bondades que brinda una arquitectura de microservicios se realiza a la construcción de un prototipo basado en los conceptos antes expuestos, se demuestra la eficiencia respecto al rendimiento realizando pruebas de carga de los componentes a ser creados, para realizar esta implementación utilizaremos lo siguiente:

De acuerdo a lo definido, para la construcción de los microservicios se utilizará el IDE Integration Developer de IBM, por ser solo una demostración y luego del análisis de las herramientas se decidió utilizar lo mencionado, esta distribución es una versión académica, idónea para realizar pruebas de tecnología.

Esta arquitectura planteada puede ser aplicada en cualquier tecnología, aplicando los diferentes lenguajes de programación.

3.1. Estándares de programación

Para el desarrollo de los microservicios es necesario seguir prácticas de programación esto dependerá de acuerdo con el lenguaje que se escoja para la implementación de los microservicios propuestos para nuestro caso utilizaremos el lenguaje ESQL.

1. Se debe inicializar las variables utilizando las declaraciones DECLARE para no tener un número de declaraciones innecesarias, se debe usar sola declaración en nuestros bloques de código.

2. Se debe declarar una REFERENCIA para no tener un exceso de navegación del árbol de mensajes se debe usar variables de referencia para reducir la sobrecarga al buscar en árboles de mensajes.

3. Se debe combinar sentencias ESQL en el número mínimo de nodos lógicos posibles, menos nodos generalmente significan menos consumo de recursos.

4. La cantidad de líneas de código utilizadas en un procedimiento o función ESQL afecta el rendimiento del flujo o sub flujo de mensajes. Se debe considerar usar una instrucción SELECT cuando se pueda usar una sola instrucción ESQL en lugar de varias sentencias SET.

5. En la cláusula WHERE se puede usar cualquier operador o función para hacer referencia a columnas de una tabla, campos de mensaje y cualquier variable o constante declarada en nuestro código.

3.2. IDE Integration Toolkit de desarrollo

El marco de trabajo propuesto se construyó en el IDE de IBM, Integration Toolkit este permite realizar componentes independientes, reutilizables que puedan ser utilizados en el desarrollo de microservicios. Se propone generar un marco de trabajo que será utilizado los lineamientos respecto a la creación de microservicios, este marco de trabajo permitirá que todos los desarrollos de microservicios cumplan con la teórica de microservicios y no tengan libertad para su construcción de componentes transversales, al tener librerías embebidas en el marco de trabajo, estas estarán encapsulados en cada desarrollo de microservicio.

En la Figura 22, se observa los componentes que tiene el IDE de desarrollo la agrupación lógica de sus componentes.

Componentes de IDE

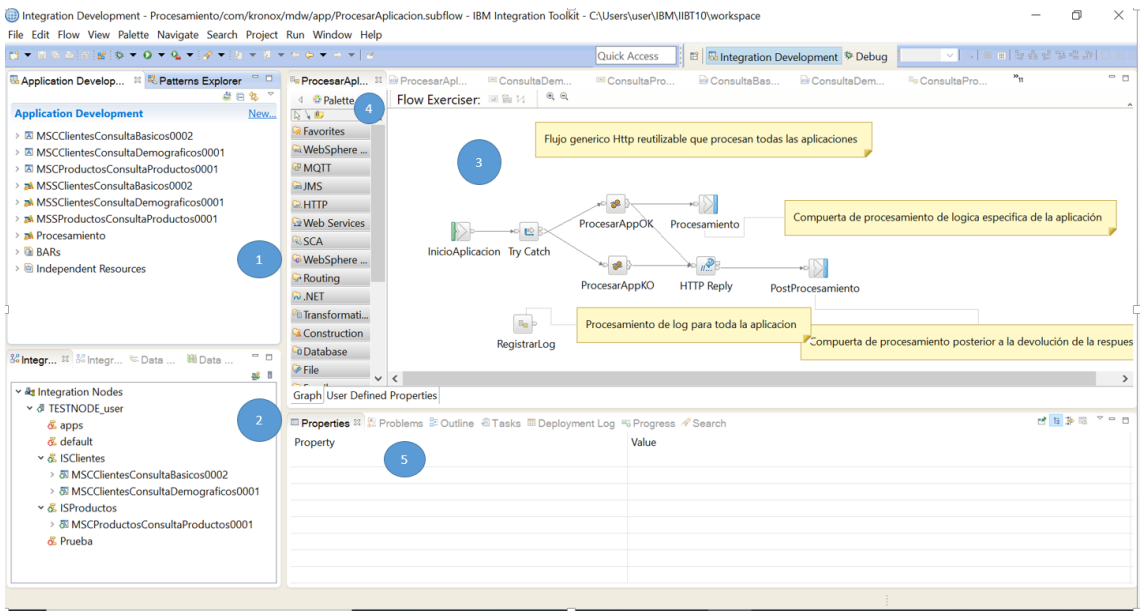

Figura 202. Descripción de componentes del IDE

Elaborado por: Diego Chicaiza

3.2.1. Desarrollo de aplicación

El IDE de desarrollo tiene varios componentes uno de ellos es el desarrollo de la aplicación en este espacio lógico de trabajo se crea la estructura de los microservicios que será poblado por el código que se implementara de acuerdo con lo solicitado por el usuario.

3.2.2. Nodos de integración

Espacio lógico de trabajo utilizado para crear servidores de integración estos componentes albergan todos los microservicios simples y compuestos, se pueden crear tantos se requiera de acuerdo a la carga transaccional existente, este espacio de trabajo nos permite aplicar

crecimiento horizontal (crear más servidores de integración aislados) de acuerdo al diseño de despliegue explicado en el capítulo anterior.

3.2.3. Editor de programación

Espacio lógico de trabajo utilizado para desarrollar componentes que son utilizados en la construcción de los microservicios simples y compuestos, aquí se ingresa el código según su estructura definida de acuerdo a la arquitectura planteada.

3.2.4. Paleta de nodos de programación

Este IDE de desarrollo proporciona componentes que agilitan el trabajo, ayudan a la construcción de componentes transversales (marco de trabajo, mecanismo de log de aplicación) y microservicios simples y compuestos para esta demostración se tiene los siguientes:

- **Nodo Compute:** se utiliza este nodo para crear mensajes de entrada o salida, estos pueden ser datos que se alojan en cualquier repositorio de información, se da uso en las siguientes acciones: Al crear un nuevo mensaje utilizando sentencias de asignación, copiar mensajes entre analizadores, convertir mensajes de código, transformar mensajes de un formato a otro.
- **Nodo http input:** Se utiliza este nodo para recibir un mensaje HTTP que proviene de un cliente HTTP para que sea procesado por un flujo de mensajes.
- **Nodo http reply:** Se utiliza el nodo parar devolver una respuesta a un cliente HTTP desde el flujo de mensajes de la invocación realizada.
- **Nodo input:** Se utiliza el nodo como compuerta de entrada para un flujo de mensaje que se integra a un sub flujo, en la llamada de entrada de una petición.
- **Nodo try catch:** Se utiliza este nodo para generar excepciones propagadas en el flujo de mensajes.

- **Nodo label:** Se utiliza este nodo para ejecutar un mensaje de entrada, que se propaga mediante otro nodo RouteToLabel que determina aleatoriamente el camino que toma el mensaje a través del flujo de mensajes.
- **Nodo trace:** Se utiliza este nodo en la construcción del prototipo, para generar registros de rastreo supervisa la salud del flujo de mensajes, ayuda a dejar el registro en la ruta del file system definido.

En la Figura 23, se observa la utilización de los nodos de programación y su integración de acuerdo a su rol explicado en la sección 3.2.4 paleta de nodos de programación.

Diseño nodos de programación

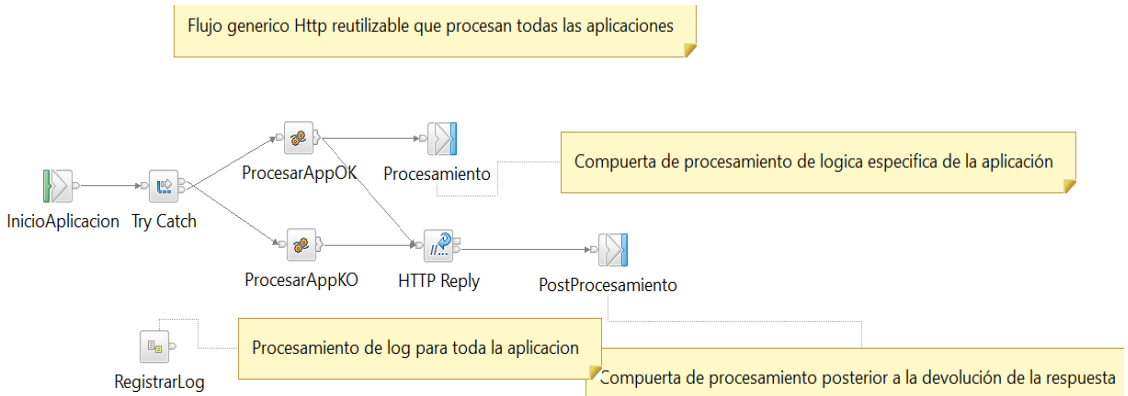

Figura 213. Nodos de programación implementados

Elaborado por: Diego Chicaiza

3.2.5. Propiedades

Las propiedades van relacionadas con los nodos de programación cada nodo tiene configuraciones propias según la lógica de uso.

En la Figura 24, se observa las propiedades del nodo compute utilizado en el desarrollo de componentes transversales (marco de trabajo, log de aplicación, microservicios).

Propiedades Nodo de programación

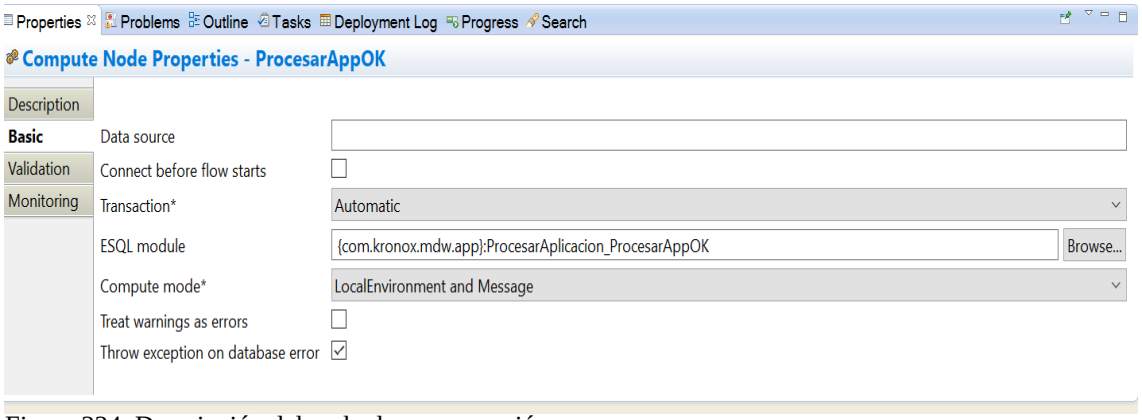

Figura 224. Descripción del nodo de programación

Elaborado por: Diego Chicaiza

3.3. Lenguaje de programación

ESQL: Es un lenguaje de programación definido por IBM Integration Developer para definir y manipular datos en un flujo de mensajes. El código ESQL creado para personalizar nodos en un flujo de mensajes dentro de un marco de trabajo se define en un archivo ESQL, generalmente denominado <nombre_flujo_mensaje>.esql, que está asociado al proyecto de integración.

3.4. Desarrollo marco de trabajo.

Todo microservicio deberá utilizar las funcionalidades embebidas en el marco de trabajo desarrollado, esto fue implementado para agilizar el tiempo de desarrollo, y para estandarizar la estructura del header (cabecera del microservicio) y body (cuerpo del microservicio) así como también el tag error (campos fijos del error propagado) que se propuso en el diseño, a través de este componente se podrá llamar a librerías que fueron desarrolladas para el mecanismo de log.

En la Figura 25, se puede visualizar el diseño de los compontes de marco de trabajo.

Compontes de marco de trabajo

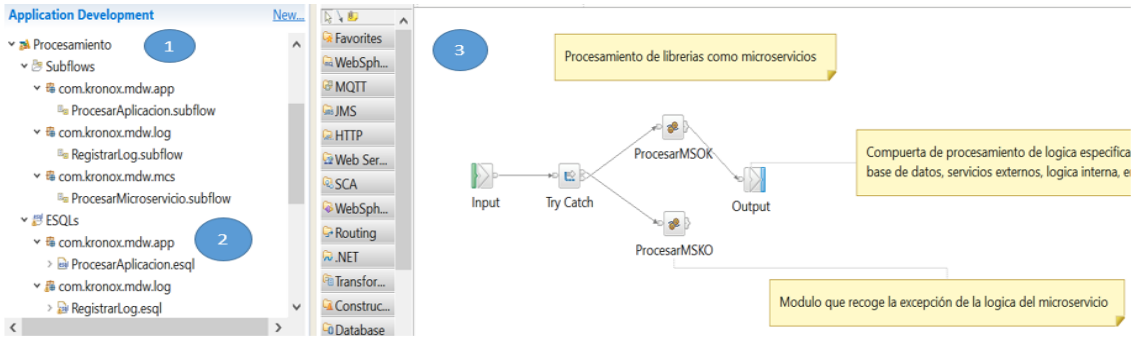

Figura 235. Diseño de los compontes de marco de trabajo

Elaborado por: Diego Chicaiza

De acuerdo a lo definido en la arquitectura el marco de trabajo contiene muchas librerías una de ellas es la de procesamiento, aquí se crea los subflujos (interacción de mensajes en forma visual utilizando los nodos de programación), estos tienen relación con el código esql que tiene cada uno de ellos y su entorno de programación.

3.4.1. Procesar aplicación

Módulo principal del marco de trabajo este realiza la integración con las librerías del mecanismo central de log de aplicación y estructura genérica de microservicios simples y compuestos, está compuesto de un flujo genérico reutilizable que embebe la lógica de programación en cada nodo de programación.

En la Figura 26, se observa el flujo de procesar aplicación como llama al componente registrar log en cada invocación a un microservicio.

Flujo procesar aplicación

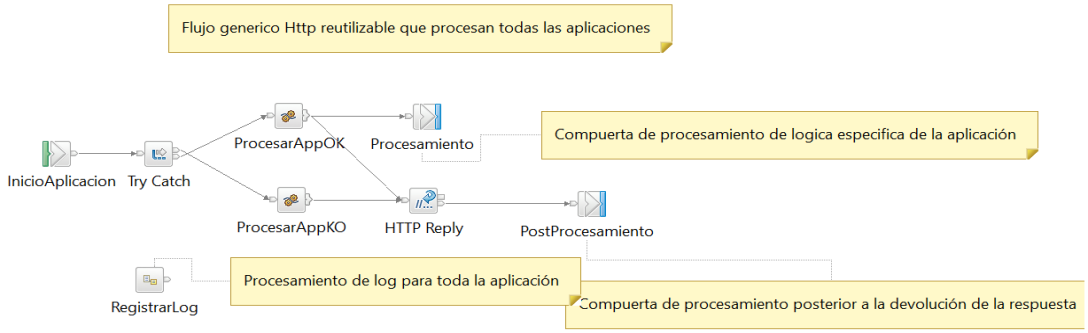

Figura 246. Flujo del procesamiento de la aplicación

Elaborado por: Diego Chicaiza

Este marco de trabajo tendrá una compuerta de entrada mediante el cual ingresaran las peticiones será controlado por un nodo try cash, que permitirá propagar las excepciones en caso de existirlas. El nodo try cash (controlador de excepciones), propagara dos tipos de ejecuciones una exitosa y otra fallida en el caso de tener excepciones. En el nodo ProcesarAppOK ingresan las peticiones exitosas aquí está programada la lógica para que luego sea llamada desde los microservicios compuestos.

Código relevante

En el desarrollo del marco de trabajo es necesario resaltar los fragmentos de código importante que ayuda a entender la lógica que tiene cada componente, a continuación, se explica las líneas de código del módulo procesar aplicación.

Figura 27, se observa la estructura del proyecto el cual tiene un paquete que embebe el archivo ESQL llamado procesar aplicación aquí se desarrolla la lógica de programación.

Estructura procesar aplicación

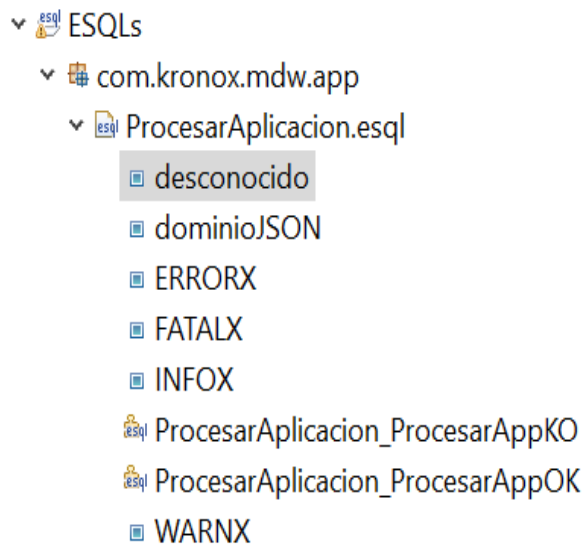

Figura 257. Descripción de la estructura de procesamiento de la aplicación

Elaborado por: Diego Chicaiza

Todo componente tiene una estructura definida para el caso se crea un proyecto llamado ESQLs el cual contiene el paquete procesar aplicación aquí se crea un archivo esql donde se alojará la lógica de programación clases métodos el código relevante a ser analizado será ProcesarAplicacion_AppOK, procesa peticiones positivas.

En la Figura 28, se observa el código relevante de acuerdo a los principios de microservicios estos módulos son simples que cumplen una función única.

Código relevante procesar aplicación

```
1  /*
   *
   *   archivo     :   ProcesarAplicacion.esql
   *   descripción :   Modulo de procesamiento de aplicaciones
   *   creador     :   Diego chicaiza
   *   fecha       :   2019 - 10 - 18
   *
   */
2  -- Dominios parceo soportados
   DECLARE dominioJSON CHARACTER 'JSON';
   DECLARE desconocido CHARACTER 'UNKNOWN';
   /*
   * Módulo de carga exitoso
   */
3  CREATE COMPUTE MODULE ProcesarAplicacion_ProcesarAppOK
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
           DECLARE campo CHARACTER NULL;
           DECLARE tiempo TIME CURRENT_TIME;
           SET OutputRoot.Properties       = InputRoot.Properties;
           SET OutputRoot.HTTPInputHeader  = InputRoot.HTTPInputHeader;
4  -- Referenciamos al mensaje de entrada
   DECLARE servicio REFERENCE TO Environment.datosAPP;

   -- Creamos estructura de salida
5  CREATE FIELD OutputRoot.JSON.Data.campo IDENTITY (JSON.Object){servicio.servicio || 'Response'};
   DECLARE salida REFERENCE TO OutputRoot.JSON.Data.{servicio.servicio || 'Response'};


6  -- Armar entrada para los logs
   SET Environment.log.usuario        = InputRoot.JSON.Data.[<].header.usuario;
   SET Environment.log.id             = SUBSTRING(CAST(InputLocalEnvironment.Destination.HTTP.RequestIdentifier AS CHARACTER) FROM 3 FOR 48 );
   SET Environment.log.guid           = InputRoot.JSON.Data.[<].header.guid;


7  -- Registramos Log
   CALL com.kronox.mdw.log.ImprimirAuditoriaTrama(Environment.log, servicio.servicio, 'Peticion', InputRoot.JSON.Data);


8   -- Cargamos el body al entorno
    SET Environment.entrada = InputRoot.JSON.Data.[<];


9  -- Generar la salida con el DOMINIO JSON.
   CREATE LASTCHILD OF Environment DOMAIN dominioJSON NAME 'salida';


10   -- Propagamos a la logica del microservicio
     PROPAGATE TO TERMINAL 'out1' DELETE NONE;
```

Figura 268. Código de procesamiento de la aplicación

Elaborado por: Diego Chicaiza

1. Todo desarrollo debe tener una documentación en su código para que sea entendible por otras personas del equipo, por esta razón en los componentes creados se tiene una etiqueta con la información básica del creador del código, para realizar mantenimientos y evaluar impactos es de mucha ayuda.

2. Declaración de mapeos de formato de datos que interpretara las tramas de entrada y salida de nuestros microservicios, este prototipo soporta Json si el formato no es correcto la petición entrara al módulo erróneo propagando una excepción tipo desconocido.
3. Creación del módulo principal del marco de trabajo tiene las funciones a ser ejecutadas de acuerdo a la petición realizada.
4. Declaración de la referencia al mensaje de entrada de los microservicios para que sea utilizada por los subflujos (lógica grafica de peticiones de mensajes) invocados.
5. Declaración de la referencia al mensaje de salida de los microservicios para que sea utilizada por los subflujos (lógica grafica de peticiones de mensajes) invocados.
6. En las sentencias se visualiza como se asigna un valor a la trama del mecanismo del log de aplicación.
7. Sentencia que permite realizar el registro del log de aplicación de los microservicios, esta sentencia será invocada de acuerdo a la lógica del log auditoria, rendimiento, excepciones, operación.
8. Asignación de un valor a la variable que contiene el body (cuerpo del microservicio).
9. Sentencia para generar la estructura genérica de salida de los microservicios.
10. Sentencia que permite propagar en mensaje de entrada o salida, se integra con la lógica de construcción del microservicio.

3.4.2. Registrar Log

En el diseño de una arquitectura de microservicios un principio importante es el monitoreo para poder saber la salud del servicio y poder dar un servicio más eficiente al cliente. Las aplicaciones invocan a los microservicios esperando una respuesta de acuerdo a la lógica del negocio, es importante tener aplicaciones o plataforma de monitoreo las cuales nos proporcionan métricas de las peticiones realizadas, respuestas exitosas o erróneas. Para esto se

plantea un mecanismo de log que permitirá explotar la información técnica y transaccional del microservicio invocado.

Este módulo permite el registro del log se integra con el módulo de procesamiento de microservicios, está compuesto de un flujo genérico reutilizable que embebe la lógica de programación en cada nodo de programación.

En la Figura 29, se observa el flujo del log como llama al componente registrar log.

Flujo registrar log

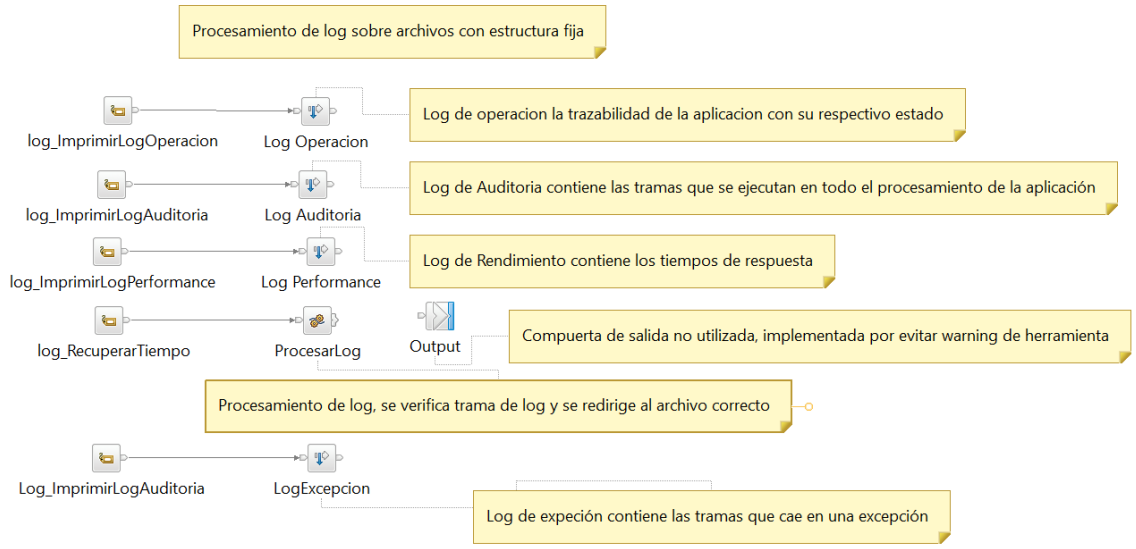

Figura 279. Descripción del flujo para registrar log

Elaborado por: Diego Chicaiza

Por medio de etiquetas recupera las peticiones que pasan por el módulo procesar aplicación, luego se integra a un nodo trace que permite guardar el registro de cada log en la ruta definida /ESB/Rastreo/logPerformance.log, aquí se define como vamos a aguardar nuestros logs en este caso serán archivos.

Código relevante

En el desarrollo del mecanismo de log es necesario resaltar los fragmentos de código importante que ayuda a entender la lógica que tiene cada componente.

Figura 30, se observa la estructura del archivo ESQL registrar log.

Estructura registrar log

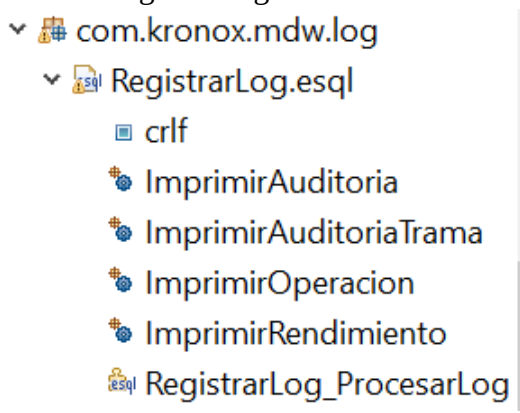

Figura 280. Descripción de la estructura para registrar log

Elaborado por: Diego Chicaiza

Este módulo tiene un paquete que embebe un archivo esql donde se alojará la lógica de programación de clases y métodos el código relevante a ser analizado será RegistrarLog_ProcesarLog, procedimiento imprimir operación, no se realiza el análisis de otros procedimientos porque su lógica es similar.

En la Figura 31, se observa el código relevante de acuerdo a los principios de microservicios estos módulos son simples que cumplen una función única.

Registro de log

Figura 291. Código relevante registrar log

Elaborado por: Diego Chicaiza

1. El procedimiento creado permite imprimir la operación de tipo severidad, módulo, código y mensaje a través de la asignación de variables que son propagadas al módulo procesar operación.

3.4.3. Procesar microservicio

Módulo que permite implementar la lógica genérica de un microservicio, se integra con el módulo procesar aplicación para completar funcionalidades específicas como guardado de registro de log, integraciones y componentes backend, está compuesto de un flujo genérico reutilizable que embebe la lógica de programación en cada nodo de programación.

En la Figura 32, se observa la relación de cada nodo de programación en el flujo procesar microservicio.

Flujo procesar microservicio

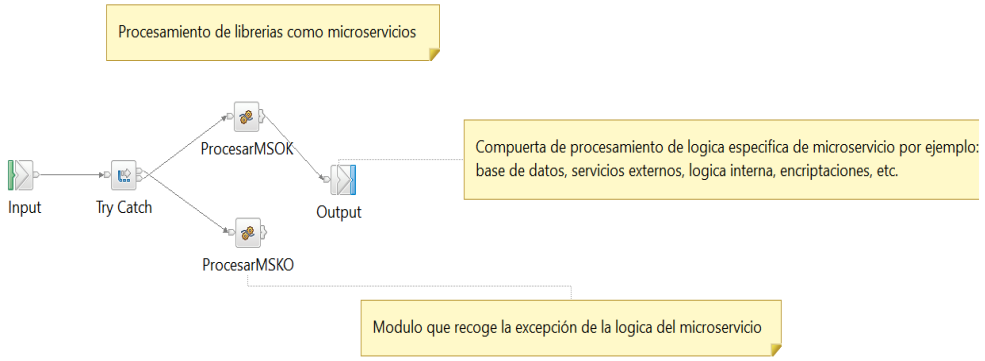

Figura 302. Descripción del flujo de procesamiento del microservicio

Elaborado por: Diego Chicaiza

Este módulo tendrá una compuerta de entrada mediante el cual ingresaran las peticiones será controlado por un nodo try cash, que permitirá propagar las excepciones en caso de existirlas. El nodo try cash (controlador de excepciones), propagara dos tipos de ejecuciones una exitosa y otra fallida en el caso de tener excepciones. En el nodo ProcesarMSOK ingresan las peticiones exitosas aquí está programada la lógica llamado a microservicios simples.

Código relevante

En el desarrollo del mecanismo de integración a microservicios es necesario resaltar los fragmentos de código importante que ayuda a entender la lógica que tiene cada componente.

Figura 33, se observa la estructura del archivo ESQL procesar microservicio

Estructura procesar microservicio

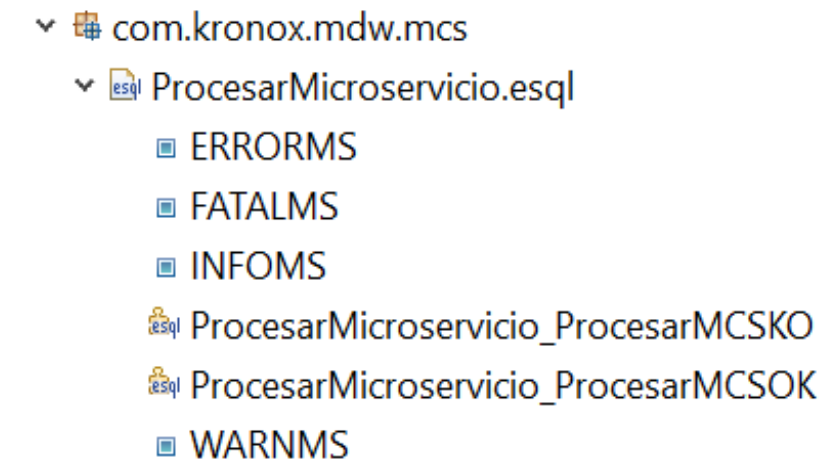

Figura 313. Descripción de la estructura de procesamiento del microservicio

Elaborado por: Diego Chicaiza

Este módulo tiene un paquete que embebe un archivo esql donde se alojará la lógica de programación de clases y métodos el código relevante a ser analizado será ProcesarMicroservicio caso exitoso ProcesarMicroservicio_ProcesarMCSOK.

En la Figura 34, se observa el código ESQL relevante que será analizado en el cual se resalta los módulos más importantes que serán explicados.

Código procesar microservicio

```
CREATE COMPUTE MODULE ProcesarMicroservicio_ProcesarMCSOK
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        DECLARE tiempo TIME CURRENT_TIME;
        DECLARE microservicio, metodo CHARACTER NULL;
        SET OutputRoot.Properties   = InputRoot.Properties;
        DECLARE flujoMicroservicio REFERENCE TO Environment.flujoMicroservicio;
        SET flujoMicroservicio.salida   = NULL;
        SET flujoMicroservicio.error    = NULL;

        -- Verificamos si existe datos correctos
        IF FIELDTYPE(flujoMicroservicio.microservicio) IS NOT NULL AND flujoMicroservicio.microservicio <> '' THEN
            SET microservicio = flujoMicroservicio.microservicio;
        ELSE
            SET flujoMicroservicio.error.codigo         = 'MSS-1001';
            SET flujoMicroservicio.error.mensaje        = 'Nombre de microservicio incorrecto';
            SET flujoMicroservicio.error.identificador  = Environment.log.id;
            SET flujoMicroservicio.error.severidad      = FATALMS;

            -- Registramos log
            CALL com.kronox.mdw.log.ImprimirOperacion(Environment.log, FATALMS, microservicio, flujoMicroservicio.error.codigo,
            CALL com.kronox.mdw.log.ImprimirRendimiento(Environment.log, FATALMS, microservicio, tiempo);
            RETURN FALSE;
        END IF;
    -- Propagamos a la logica del servicio
    PROPAGATE TO TERMINAL 'out1' DELETE NONE;
```

Figura 324. Código relevante para procesar el microservicio

1. Creación del módulo procesar microservicio permite declarar variables y asignar responsabilidades de integración con sentencias internas del flujo procesar microservicio.
2. Permite verificar la estructura y nombrado de los microservicios.
3. Esta sentencia tiene relación con el mecanismo de log permite imprimir el log del microservicio.
4. Sentencia que permite la integración a la lógica individual del microservicio.

3.5. Construcción de microservicios

Siguiendo la teoría de microservicios tamaño pequeño, desarrollo de forma autónoma, implementado de forma independiente, descentralizado (Newman, 2015), se procede a la construcción de los siguientes microservicios:

- MSCClientesConsultaDemograficos0001
- MSCClientesConsultaBasicos0002
- MSCProductosConsulta0001

Los microservicios compuestos poseen una interfaz definida de acuerdo a los parámetros de input (datos de entrada del microservicio) y output (datos de salida del microservicio) definido en las historias de usuario, estos microservicios llamarán a los microservicios simples que son los encargados de la integración a los backend, estos microservicios realizan una tarea puntual que será recuperada en los microservicios compuestos, y expuestos a las aplicaciones de lo invocaron.

3.5.1. MSCClientesConsultaDemograficos0001

El siguiente microservicio tendrá una interfaz definida que está compuesto por un header (cabecera del microservicio) parte fija y un body (cuerpo del microservicio) parte variable esto depende de los requerimientos del usuario.

Definición interfaz

De acuerdo a los principios de microservicios este contrato está definido en formato Json, su estructura tiene un nombre descriptivo que hace referencia a la funcionalidad del microservicio el cual encapsula al header (cabecera del microservicio) y el body (cuerpo del microservicio), explicados en el capítulo de diseño, esta estructura permitirá guardar los registros en el log de una forma estructurada.

Campos de Entrada

De acuerdo a la lógica de negocio historia de usuario HU002, se define los siguientes campos de entrada.

En la Figura 35, se observa los datos de la interfaz de entrada, esta información es mapeada en el cuerpo (body) del microservicio MSCClientesConsultaDemograficos0001.

Campos de entrada

MSCClientesConsultaDemograficos0001


```
{
 "ConsultaDemograficos0001": {
  "header": {
   "canal": "WEB",
   "guid": "a7gb9d89g9d3g89gd9898q9b9qb7w6q6w9qw15",
   "estacion": "0012",
   "usuario": "GENERICO",
   "ip": "10.0.57.85"
  },
  "body": {
   "identificacion": "1719348748",
   "tipoIdentificacion": "C"
  }
```


Figura 335. Ejemplo campos de entrada MSCClientes

Elaborado por: Diego Chicaiza

Campos de Salida

De acuerdo a la lógica de negocio historia de usuario HU002, se define los siguientes campos de salida.

En la Figura 36, se observa los datos de la interfaz de salida, esta información es mapeada en el cuerpo (body) del microservicio MSCClientesConsultaDemograficos0001.

Campos de salida

MSCClientesConsultaDemograficos0001


```
{"ConsultaDemograficos0001Response": {
  "header":   {
    "canal": "WEB",
    "guid": "a7gb9d89g9d3g89gd9898q9b9qb7w6q6w9qw15",
    "estacion": "0012",
    "usuario": "GENERICO",
    "ip": "10.0.57.85"
  },
  "body":   {
    "codigo": 5,
    "nacionalidad": "Ecuatoriana",
    "genero": "Femenino",
    "estudios": "Maestria",
    "fechaNacimiento": "1990-06-12"
  },
  "error":   {
    "codigo": "0",
    "mensaje": "OK",
    "identificador": "485454500000000000000000e99dea57802e000000000000",
    "severidad": "INFO"
  }
}}
```


Figura 346. Ejemplo campos de salida MSCClientes

Elaborado por: Diego Chicaiza

Flujo de mensajes microservicio MSCClientesConsultaDemograficos0001

Los microservicios compuestos son los únicos que poseen interfaz por ende se desarrolla un flujo de mensajes que realizara el llamado a los microservicios simples de acuerdo a la arquitectura definida.

En la Figura 37, se puede observar la estructura del microservicio los componentes que serán utilizados para la generación del mensaje.

Estructura del microservicio

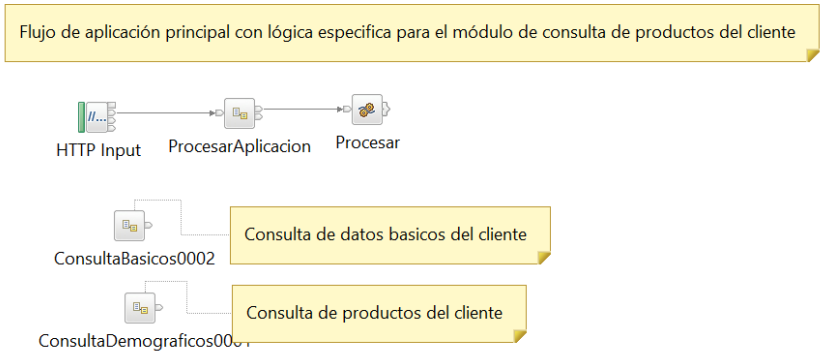

Figura 357. Descripción de la estructura del microservicio

Elaborado por: Diego Chicaiza

Llamado a microservicios simples.

Estos microservicios no realizan integraciones a bankend para completar la función solicitada deben llamar a los microservicios simples.

MSSClientesConsulta Basicos0002.

Este micro tendrá la responsabilidad de extraer los datos básicos de un cliente almacenados en la base de datos.

Código relevante

En el desarrollo del microservicio es necesario resaltar los fragmentos de código importante que ayuda a entender la lógica que tiene cada componente, a continuación, se explica las líneas de código del microservicio simple.

En la Figura 38, se visualiza el código relevante del microservicio.

Código microservicio MSSClientesConsultaBasicos0002

```
1  -- Validamos si existe datos para consulta
   SET OutputLocalEnvironment.respuestaBDD[] = SELECT P.codigo, P.nombre1, P.nombre2, P.apellido1, P.apellido2, P.codigo, P.tipo
               FROM Database.PostgreSQLClient.clientes."inf_basic_cli" AS P
               WHERE P.identificacion = entrada.identificacion
               AND P.tipoIdentificacion = entrada.tipoIdentificacion;
```

Figura 368. Código relevante del microservicio MSSClientesConsultaBasicos0002

Elaborado por: Diego Chicaiza

1. La siguiente sentencia realiza una consulta a la base de datos, tabla información básica.

MSSClientesConsultaDemograficos0001

Este microservicio tendrá la responsabilidad de extraer los datos demográficos de un cliente almacenados en la base de datos.

Código relevante

En el desarrollo del microservicio es necesario resaltar los fragmentos de código importante que ayuda a entender la lógica que tiene cada componente, a continuación, se explica las líneas de código del microservicio simple.

En la Figura 39, se visualiza el código relevante del microservicio.

Código microservicio MSSClientesConsultaDemograficos0001

```
1  -- Validamos si existe datos para consulta
   SET OutputLocalEnvironment.respuestaBDD[] = SELECT P.codigo, P.nacionalidad, P.fechaNacimiento, P.genero, P.estudios
               FROM Database.PostgreSQLClient.clientes."inf_dmgrfc_cli" AS P
               WHERE P.codigo = entrada.codigoCliente;
```

Figura 379. Código relevante del microservicio MSSClientesConsultaDemograficos0001

Elaborado por: Diego Chicaiza

1. La siguiente sentencia realiza una consulta a la base de datos tabla demográficos.

3.5.2. MSCClientesConsultaBasicos0002

El siguiente microservicio tendrá un contrato definido que está compuesto por un header (cabecera del microservicio) parte fija y un body (cuerpo del microservicio) parte variable esto depende de los requerimientos del usuario, en este microservicio se realiza la lógica de construcción de interfaz.

Definición interfaz: De acuerdo a los principios de microservicios este contrato está definido en formato Json, su estructura tiene un nombre descriptivo que hace referencia a la funcionalidad del microservicio el cual encapsula al header (cabecera del microservicio) y el body (cuerpo del microservicio), explicados en el capítulo de diseño, esta estructura permitirá guardar los registros en el log de una forma estructurada.

Campos de Entrada

De acuerdo a la lógica de negocio historia de usuario HU003, se define los siguientes campos de entrada. En la Figura 40, se observa los datos de la interfaz de entrada, esta información es mapeada en el cuerpo (body) del microservicio MSCClientesConsultaBasicos0002.

Campos de entrada del microservicio MSSClientesConsultaBasicos0002

```
{
  "ConsultaBasicos0002": {
    "header": {
      "canal": "WEB",
      "guid": "a7gb9d89g9d3g89gd9898q9b9qb7w6q6w9qw15",
      "estacion": "0012",
      "usuario": "GENERICO",
      "ip": "10.0.57.85"
    },
    "body": {
      "identificacion": "1719348748",
      "tipoIdentificacion": "C"
    }
```

Figura 380. Ejemplo de campos de entrada del microservicio MSSClientesConsultaBasicos0002

Elaborado por: Diego Chicaiza

Campos de Salida

De acuerdo a la lógica de negocio historia de usuario HU003, se define los siguientes campos de salida.

En la Figura 41, se observa los datos de la interfaz de salida, esta información es mapeada en el cuerpo (body) del microservicio MSCClientesConsultaBasicos0002.

Campos de salida del microservicio MSSClientesConsultaBasicos0002


```
{"ConsultaBasicos0002Response": {
  "header":   {
    "canal": "WEB",
    "guid": "a7gb9d89g9d3g89gd9898q9b9qb7w6q6w9qw15",
    "estacion": "0012",
    "usuario": "GENERICO",
    "ip": "10.0.57.85"
  },
  "body":   {
    "codigo": 5,
    "nombre1": "JESSICA",
    "nombre2": "FERNANDA",
    "apellido1": "BERMEJO",
    "apellido2": "LOPEZ",
    "tipo": "Natural"
  },
  "error":   {
    "codigo": "0",
    "mensaje": "OK",
    "identificador": "485454500300000000000008a2b6369f40c000000000000",
    "severidad": "INFO"
  }
}}
```


Figura 391. Ejemplo de campos de salida del microservicio MSSClientesConsultaBasicos0002

Elaborado por: Diego Chicaiza

Flujo de mensajes microservicio MSSClientesConsulta Basicos0002

Los microservicios compuestos son los únicos que poseen interfaz por ende se desarrolla un flujo de mensajes que realizara el llamado a los microservicios simples de acuerdo a la arquitectura definida.

En la Figura 42, se puede observar la estructura del microservicio los componentes que serán utilizados para la generación del mensaje.

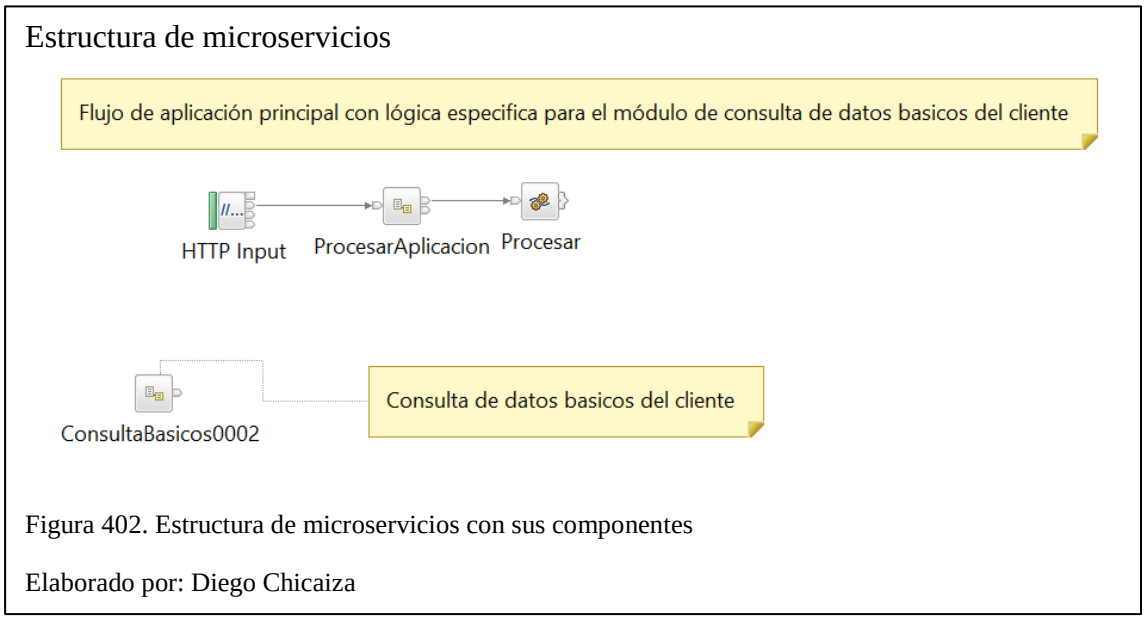

Figura 402. Estructura de microservicios con sus componentes

Elaborado por: Diego Chicaiza

No se explica la lógica del llamado a microservicios porque reutiliza el microservicio simple MSSClientesConsulta Basicos0002.

3.5.3. MSCProductosConsulta0001

El siguiente microservicio tendrá un contrato definido que está compuesto por un header (cabecera del microservicio) parte fija y un body (cuerpo del microservicio) parte variable esto depende de los requerimientos del usuario, en este microservicio se realiza la lógica de construcción de interfaz.

Definición interfaz: De acuerdo a los principios de microservicios este contrato está definido en formato Json, su estructura tiene un nombre descriptivo que hace referencia a la funcionalidad del microservicio el cual encapsula al header (cabecera del microservicio) y el body (cuerpo del microservicio), explicados en el capítulo de diseño, esta estructura permitirá guardar los registros en el log de una forma estructurada.

Campos de Entrada: De acuerdo a la lógica de negocio historia de usuario HU004, se define los siguientes campos de entrada.

En la Figura 43, se observa los datos de la interfaz de entrada, esta información es mapeada en el cuerpo (body) del microservicio MSCProductosConsulta0001.

Campos de entrada de microservicios MSCProductosConsulta0001


```
{
 "ConsultaProductos0001": {
  "header": {
   "canal": "WEB",
   "guid": "a7gb9d89g9d3g89gd9898q9b9qb7w6q6w9qw15",
   "estacion": "0012",
   "usuario": "GENERICO",
   "ip": "10.0.57.85"
  },
  "body": {
   "identificacion": "1754089365",
   "tipoIdentificacion": "C"
  }
 }
}
```


Figura 413. Ejemplo de campos de entrada de microservicios MSCProductosConsulta0001

Elaborado por: Diego Chicaiza

Campos de Salida: De acuerdo a la lógica de negocio historia de usuario HU004, se define los siguientes campos de salida.

En la Figura 44, se observa los datos de la interfaz de salida, esta información es mapeada en el cuerpo (body) del microservicio MSCProductosConsulta0001.

Campos de salida de microservicios MSCProductosConsulta0001


```
{"ConsultaProductos0001Response": {
  "header":   {
    "canal": "WEB",
    "guid": "a7gb9d89g9d3g89gd9898q9b9qb7w6q6w9qw15",
    "estacion": "0012",
    "usuario": "GENERICO",
    "ip": "10.0.57.85"
  },
  "body":   {
    "codigo": 1,
    "nombre": "ANDRES FRANCISCO ALVAREZ ANDRADE",
    "productos": {"producto":      [
            {
        "numero": "000124579661",
        "tipo": "ahorro",
        "subtipo": "unico",
        "moneda": "USD"
      },
            {
        "numero": "000253647894",
        "tipo": "corriente",
        "subtipo": "unico",
        "moneda": "USD"
      }
    ]}
  },
  "error":   {
    "codigo": "0",
    "mensaje": "OK",
    "identificador": "485454500000000000000000029f862587c2d000000000000",
    "severidad": "INFO"
  }
}}
```


Figura 424. Ejemplo de campos de salida de microservicios MSCProductosConsulta0001

Elaborado por: Diego Chicaiza

Flujo de mensajes microservicio MSCProductosConsulta0001

Los microservicios compuestos son los únicos que poseen interfaz por ende se desarrolla un flujo de mensajes que realizara el llamado a los microservicios simples de acuerdo a la arquitectura definida.

En la Figura 45, se observar cómo invocamos a los microservicios simples desde la implementación de un microservicio compuesto.

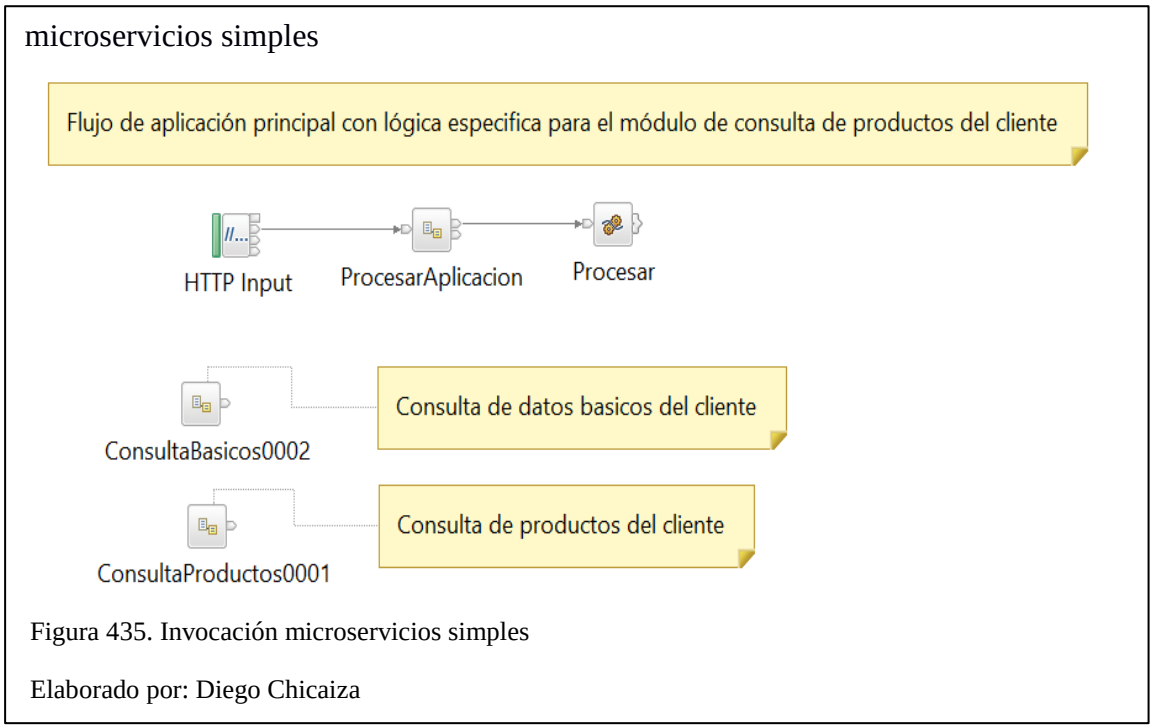

Figura 435. Invocación microservicios simples

Elaborado por: Diego Chicaiza

Llamado a microservicios simples: Estos microservicios no realizan integraciones a bankend para completar la función solicitada deben llamar a los microservicios simples.

MSSProductosConsultaProductos0001

Este microservicio tendrá la responsabilidad de extraer los de los productos que dispone un cliente almacenados en la base de datos.

Código relevante

En el desarrollo del microservicio es necesario resaltar los fragmentos de código importante que ayuda a entender la lógica que tiene cada componente, a continuación, se explica las líneas de código del microservicio simple.

En la Figura 46, se visualiza el código relevante del microservicio.

Invocación microservicios simples MSSProductosConsultaProductos0001

```
1  -- Validamos si existe datos para consulta
   SET OutputLocalEnvironment.respuestaBDD[] = SELECT P.codigo, P.tipo, P.numero, P.subtipo, P.moneda
               FROM Database.PostgreSQLProduc.productos."inf_prdcto_cli" AS P
               WHERE P.codigo = entrada.codigoCliente;
```

Figura 446. Código de invocación Microservicios simples MSSProductosConsultaProductos0001

Elaborado por: Diego Chicaiza

1. La siguiente sentencia realiza una consulta a la base de datos tabla productos.

De acuerdo a los conceptos planteados en los capítulos marco teórico, análisis y diseño se valida que la implementación de los microservicios están bajo los lineamientos (simples, independientes, pequeños) que proponen varios autores (Posta, 2016), (Newman, 2015), (Richardson, 2019), por citar los más importantes.

3.6. Registro mecanismo de log de aplicación.

Es importante validar el registro del log, para lo cual se realiza una invocación al microservicio desde SOAP UI.

En la Figura 47, se observa una petición realizada desde el aplicativo SOAP UI, del microservicio MSCClientesConsultaBasicos0002.

Petición al microservicio desde SOAP UI

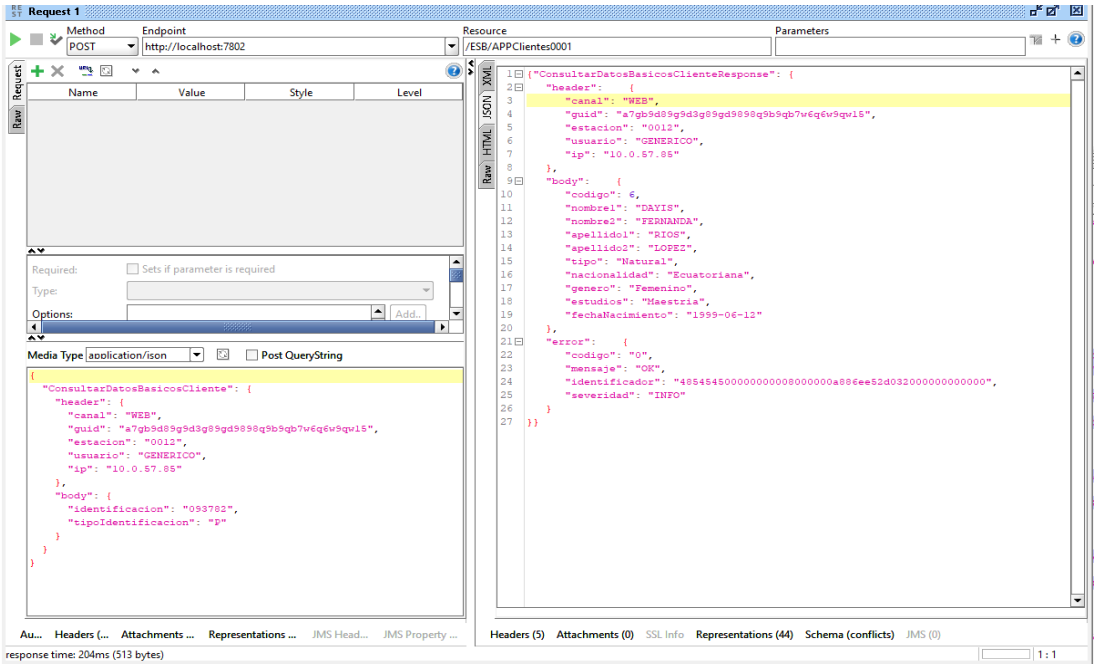

Figura 457. Ejemplo de petición realizada al microservicio desde SOAP UI

Elaborado por: Diego Chicaiza

En la Figura 48, se puede visualizar el registro del log

Registro del log

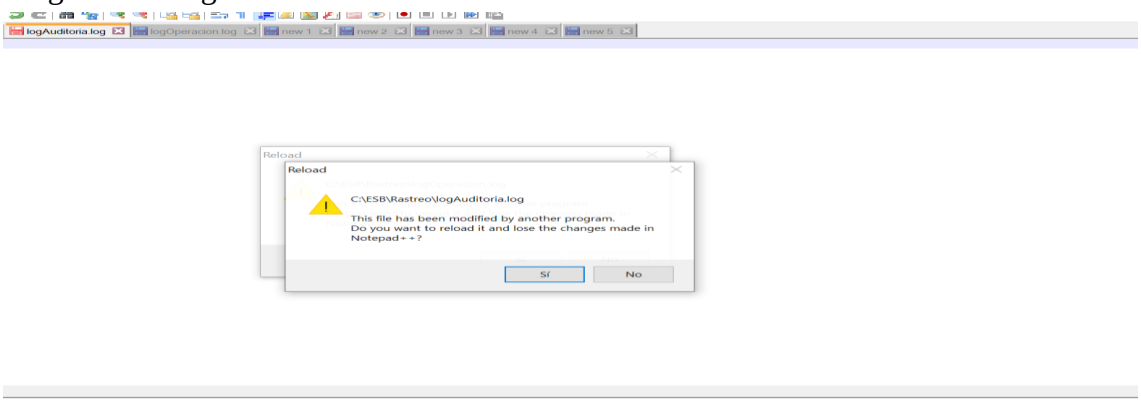

Figura 468. Visualización del registro del log

Elaborado por: Diego Chicaiza

3.6.1. Registros guardados en el log

Log de auditoría, en este log se encontrará 4 registros por petición la llamada del microservicio compuesto trama de entrada y salida, al igual de los micros simples invocados.

En la Figura 49, se visualiza el registro del microservicio.

Registros de petición

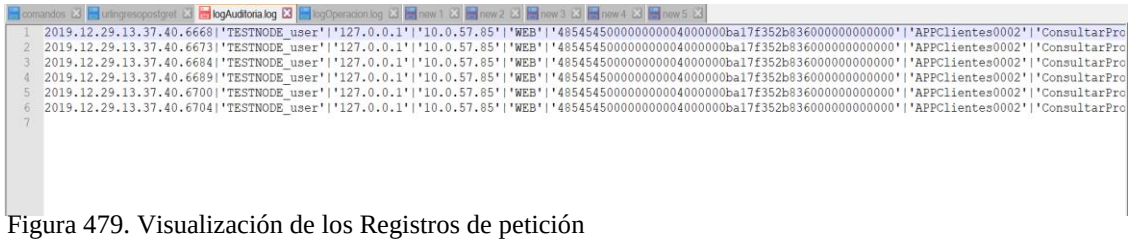

Figura 479. Visualización de los Registros de petición

Elaborado por: Diego Chicaiza

En la Figura 50, visualiza el flujo de invocación, con el detalle de los datos enviados en el cuerpo del contrato tanto en la entrada como en la salida.

Flujos de innovación

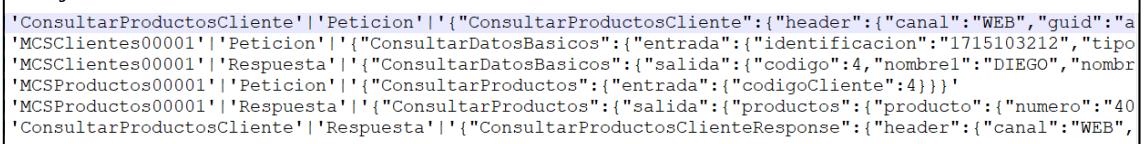

Figura 480. Visualización de flujos de innovación

Elaborado por: Diego Chicaiza

La lógica de guardado de log aplica para los 4 logs (auditoria, rendimiento, excepciones, operación) propuestos en la arquitectura para el ejemplo se presenta el log de auditoria.

3.7. Resultados de pruebas

3.7.1. Resultados Prueba de carga

Los resultados a continuación mostrados son el resultado de las pruebas de carga realizados, en esta prueba se ejecutó con 300 usuarios en un periodo de 6:25 minutos. En la siguiente pantalla se puede observar el resultado de las peticiones procesadas, teniendo respuestas satisfactorias.

En la Figura 51, se observa el árbol de peticiones realizadas en el periodo de tiempo 6:25 minutos, en los datos de respuesta visualizamos la trama de respuesta, en icono verde significa que las peticiones fueron satisfactorias.

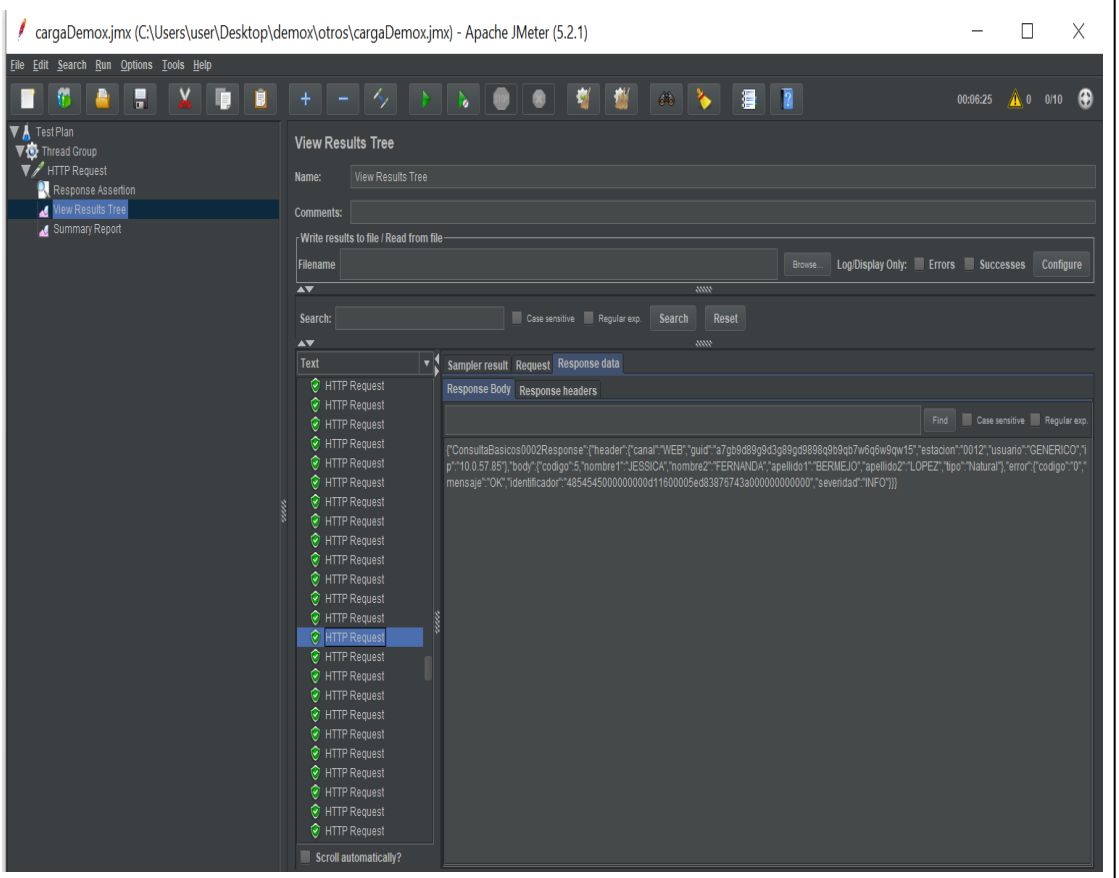

Figura 491. Resultados con respuestas satisfactorias

Elaborado por: Diego Chicaiza

Respuestas satisfactorias

En la Figura 52, se observa el reporte de resultados de la ejecución realizada al microservicio las tramas procesadas son 266152 en el periodo de tiempo 6:25 obteniendo una efectividad del 99%, esta interpretación se da porque en la gráfica se visualiza un margen de error del 0.04%, esto se da por el aumento de usuario en un determinado tiempo.

Peticiones ingresadas y procesadas

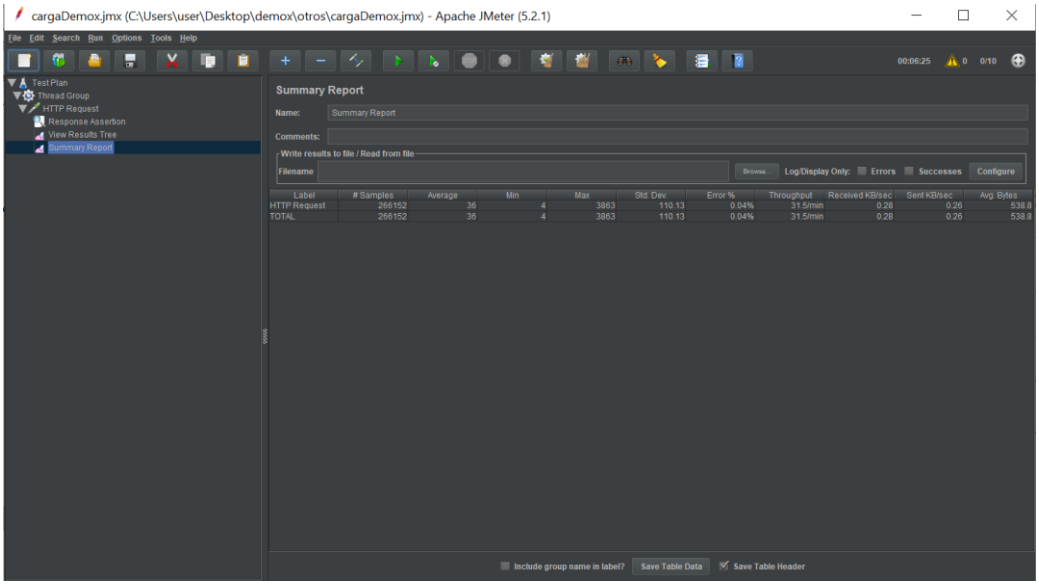

Figura 502. Resultados con peticiones ingresadas y peticiones procesadas

Elaborado por: Diego Chicaiza

Log de operación

Para la validación de las peticiones registradas en el log procedemos a verificar en el log de operación la peticiones registradas y exitosas.

En la Figura 53, se observa los registros de las peticiones realizas en la prueba de carga.

Log de operación

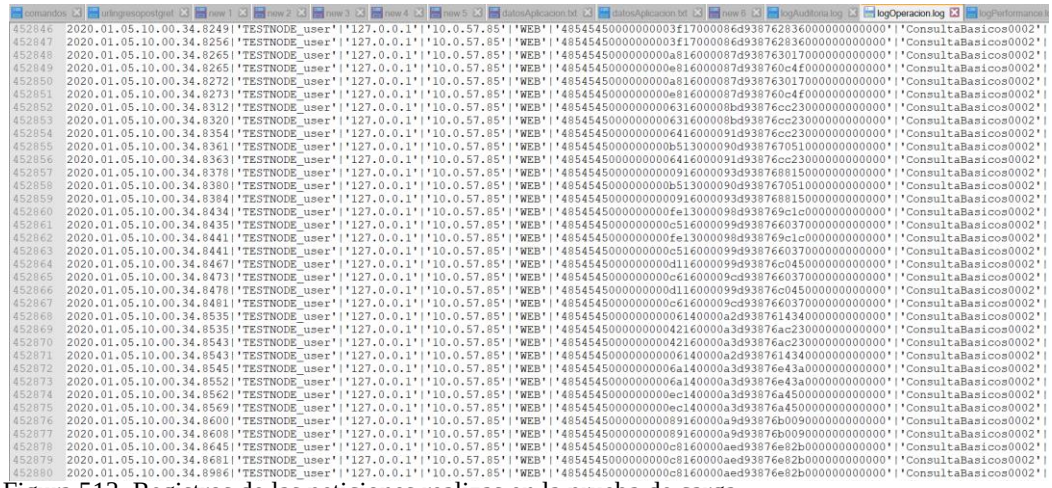

Figura 513. Registros de las peticiones realizas en la prueba de carga

Elaborado por: Diego Chicaiza

Verificación de la ruta del repositorio de log, se valida el tamaño del archivo generado en esta prueba.

En la Figura 54, se observa los archivos generados

Archivos generados

Nombre	Fecha de modificación	Tipo	Tamaño
logAuditoria.log	5/1/2020 10:07	Documento de texto	446,472 KB
logOperacion.log	5/1/2020 10:07	Documento de texto	111,675 KB
logPerformance.log	5/1/2020 10:07	Documento de texto	109,463 KB

Figura 524. Verificación de ruta del repositorio de log

Elaborado por: Diego Chicaiza

El resultado obtenido de esta prueba es satisfactorio tomando en cuenta que esta prueba fue ejecutada de un computador personal no en servidores de ambientes productivos con más recursos, de acuerdo a este documento para obtener una mejor respuesta en una arquitectura de microservicios un factor importante son los recursos físicos hardware que debemos disponer para tener una mejor disponibilidad.

3.7.2. Resultados prueba de stress

Para realizar esta prueba se inició con 100 usuarios y se aumentó a 100 hasta llegar a 900 usuarios de forma masiva en un periodo de 15 minutos.

Figura 55, se puede observar los resultados de las peticiones ingresadas total peticiones procesadas 439751, con una margen de error del 0.09%, el margen de error sube de acuerdo a la cantidad de usuarios ejecutados.

Prueba inicial 100 usuarios

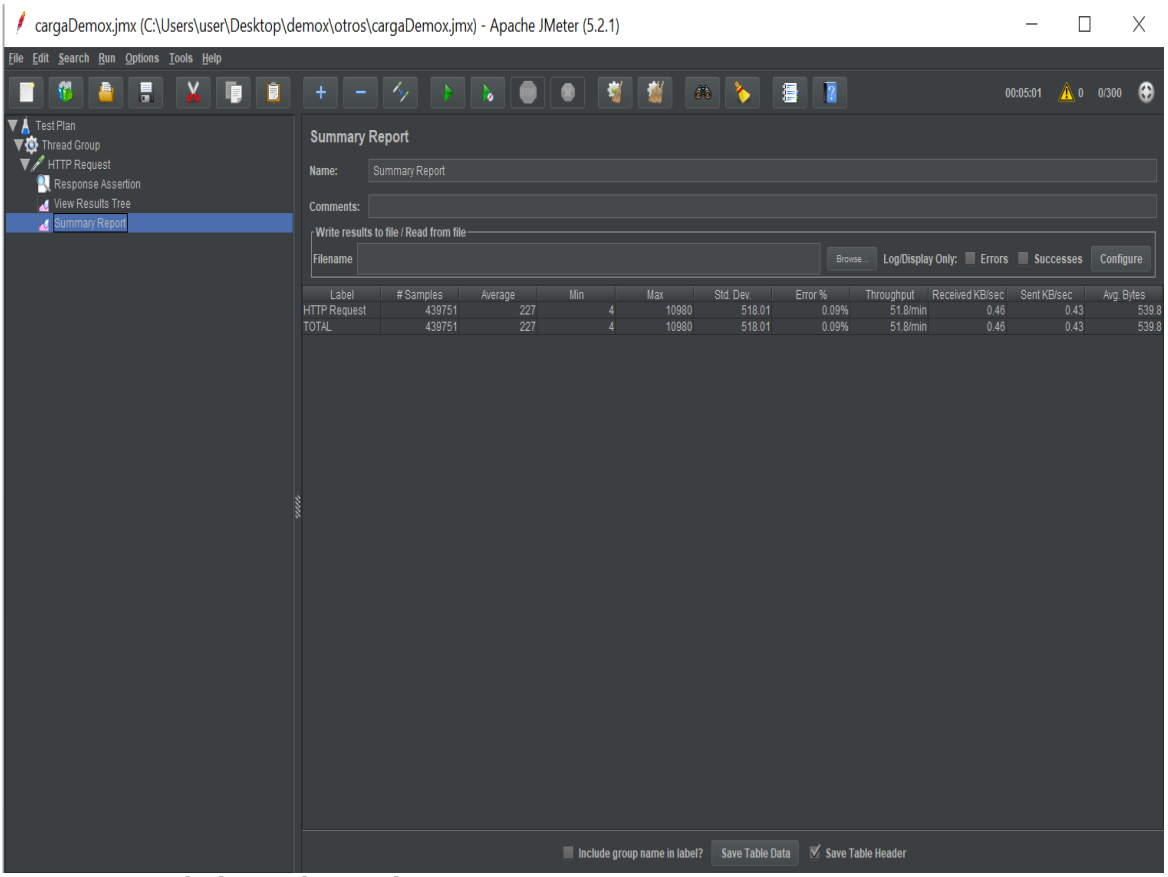

Figura 535. Resultados prueba inicial 100 usuarios

Elaborado por: Diego Chicaiza

En el siguiente gráfico se puede validar que hasta los 900 usuarios se tiene peticiones efectivas.

En la Figura 56, muestra el árbol de resultados con la cantidad de usuarios el sistema responde, se visualiza las tramas de respuesta.

Prueba 900 usuarios

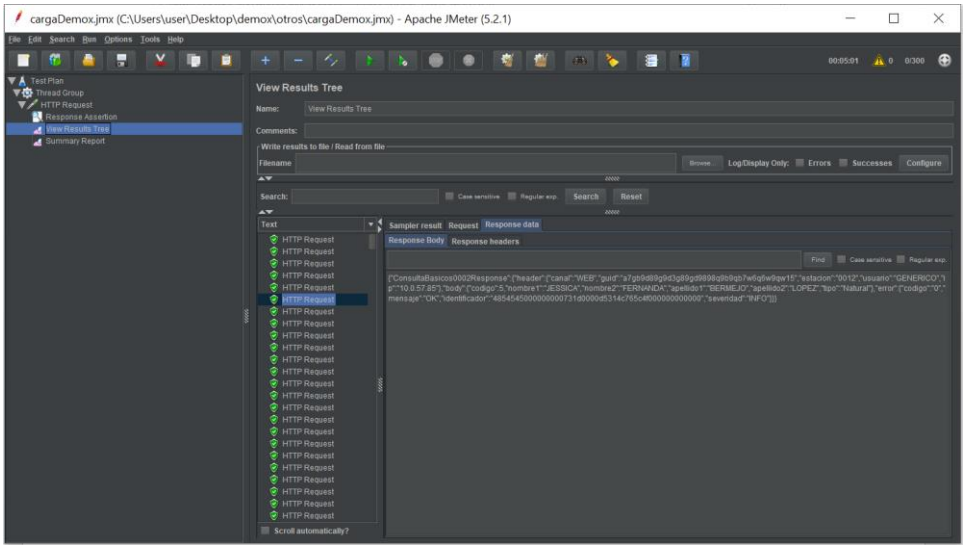

Figura 546. Resultados prueba 900 usuarios

Elaborado por: Diego Chicaiza

En la Figura 57, se valida si una aplicación invocara a los microservicios estos responderían sin ningún problema en el tiempo establecido, tiempo de respuesta 75ms (milisegundos).

Prueba Invocación a microservicios

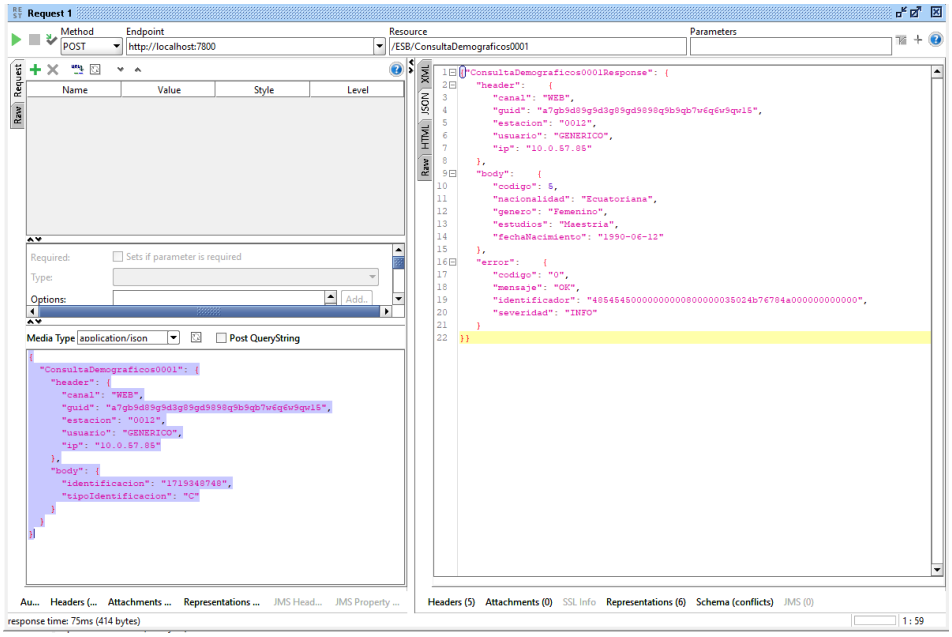

Figura 557. Interfaz de prueba para invocación de microservicios

Elaborado por: Diego Chicaiza

Al subir un bloque de usuario (1000 masivamente) empezamos a tener problemas con la respuesta del microservicio.

En la Figura 58, muestra el árbol de resultados no todas las respuestas son satisfactorias, las peticiones empiezan a fallar por el aumento masivo de usuarios.

Prueba 1000 usuarios

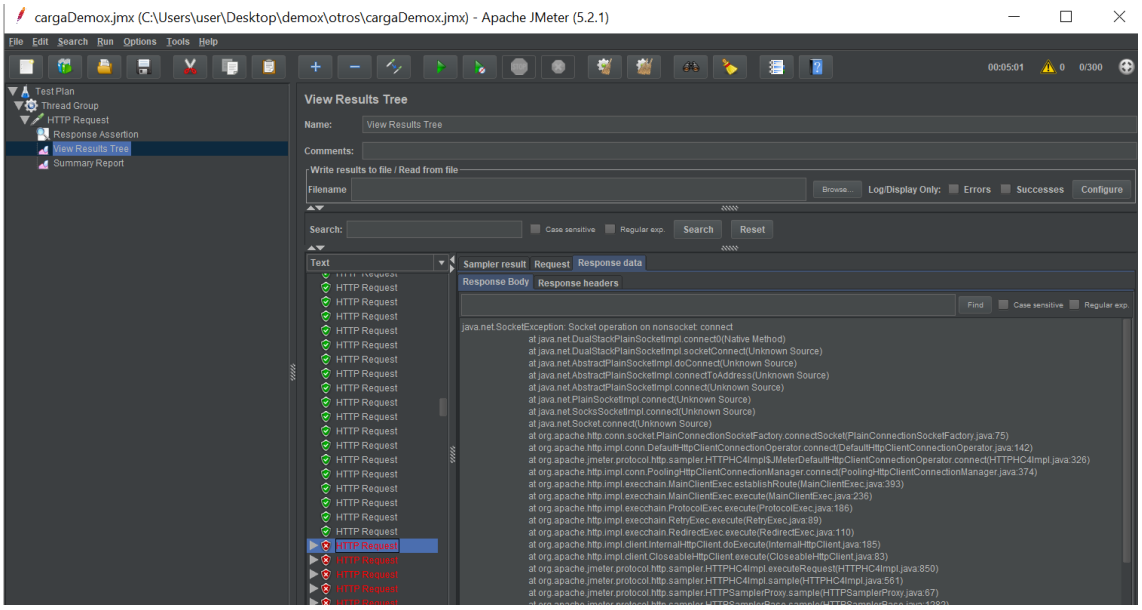

Figura 568. Resultados prueba 1000 usuarios

Elaborado por: Diego Chicaiza

Con este escenario y con el tuning realizado se puede concluir que nuestros microservicios soportan 900 usuarios concurrentes en tiempos óptimos y manteniéndose en el tiempo, se puede mejorar realizando un tuning a la base de datos como se puede visualizar en el detalle hay que afinar el socket de conexión,

Como se puede validar implementar una arquitectura de microservicios es eficiente para transacciones altamente transaccionales, pero esto también exige suficientes recursos para la ejecución a nivel de todos los componentes de la aplicación.

Nuestro alcance esta demostrar la arquitectura en la parte de integración microservicios a nivel de un middleware. Con esta prueba se puede extrapolar para saber que estos microservicios no van a tener problemas de perfomance en ambientes productivos porque estos ambientes cuentan con infraestructura necesaria para responder en tiempos óptimos a las peticiones realizadas por el usuario.

En la Figura 59, se observa el resultado de escenario de pruebas para cada microservicio en el periodo de tiempo establecido, se inicia con un número determinado de usuarios cada uno de ellos con ciertos parámetros establecidos, cada periodo tiene un tiempo de ejecución en cual tendrá un tiempo mínimo de establecimiento el think time (tiempo de espera) se configura de acuerdo a la cantidad de tps (transacciones por segundo) esperados, en las primeras etapas se tiene una efectividad del 100% esto se da porque los periodos de tiempo son muy cortos, cuando se agrega más usuarios en determinado tiempo la efectividad varia en el ejemplo se observa que se tiene un 99% y 98%, esto se da por el tiempo de estabilización en este escenario de pruebas alcanzamos más de 1000 transacciones.

Escenarios de Pruebas

	Escenario				Resultados			
Nombre de Micro Servicio	Concurrencia de Usuarios	Think time / Time Out	Tiempo Establecimiento	Tiempo ejecución	Efectividad	TPS	Tiempo de respueta en segundos	Cantidad Transacciones Exitosas
MSCClientesConsultaDemograficos0001	100	5 S / 15 S	1s / 1min	1 - 2min	100	40	10ms	66538
MSCClientesConsultaDemograficos0001	200	5 S / 15 S	1s / 1min	2- 4min	100	49	10ms	100231
MSCClientesConsultaDemograficos0001	300	15 S / 30 S	1s / 1min	4 - 6 min	99	55	30ms	266152
MSCClientesConsultaDemograficos0001	900	15 S / 30 S	1s / 1min	15 min	98	70	70ms	439751
MSCClientesConsultaBasicos0002	100	5 S / 15 S	1s / 1min	1 - 2min	100	44	10ms	66538
MSCClientesConsultaBasicos0002	200	5 S / 15 S	1s / 1min	2- 4min	100	52	10ms	100231
MSCClientesConsultaBasicos0002	300	15 S / 30 S	1s / 1min	4 - 6 min	99	58	30ms	266152
MSCClientesConsultaBasicos0002	900	15 S / 30 S	1s / 1min	15 min	98	72	70ms	439751
MSCProductosConsulta0001	100	5 S / 15 S	1s / 1min	1 - 2min	100	39	10ms	66538
MSCProductosConsulta0001	200	5 S / 15 S	1s / 1min	2- 4min	100	48	10ms	100231
MSCProductosConsulta0001	300	15 S / 30 S	1s / 1min	4 - 6 min	99	58	30ms	266152
MSCProductosConsulta0001	900	15 S / 30 S	1s / 1min	15 min	98	71	70ms	439751

Figura 579. Resultados del escenario de pruebas

Elaborado por: Diego Chicaiza

En la Figura 60, se observa la cantidad de tps (cantidad de transacción) y el porcentaje de efectividad que arrojo la prueba de carga realizada.

Prueba de carga de microservicio

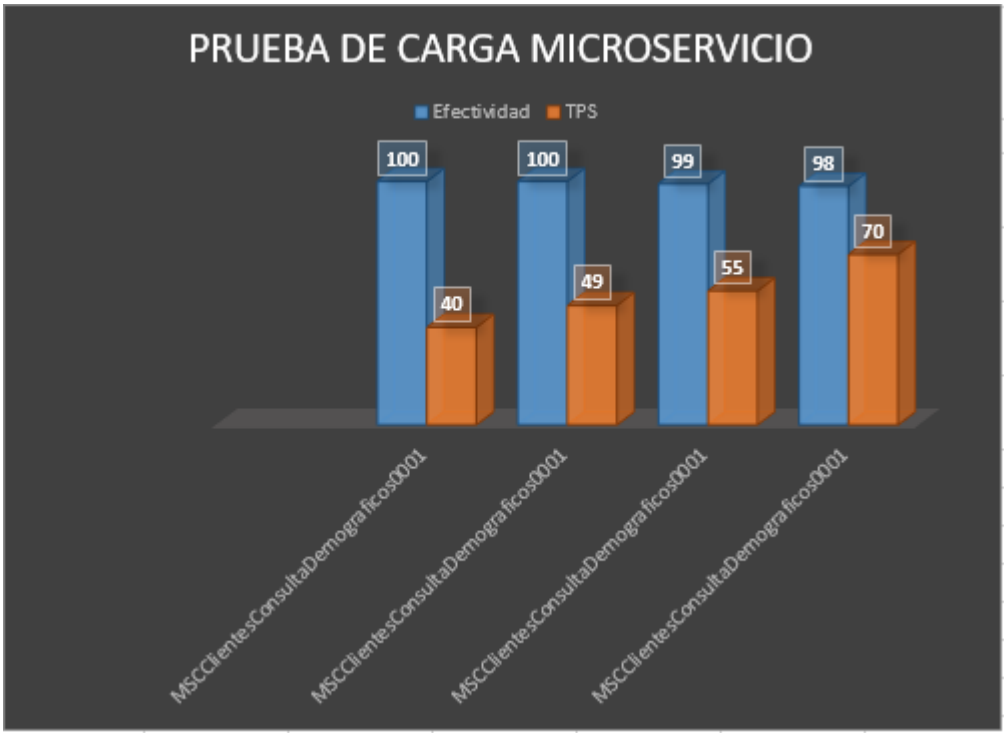

Figura 60. Resultados comparativo de pruebas ejecutadas

Elaborado por: Diego Chicaiza

Estos valores pueden variar de acuerdo de las características del equipo en el cual se realicen las pruebas de carga, los ambientes de desarrollo, test y producción siempre tienen recursos físicos (hardware), diferente.

CONCLUSIONES

- Para caracterizar la arquitectura monolítica y la arquitectura basada en microservicios, se realizó un estudio sobre arquitecturas monolíticas tradicionales, en el que estaba claro que era posible identificar las lagunas que poseían, así como diferentes alternativas que se han desarrollado con el tiempo para llegar al microservicio. En consecuencia, las arquitecturas orientadas a servicios han recibido una excelente recepción, pero la flexibilidad que ofrecen los microservicios puede mejorar el escalado, la estabilidad, la velocidad y otros, dependiendo del contexto de uso.
- Una arquitectura de microservicios bien diseñada y aplicada en el desarrollo de los microservicios ayudaría a la reducción de intermitencias y cuellos de botella que los sistemas informáticos altamente transaccionales, en esta arquitectura los componentes no dependen unos de otros, respondiendo a las peticiones de acuerdo a la lógica de invocación que realice la aplicación.
- Para tener una arquitectura de microservicios completa se debe aplicar a todos los componentes (front, middleware, backend) de un sistema informático, caso contrario la arquitectura estará incompleta no tendremos beneficios de escalado, estabilidad y velocidad, el cuello de botella estará presente en los componentes que no estén en la capacidad de procesar solicitudes en los tiempos definidos, milisegundos dependiendo del escenario.
- El enfoque no es aplicable a todos los sistemas, en realidad se enfoca en sistemas que han crecido y excedido sus límites definidos inicialmente, que requieren alta transaccionalidad y estabilidad, pero, sobre todo, tienen un monolito con una estructura claramente definido, porque sin él no podemos esperar que los resultados sean como nos gustaría.

RECOMENDACIONES

- En cuanto a la etapa de descomposición se recomienda establecer previamente como evoluciono el aplicativo a lo largo del tiempo, ya que existen funcionalidades que pueden ser descartables por falta de uso o, al contrario, funcionalidades que necesiten mayor escalabilidad para determinados escenarios. Un eje clave para esta sección es el modelo de dominio, ya que nos permite definir las funciones del aplicativo y definir un punto de partida para una estrategia de descomposición.
- Se recomienda verificar también no solo la infraestructura física actual en donde corre el sistema, sino también los posibles cambios que se pudieran tener ya planificados, y gestionar la manera de incluir los microservicios de forma incremental para ir controlando el impacto de cambio.
- La propuesta metodológica a través de la arquitectura de microservicios puede llegar a convertirse en un punto de referencia para facilitar una migración partiendo de un monolito (servicio que contiene mucha funcionalidad técnica funcional).
- Además, es importante que los trabajos futuros no solo se enfoquen en el diseño, sino también en el despliegue, ya que, si bien el diseño es importante, la implementación de dicho diseño en código es la única que logra descubrir falencias que quizás no fueron consideradas.
- Finalmente se recomienda realizar investigación en este campo de la ingeniería de software puesto que se encuentra en evolución, y es necesario una constante actualización el modelo propuesto.
- En este documento se dan muchas pautas para iniciar varios temas de investigación relacionados con la arquitectura de microservicios en sistemas alta mente transaccional, enfocada a la banca, redes sociales.

LISTA DE REFERENCIAS

Álvarez, C. (25 de 06 de 2018). *https://www.bbva.com/es/futuro-banca-adaptarse-morir/.* Obtenido de https://www.bbva.com/es/futuro-banca-adaptarse-morir/.

ASOBANCA. (11 de 2010). *La importancia de la profundización financiera y bancarización en el ecuador.* Obtenido de boletín informativo de la asociación de bancos privados del ecuador: https://www.asobanca.org.ec/sites/default/files/noviembre.pdf

ASOBANCA. (2016). *Directorio de la Asociación de Bancos.* Obtenido de INFORME ANUAL DE ACTIVIDADES DE LA ASOCIACIÓN DE BANCOS DEL ECUADOR: https://www.asobanca.org.ec/sites/default/files/Anuarioii.pdf

Emilio Ontiveros Baeza, Á. M. (2012). *LAS TIC Y EL SECTOR FINANCIERO.* Barcelona: Ariel.

Fanjul Suárez, J. L., & Valdunciel Bustos, L. (7 de 11 de 2008). IMPACTO DE LAS NUEVAS TECNOLOGÍAS EN EL NEGOCIO. *Investigaciones Europeas de Dirección y Economía de la Empresa*, 14.

Gartner. (10 de 01 de 2020). *Gartner Glossary*. Obtenido de Gartner Glossary: https://www.gartner.com/en/information-technology/glossary/magic-quadrant

Grubor, S. (2017). *Deployment with Docker.* Birmingham: Packt Publishing.

Gupta, Y. (2015). *Kibana Essentials.* 35 Livery Street Birmingham B3 2PB, UK.: Packt Publishing Ltd.

Hunter, T. (2017). *Advanced Microservices.* San Francisco, California, US: Todd Green.

IBM. (20 de 12 de 2019). *knowledge center*. Obtenido de knowledge center: https://www.ibm.com/support/knowledgecenter/es/SSMKHH_10.0.0/com.ibm.etools.mft.doc/ae67700_.htm#ae67700___entry

Lowe, D. H. (2009). *SmartDraw.* India: Wiley Publishing.

Namit Tanasseri, R. R. (2017). *Microservices with Azure.* BIRMINGHAM - MUMBAI: Packt Publishing .

Newman, S. (2015). *Building Microservices* . United States of America: O'Reilly Medi.

Patricio Letelier, M. C. (15 de 01 de 2006). *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Obtenido de Artículo: http://www.cyta.com.ar/ta0502/v5n2a1.htm#

Posta, C. (2016). *Microservices for Java Developers.* United States of America: Nan Barber and Susan Conant.

Pressman, R. S. (2015). *Software Engineering.* New York: McGraw-Hill.

Richardson, C. (2019). *Microservices Patterns.* Shelter Island: Manning.

Sai Matam, J. J. (2017). *Pro Apache JMeter, Web Application Performance Testing.* Dewas, Madhya Pradesh, India: Sai Matam and Jagdeep Jain.

SERES, G. D. (19 de 01 de 2017). *EDI en el sector financiero: revolución en las transacciones.* Obtenido de EDI en el sector financiero: revolución en las transacciones: http://blog.groupseres.com/edi-en-el-sector-financiero-la-revoluci%C3%B3n-en-la-relaci%C3%B3n-entre-entidades

Tapia, E. (12 de Febrero de 2018). *Transacciones Banca Internet*. (E. Comercio, Editor) Obtenido de Transacciones Banca Internet: https://www.elcomercio.com/actualidad/transacciones-banca-internet-aplicaciones-digitales.html

TOGAF. (2014). *TOGAF® Certification - Level 1 & 2.* The Open Group.

Torre, C. d. (2018). *Microservicios .Net: Arquitectura para Aplicaciones .Net Contenerizadas.* Washington: Microsoft Developer Division, .NET and Visual Studio product teams A division of Microsoft Corporation .

Torres, P. L. (2003). Metodologías Ágiles en el desarrollo de software. *Taller realizado en el marco de las VIII Jornadas de Ingeniería del Software y Bases de Datos, JISBD 2003* (pág. 59). Alicante: Grupo ISSI.

Udagawa, F. (30 de 11 de 2016). *Transformación digital: la evolución de la banca en la vida de las personas y las empresas*. Obtenido de Transformación digital: la evolución de la banca en la vida de las personas y las empresas: https://www.bbva.com/es/transformacion-digital-la-evolucion-la-banca-la-vida-las-personas-las-empresas/

Wire., I. (07 de 07 de 2011). *BIAN anuncia un marco de Referencia para la industria bancaria*. Obtenido de BIAN anuncia un marco de Referencia para la industria bancaria: https://www.estrategiasdeinversion.com/actualidad/noticias/empresas/bian-anuncia-un-marco-de-referencia-para-la-industria-n-142956

Wolff, E. (2017). *Microservices Flexible Software Architecture.* Boston: Pearson Education, Inc.