ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

ARBOLEDA COLA CARLOS AUGUSTO

carlos.arboleda@epn.edu.ec

DIRECTOR: PhD. FLORES NARANJO PAMELA CATERINE

pamela.flores@epn.edu.ec

CO-DIRECTOR: MSc. ANCHUNDIA VALENCIA CARLOS EDUARDO

carlos.anchundia@epn.edu.ec

Quito, noviembre de 2017

DECLARACIÓN

Yo, Arboleda Cola Carlos Augusto, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mi derecho de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Arboleda Cola Carlos Augusto

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Arboleda Cola Carlos Augusto, bajo mi supervisión.

_____

Flores Naranjo Pamela Caterine

DIRECTOR DE PROYECTO

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Arboleda Cola Carlos Augusto, bajo mi supervisión.

_____

Anchundia Valencia Carlos Eduardo

CO-DIRECTOR DE PROYECTO

AGRADECIMIENTO

El sentimiento más profundo de gratitud que puedo tener es a mis padres y hermanos, por las enseñanzas y apoyo incondicional que me supieron dar. Así como la paciencia que han tenido conmigo en los aciertos y desaciertos de mi vida.

De la misma manera he tenido la bendición de conocer seres humanos maravillosos, de los cuales quiero destacar a aquellos que me han acompañado en momentos complicados, ya sea de la universidad, de trabajo, de deporte, o de una simple salida a conversar para brindarme un consejo.

Hago una mención especial a Pamela Flores, directora de este proyecto, que, con su conocimiento, dedicación y apoyo, no he logrado solo culminar este trabajo, sino además me permitió conocer una persona espectacular a la cual admiro y le deseare siempre el mejor de los éxitos.

DEDICATORIA

Dedico este trabajo y los proyectos que aún me propongo por culminar a Hugo y Rosario, mis padres, que han sabido guiarme con su ejemplo para formarme como persona, impulsarme en cumplir mis sueños, y entender la importancia de valores como la perseverancia, honestidad y respeto. Así como a Vico y Damián, mis sobrinos, que llegaron a iluminar nuestras vidas y llenarlas de sonrisas con su existencia.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

La Ingeniería de Software ha buscado desde sus inicios el diseño, construcción y mantenimiento de aplicaciones informáticas, acorde a las necesidades organizaciones y requerimientos técnicos dados por un consumidor. Así pues, nace la Arquitectura Basada en Microservicios como un enfoque para el mejoramiento de arquitecturas tradicionales, centrada en la estabilidad, confiabilidad, alto grado de resiliencia y fácil escalamiento.

Este proyecto plantea el desarrollo de una propuesta metodológica que permita facilitar la migración hacia microservicios enfocada en la descomposición de una arquitectura monolítica tradicional. Para esto se parte de una caracterización de ambas arquitecturas, seguido de un análisis de los patrones de descomposición existentes, y posteriormente se realiza la identificación de los componentes de la arquitectura basada en microservicios. Esto se complementa con la elaboración del primer prototipo *"Scheme for Migration towards Microservices" (SMMicro),* que, si bien no existe un proceso mecánico a seguir para una migración óptima, SMMicro permitirá establecer un punto de referencia al identificar directrices y lineamientos que faciliten una transición de arquitecturas. Finalmente, se valida *SMMicro* sobre un estudio de caso seleccionado de la *"Dirección de Gestión de Información y Procesos"* (DGIP) de la Escuela Politécnica Nacional del Ecuador.

Palabras clave: Ingeniería de Software, Microservicios, Migración, Arquitectura Monolítica.

ABSTRACT

Software Engineering has sought from the outset the design, construction and maintenance of computer applications, according to the needs of organizations and technical requirements given by a consumer. Thus, Microservice Based Architecture is born as an approach for the improvement of traditional architectures, centered on stability, reliability, high resilience and easy scaling.

This project proposes the development of a methodological proposal that facilitates the migration towards micro-services focused on the decomposition of a traditional monolithic architecture. For this, a characterization of both architectures is followed, followed by an analysis of the existing decomposition patterns, and later the identification of the components of the architecture based on microservices. This is complemented by the development of the first SMMicro prototype (Scheme for Migration towards Microservices), which, although there is no mechanical process to be followed for optimal migration, SMMicro will allow establishing a benchmark in identifying guidelines that facilitate a transition of architectures. Finally, SMMicro is validated on a case study selected from the *"Dirección de Gestión de Información y Procesos"* (DGIP) of the National Polytechnic School of Ecuador.

Keywords: Software Engineering, Microservices, Migration, Monolithic Architecture.

INTRODUCCIÓN

Este capítulo se centra en el entendimiento general del proyecto, sus limitaciones, explicar cómo se ha planteado su desarrollo, así como establecer su aporte a la ciencia dentro de la Ingeniería de Software.

Para comenzar debemos tener en consideración, que una metodología posee como objeto de investigación el sistema de procedimientos científicos que orientan de forma lógica al investigador en el hallazgo de los resultados pretendidos [1]. De esta forma en la Ingeniería de Software existen varias metodologías que han sido probadas y analizadas en diferentes escenarios para poder ser validadas y aceptadas como tal. Así pues, en este proyecto se plantea únicamente una propuesta metodológica, enfocada a la migración de arquitecturas de software en sistemas web.

El desarrollo de esta propuesta, parte de un sólido análisis teórico y conceptual de varios estudios realizados sobre Arquitecturas Orientadas a Microservicios, así como su inminente arribo para sustituir a las arquitecturas monolíticas tradicionales. Por ende, se realiza previamente una caracterización de ambas arquitecturas, se detalla las problemáticas existentes, además de establecer una comparativa tomando los puntos más relevantes de *"The Software Architecture Comparison Analysis Method*" (SACAM) [2], que nos proporcionara una base de conocimiento para proceder a analizar patrones de descomposición, sus implicaciones en las funcionalidades del sistema, y posteriormente como asociar servicios de tal manera que la lógica de negocio no se vea afectada.

Esta propuesta metodológica se canalizará a través de un conjunto de actividades y lineamientos propuestos, en el cual están incluidos aspectos importantes sobre el impacto en la organización, descomposición de monolitos, asociación en un ecosistema de microservicios y su correspondiente validación. A su vez, se construye una primera versión del esquema grafico denominado "*Scheme for Migration towards Microservices"* (*SMMicro)* que facilite su comprensión y sirva de eje principal en la transición de arquitecturas, mismo que se convierte en el punto de referencia para trabajos futuros que conlleven la parte experimental con los equipos de desarrollo de un sistema a migrar.

Al culminar el modelamiento de *SMMicro* en conjunto con la propuesta metodológica, serán validados en un estudio de caso, que muestre su eficiencia, posibles correcciones, y recomendaciones para futuras versiones. Se debe destacar que este modelo y por ende su validación, se caracteriza por ser en su mayoría teórico, por lo cual existen implicaciones empíricas que quedan fuera del alcance del proyecto.

CAPÍTULO 1 – ESTADO DEL ARTE

El estudio de diferentes arquitecturas de software se ha convertido en un punto crucial dentro la Ingeniería de Software, es así como en el proceso de continuo mejoramiento se ha llegado a tratar el tema de una Arquitectura Basada en Microservicios. Partiendo de las dificultades que atraviesa el enfoque monolítico tradicional, Villamizar [3] realiza un análisis de la evolución a microservicios a través de Service Oriented Architecture (SOA), así como de estrategias Development and Operations (DevOps), enfatizando la colaboración que debe existir entre el analista de tecnología y el desarrollador de software, lo que conlleva a una serie de desafíos como una nueva cultura de desarrollo, complementado por un conjunto de directrices y buenas prácticas organizacionales.

Por su parte Balalaie [4] además de tratar el tema DevOps se centra en porque utilizar microservicios y los diferentes componentes que conlleva una arquitectura distribuida. De una manera más detallada Salah [5] muestra un análisis claro de cómo ha sido la evolución de este enfoque y los tipos de herramientas que se han venido utilizando, como Remote Procedure Call (RPC), Simple Object Access Protocol (SOAP), Representational State Transfer (REST), entre otros. En este contexto, se trata de forma poco detallada la descomposición de sistemas monolíticos, a diferencia de investigadores como Escobar [6], Levcovitz [7] o Ahmadvand [8] que definen claramente este aspecto como uno de los más relevantes dentro de una migración, tratan varias metodologías de descomposición y algunas problemáticas y limitantes que se pueden encontrar en estudio de casos. Gouigoux [9] por su parte trata estos temas asociado a su experiencia y lecciones aprendidas en diferentes sistemas migrados, que resaltan una selección adecuada de granularidad al descomponer, así como Mosleh [10] enfatiza elegir un nivel adecuado de modularidad.

Otro punto que considerar son los principios que se deben respetar al hablar de una arquitectura basada en microservicios, como lo son: estabilidad, confiabilidad, escalabilidad, tolerancia a fallos, preparación para catástrofes, desempeño y monitoreo. Todos estos tratados por autores como Fowler [11] o Dragoni [12] que analizan el impacto a futuro que podrá tener un traslado a microservicios y por consiguiente los elementos que se deben discutir si se desea un exitoso cumplimiento de objetivos. En este camino es de vital importancia asociar estos principios con los diferentes componentes que estructuran los microservicios, así pues, Bakshi [13], y Sanchez [14] desglosan algunos elementos claves como Discovery Server, Configuration Server, Load Balancing, Circuit Breaker, Edge Server que facilitan la construcción y despliegue de arquitecturas distribuidas.

Finalmente, la arquitectura de microservicios como tal, se encuentra en constante evolución y crecimiento, por ende, cada uno de los autores mencionados limitan y centran su estudio en algún aspecto relevante y procura explotarlo, sin dejar de aportar intrínsecamente en otras perspectivas. Es así como si bien dentro de las metas propuestas para este trabajo, se encuentra la construcción de un esquema que facilite la transición a microservicios, éste se encuentra enfocado en la descomposición de sistemas monolíticos web y los diferentes elemento que implica una migración. Se elabora este trabajo procurando integrar y organizar ideas, técnicas y metodologías ya propuestas, así como validarlo sobre un estudio de caso que nos permita afinar ciertos aspectos empíricos y descubrir nuevas experiencias que complemente nuestro estudio. Así pues, la contribución que se obtiene es significativo en tres aspectos bien definidos:

- Facilitar las decisiones de diseño para transición a microservicios.
- Reducir el número de riesgos de software al migrar a microservicios.
- Mejorar la comprensibilidad de los principales cambios en una descomposición a microservicios.

De igual forma, existen seis ejes sobre los cuales se delimita el alcance de este proyecto, los cuales se los puede apreciar en la Figura 1, tomando en consideración que estos mismos pilares son aquellos tomados como referencia por los principales autores en los últimos años, tal como lo muestra la Figura 2.

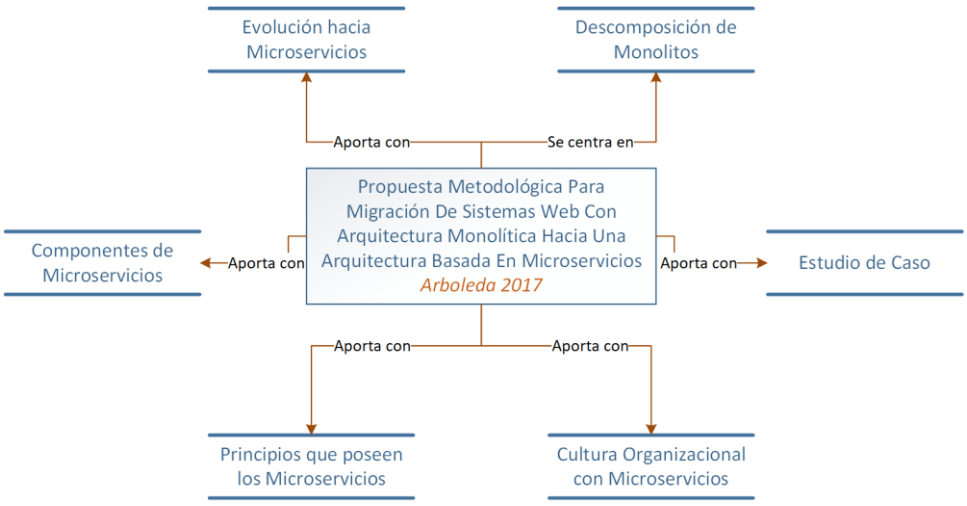


Fig. 1: Ejes principales del Proyecto de Titulación

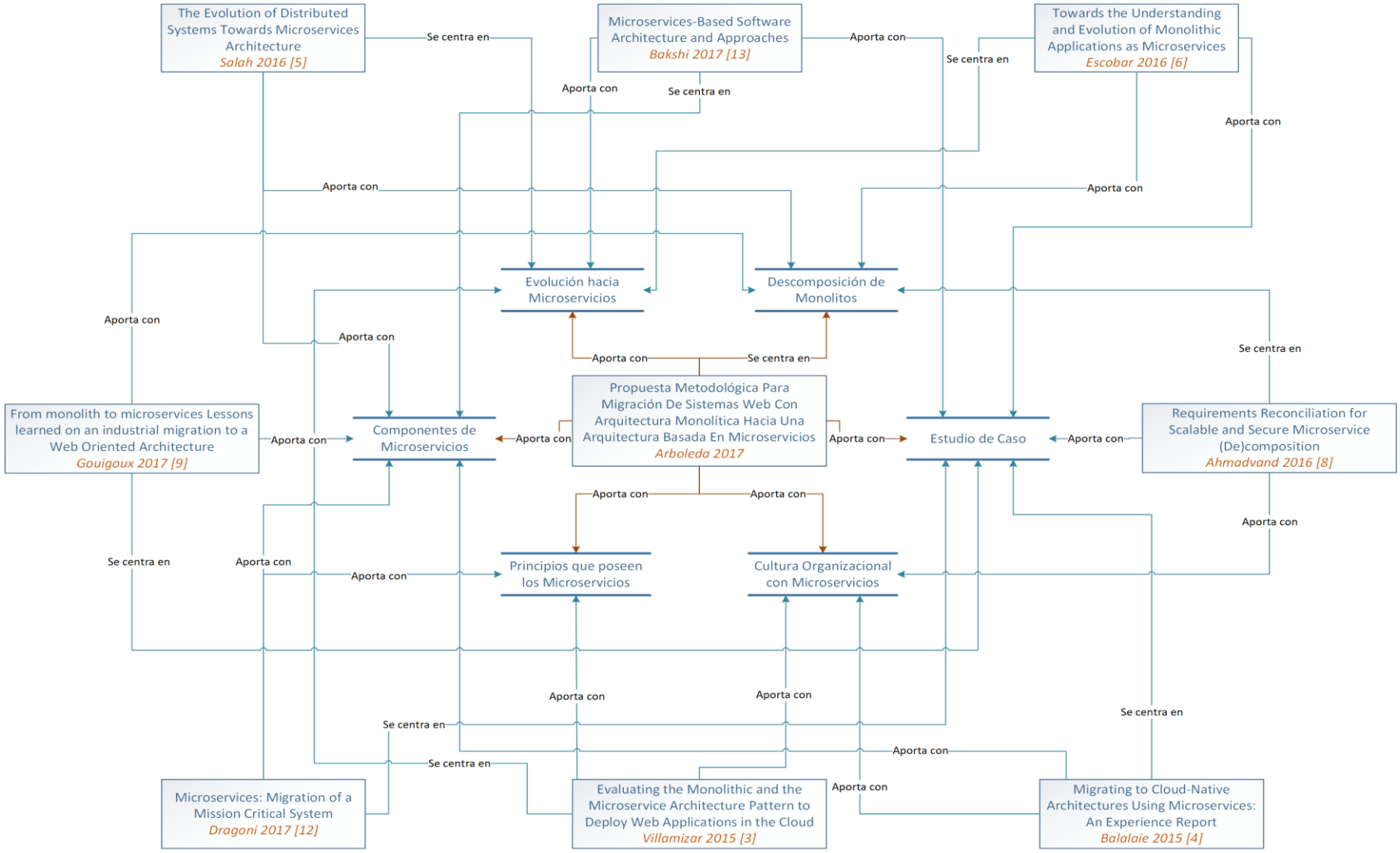


Fig. 2: Resumen de contribuciones relevantes sobre microservicios

CAPÍTULO 2 – MARCO TEÓRICO

Este capítulo tiene como objetivo definir conceptos generales y esclarecer dudas referentes a las arquitecturas de software monolíticas y las basadas en microservicios. Además de caracterizarlas y realizar una comparativa en el contexto que tendrá el proyecto de titulación.

El proceso de ingeniería en el software nos ha permitido el desarrollo de aplicativos a lo largo del tiempo de una manera ordenada, coherente, metodológica y teniendo como uno de sus ejes principales el diseño arquitectónico de construcción de un sistema. Para esto se han utilizado diferentes metodologías y lineamientos asociados con el continuo cambio de necesidades que el cliente demanda especialmente en entornos web. Bass [15], define una arquitectura de software como "*El conjunto de estructuras necesarias para razonar sobre el sistema, que comprenden los elementos de software, sus relaciones y propiedades."*

Aunque la mayoría de sistemas se construyen para que puedan permanecer en operatividad por largo tiempo, [16] menciona que la planificación para un uso productivo a futuro está plagada de incertidumbre debido a factores de mercado, sociales, económicos, y tecnológicos. Cuando los cambios son demasiado costosos o tardan en hacerse, el sistema deja de evolucionar, no cumple sus objetivos de negocio y cae en la obsolescencia. Así pues, existen arquitecturas como la monolítica, distribuida basada en cliente servidor, distribuida basada en agentes móviles, distribuida basada en microservicios, etc. Que han venido acoplándose a requerimientos cada vez más exigentes por parte de los consumidores, y que, dependiendo del entorno, pueden o no resultar exitosas.

2.1. Arquitectura de Software Monolítica

Si bien se entiende un servicio como una pieza de funcionalidad del sistema, que busca optimizar el aplicativo para la máxima utilidad del consumidor, en donde idealmente cada servicio es diferente ya que se optimiza de forma independiente para los casos de uso de sus clientes. En este contexto, una aplicación monolítica web significa un aplicativo con una única base de código, que ofrece una variedad de servicios que utilizan diferentes interfaces tales como páginas HTML, encapsulados a un mismo repositorio [3], como se muestra en la Figura 3.

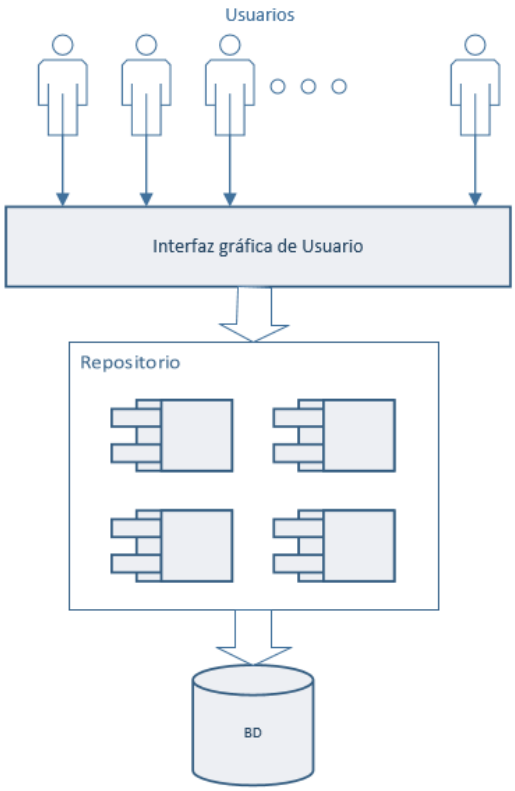


Fig. 3: Ejemplo Arquitectura Monolítica

2.1.1. Inconvenientes en Arquitecturas Monolíticas

Para comprender de mejor manera el motivo por el cual una arquitectura monolítica no siempre es la mejor opción por tomar, se muestra los siguientes inconvenientes:

- Si un servicio o funcionalidad dentro del sistema es demandado en exceso, y este necesita ser escalado, el cambio afectara a todo el conjunto de funcionalidades, produciendo uso de recursos innecesarios.
- Si dentro de un aplicativo existe un solo repositorio de código, concatenado con una única base de datos, al no tratárselos de manera adecuada, estos se vuelven un punto de fallo de único para todo el sistema.
- Si existe un cambio de tecnología continuo, el software se vuelve frágil, complicado de actualizar y por consiguiente propenso a errores.
- Si se requiere implementar reutilización de código eficiente, pueden existir limitantes con otros equipos de desarrollo para mantener un acoplamiento adecuado.

2.1.2. Alternativas a una Arquitectura Monolítica

Lo que se busca es un enfoque para aumentar la flexibilidad, capacidad de respuesta, así como entrega de software continua, separación funcional, reutilización adecuada, resiliencia, entre otros. Entre las principales propuestas que se han planteado a lo largo de los años tenemos, por ejemplo [5]:

- Una arquitectura basada en agentes móviles, en donde un agente es una pieza de software que tiene la capacidad de moverse con su código, datos y estado de un anfitrión a otro, para cumplir una tarea específica. Sin embargo, estaban destinados a ser utilizados en redes lentas, y a menudo fuera de línea.
- Una arquitectura tradicional cliente-servidor que utiliza llamadas de procedimientos remotos, teniendo como principal problema el mantenimiento de actualizaciones frecuentes y sobrecarga en la red.
- Una arquitectura orientada a servicios utilizando SOAP, teniendo como resultado acoplamiento fuerte que dificulta el mantenimiento y corrección de errores. Además, se puede considerar como una arquitectura orientada a servicios, pero de forma monolítica, puesto que no existe una descomposición eficiente dentro del software.

2.2. Arquitectura de Software Basada en Microservicios

RedHat [17] define la Arquitectura de Microservicios como "*Un nuevo estilo arquitectónico para crear servicios de bajo acoplamiento, pero autónomos*", por su parte Martin Fowler [18] menciona que "*El término Arquitectura de Microservicios ha surgido en los últimos años para describir una forma particular de diseñar aplicaciones de software como suites de servicios desplegables de forma independiente*".

Con este tipo de arquitectura se busca descomponer un software complejo en pequeñas aplicaciones independientes comunicándolas con protocolos ligeros, que sean modificables sin afectar el compartimiento general del sistema [19]. De esta manera se entiende que es diferente a la arquitectura tradicional en donde se construía una sola aplicación que lo hacia todo. Teniendo también en cuenta que, como cualquier otra arquitectura, dependerá el entorno y fines para el cual se utilice. Un ejemplo clásico de un sistema descompuesto en microservicio se puede apreciar en la Figura 4.

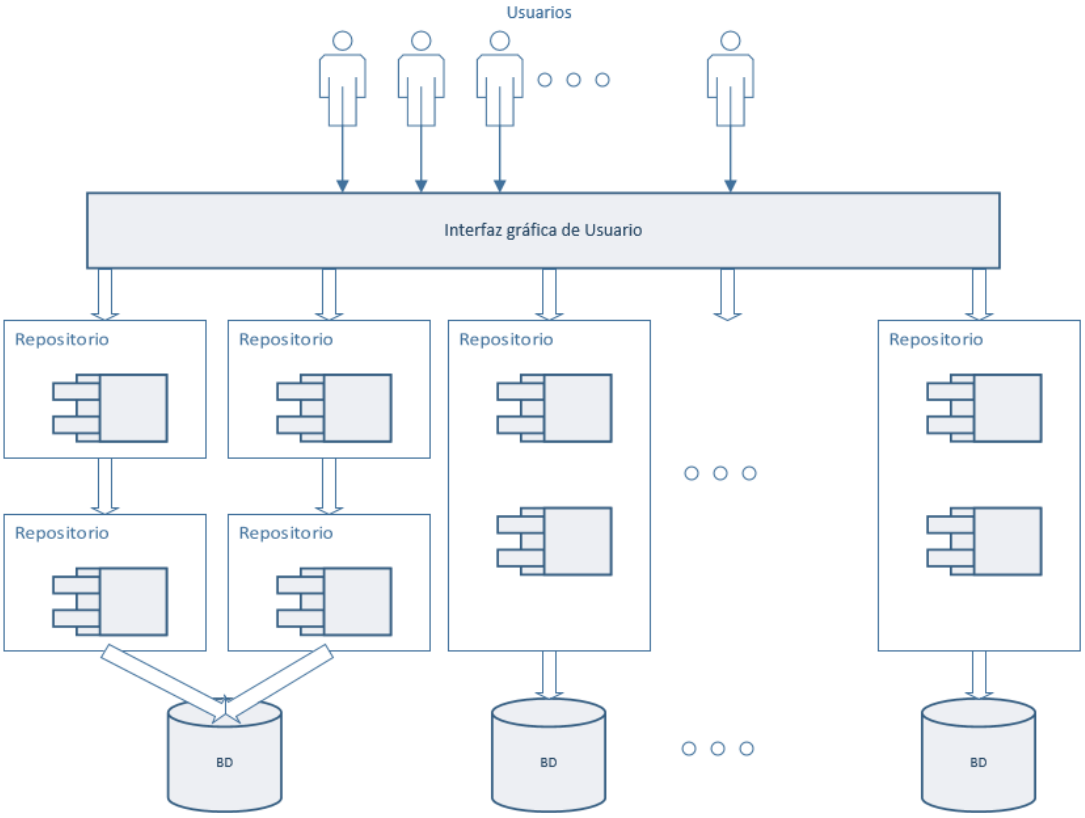


Fig. 4: Ejemplo Arquitectura Orientada a Microservicios

2.2.1. SOA y Microservicios

Al tratar el tema de microservicios siempre existe una relación con una arquitectura netamente basada en servicios, sin embargo, los enfoques son diferentes, aunque esto no quiere decir que el uno sea mejor que otro, todo dependerá cual es el fin de su utilización: SOA (Service Oriented Architecture) se basa principalmente en una orquestación de sus servicios, mientras que la arquitectura basada en microservicios se enfocan en una coreografía de servicios por así decirlo [20]. Sin embargo, los microservicios surgen como una respuesta de mayor flexibilidad y escalamiento horizontal en cuanto al rendimiento general de la aplicación se refiere.

SOA se centra en organizaciones en las que existen equipos de desarrollo para back-end, y por separado equipos de desarrollo para interfaces de usuario, mientras que, en un enfoque basado en microservicios, un equipo debe implementar todo para facilitar la comunicación y así acelerar la implementación de las características. En SOA, una nueva característica puede implicar cambios en numerosos servicios y por lo tanto requieren la comunicación entre un gran número de equipos, mientras que los microservicios tratan de

evitarlo [21]. Si bien a nivel técnico hay puntos comunes como que ambos conceptos se basan en servicios, incluso la granularidad del servicio puede ser similar. Sin embargo, desde puntos de vista conceptuales, arquitectónicos y organizacionales ambos enfoques tienen efectos diferentes. En la Tabla 1 se sintetizan algunas diferencias entre SOA y Microservicios.

	SOA	**Microservices**
Scope	Enterprise-wide architecture	Architecture for one project
Flexibility	Flexibility by orchestration	Flexibility by fast deployment and rapid, independent development of Microservices
Organization	Services are implemented by different organizational units	Service are implemented by teams in the same project
Deployment	Monolithic deployment of several services	Each Microservice can be deployed individually
UI	Portal as universal UI for all services.	Service contains UI

Tabla 1: Diferencias entre SOA y Microservicios [20]

2.2.2. Ventajas de utilización de Microservicios

Existe una gran variedad de características por las cuales se puede elegir una arquitectura basada en microservicios, a continuación, se describe un breve compendio de las razones principales tratadas en [20] y [21]:

Escalamiento. - Cada Microservicio puede escalarse independientemente de otros servicios. Esto evita la necesidad de escalar todo el sistema cuando son sólo unas pocas funcionalidades que se utilizan intensamente. Esto a menudo será una simplificación decisiva.

Resiliencia. - Los microservicios pueden ser reemplazados fácilmente, por ende, recuperarse de un fallo es relativamente sencillo. Un nuevo microservicio no necesita utilizar parte de la base de código ni las tecnologías del antiguo microservicio. Pequeños microservicios facilitan además su actualización y cuando la decisión de una tecnología o

enfoque se limita a un microservicio, este puede ser completamente reescrito si es necesario.

Modularización. - Los microservicios son un concepto de modularización fuerte. Cada vez que un sistema se construye a partir de diferentes componentes de software, las dependencias no deseadas pueden aparecer fácilmente si alguien hace referencia a una clase o función en un lugar donde no se supone que sea usado. Después de un corto tiempo se habrán acumulado tantas dependencias que el sistema ya no podrá ser mantenido o desarrollado. Los microservicios, por el contrario, se comunican a través de interfaces explícitas, utilizando mecanismos como mensajes o Representational State Transfer (REST).

Velocidad. - Los microservicios permiten un mejor tiempo de salida al mercado. Si los equipos que trabajan en un sistema grande son responsables de uno o varios microservicios y si las características requieren solamente cambios a estos microservicios, cada equipo puede desarrollar y traer características en la producción sin el tiempo que consume la coordinación con otros equipos.

De esta manera muchos equipos pueden trabajar en numerosas características en paralelo y traer más características a la producción dentro de un tiempo determinado de lo que habría sido posible con un monolito. Dicho de otra forma, los microservicios son ventajosos para la entrega continua, ya que son pequeños y se pueden desplegar independientemente uno del otro. El despliegue de un único microservicio se asocia con menos riesgo que el despliegue de un gran monolito.

Flexibilidad. - Al desarrollar microservicios no hay restricciones en cuanto al uso de tecnologías. Es posible utilizar tecnologías específicas para funcionalidades específicas, ya sea por ejemplo para la base de datos. Esto reduce el riesgo potencial y permite decisiones tecnológicas independientes para diferentes microservicios. Además, facilita la decisión de probar y evaluar nuevas tecnologías altamente innovadoras. Esto aumenta la productividad de los desarrolladores y evita que la plataforma tecnológica se vuelva obsoleta. Además, el uso de tecnologías modernas atraerá a desarrolladores más calificados.

2.2.3. Contexto de utilización para Microservicios

Se identifican tres enfoques principales en la utilización de microservicios:

- Los microservicios son ideales en aplicativos con gran demanda de peticiones, ya que entre sus principales objetivos esta la alta disponibilidad. Para un sistema con

baja transaccionalidad una arquitectura monolítica puede adecuarse fácilmente a las necesidades del consumidor.

- Al hablar de migración a microservicios se debe considerar que el monolito de partida debe tener una estructura bien definida, no se puede realizar un traslado a una nueva forma de arquitectura si no tenemos un punto de partida claro.
- Los microservicios son ideales para sistemas grandes, es decir sistemas que crecen en tamaño más allá de los límites que se hayan definido inicialmente, ya que se plantean problemas particulares cuando se trata de cambiarlos. En otras palabras, surgen nuevos problemas debido a su escalamiento.

2.3. Comparativa de Arquitecturas

Una vez caracterizadas las arquitecturas monolítica y basada en microservicios, para finalizar este capítulo se procede a compararlas para obtener una perspectiva objetiva del porque es, o no, factible una migración de arquitectura. Así pues, el objetivo de evaluar una arquitectura de software es determinar que satisface los requerimientos de calidad, y asegurarse que la arquitectura seleccionada satisface las necesidades del proyecto.

Hay muchos enfoques para evaluar las arquitecturas de software como: basado en la simulación, basado en escenarios, basado en la experiencia y basado en modelos matemáticos. Algunos de ellos son [22]: SAAM-Scenario-based software architecture analysis method; ATAM-Architecture-based Trade off Analysis Method; ALPSM-Architecture Level Prediction of Software Maintenance; ALMA-Architecture-Level Modifiability Analysis; SBAR-Scenario Based Architecture Reengineering; SALUTA-Scenario-based Architecture Level Usability Analysis; SAAMCS-SAAM for Complex Scenarios; ESAAMI-Extending SAAM by Integration in the Domain; ASAAM-Aspectual Software Architecture Analysis Method; SACAM-Software Architecture Comparison Analysis Method.

Dado que nuestro objetivo es lograr un enfoque comparativo entre dos arquitecturas, se procederá a utilizar los aspectos más relevantes encontrados en SACAM (The Software Architecture Comparison Analysis Method) elaborado por el Software Engineering Institute de la Universidad Carnegie Mellon. Este método fue creado para proporcionar una justificación para un proceso de selección de arquitectura, comparando la aptitud de dos o más candidatos para un sistema requerido [2].

2.3.1. SACAM

Esta metodología se basa por un lado en la extracción de vistas arquitectónicas comparables de cada candidato a niveles de abstracción similares y por otro lado en la clasificación de los criterios y análisis de las arquitecturas candidatas, todo esto estructurado en una serie de etapas o pasos mostrados en la Fig. 5.

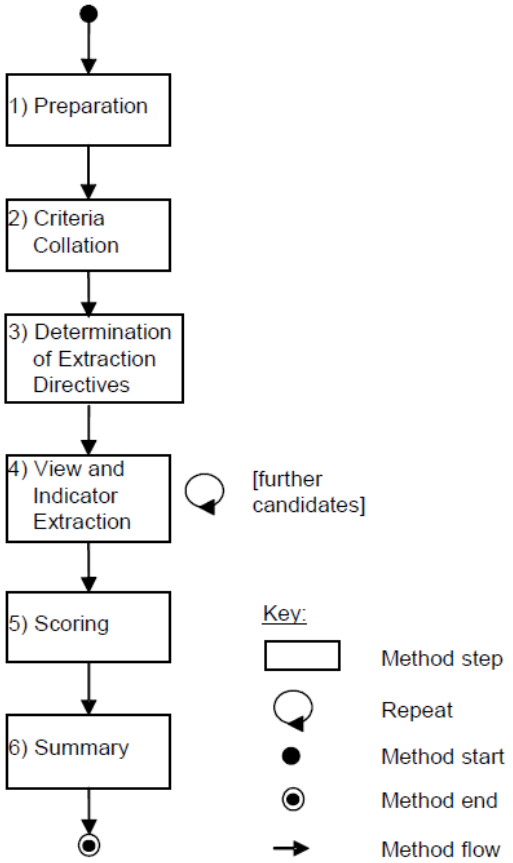


Fig. 5: Método SACAM [2]

Etapa de Preparación

Establece los objetivos de negocio relevantes y los documentos de las arquitecturas de diseño a ser comparados. Se identifican el contexto de la aplicación, las metas del negocio para el sistema previsto, los candidatos de comparación, el alcance y las partes interesadas en la comparativa [2].

Candidatos comparativos

En este caso los candidatos a realizar la comparativa son la Arquitectura Monolítica y la Arquitectura basada en Microservicios, teniendo a consideración que los interesados son

todos aquellos Ingenieros de Sistemas, Ingenieros de Software, Arquitectos de Software, etc., que deseen establecer la diferencia entre mencionadas arquitecturas.

Objetivos empresariales

Los objetivos empresariales varían acorde al enfoque que posea la empresa, sin embargo, al tratar con productos de software se los puede generalizar de manera que equilibren el tiempo, el costo y la calidad con el valor comercial esperado del software producido.

Así pues, Steve Masters [23] de Carnegie Mellon University menciona algunos de los siguientes objetivos empresariales:

- Entregar productos dentro del presupuesto, tiempo y recursos estimados.
- Mejora continua del producto en un porcentaje especificado dentro de un plazo determinado.
- Creciente productividad de equipo de trabajo y fácil adaptabilidad de nuevos miembros.
- Mantener buenas calificaciones de satisfacción con el cliente o consumidor.
- Actualización de servicios manteniendo un rango de costo-beneficio adecuado.
- Adaptabilidad al entorno empresarial y facilidad de reemplazo en caso de ser necesario.

Etapa de clasificación de criterios

Los objetivos de negocio se utilizan para crear un conjunto de criterios de comparación que sirven como el criterio para la comparación global. Los criterios se refinan en escenarios de atributos de calidad [2].
Para definir de mejor manera la priorización de atributos de calidad, utilizaremos la norma ISO/IEC 9126 y el criterio realizado por Senthilkumar y Arunkumar [24], en donde establece lo siguiente:

Atributo de Calidad	*Prioridad*
Funcionalidad	Alta
Fiabilidad	Alta
Usabilidad	Alta
Eficiencia	Media
Mantenibilidad	Media
Portabilidad	Baja

Tabla 2: Prioridad de Atributos de Calidad [24]

Una vez definida la prioridad para los atributos de calidad, se procede a relacionarlos con los objetivos empresariales definidos en el paso anterior. Cada objetivo empresarial puede corresponder a uno a varios atributos de calidad, sin embargo, la prioridad global del criterio define cual será el valor cualitativo (Alta, Media, Baja) que se utilizará en el resto de la metodología.

En la tabla 3. Podemos observar la clasificación de criterios que deriva de los objetivos de negocio.

		Atributos de calidad relacionados	***Prioridad Global del Criterio***
1	Entregar productos dentro del presupuesto, tiempo y recursos estimados.	Funcionalidad, Eficiencia	Alta
2	Mejora continua del producto en un porcentaje especificado dentro de un plazo determinado.	Funcionalidad, Fiabilidad	Alta
3	Creciente productividad de equipo de trabajo y fácil adaptabilidad de nuevos miembros.	Eficiencia, Funcionalidad	Alta
4	Mantener buenas calificaciones de satisfacción con el cliente o consumidor.	Usabilidad, Portabilidad	Media
5	Actualización de servicios manteniendo un rango de costo-beneficio adecuado.	Mantenibilidad, Eficiencia	Media
6	Adaptabilidad al entorno empresarial y facilidad de reemplazo en caso de ser necesario.	Funcionalidad, Portabilidad	Media

Tabla 3: Clasificación de criterios SACAM

Etapa de determinación de directivas de extracción

Genera características que proporcionan evidencia a favor o en contra de los escenarios de atributos de calidad del paso 2 en base a características arquitectónicas y a su vez directivas que permitan identificar indicadores para el paso 4 en donde se comparan los candidatos [2].

Candidato Arquitectura Monolítica

- El contexto de la arquitectura monolítica a pesar de ser el tipo predominante de arquitectura actual requiere un compromiso a largo plazo para un stack de tecnología en particular. Por ejemplo, para probar un nuevo marco de

infraestructura, a veces es posible que necesite volver a escribir el código de toda la aplicación.

En la mayoría de los casos, se tendera utilizar el stack de tecnología existente durante todo el ciclo de desarrollo. Además, escalar partes "lógicas" de una aplicación monolítica requiere recursos significativos (en términos de presupuesto, tiempo, etc.). Normalmente, el único camino disponible es escalar la aplicación completa verticalmente actualizando o comprando nuevo hardware, migrando a una infraestructura más eficiente, y así sucesivamente [25].

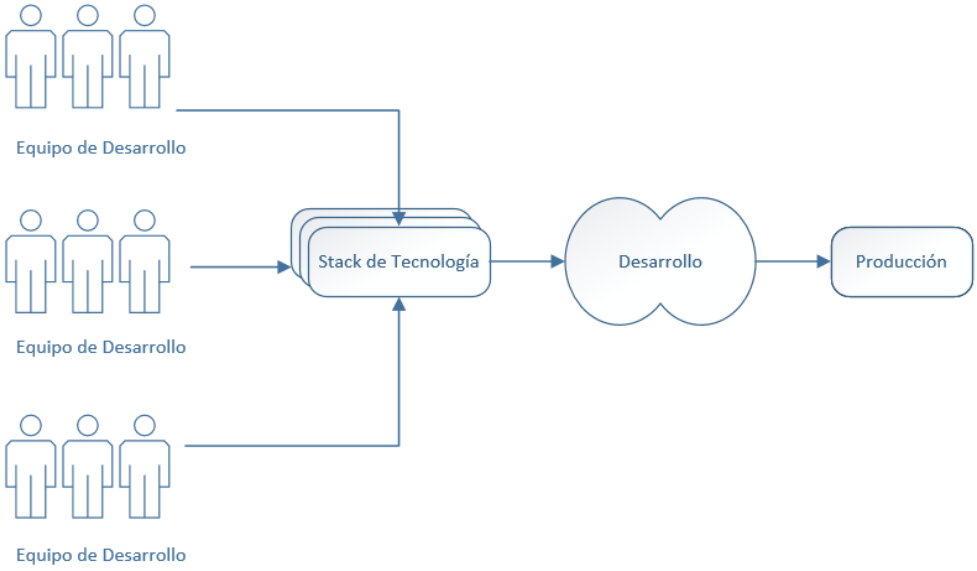


Fig. 6: Dependencia Stack Tecnológico Monolítico

- Por otro lado, siempre existirán equipos trabajando en diferentes características nuevas. Sin embargo, el trabajo paralelo es complicado: la estructura del software no lo soporta realmente. El back-end, por ejemplo, no puede funcionar realmente sin obtener la entrada de la base de datos, y la interfaz de usuario no puede funcionar sin la entrada desde el back-end.

  Cuando los equipos trabajan en sprints, estas dependencias causan retrasos: el equipo de base de datos genera en su primer sprint los cambios necesarios, dentro del segundo sprint el equipo de back-end implementa la lógica y en el tercer sprint se trata la interfaz de usuario. De esta manera se necesitan tres sprints para implementar una sola característica [20]. El trabajo de los diferentes equipos tiene que ser coordinado de tal manera que sus esfuerzos encajen temporalmente entre sí.

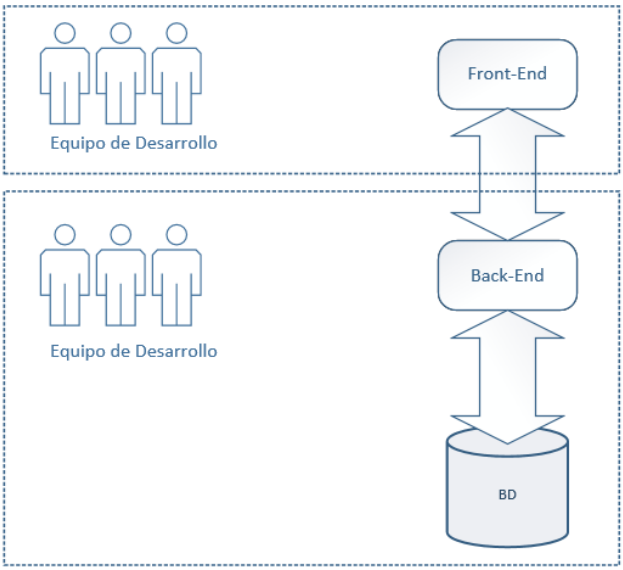


Fig. 7: División de Equipos Arquitectura Monolítica

- Otro aspecto por considerar es que en esta arquitectura comúnmente la aplicación web publica los servicios de los servicios REST a través de Internet, que son consumidos por la aplicación MVC front-end ejecutada en el navegador. La arquitectura monolítica se desplegaría en una solución Cloud utilizando el despliegue ilustrado en la Figura 8. Para escalar la aplicación, la arquitectura requiere el uso de un equilibrador de carga, varios servidores web donde se despliega la base de código de la aplicación y una base de datos relacional para almacenar Información.

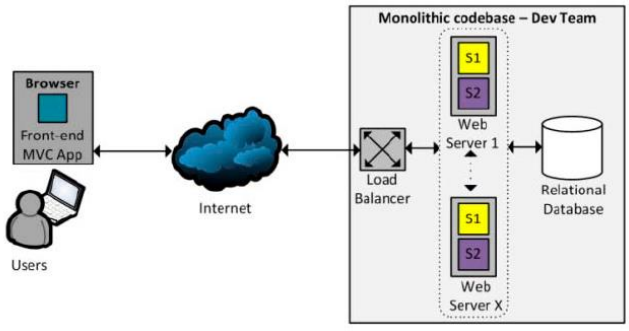


Fig. 8: Despliegue Arquitectura Monolítica [3]

Candidato Arquitectura Basada en Microservicios

- Al tratar con microservicios la línea de desarrollo es diferente para cada equipo de trabajo, por ende no existe una dependencia con una sola pila de tecnológica, se puede utilizar diferentes frameworks en base a la necesidad del servicio que se brinda, así pues la fase de despliegue y prueba se lo realiza para cada microservicio independientemente de los demás, cabe recalcar que para lograr esto debe existir posteriormente una buena capa de integración que permita consumir todo el aplicativo para facilidad del cliente.

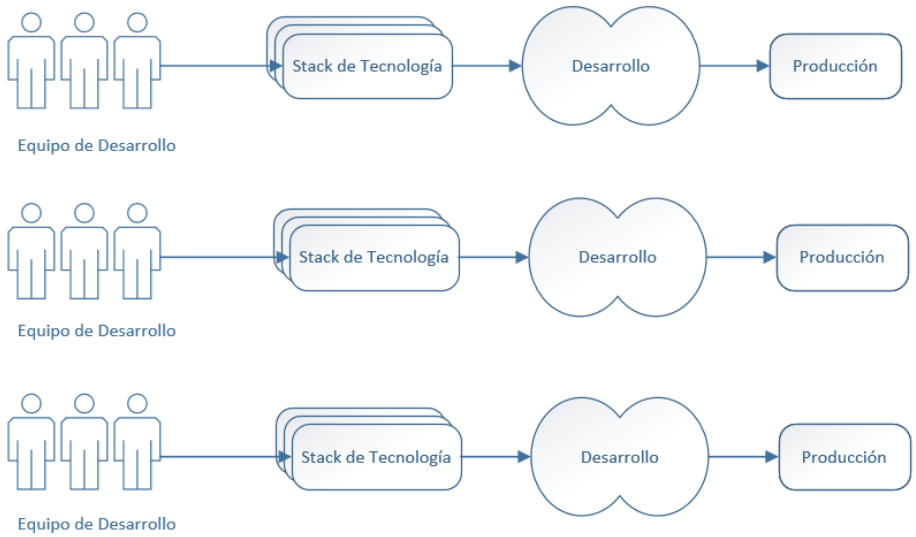


Fig. 9: Independencia Stack Tecnológico en Microservicios

- Este enfoque corresponde con la idea de los llamados equipos interfuncionales, tal como lo proponen los métodos con Scrum. Estos equipos deben abarcar diferentes funciones para que puedan cubrir un amplio espectro de tareas. Sólo un equipo diseñado a lo largo de estos principios puede estar a cargo de un componente desde los requisitos de ingeniería hasta la implementación y operación. La división en artefactos técnicos y la interfaz entre los artefactos se puede resolver dentro de los equipos [20]. Aunque en este tipo de arquitectura los cambios se pueden introducir más rápido que en el monolito de despliegue, produciendo una gobernanza descentralizada momentánea eficiente, se debe tener en consideración que si no se realiza una separación adecuada de microservicios no existirá realmente una independencia.

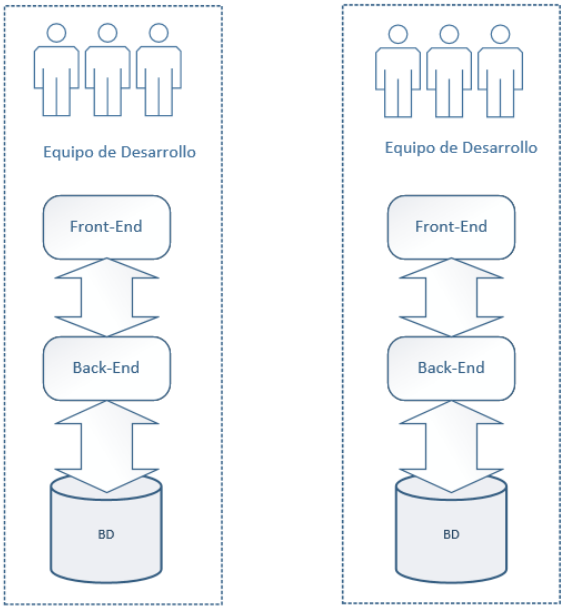


Fig. 10: División de equipos en Microservicios

- Finalmente, para un escalamiento de esta arquitectura, la puerta de enlace o capa de integración que se haya utilizado y cada microservicio pueden escalarse de forma independiente. El microservicio uS1 se despliega utilizando un equilibrador de carga y varios servidores web. El microservicio uS2 también se implementa utilizando un equilibrador de carga y varios servidores web, pero también utiliza una base de datos relacional (o si se desea no relacional) para almacenar información. La puerta de enlace se despliega utilizando un equilibrador de carga y varios servidores web que bien pueden ser alojados en servicios cloud como IaaS (Infrastructure as a service).

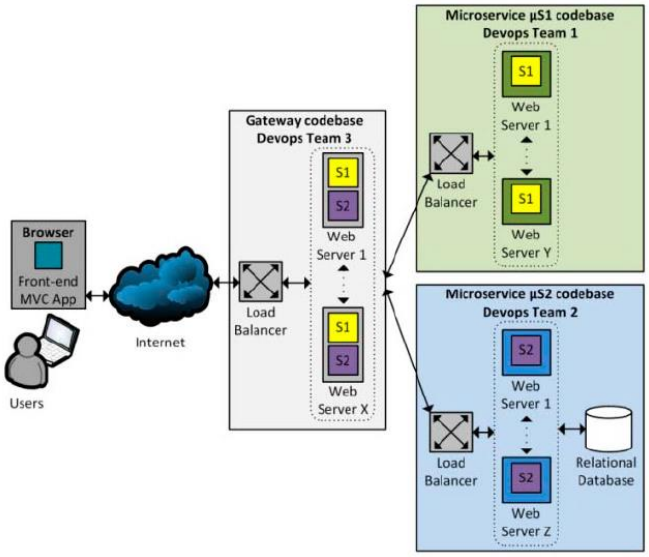


Fig. 11 Despliegue Arquitectura Microservicios [3]

Etapa de extracción de la vista y del indicador

Los tipos de vista seleccionados del paso 3 tienen que ser extraídos y refinados hasta que ofrezcan un nivel apropiado de detalle para una comparación de arquitectura. Procurar detectar indicadores que soportan los escenarios de atributos de calidad del paso 2 [2].

Para establecer los indicadores se toma como referencia información adicional sobre la evolución hacia sistemas basados en microservicios de Salah [5], y la caracterización realizada previamente en la tercera etapa de SACAM.

		Monolítico	***Microservicios***
1	Secuencia de desarrollo multifuncional	No	Si
2	Stack de desarrollo dinámico	No	Si
3	Fase de despliegue y pruebas no limitado	No	Si
4	Equipos de desarrollo interfuncionales	No	Si
5	Comunicación en su mayoría ligera	No	Si
6	Gobernanza descentralizada	No	Si
7	Voluntaria separación de servicios	Si	No
8	Balanceador de carga	Si	Si
9	Monitorización de intercomunicación opcional	Si	No
10	Gateway para integración	Si	Si
11	Utilización de cortacircuitos de fallos	No	Si
12	Funcionalidad independiente de red	Si	No

Tabla 4: Indicadores SACAM

Etapa de puntuación

Proporciona a cada candidato a la arquitectura una puntuación que indica qué tan bien soporta los escenarios de atributos de calidad generados en el taller de clasificación de criterios de la Etapa 2 [2]. Para califica la aptitud de las arquitecturas candidatas se valorar sobre un peso máximo de 1.5 en caso de ser Buena, 1 en caso de ser Media y 0.5 en caso de ser baja.

		Monolítico	*Microservicios*	*Justificación*
1	Entregar productos dentro del presupuesto, tiempo y recursos estimados.	1.5/1.5	1.5/1.5	Ambas arquitecturas cumplen adecuadamente la entrega ya que esto depende de la planificación inicial del proyecto.
2	Mejora continua del producto en un porcentaje especificado dentro de un plazo determinado.	1/1.5	1.5/1.5	Para lograr una mejora continua en la arquitectura monolítica se debe hacer un cambio general de todo el aplicativo, mientras que en microservicios no.
3	Creciente productividad de equipo de trabajo y fácil adaptabilidad de nuevos miembros.	1/1.5	1/1.5	Si bien el enfoque utilizado en monolítico no es la ideal, en microservicios aún se debe mejorar también varios aspectos para una adaptación de nuevos desarrolladores.
4	Mantener buenas calificaciones de satisfacción con el cliente o consumidor.	1/1	1/1	Ambas arquitecturas mantienen la misma satisfacción puesto que su mayor diferencia se encuentra en back-end y no en front-end.
5	Actualización de servicios manteniendo un rango de costo-beneficio adecuado.	0.5/1	0.5/1	Microservicios mejora costos de mantenimiento, sin embargo, al iniciar un proyecto monolítico no necesitamos pagar por los servicios cloud que si consume microservicios.
6	Adaptabilidad al entorno empresarial y facilidad de reemplazo en caso de ser necesario.	0.5/1	1/1	La arquitectura microservicios posee mejor capacidad de reemplazo debido a su independencia de servicios.

Tabla 5: Puntuación SACAM

Resumen

Proporciona un resumen de los resultados, junto con recomendaciones específicas, se toma en cuenta los escenarios de atributos de calidad del paso 2, indicadores arquitectónicos extraídas desde el paso 4, puntuación y razonamiento desde el paso 5 [2].

Resumen Datos Obtenidos en SACAM

Se realizó una comparación en términos de las características más deseadas que se encuentran relacionadas con los objetivos de negocio para una empresa enfoca en software. En donde se

puede concluir que una arquitectura monolítica no posee dificultades en realizar un aplicativos de manera eficiente, e inclusive en cuanto a consumo de recursos se refiere, una arquitectura monolítica no requiere la misma cantidad de componentes que evidencia la arquitectura distribuida. Así pues, la mejora continua de un sistema es quizás la mayor ventaja en cuanto a microservicios, junto a la adaptabilidad, puesto que la construcción de cada microservicio se lo hace de forma independiente y con un equipo de desarrollo que se centra en el requerimiento del cliente y como se puede emplear inclusive un stack tecnológico diferente para cada servicio.

Dentro del análisis realizado se debe resaltar además las desventajas encontradas para la utilización de microservicios, entre ellas: un riguroso cuidado al momento de realizar una descomposición de microservicios, puesto que es el punto de partida de la arquitectura; también el hecho de que siempre se encontrara resistencia al cambio dentro de una empresa cuando el equipo de desarrollo probablemente ya se encuentra adecuado a una metodología de trabajo; además lo que implica un uso mayor de recursos y costos a comparación de los monolitos. Aunque hoy en día este último punto es relativamente fácil manejar gracias al uso de servicios cloud como IaaS (Infrastructure as a service), PaaS (Platform as a service) y SaaS (Software as a service) que detallaremos más adelante.

A pesar de que las ventajas e inconvenientes de los microservicios y monolitos fueron tomadas en cuenta para mostrar una caracterización objetiva, aún quedan consecuencias conocidas y desconocidas dependientes de casa sistema, puesto que como ya se mencionó, el tamaño del aplicativo influye considerablemente en cuanto a microservicios se refiere, por ende, es importante analizar antes si la aplicación se beneficiara de escalabilidad, integridad, resiliencia y agilidad. Finalmente utilizando el método SACAM se recomienda analizar la factibilidad de definir directrices de migración de una arquitectura monolítica hacia microservicios, puesto que si representa una mejora significativa para empresas dedicadas a la producción de software.

CAPÍTULO 3 - PROPUESTA METODOLÓGICA

En este capítulo se desarrolla la propuesta metodológica que definirá las directrices para una migración de una arquitectura monolítica hacia una arquitectura basada en microservicios, tomando como referencia el análisis teórico realizado en los capítulos previos. Así como la construcción del esquema SMMicro, ya que si bien no existe un proceso mecánico a seguir, hay varias estrategias definidas por autores que facilitan la extracción de lineamientos de transición que trataremos a lo largo de este capítulo.

3.1.Migración de Arquitectura Monolítica a Microservicios

Es importante recalcar que, desde el punto de vista del sistema de microservicio, el diseño organizacional incluye la estructura, dirección de autoridad, granularidad y composición de los equipos. Un buen diseño de sistemas de microservicio abarca las implicaciones de cambiar estas propiedades organizacionales y que el buen diseño de servicios es un subproducto del buen diseño organizacional. Así como también es importante analizar cómo se encuentra estructurado el aplicativo previo a una migración.

3.1.1. Impacto Organizacional y análisis inicial

En el artículo de 1967 titulado "How Committees Invent", el informático Melvin Conway argumentó que el diseño de un sistema de software imitará la estructura de comunicación de la organización que la produjo. "La ley de Conway", como llegó a ser conocido, ha tenido un renacimiento reciente en círculos de microservicios. La mayoría de los pioneros de la arquitectura de microservicios comenzaron su búsqueda de una entrega de software más rápida optimizando el diseño organizacional antes de abordar la arquitectura del software. Dada esta progresión, estas organizaciones aterrizaron en la arquitectura de microservicios como estilo que alineó con sus equipos pequeños alineados negocio, revelando así la sabiduría de la afirmación de décadas de Conway [21].

Por ende, es crucial examinar la estructura organizativa y el estado actual del sistema a migrar, para aplicar una estrategia de transformación se puede:

- Analizar el estado y documentación del sistema a migrar en relación con la organización.
- Identificar áreas problemáticas en su diseño organizacional.
- Identificar transformaciones seguras (cambios que no cambian el comportamiento existente).

3.1.2. Descomposición de sistemas de software monolíticos

Las dificultades de dividir un monolito en microservicios dependen enteramente de la complejidad de la aplicación monolítica. Una aplicación monolítica con muchas características requerirá gran esfuerzo arquitectónico y deliberación cuidadosa para descomponerse con éxito en microservicios, y la complejidad adicional se introduce por la necesidad de reorganizar y reestructurar equipos. La decisión de pasar a microservicios debe convertirse siempre en un esfuerzo de toda el área de desarrollo.

Existen varios pasos para fragmentar un monolito, entre estos, identificar los componentes que pueden ser utilizados como servicios independientes, así pues, autores como Eberhard Wolff [20] y Chris Richardson [26] recomiendan seleccionar las principales funcionalidades generales del monolito y posteriormente dividir esas funcionalidades en pequeños componentes independientes. Sin embargo, existen otras propuestas como descomposición por contexto, madurez o acceso a datos, cada una de ellas empleadas de tal forma que respete que los microservicios deben ser tan simples como sea posible, o bien, la empresa se arriesga a reemplazar un software monolítico con varios monolitos más pequeños.

Descomposición por funcionalidad

Esta primera directriz es quizás la más tradicional en cuanto a microservicios se refiere, destaca la división del dominio de un problema grande en las funcionalidades y características que ofrece. Esto dependerá de los requerimientos que hayan sido definidos para el sistema, se identifican las piezas individuales de funcionalidad que en general proporciona el aplicativo, y se las agrupa en paquetes, verificando que la lógica de negocio no haya sido afectada [27]. Algunos de estos candidatos a convertirse en un servicio independiente pueden por ejemplo ser: el servicio de registro de clientes, servicio de compra y venta, servicio de devoluciones, etc.

Netflix [28] adopta un enfoque basado en funcionalidad para dividir su sistema en servicios. De hecho, se podría argumentar que están siguiendo este enfoque hasta el extremo, en el sentido de que tienen una guía para tener una sola funcionalidad por servicio, como recomendar una lista de películas basadas en un usuario dado y sus calificaciones para diferentes películas.

Descomposición por madurez

Analizar el nivel de madurez es otra manera para identificar secciones candidatas de una aplicación a convertirse en microservicios. Por lo general al hablar de organización empresarial tenemos que tener en cuenta que ya debieron existir varias iteraciones de definición de objetivos y especificaciones para el aplicativo. Las partes cuyos requerimientos funcionales y no funcionales son estables y bien comprendidos deben ser agrupadas, mientras que las cosas que son menos estables y sujetas a iteraciones más y más rápidas podrían pasar a otros servicios separados versionados. En otras palabras, se podría considerar agrupar servicios que se encuentren estables a lo largo del tiempo como por ejemplo registro de usuarios, y separar aquellos que no varían con el tiempo y la demanda de usuario como el servicio de compra y venta.

Descomposición por patrón de acceso a datos

Para poder analizar este tipo de descomposición debemos tener en cuenta la eficacia de la persistencia y las entidades sensibles por número de relaciones que posean. Carneiro [27] recomienda analizar el acceso de almacenamiento esperado para las distintas funcionalidades y clasificarlas de acuerdo con "lectura intensiva", "escritura intensiva" y "lectura y escritura balanceadas". La ventaja de este enfoque es que los esquemas de un servicio, las consultas de acceso o incluso el motor de almacén de datos (ya sea un relacional, no relacional, almacén de valores clave, basado en documentos u otro método) se pueden optimizar para el acceso de lectura o escritura intensiva en aislamiento. De forma similar, se pueden aplicar diferentes estrategias de almacenamiento en caché de datos para cada uno de los servicios. Por ejemplo, uno podría decidir no introducir la complejidad relacionada con el almacenamiento en caché si un servicio se ocupa principalmente de datos de escritura intensiva.

Descomposición por Contexto

Por último, una descomposición por contexto puede ser una solución si encontramos servicios que no se definan igual en diferentes contextos, esto dependerá del nivel de complejidad del sistema analizado, puesto que, en un sistema con funcionalidades bien definidas para cada actor, siempre contendrá atributos que varíen su función para determinados escenarios.

En el artículo del experto en microservicios Martin Fowler *“Bounded Context”* [29] explica que, por ejemplo, un cliente puede ser polisémico, es decir un cliente puede significar dos cosas diferentes en diferentes contextos. Los atributos y la lógica de negocio asociados

con un cliente diferirán grandemente para la unidad de la atención al cliente que para el contexto del negocio de la unidad de ventas. Tal separación podría ir tan lejos que los únicos datos compartidos entre tales servicios centrados en el cliente serían un identificador de cliente único. Ambos servicios de cliente específicos de dominio también podrían hacer referencia cruzada a un tercer servicio de cliente de nivel inferior para datos compartidos como la dirección del cliente u otra información de contacto.

Tipos de Descomposición

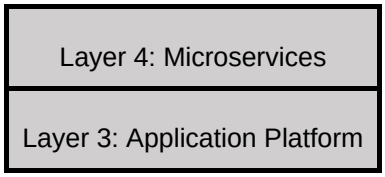
Por Funcionalidad	Por Madurez	Por Acceso a Datos	Por Contexto
Se analiza: piezas individuales de funcionalidad que en general proporciona el aplicativo. • Complejidad estimada: Media	Se analiza: Partes cuyos requerimientos funcionales y no funcionales son estables y aquellos que no lo son. • Complejidad estimada: Media	Se analiza: La eficacia de la persistencia y las entidades sensibles por número de relaciones que posean. • Complejidad estimada: Alta	Se analiza: Servicios que no se definan igual en diferentes contextos. • Complejidad estimada: Alta

Fig. 12: Tipos de Descomposición

3.1.3. Ecosistema de microservicios

Se considera como ecosistema de microservicios al entorno en donde serán construidos y desplegados, esto debido a que los microservicios se extraen de toda una infraestructura que incluye principalmente hardware, redes, pila de generación, distribución, detección de servicios y balanceadores de carga [11]. Todo esto es parte de la infraestructura del ecosistema de microservicios, su construcción, estandarización y mantenimiento debe ser por consiguiente estable, escalable, tolerante a fallos y confiable.

Una propuesta que abarca estos aspectos para microservicios puede dividirse en cuatro capas, aunque los límites de cada una no siempre están claramente definidos: algunos elementos de la infraestructura tocarán cada parte del modelo. Las tres capas inferiores son las capas de infraestructura: en la parte inferior del modelo encontramos la capa de hardware y, además, la capa de comunicación, seguida por la plataforma de aplicación y finalmente la cuarta capa (superior) es donde viven todos los microservicios individuales.



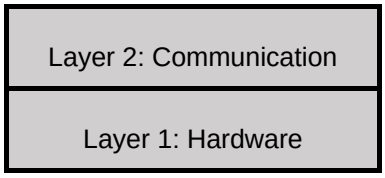


Fig. 13: Modelo de cuatro capas de Ecosistema de Microservicios [11]

Capa Uno – Hardware

Se enfoca puramente en las herramientas físicas como servidores y equipos de cómputo que se utilizarán para montar el resto de componentes de software. Estos equipos pueden ser propiedad de la empresa o pagar por servicios cloud que es lo más rentable como, por ejemplo: Amazon Web Services; Elastic Compute Cloud (AWS EC2); Google Cloud Platform (GCP) o Microsoft Azure.

La gestión de estos servidores también debe ser considerado, para esto el sistema operativo que se elija dependerá del enfoque que posea la empresa en cuanto a la utilización de distribuciones libres o propietarias. Una vez instalado el sistema operativo, se debe usar una herramienta de administración de configuración para instalar todas las aplicaciones y establecer todas las alineaciones necesarias. Posteriormente los hosts necesitan una adecuada supervisión y registro de nivel de host para que si existe falla de disco, fallo de red o si la utilización de CPU sobrepasa el límite, puedan ser fácilmente diagnosticado, mitigado, y resuelto. Todas estas consideraciones son administradas de mejor manera con tecnologías IaaS, PaaS y SaaS que detallaremos más adelante.

Capa Dos – Comunicación

Esta capa se centra en la comunicación de microservicios, e interactúa con las otras capas del ecosistema. Utilizando un protocolo específico, un microservicio enviará datos en un formato estandarizado a través de la red a otro microservicio que puede estar en un punto final del API de microservicios, o lo enviará a un intermediario de mensajes que se asegurará de que los datos se envíen al otro punto dado.

Así pues, en una arquitectura monolítica, el tráfico solo necesita ser enviado a una aplicación y distribuido apropiadamente a los servidores que alojan la aplicación. En la arquitectura de microservicios, el tráfico debe ser encaminado apropiadamente a un gran número de aplicaciones diferentes, y luego distribuirse apropiadamente a los servidores que alojan cada microservicio específico, y para esto se utilizan tecnologías como *service discovery, service registry, y load balancing.*

Capa Tres – Plataforma de Aplicación

Encargada de todos los servicios y herramientas internas que son independientes de los microservicios, mismos que deben construirse de tal forma que los equipos de desarrollo no tengan inconvenientes en diseñar, construir o mantener nada que no sea sus propios microservicios. Una plataforma de aplicaciones puede incluir una herramienta interna de autoservicio para desarrolladores, un proceso de desarrollo estandarizado, un sistema de compilación y despliegue automatizado y centralizado, pruebas automatizadas, una solución de despliegue estandarizada y centralizada, un registro centralizado, y monitoreo a nivel de microservicio.

Capa Cuatro – Microservicios

Esta sección se centra exclusivamente en los microservicios y cualquier cosa delimitada a ellos, independientemente de las capas inferiores de la infraestructura, excepto por las configuraciones específicas para cada microservicio en cuanto a uso de herramientas, teniendo en cuenta que estas configuraciones se pueden almacenar en el mismo repositorio del microservicio para ser accedidas por las herramientas de las capas inferiores, evitando cambios directos sobre las mismas.

3.1.4. Componentes para arquitecturas basadas en microservicios.

Una vez analizados los diferentes aspectos del ecosistema de microservicios, podemos realizar un estudio más detallado al especificar cada componente por separado para después encajarlos en cada capa y así obtener un esquema más claro de cómo luciría una arquitectura basada en microservicios.

Cabe recalcar que el uso de los componentes a continuación detallados dependerá netamente del sistema analizado y los requerimientos que necesite.

Discovery Server

El denominado servidor de registro o descubrimiento sirve para delimitar y proveer los end-points de los servicios a ser consumidos. A su vez obtiene instancias disponibles de cada microservicio y permite que cada microservicio sepa lo que otro ejecuta en caso de tener algún tipo de relación de comunicación [11]. Cabe considerar que también se puede incluir un componenete *Configuration Server* que centralice y provea remotamente la configuración de cada microservicio, de manera simple es la encargada de realizar cambios en nuestras API sin la necesidad de volver a implementar código. Esto además hace

referencia a la división del ecosistema en capas previamente estudiado y la mínima dependencia que debe existir entre cada servicio.

Load Balancing

Permite un equilibrio entre las distintas instancias de forma transparente al momento de consumir un servicio, se lo puede implementar como parte de la orquestación de servicios y además puede obtener instancias disponibles del servidor de registro [20]. Su funcionalidad es primordial al poseer un aplicativo que demanda gran cantidad de consumidores, en donde existen servicios más demandados que otros, y de la misa forma existen servicios que devuelven una gran cantidad de información que debe ser administrada de tal forma que otros clientes que consuman el mismo servicio no se vean afectados.

Circuit Breaker

Al tener una arquitectura distribuida el riesgo de fallo en la comunicación entre microservicios es evidente, por tal motivo este componente se encarga de evitar la propagación de un fallo hacia el resto del aplicativo, y a su vez, en lo mínimo el rendimiento de los demás servicios. Se basa principalmente en tres estados [20]:

- Cerrado: Ocurre cuando el micro servicio funciona en total normalidad.
- Abierto: Ocurre al momento de presentarse algún tipo de fallo.
- Semi Abierto: Ocurre cada determinado tiempo para determinar si el servicio ya se encuentra disponible para habilitarlo o sigue caído para mantenerlo aislado.

De esta manera se lo puede configurar para actuar de una forma distinta dependiendo el escenario en el cual se haya producido el daño. Cabe recalcar que también es aplicable cuando se producen errores al consumir servicios externos al sistema, ya que el aplicativo no debería seguir gastando recursos al mandar peticiones continuas, si no tiene respuesta en un determinado tiempo o número de intentos.

Edge Server

Este servicio de borde o perimetral en la mayoría de casos es un API Gateway en el que se exponen los servicios a consumir. Su principal función es convertirse en un punto de entrada único que permita manejar las solicitudes de los clientes, independientemente del medio por el cual accedan (aplicación móvil o aplicación web), o inclusive, aunque sea más complejo se puede crear varios API dependiendo los requerimientos de cada grupo de consumidores [26]. De esta forma aísla a los clientes de cómo se divide la aplicación en

microservicios y todo el tráfico del exterior es encaminado a los servicios internos correspondientes.

Message Broker

Al hablar de este componente podemos encontrarnos en dos contextos diferentes, por un lado, tenemos un patrón de mensajería entre microservicios, y, por otro lado, a una arquitectura basada en eventos para manejar datos [26]. Así pues, se aclara que la comunicación entre microservicios puede ser sincrónica si se comunican directamente entre ellos, o asincrónica si existe un message broker que administre su comunicación.

Sin embargo, este componente posee mayor relevancia en cuanto a manejo de datos se refiere, dado que cada microservicio maneja su propia base de datos relacional o no relaciones, siempre existirá una dependencia entre ellos y por ende un message broker se encarga de gestionar sus transacciones y guarda un log de eventos que nos permita después poder garantizar la transaccionalidad en caso de existir un fallo. Dada su estrecha relación entre la capa de comunicación y aplicación, se lo puede aplicar en cual quiera de ellas.

Logging

En este punto tratamos la monitorización, y como uno de sus ejes principales un registro de logs de toda la información relevante, lo que permite a los desarrolladores comprender el estado del microservicio con un histórico de información para poder prevenir problemas futuros [11]. Una vez que ha madurado nuestro registro de logs se puede proponer un componente adicional que es el uso de *Alertas* para notificar cuando un microservicio está teniendo una conducta diferente a la normal. Por lo general este tipo de alertas se las programa en base a un registro del comportamiento que ha tenido un servicio ya sea en estado normal, peligroso o crítico, y que posteriormente nos permitirá establecer reglas de comportamiento para identificar daños mínimos que puedan evolucionar en catástrofes.

Dashboards

Se centran en monitorear que todas las funcionalidades del sistema se encuentren activas y sus respectivas conexiones con la base de datos [11]. De esta forma reflejan con precisión la salud de los microservicios, se acoplan fácilmente con los componentes como Service Discovery o Message Broker, y están construidos de tal manera que cualquier persona podría verlos y entender el estado del software sin dificultad.

Hardware

Si bien el hardware no es un componente como tal, al tratar con microservicios debemos tener presente la importancia de elegir un entorno físico adecuado en donde se ejecuten, para esto actualmente existe una gran cantidad de tecnologías que nos ofrecen los servicios en la nube, para no tener la necesidad de gastar en servidores físicos donde alojar nuestros componentes de software. Así pues, existen tres definiciones importantes [30]:

- IaaS: Infraestructura como servicio, proporciona a los usuarios acceso a recursos informáticos tales como servidores, almacenamiento y redes a través de Cloud.
- PaaS: Proporciona una plataforma con herramientas para probar, desarrollar y alojar aplicaciones en el mismo entorno.
- SaaS: Brinda a los usuarios acceso al software basado en la nube de un proveedor. Los usuarios no instalan aplicaciones en sus dispositivos locales. En cambio, las aplicaciones residen en una red remota en la nube a la que se accede a través de la web o una API.

Una vez aclarado esto, es notorio que los aspectos más relevantes a analizar en una migración son el uso de IaaS y PaaS, puesto que envuelven nuestros microservicios de tal manera que se aíslan en contenedores o máquinas virtuales para lograr una mejor estabilidad y resistencia. No obstante, esto dependerá de la cantidad de recursos que utilice el sistema a ser migrado, y también de los recursos disponibles por la empresa, ya que se puede contratar los servicios en la nube o utilizar servidores físicos propios.

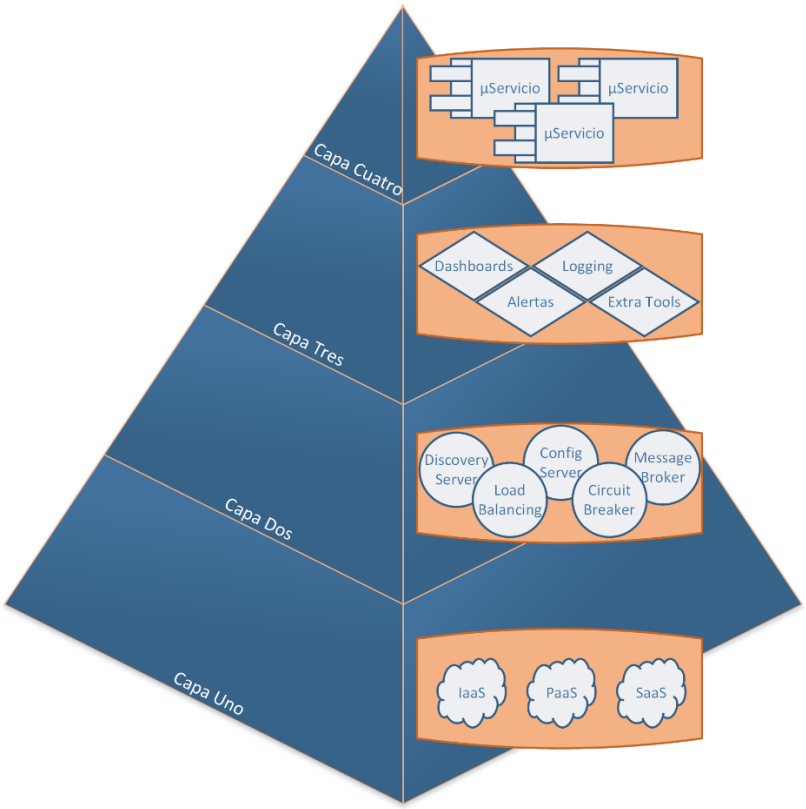


Fig. 14: Ecosistema y Componentes de microservicios

3.2. Delimitación de un modelo conceptual

Dentro de la propuesta metodológica existe un punto crucial que es la delimitación de un modelo conceptual que permita encapsular las directrices que posteriormente serán empleadas para la migración a microservicios. Un modelo conceptual, si se aplica correctamente, debe lograr los siguientes objetivos fundamentales [31], [32]:

- Mejorar la comprensión de los individuos del sistema representativo.
- Proporcionar un punto de referencia para extraer especificaciones adicionales futuras.
- Facilitar documentación de requisitos previos para la tarea de modelamiento.
- Capturar y comunicar la conceptualización con escenarios previstos y potencialmente imprevistos.
- Ser suficientemente transparente para que los interesados en el proyecto se sientan cómodos en utilizarlo, como un medio para discutir los mecanismos dentro del sistema que tienen relevancia para caracterizar su comportamiento.

En base al análisis realizado a lo largo del proyecto se establece un modelo conceptual estructurado en 4 etapas: compresión, descomposición, asociación y validación.

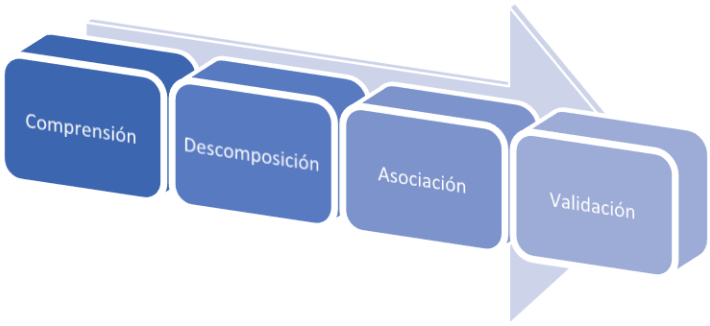


Fig. 15: Modelo Conceptual Migración Microservicios

3.2.1. Comprensión

Una vez analizado los diferentes estudios y teorías sobre microservicios, se plantea como primera etapa la comprensión de las actividades a realizar por parte de los directivos de la empresa propietaria del sistema. Se debe informar y motivar a la institución sobre cual es cronograma por seguir, objetivos, alcance y los beneficios que obtendrán al incluir una transformación no solo del aplicativo, sino además de posibles cambios en la cultura organizacional que se posee para desarrollar software.

Es así como, como herramientas de ayuda para facilitar la apertura al cambio dentro de la empresa se plantean:

- Reuniones sobre la planificación de migración.
- Reuniones sobre limitaciones y alcance.
- Reuniones sobre implicaciones de microservicios.

Cabe enfatizar que no se puede realizar una migración adecuada si no se tiene la apertura al cambio por parte de los directivos.

Al hablar de comprensión también nos referimos al entendimiento del sistema a ser migrado. Esto es importante debido a que se debe entender como fue construido el aplicativo, incluyendo sus antecedentes, metodología de desarrollo, modelo de dominio, estructura de base de datos, arquitectura, entre otros. Cabe recalcar que en caso de que la información no se encuentre actualizada se recomienda realizar un análisis dividido en dos partes, primero una sección de antecedentes que nos ayude a comprender los requerimientos por los cuales fue creado el aplicativo, y segundo, un análisis del estado

actual que nos permita comprender como evoluciono el sistema y las implicaciones de realizar una transformación a microservicios sobre ese aplicativo.

Para realizar esto se puede utilizar un análisis de:

- Metodología de desarrollo.
- Requerimientos Funcionales.
- Requerimientos No Funcionales.
- Diagrama Global de Paquetes.
- Clases pertenecientes a paquetes del sistema.
- Esquema Lógico de Base de Datos.
- Esquema Físico de Base de Datos.
- Modelo de Dominio Inicial.
- Patrón de Diseño.
- Arquitectura Física.

Toda esta información se debe recolectar de la documentación del sistema o en su defecto, extraerla manualmente en caso de no tenerla formalmente. Existen muchos otros diagramas, esquemas y herramientas que nos pueden ayudar a comprender el aplicativo, sin embargo, en este modelo se sugieren éstas por ser las que poseen mayor información y son más accesibles de adquirir en cualquier sistema.

3.2.2. Descomposición

Una vez cumplida la etapa de comprensión se continúa con la descomposición del monolito. Para esto se analiza los diferentes patrones de descomposición existentes y se selecciona cuáles de ellos se adaptan de mejor manera a nuestro aplicativo, tanto en módulos de software como en base de datos.

Para una descomposición optima como ya lo menciona [26], es primordial definir las operaciones que realiza el sistema, así pues, nos podemos valer de las siguientes herramientas para facilitar su estudio:

- Software de Modelamiento de clases
- Modelo de Dominio
- Tabla con funciones del sistema.

En este punto se debe enfatizar que esta propuesta se limita a sistemas mínimamente construidos con modelo por capas de software, ya que una vez identificadas las funciones

del sistema, procedemos a agrupar esas funcionalidades en futuros microservicios sobre la capa de negocio y datos, mediante:

- Lista de posibles microservicios.
- Diagrama de Modelo de Dominio Dividido.
- Diagrama de Base de Datos Descompuesto.
- Software de Modelamiento de Entidades de la Base de Datos.
- Tabla con entidades sensibles por número de relaciones.
- Querys sobre la Base de Datos.
- Reuniones sobre el funcionamiento con usuarios del sistema.
- Cuadro comparativo de lectura y escritura de entidades.
- Diagrama de Base de Datos Descompuesto.

Ya estructurados los diagramas de posibles descomposiciones, se realiza una comparativa para determinar el nivel de dependencia entre la capa de negocio y datos, para finalmente definir una descomposición óptima. Se plantea la utilización y realización de:

- Pruebas en ambiente de producción del sistema.
- Software de Monitorización.
- Matriz con llamadas de la capa de persistencia a la Base de Datos.
- Diagrama final de descomposición.

3.2.3. Asociación

Una vez realizada la descomposición final, se debe acoplar cada elemento dentro las capas del ecosistema de microservicios. Esto se lo plantea como la tercera etapa de nuestro modelo, para lo cual se debe utilizar tanto los componentes como herramientas que se adapten mejor a nuestro estudio de caso.

Se debe tener en cuenta que, si bien todas las capas del ecosistema son necesarias, los componentes que cada una utilice dependerán del aplicativo analizado, es decir, en base a los recursos de la empresa y los diferentes escenarios en el cual se plantea poner en producción el sistema, se deberán seleccionar varios, todos, o inclusive más componentes de los que se han planteado a lo largo de esta propuesta: Dashboards, Logging, Alertas, Discovery Server, Load Balancing, Config Server, Circuit Breaker, Message Broker, IaaS, PaaS, SaaS.

Así pues, se parte de la capa cuatro en donde se construye nuestro núcleo del ecosistema con los microservicios previamente descompuestos, tanto en la capa de negocio, como en la capa de datos. Continuamos con la capa tres en donde analizamos todo lo que es plataforma de aplicación, para en conjunto con los componentes necesarios que se puedan adaptar a la capa cuatro, se forme lo referente a comunicación y finalmente despliegue de las funcionalidades que se planteó para cada microservicio. Para todo esto nos valemos consecutivamente de:

- Lista de componentes factibles para integrar en sistema analizado.
- Software de Modelamiento.
- Diagrama de sistema en base a ecosistema de Microservicios.

Cabe recalcar que no se ha hecho referencia a la capa de vista, puesto que desde 2016 ya existen nuevas tendencias y estudios en evolución referente a lo que es “Micro Frontends”, que se enfoca netamente en trabajar la interfaz de usuario de forma descompuesta, varias referencias interesantes e información más detallada se lo puede encontrar en el artículo de Michael Geers [33]. Aunque estos conceptos quedan fuera del alcance de este proyecto ya que aún se encuentran en desarrollo, y únicamente se ha tomado referencias ya comprobadas en diferentes casos de estudio y por diferentes autores, en donde poseemos una mayor confiabilidad de nuestras fuentes bibliográficas, ya sea de libros o artículos.

3.2.4. Validación

Finalmente, una vez establecida la arquitectura distribuida dentro del ecosistema de microservicios, procedemos a validar que se cumplan todos los requerimientos y características que poseen.

En este punto cabe recalcar la diferencia entre Validación y Verificación, acorde a la IEEE tenemos [34]:

- Verification: “The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase”.
- Validation: “The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements”.

Por ende, se entiende que la verificación es aplicable a cada una de las fases en las cuales se desarrolló un producto, es decir, si se construyó el producto de forma correcta. Mientras que la validación evalúa que el producto final cumpla con los requerimientos, es decir, si se construyó el producto correcto.

Para determinar si hemos construido el producto deseado, en nuestro caso el esquema del sistema migrado a microservicios, lo dividiremos en dos secciones, la primera comprende lo estrictamente relacionado con el modelo propuesto, y la segunda parte en donde ya se encuentre en producción el sistema. Teniendo en cuenta que ésta segunda parte se plantea para trabajos futuros.

Además, se debe evaluar consideraciones para producción que posee un sistema de microservicios. Es decir, no se puede implementar todo el análisis realizado para la migración, si no se siguen patrones y estrategias de despliegue que posteriormente se utilizaran con el departamento encargado de la transición a un sistema distribuido.

Una manera de validar los aspectos de microservicios los plantea Susan Fowler [11] en donde recopila todos los aspectos de un ecosistema de microservicios en varias preguntas, sin embargo, se debe seleccionar las más adecuadas puesto que existen preguntas que corresponden a un sistema ya en producción o a temas estrictamente relacionados con su libro como testeo de resiliencia o documentación de microservicios, que además incluye un checklist aplicable cuando el aplicativo ya se encuentre en etapas de prueba, este se lo puede observar en el Anexo 2.

Por nuestra parte para esta etapa de SMMicro se seleccionaron preguntas que puedan ser aplicadas a varios estudios de caso, acorde a toda la información recolectada a lo largo del trabajo, en conjunto con algunas de las propuestas por Susan Fowler estrictamente a lo que se refiere con estabilidad, confiabilidad, escalabilidad y rendimiento. El cuestionario completo se lo puede observar en el Anexo 3.

	Estabilidad y Confiabilidad
1	¿El microservicio tiene un repositorio central donde se almacena todo el código?
2	¿El ecosistema de microservicio tiene una línea de implementación estandarizada?
3	¿Qué acceso tiene el entorno de transición a los servicios de producción?
4	¿Las implementaciones para la producción se hacen todas al mismo tiempo, o se implementan incrementalmente?
5	¿Qué son estas dependencias de microservicios?
6	¿Cómo este microservicio mitiga los fallos de dependencia?
7	¿Hay copias de seguridad, alternativas, retrocesos o almacenamiento en caché defensivo para cada dependencia?
8	¿Son confiables los controles de salud al microservicio?

9	¿Hay interruptores en su lugar para evitar que el tráfico de producción se envíe a hosts y microservicios poco saludables?
10	¿Existen procedimientos para depreciar los puntos finales API de microservicios?
	Escalabilidad y Rendimiento
1	¿Cuáles son los cuellos de botella de recursos de este microservicio?
2	¿Escalarán las dependencias con el crecimiento esperado de estos microservicios?
3	¿Se puede enrutar automáticamente el tráfico a otros centros de datos en caso de falla?
4	¿El microservicio está escrito en un lenguaje de programación que permitirá que el servicio sea escalable y funcional?
5	¿Hay alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio maneja las solicitudes?
6	¿Existe alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio procesa las tareas?
7	¿Este microservicio maneja los datos de una manera escalable y eficiente?
8	¿Qué tipo de datos necesita cada microservicio para almacenar?
9	¿Cuál es el esquema necesario para sus datos?
10	¿Este microservicio necesita un mayor rendimiento de lectura o escritura?
11	¿Es de lectura pesada, de escritura pesada o ambas cosas?
12	¿La base de datos de este servicio se escala horizontal o verticalmente? ¿Es replicado o particionado?
13	¿Es este microservicio usando un dedicado una base de datos compartida?
14	¿El microservicio tiene un único punto de falla?
15	¿Se han identificado todos los escenarios de falla y posibles catástrofes del microservicio?
16	¿Cuáles son las fallas comunes en el ecosistema de los microservicios?
17	¿Cuáles son los escenarios de falla de la capa de hardware que pueden afectar este microservicio?
18	¿Qué fallas en la capa de comunicación y en la capa de aplicación pueden afectar este microservicio?
19	¿Cómo impactan las fallas y las interrupciones de este microservicio en el negocio?
20	¿Hay niveles de falla bien definidos?

Tabla 6: Preguntas seleccionadas para Evaluación de Microservicios

3.3.Lineamientos de transición para una descomposición en microservicios.

En este punto se definen lineamientos dentro del modelo conceptual previamente definido. La imagen a continuación presentada es únicamente referencial, para poder observar a detalle todos los campos, se encuentra el Anexo 4 en donde se puede visualizar de mejor manera cada sección.

	Lineamientos	Justificación	Actividades	Herramientas	Consideraciones y Recomendaciones
Comprensión	Análisis de cambio, en base a lógica de negocio.	Constatar que la institución impulsará un cambio en la forma de estructurar software.	Comprender cambios con director o jefe de proyectos.	Documentación de planificación de la migración.	No existe un formato específico para estos documentos, existira una variación acorde a la empresa, pero lo escencial es que cumpla con la finalidad propuesta.
			Analizar con equipo de desarrollo posibles implicaciones de migración.	Documentación sobre alcance de migración.	
	Estudio de los antecedentes del sistema a migrar.	Comprender el origen de creación del sistema a ser migrado para entender las implicaciones de su evolución en el tiempo.	Extraer información general del aplicativo.	Metodologías de desarrollo.	Es primordial analizar y entender el estado y documentación del sistema a migrar en su origen con relación a la organización. Así como identificar áreas problematicas en su diseño inicial y como esto puede afectar el nuevo cambio de arquitectura.
				Requerimientos Funcionales.	
				Requerimientos No Funcionales.	
			Extraer información de los Módulos Iniciales del aplicativo.	Diagrama Global de Paquetes.	
				Clases pertenecientes a paquetes del sistema.	
			Extraer información de la Base de Datos del aplicativo.	Esquema Lógico.	
				Esquema Físico.	
	Estudio del estado actual del sistema a migrar.	Identificar las deficiencias actuales del sistema, así como analizar que la migración de arquitectura creará un cambio significativo a futuro.	Extraer información de los Módulos Actuales del aplicativo.	Diagrama Global de Paquetes.	Se procura identificar transformaciones seguras, es decir cambios que no afectarán el comportamiento existente del aplicativo. Sin embargo, a pesar que el usuario final no distinga una variación considerable en su experiencia, para el sistema como tal, si, puesto que se busca mejorar su rendimiento en backend.
				Modelo de Dominio Inicial.	
			Extraer información de la Base de Datos del aplicativo.	Esquema Lógico.	
				Esquema Físico.	
			Extraer información de la Arquitectura del aplicativo.	Patrón de Diseño.	
				Arquitectura Física.	
Descomposición	Análisis de la información de las operaciones del sistema.	Establecer un punto de referencia para la descomposición	Crear un nuevo esquema que muestre las clases principales del aplicativo.	Modelo de clases.	Las operaciones del sistema definidas nos ayudarán a definir posteriormente la granularidad de los servicios, puesto que las funcionalidades del aplicativo deben perdurar sin cambios.
				Modelo de Dominio.	
			Describir el comportamiento del sistema en base al Modelo de Dominio.	Tabla con funciones del sistema.	
	Aplicación de patrones de descomposición en la capa de negocio del aplicativo.	Abstraer una fragmentación adecuada manteniendo de la lógica de negocio del sistema.	Dividir la capa de negocio en base al Modelo de Dominio.	Lista de posibles microservicios.	La capa de negocio debe mantener la lógica de negocio a pesar de la división del modelo de dominio, se recomienda analizar las funcionalidades del aplicativo en producción para apreciar de mejor manera los posibles microservicios.
			Definir la primera propuesta de descomposición de capa de negocio.	Diagrama de Modelo de Dominio Dividido.	
			Extraer la primera propuesta de descomposición de la base de datos.	Diagrama de Base de Datos Descompuesto.	
				Modelo de Entidades de Base de Datos.	
	Aplicación de patrones de descomposición en la capa de datos del aplicativo.	Abstraer una fragmentación adecuada manteniendo de la lógica de negocio del sistema.	Definir el número de relaciones padre e hija de tablas.	Tabla con entidades sensibles por número de relaciones.	Es primordial buscar que cada microservicio aisle al máximo sus datos, sin embargo, al tratarse de una base de datos relacional se deben realizar varios posibles escenarios de descomposición puesto que existen varios factores que influiran en el rendimiento final del sistema, estre ellas aquellas entidades estrechamente relacionadas que deberán mantener cierta relación de dependencia.
			Definir el tamaño y número de registros de las tablas.	Querys sobre la Base de Datos.	
			Clasificar las tablas en base a lectura y escritura intensiva.	Cuadro comparativo de lectura y escritura de entidades.	
			Abstraer la segunda propuesta de descomposición de la Base de Datos.	Diagrama de Base de Datos Descompuesto.	
	Identificación de una propuesta de descomposición final.	Obtener un diagrama de descomposición que se acople con la lógica de negocio del sistema.	Comparar la independencia entre propuestas de capa de negocio con descomposición en Base de Datos.	Pruebas en ambiente de producción del sistema.	Para seleccionar una descomposición final, se debe analizar las posibles variaciones que tendra la Base de Datos en conjunto con la capa de negocio, sus llamadas, relaciones e instancias. Utilizar un ambiente de producción facilitará la visualización de todos estos factores de comportamiento.
				Matriz con llamadas de la capa de persistencia a Base de Datos.	
			Elegir mejor propuesta de descomposición entre capa de negocio y BD.	Diagrama final de descomposición.	
Asociación	Selección de componentes para acoplamiento en capa cuatro de ecosistema de microservicios.	Estructurar los microservicios que en sí facilitaran el núcleo de la nueva arquitectura.	Analizar el acoplamiento y configuración de los microservicios previamente definidos.	Lista de componentes factibles para integrar en sistema analizado.	Esta sección se centra exclusivamente en los microservicios y cualquier cosa delimitada a ellos, teniendo en cuenta que sus configuraciones se pueden almacenar en el mismo repositorio del microservicio para ser accedidas por las herramientas de las capas inferiores, evitando cambios directos sobre las mismas.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.	Diagrama de componentes en capa cuatro de ecosistema de microservicios.	
				Diagrama de sistema en base a ecosistema de Microservicios.	
	Selección de componentes para acoplamiento en capa tres de ecosistema de microservicios.	Integrar los componentes que faciliten la configuración y monitorización de los microservicios.	Analizar el acoplamiento de los componentes para la plataforma de aplicación de microservicios.	Lista de componentes factibles para integrar en sistema analizado.	Encargada de todos los servicios y herramientas internas que son independientes de los microservicios, mismos que deben construirse de tal forma que los equipos de desarrollo no tengan inconvenientes en diseñar, construir o mantener nada que no sea sus propios microservicios.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.	Diagrama de componentes en capa tres de ecosistema de microservicios.	
				Diagrama de sistema en base a ecosistema de Microservicios.	
	Selección de componentes para acoplamiento en capa dos de ecosistema de microservicios.	Integrar los componenetes que faciliten el despliegue y comunicación de los microservicios.	Analizar el acoplamiento de los componentes para la comunicación entre microservicios.	Lista de componentes factibles para integrar en sistema analizado.	Se centra en la comunicación de microservicios, e interactúa con las otras capas del ecosistema. El tráfico debe ser encaminado apropiadamente a un gran número de aplicaciones diferentes, y luego distribuirse apropiadamente a los servidores que alojan cada microservicio específico.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.	Diagrama de componentes en capa dos de ecosistema de microservicios.	
				Diagrama de sistema en base a ecosistema de Microservicios.	
	Selección de componentes para acoplamiento en capa uno de ecosistema de microservicios.	Integrar los componentes que faciliten el despliegue y comunicación de los microservicios.	Analizar el acoplamiento en cuanto a requerimientos de hardware requieren los microservicios.	Lista de componentes factibles para integrar en sistema analizado.	Se enfoca puramente en las herramientas físicas como servidores y equipos de cómputo que se utilizarán para montar el resto de componentes de software. Estos equipos pueden ser propiedad de la empresa o pagar por servicios cloud.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.	Diagrama de componentes en capa uno de ecosistema de microservicios.	
				Diagrama de sistema en base a ecosistema de Microservicios.	
Validación	Evaluación de consideraciones para producción que posee un sistema de microservicios.	Validar que el aplicativo a migrar puede ser tratado como un sistema basado en microservicios.	Analizar los patrones para un despliegue de microservicios.	Compendio de patrones utilizables durante la migración.	Si no se valida la posibilidad de utilización de patrones y estrategias para la implementacion de la propuesta en microservicios se puede caer en fallos o errores graves ya estando en producción.
			Analizar las estrategias para un despliegue de microservicios.	Compendio de estrategias utilizables durante la migración.	
	Evaluación de cumplir las características básicas de un sistema distribuido.	Comprobar que el sistema analizado puede cumplir los requisitos de un ecosistema de microservicios.	Analizar el sistema como un ecosistema de microservicios	Documento con Preguntas sobre Estabilidad y Confiabilidad.	Un sistema distribuido posee características únicas que deben ser validadas, una vez evaluado esto se procede a evaluarlo como microservicios.
				Documento con Preguntas sobre Escalabilidad y Rendimiento.	
				Checklist de validación de componentes de sistema distribuido.	

Fig. 16: Lineamientos para descomposición en microservicios

3.4. SMMicro (Scheme for Migration towards Microservices).

SMMicro se lo construye con el fin de encapsular todos los aspectos antes vistos, de una forma fácil de comprender y utilizar. La imagen a continuación presentada es únicamente referencial, para poder observar a detalle todos los campos, se encuentra el Anexo 5 en donde se puede visualizar de mejor manera cada sección.

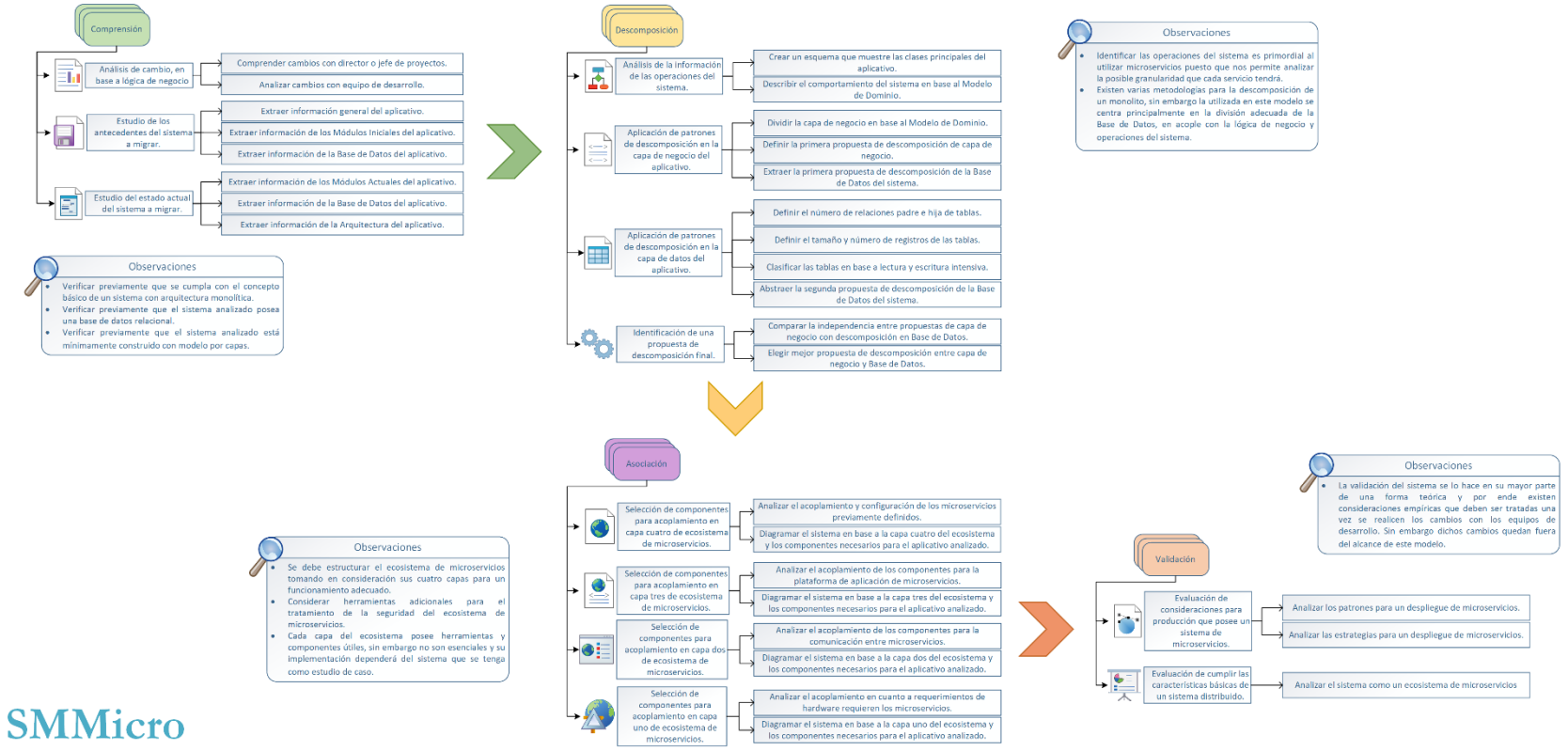


Fig. 17 Scheme for Migration towards Microservices

CAPÍTULO 4 – ESTUDIO DE CASO

Este capítulo se centra en la aplicación de la propuesta metodológica en conjunto con el esquema SMMicro, sobre un estudio de caso que valide sus funcionalidades, permita identificar posibles correcciones, y recolectar recomendaciones para futuras versiones que optimicen una migración.

4.1. Selección del Estudio de Caso

Para poder identificar un estudio de caso conveniente partimos definiendo el conjunto de requerimientos en el cual nuestro esquema se desarrollará de una manera cuasi ideal, para esto es primordial la documentación en cuanto a sistema, su desarrollo y base de datos. En este punto se ha solicitado apoyo a la Dirección de Gestión de Información y Procesos (DGIP) de las Escuela Politécnica Nacional (EPN), quienes nos han facilitado el acceso a su gama de sistemas en producción; dicha petición se ha realizado de manera formal, como se muestra en el Anexo 1.

Una vez obtenidos los sistemas candidatos evaluamos la lógica de negocio que mejor se adapte a un enfoque sobre microservicios. Cabe recalcar que, dado el alcance de este proyecto, no se puede aplicar el esquema sobre un aplicativo de gran magnitud, sin embargo, se lo aplicara sobre un sistema que cumpla los requerimientos mínimos para poder identificar el cambio de arquitectura.

De esta manera se procede a trabajar sobre el Sistema de Gestión Centralizada de Laboratorios (SISLAB), perteneciente al Sistema Integrado de Información (SII) de la Escuela Politécnica Nacional, principalmente por los siguientes motivos:

- SISLAB se encuentra en un estado de cambio continuo, sin embargo, está limitado a una base de tecnología tradicional desarrollada en el año 2009 y por ende posee grandes limitaciones en cuanto a escalamiento se refiere.
- SISLAB es un sistema web que se encuentra en producción y funciona como servicio para un gran conjunto de laboratorios dentro de la EPN, sin embargo, existen otras entidades que demandan funcionalidades específicas que posee este aplicativo, pero por la intrínseca relación que posee como monolito no es posible extraer únicamente los servicios requeridos.
- SISLAB requiere un nivel de estabilidad y resiliencia alto puesto que posee un conjunto de consumidores bastante amplio, por ende, debe estar activo la mayor cantidad de tiempo posible y para esto se necesita reforzar su estructura y por consiguiente su arquitectura.

4.2. Aplicación de propuesta Metodológica

Una vez identificado el sistema a migrar procedemos a aplicar la propuesta metodológica estructura en su esquema que consta de las cuatro etapas ya explicadas: compresión, descomposición, asociación y validación.

4.2.1. Etapa Uno - Comprensión

ANÁLISIS DE CAMBIO

Para realizar el análisis de cambio se mantuvo reuniones iniciales con la Ing. Geovanna Saltos, coordinadora del área, en donde se informó las implicaciones del proyecto con la planificación del trabajo a realizarse. De igual forma dentro del área de desarrollo se mantuvo una constante comunicación con la Ing. Sara Cruz para facilitar la limitación del alcance, sin embargo, estas reuniones no tuvieron la necesidad de ser documentadas debido a que la información no era para personas externas a la Escuela Politécnica Nacional y que además se firmó un acta de confidencialidad que permitía el libre acceso a todo el Sistema de Gestión y Administración de Laboratorios.

ESTUDIO DE ANTECEDENTES

La información mostrada a continuación corresponde al año 2009-2010, año en que fue planteado y tuvo inicio el desarrollo del SISLAB, esta información es un resumen con los aspectos más relevantes de la documentación oficial proporcionada por la DGIP.

a) Información General

El proyecto de desarrollo e implementación de un Sistema de Administración Centralizada de Laboratorios en la EPN permitirá llevar el control centralizado de la facturación y del inventario de los laboratorios que brindan servicio a la comunidad politécnica y al público en general; y será desarrollado por la Unidad de Gestión de Información.

Metodología de Desarrollo

La metodología para el Desarrollo de software que utilizaremos es RUP – Rational Unified Process, esta metodología se divide en 4 fases el desarrollo del software: Inicio, Definición de la Arquitectura, Construcción, Implementación, Documentación.

En esta fase se documentará a lo largo del desarrollo del proyecto los documentos del diseño (Requerimiento, BDD), Documentos de la Aplicación y Documentos para el Usuario final (manuales de usuario)

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, el cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

El ciclo de vida que se desarrolla por cada iteración se maneja bajo dos disciplinas:

Disciplina de Desarrollo

- Ingeniería de Negocios: Entendimiento de las necesidades del negocio.
- Requerimientos: Trasladar las necesidades del negocio a un sistema automatizado.
- Análisis y Diseño: Trasladar los requerimientos a la arquitectura de software.
- Implementación: Crear el software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- Pruebas: Asegurarse que el comportamiento requerido es el correcto y que todo los solicitado está presente.

Disciplina de Soporte

- Configuración y administración del cambio: Archivar y mantener todas las versiones del proyecto.
- Administrando el proyecto: Administrando horarios y recursos.
- Ambiente: Administrar el ambiente de desarrollo.
- Distribución: Hacer todo lo necesario para la implantación y funcionamiento del proyecto

b) Módulos

El proyecto pretende diseñar, construir e implementar un sistema de administración centralizado de laboratorios, el mismo que permita obtener información actualizada cuando se requiera. En el análisis realizado se han identificado los siguientes módulos:

1. Gestión de Inventarios,
2. Facturación,
3. Gestión de Usuarios,
4. Gestión de Servicios,
5. Ordenes de Trabajo,

6. Información Gerencial
7. Reportes

Sin embargo, en el acta de cierre del proyecto en su Versión 1, comprendía la entrega de los siguientes módulos:

1. Gestión de Inventarios
2. Gestión de Servicios
3. Ordenes de Trabajo
4. Reportes

Diagrama Global de Paquetes

El primer diagrama muestra la disposición de las partes integrantes de la aplicación y las dependencias entre los distintos módulos de la aplicación.

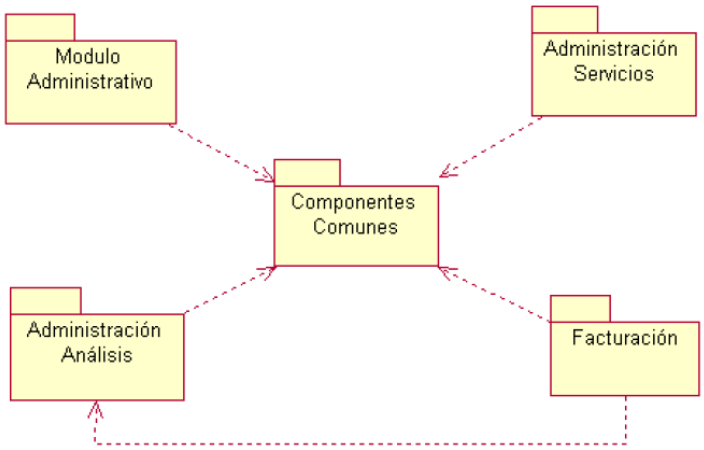


Fig. 18: Diagrama Global de Paquetes Inicial SISLAB [35]

Modelo del Dominio

El modelo de dominio es el encargado de representar los conceptos (objetos) importantes en el dominio del problema. El diseño de un modelo de dominio constituye una actividad clásica del análisis orientado a objetos. Cuando hablamos de objetos, nos referimos a objetos del domino (diccionario visual) no a objetos de software.

Un modelo de dominio incluye:

- Identificar las clases de objetos de interés en el dominio
- Identificar los atributos de las clases de objetos, y;
- Las relaciones entre las clases de objetos.

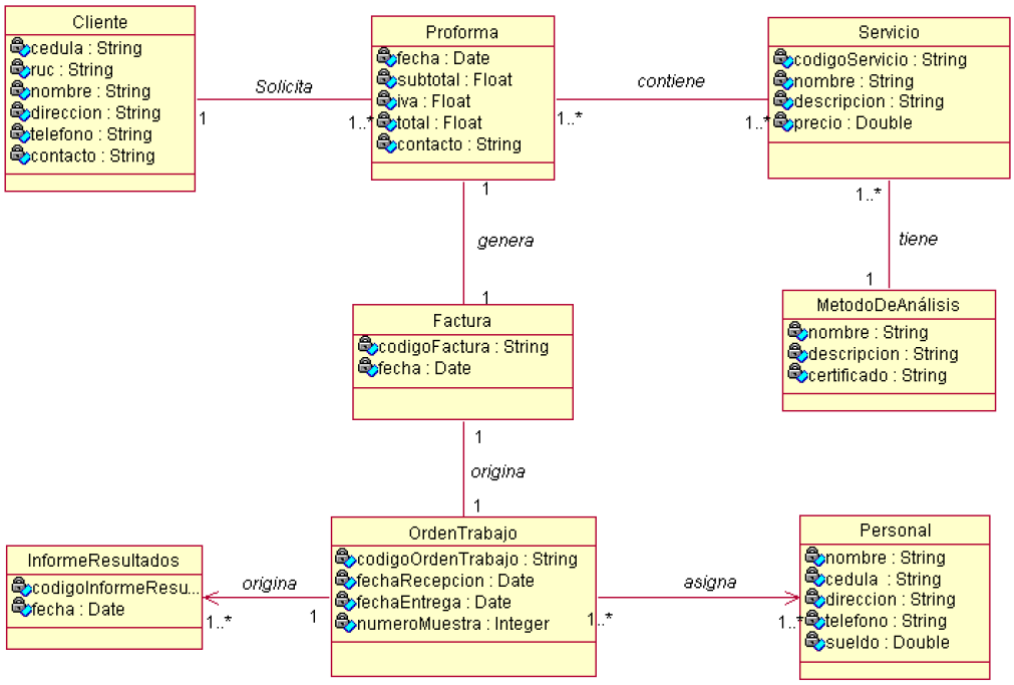


Fig. 19 Modelo de Dominio Inicial SISLAB [35]

A continuación, se presenta el diagrama de clases elaborado para el desarrollo del sistema de Gestión de Laboratorios donde se muestra la iteración entre las clases caracterizadas por sus atributos y operaciones. Cabe destacar que esta imagen es únicamente referencial, si se desea observar la imagen original se la puede observar en [35].

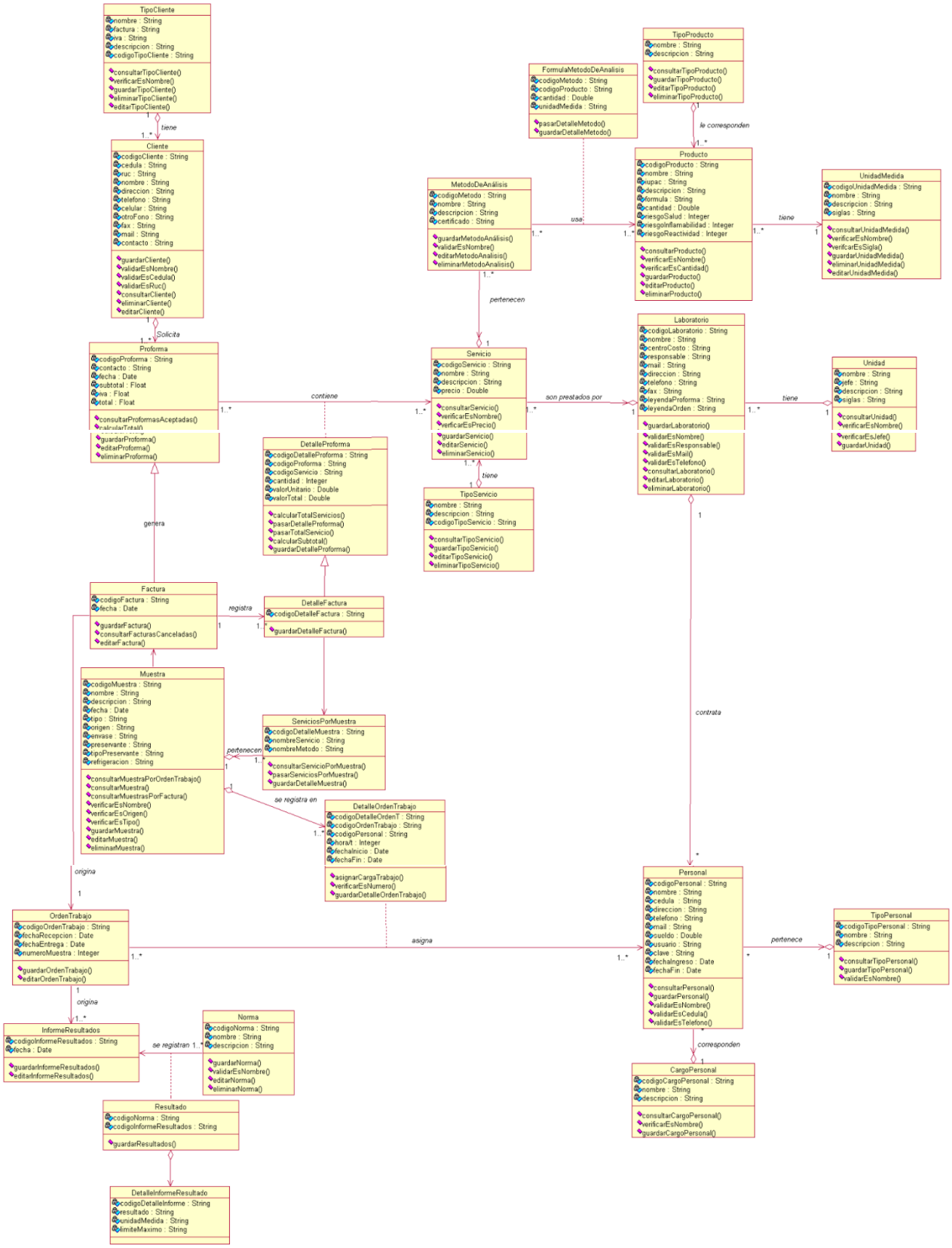


Fig. 20: Diagrama de Clases Inicial SISLAB [35]

c) Base De Datos

El modelo de datos del SISLAB no se encontró legible dentro de los documentos proporcionados por la DGIP, sin embargo, se pudo extraer el nombre de las tablas y numero de relaciones que poseían cuando inició el sistema en producción.

TABLA	**N.º TABLAS HIJAS**	**N.º TABLAS PADRE**
producto	1	3
tipoproducto	1	0
unidadmedida	2	0
detalleinforme	0	3
norma	1	0
cliente	1	1
tipocliente	1	0
unidad	3	0
detallemetodo	0	2
informeresultados	1	1
proforma	2	1
ordentrabajo	2	2
metodo	2	0
laboratorio	2	1
detalleproforma	0	3
detalleorden	0	4
personal	2	3
cargospersonal	1	0
tipopersonal	1	0
servicio	3	2
factura	3	1
detallefactura	0	2
muestra	2	1
detallemuestra	0	1
tiposervicio	1	0
usuario	0	1

Tabla 7: Número de Tablas Padre e Hijas iniciales en SISLAB

ESTUDIO DE ESTADO ACTUAL

El sistema desde su creación ha tenido varios cambios es toda su estructura física y lógica. La mayoría de los cuales no han sido documentados, por lo cual se procede a obtener la siguiente información a través de un análisis del SISLAB actualizado al año 2017.

a) Información General

El objetivo general del Sistema de Administración Centralizada de Laboratorios mantiene su meta de permitir llevar el control centralizado de la facturación y del inventario de los laboratorios que brindan servicio a la comunidad politécnica y al público en general; sin embargo, existen algunos cambios en la lógica de negocio puesto que se implementaron algunas adaptaciones debido al gran escalamiento de laboratorios que posee actualmente la EPN.

Metodología de Desarrollo

Si bien la metodología para el desarrollo del sistema fue RUP, en la actualidad dentro de la DGIP, se utiliza únicamente SCRUM como marco de referencia para desarrollo con metodología ágil, así como para correcciones y actualización de sistemas de software.

b) Módulos

Existen un aumento en el número de módulos funcionales que posee la aplicación, a la fecha de este análisis se encontraron en funcionamiento:

1. Gestión de Inventarios
2. Gestión de Servicios
3. Ordenes de Trabajo
4. Reportes
5. Facturación

Paquetes

En el proyecto se encontraron los siguientes paquetes

- Paquete Persistencia: Utilizado para el almacenamiento de información de las clases en las respectivas tablas de la base de datos.
- Paquete VO: Contiene los métodos que serán utilizados por cada clase para los módulos dentro del SISLAB.

- Paquete Recurso: Posee recursos adicionales que serán de utilidad para todo el sistema, como por ejemplo encriptación de información.
- Paquete Conexión: Utilizado exclusivamente para la conexión a la base de datos Postgres del SISLAB.
- Paquete Reporte: Contiene la clase destinada a la configuración de parámetros adicionales para la producción adecuada de reportes.
- Paquete Servicios: Posee servicios consumidos para funcionalidades específicas como por ejemplo la consulta de datos del registro civil.

Cabe mencionar que al estar basado en el patrón Modelo-Vista-Controlador, los diferentes archivos de presentación (html, css, js, etc.) se encuentran almacenados en otra sección diferente de los paquetes previamente mencionados.

A continuación se muestra un diagrama de los paquetes encontrados en SISLAB, con sus respectivas dependencias, teniendo en consideración que *ec.edu.epn.laboratorios.reporte* y *ec.edu.epn.laboratorios.servicios* no presentan dependencia con ningún otro paquete.

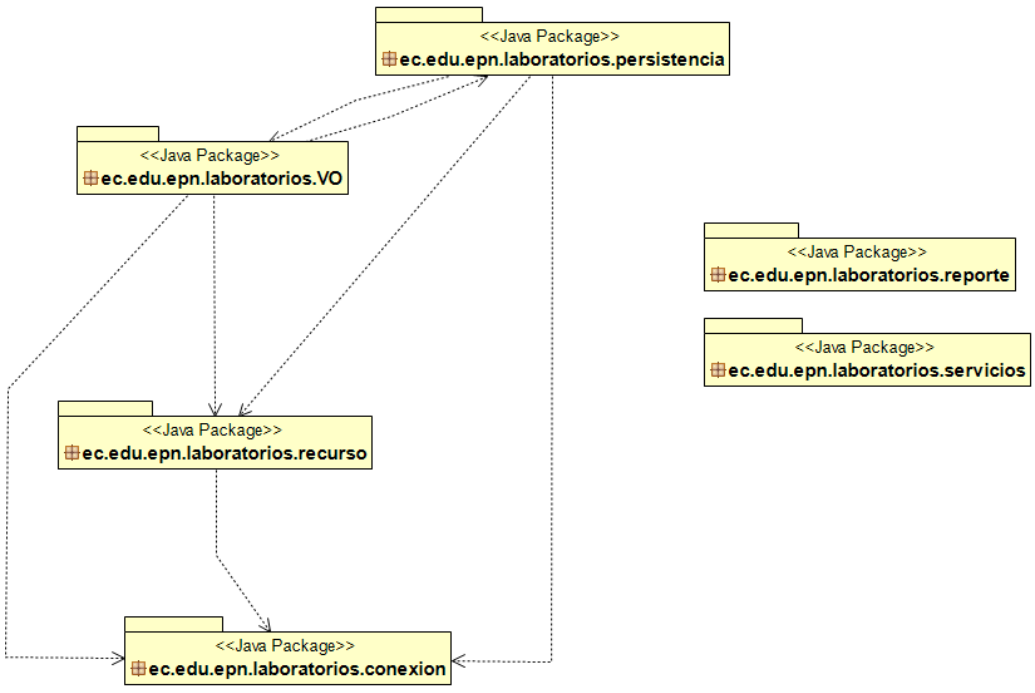


Fig. 21: Diagrama de Paquetes actuales en SISLAB

Diagrama de Clases

Se procede a analizar las clases de los paquetes con relaciones más importantes con la lógica de negocio del sistema, por ende, se realiza un completo barrido de la información que posee el aplicativo en producción.

Además, se considera que el objetivo de este análisis es identificar posibles entidades que posteriormente serán agrupadas o separas acorde a como se adapte la arquitectura basada en microservicios.

- Paquete Recurso: Las clases encontradas dentro de este paquete fueron:

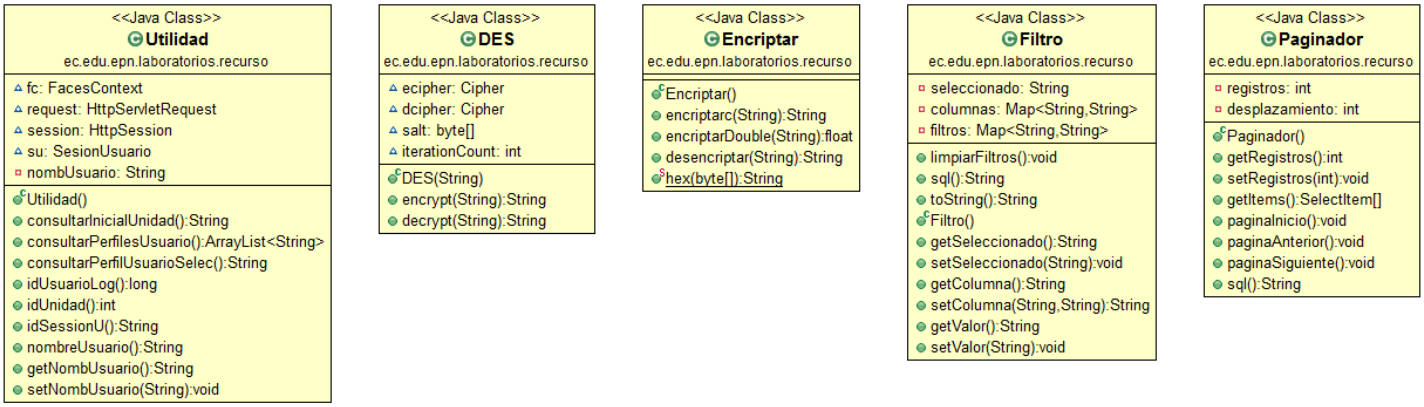


Fig. 22: Clases en Paquete Recurso de SISLAB

- Paquete Conexión. - Las clases encontradas dentro de este paquete fueron:

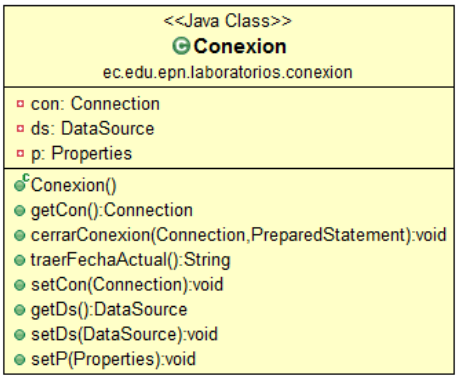


Fig. 23: Clases en Paquete Conexión de SISLAB

- Paquete Persistencia: Se encontró un aumento significativo del número de clases con el que inicio el sistema en 2010, lo que implica que la base de datos de igual manera debió ser modificada:

Fig. 24: Clases en Paquete Persistencia de SISLAB

Se debe considerar que tanto la gráfica de las clases del paquete persistencia, como el del paquete VO se muestran únicamente sus nombres para una mejor visualización. Sin embargo, el análisis se lo realizo con sus atributos y métodos como en los demás paquetes del sistema.

- Paquete VO. - Las clases encontradas dentro de este paquete fueron:

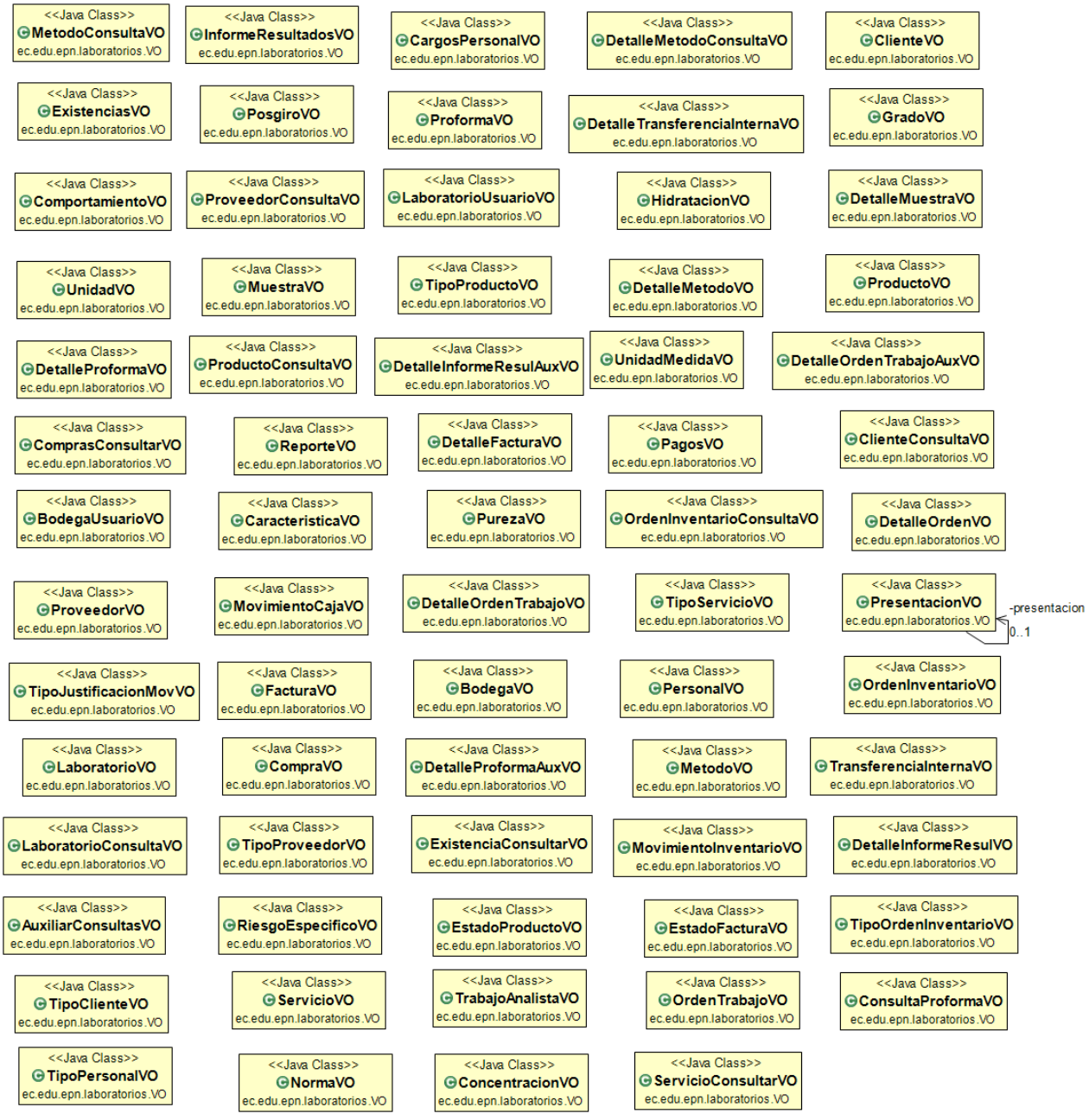


Fig. 25: Clases en Paquete VO de SISLAB

Un hecho a destacar es que se encontró un exceso de dependencia hacia la clase Conexión, demostrando claramente un único punto de fallo hacia la base de datos como se muestra en la Figura 26.

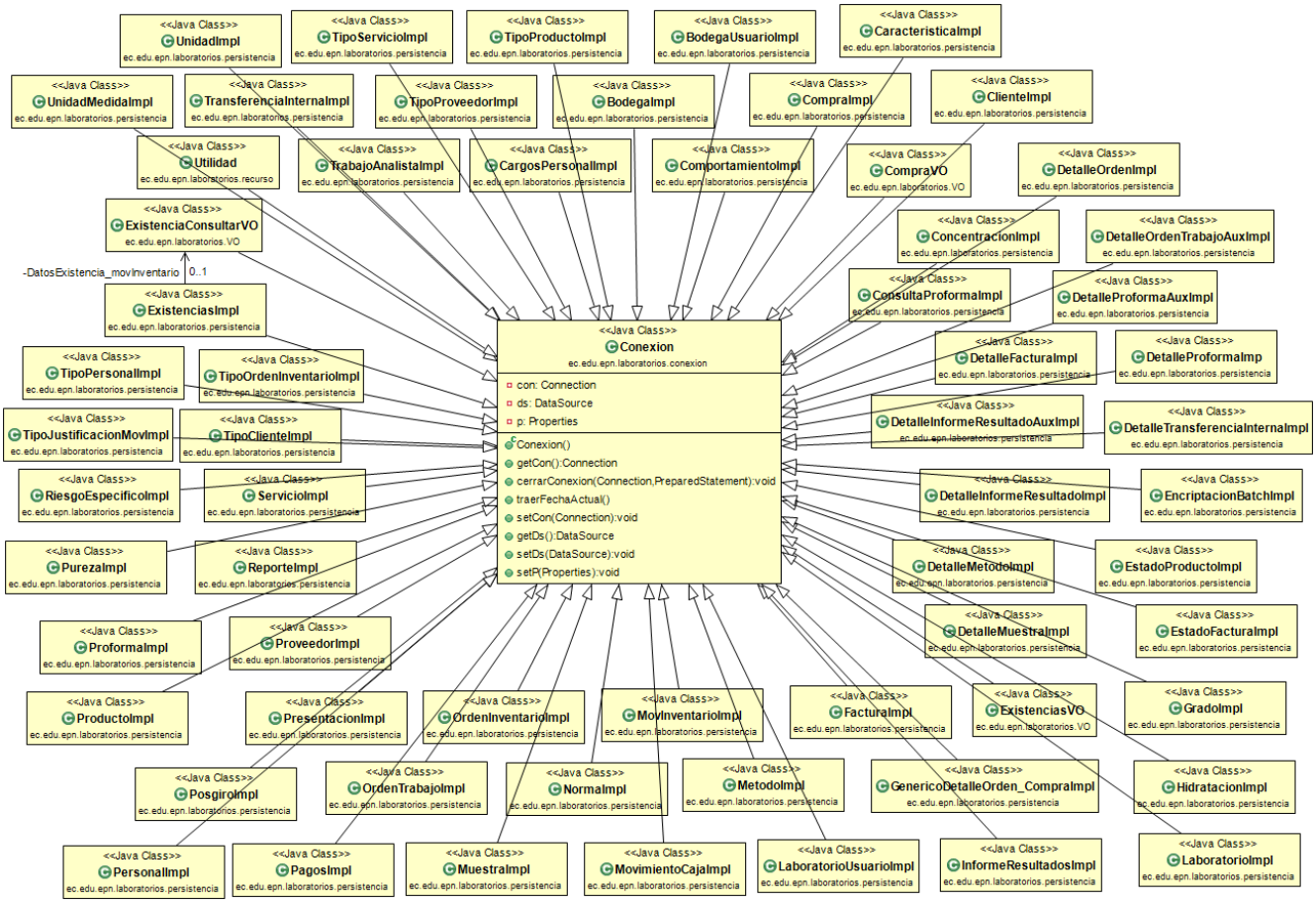


Fig. 26: Dependencias a clase Conexión en SISLAB

c) *Base De Datos*

La base de datos ha sufrido varios cambios considerables, entre los cuales destaca la creación de tablas adicionales y aislamiento de tablas ya existentes en otros esquemas. El número inicial de tablas en la creación del sistema fue de 26, a la fecha se encontraron un total de 44 tablas en el primer esquema analizado, puesto que ahora existe un esquema separado destinado exclusivamente a facturación. A continuación, se muestra un resumen del esquema "*bddcorpepn.Laboratorios"* con los respectivos comentarios realizados por los creadores de las tablas.

TABLA	*N.º TABLAS HIJAS*	N.º TABLAS PADRE	COMENTARIOS
bodega	3	1	Bodegas de la EPN
bodega_usuario	0	1	
caracteristica	2	0	
caracteristica_usuario	0	1	
cargospersonal	1	0	Cargo que tiene un Personal
cliente	2	1	Clientes de la EPN
compra	1	2	
concentracion	1	0	
control_existencia_metodo	0	4	
detalle	0	10	
detalle_proforma	0	4	
detallemetodo	1	3	
detalleorden	1	3	
estadoproducto	1	0	
existencias	4	12	Existencias reales en inventario.
grado	1	0	
hidratacion	1	0	
laboratorio	2	1	Laboratorios pertenecientes a una Unidad
laboratorio_usuario	0	0	
metodo	4	1	
movimientosinventario	0	3	
muestra	2	1	
norma	0	0	
orden_trabajo	3	1	
ordeninventario	1	4	
personal	2	3	No existe relación de tipo foreign key entre personal -

			laboratorio y personal – unidad.
posgiro	1	0	
presentacion	1	0	
producto	1	2	
proforma	2	1	
proveedor	1	1	
pureza	0	0	
riesgoespecifico	1	0	
saldo_existencia	0	0	
servicio	3	2	
tipo_justificacion	2	0	
tipocliente	1	0	Tipos definidos para identificar a los Clientes
tipopersonal	1	0	Tipos definidos para clasificar a Personal
tipoproducto	2	0	Tipos definidos para identificar a los Productos
tipoproveedor	1	0	Tipo definido para clasificar a un proveedor
tipordeninv	1	0	
tiposervicio	1	0	
unidad	7	0	Unidad organizacional de la Politécnica
unidadmedida	3	0	

Tabla 8: Entidades en Base de Datos de SISLAB

El modelamiento de la estructura de la base de datos actual se lo realizo para poseer una mejor vista de tablas sensibles a cambio y numero de relaciones totales, teniendo como resultado el siguiente diagrama:

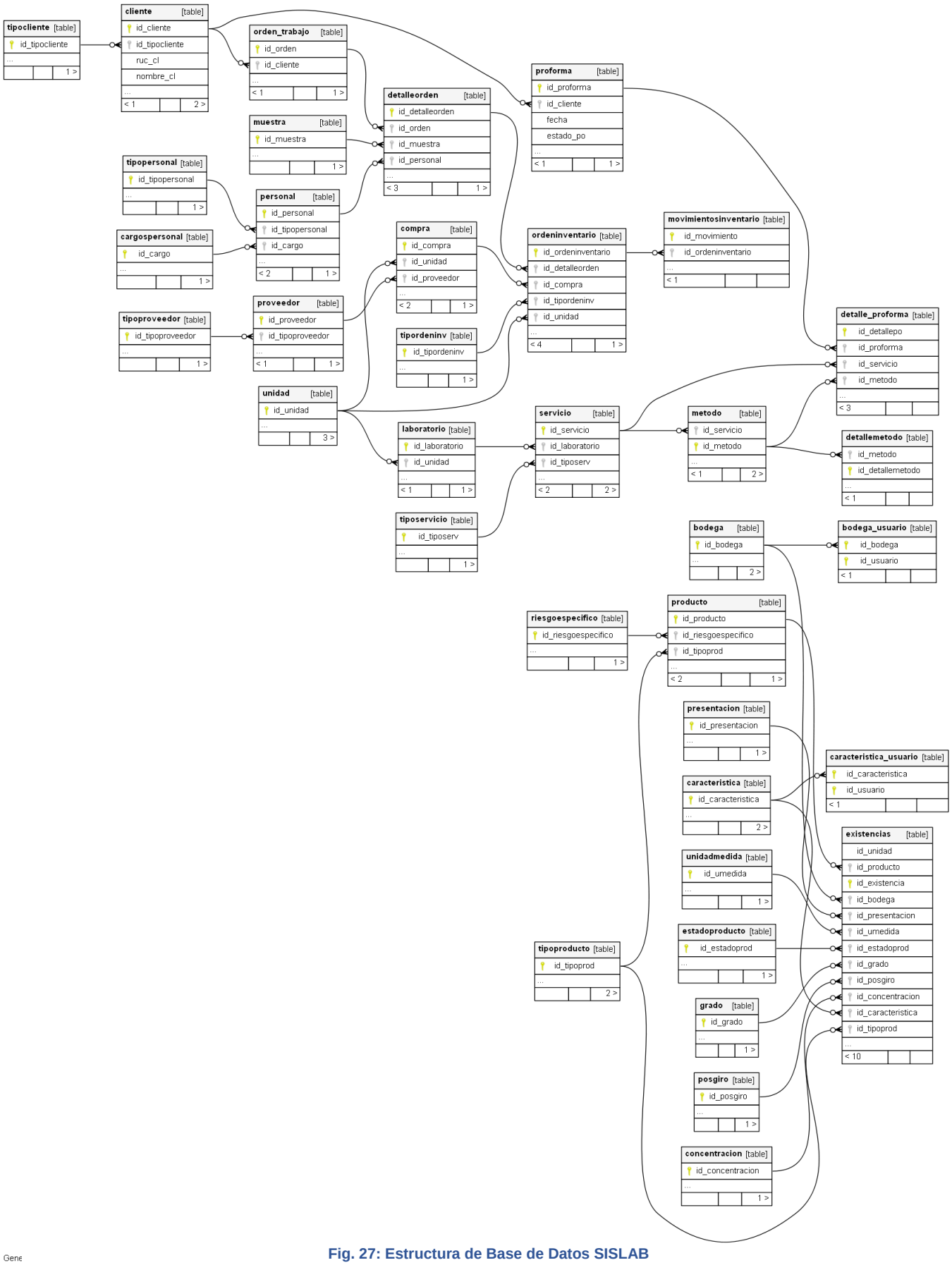


Fig. 27: Estructura de Base de Datos SISLAB

d) Arquitectura

Plataforma multicapa

La arquitectura física impuesta por el modelo de seguridad de red define un modelo de tres capas. Dicho modelo presenta además una distribución de componentes software, distinguiéndose tres niveles lógicos con funciones distintas: uno para los servicios de usuario, otro para los servicios de la lógica de negocio (la lógica principal de la aplicación), y otro nivel para los servicios de datos.

- Presentación: En la capa de presentación se engloban al menos los contenidos estáticos necesarios para la aplicación (CSS, JavaScript, imágenes, etc.) que se desplegarán en el servidor web JBoss del contexto y entorno correspondiente.
- Negocio o aplicación: Como medio para la implementación y soporte de esta capa lógica, así como para el funcionamiento de toda esta arquitectura, estará el servidor de aplicaciones para tecnología Java. El servidor proporciona los servicios necesarios para gestionar el tratamiento de los componentes de negocio desarrollados, así como su control transaccional.
- Datos o Back-end: Se implementan estrategias de persistencia JPA para independizar esta capa propiamente del motor de base de datos escogido. La capa de datos estará en el servidor de Base de Datos PostgreSQL y se administrará utilizando la herramienta Pg Admin.
- Integración: Se añade una capa de integración como:

    a) Intermediario y agregador de funciones de interés general suministrados por distintos sistemas para su exposición y consumo.

    b) Suministrador de servicios con formato de servicio web.

    c) Ámbito de despliegue de procesos de negocio inter-sistemas (orquestación de servicios)

    d) Proveedor de adaptadores que faciliten el acceso a distintas tecnologías subyacentes.

    e) Propuesta de ámbitos de publicación y consumo de eventos con un único punto de entrada, pero con diversas tecnologías de publicación y recolección.

Todos estos componentes se distribuyen entonces en capas lógicas, pero con disposición en forma de servicios para aquellas funciones de interés general como una Arquitectura Orientada a Servicios (SOA)

El sistema fue desarrollado como una aplicación web que permite a los usuarios gestionar la información de los laboratorios de la EPN. Es por esta razón que el proyecto fue desarrollado en base al estándar empresarial J2EE, utilizando la arquitectura de tres capas: Capa del Cliente, Capa Intermedia la cual contiene la presentación y la lógica del negocio y la Capa de Datos.

La aplicación fue completamente codificada en lenguaje JAVA y se implementó las tres capas descritas anteriormente de la siguiente manera:

- La Capa de Datos fue implementada utilizando PostgreSQL 8.4.
- La Capa Intermedia fue implementada utilizando el Servidor JBoss y el framework JSF.
- El Cliente se comunica con la lógica del negocio a través de un navegador como el Mozilla Firefox.

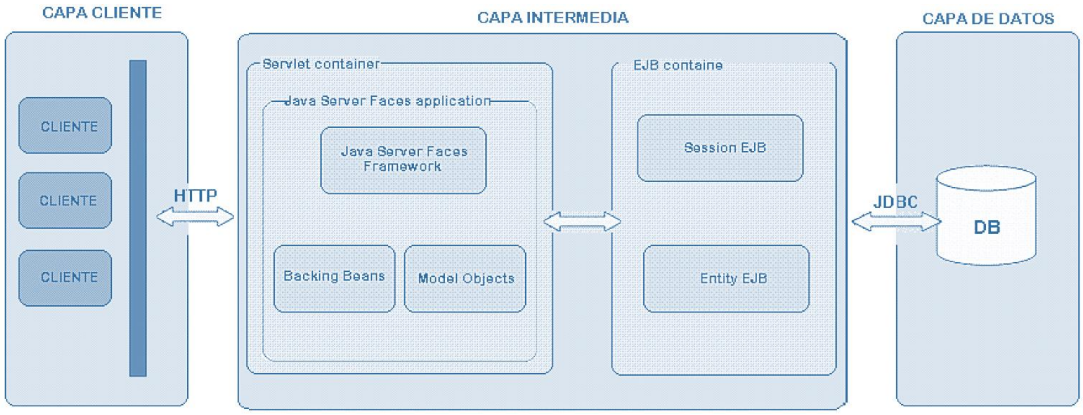


Fig. 28: Arquitectura por Capas Inicial SISLAB [35]

Patrón de diseño

Se utiliza el patrón MVC (Modelo Vista Controlador) durante el desarrollo, para separar la lógica de control, la lógica de negocio y la lógica de presentación, a continuación, se presentan algunas consideraciones dentro de SISLAB.

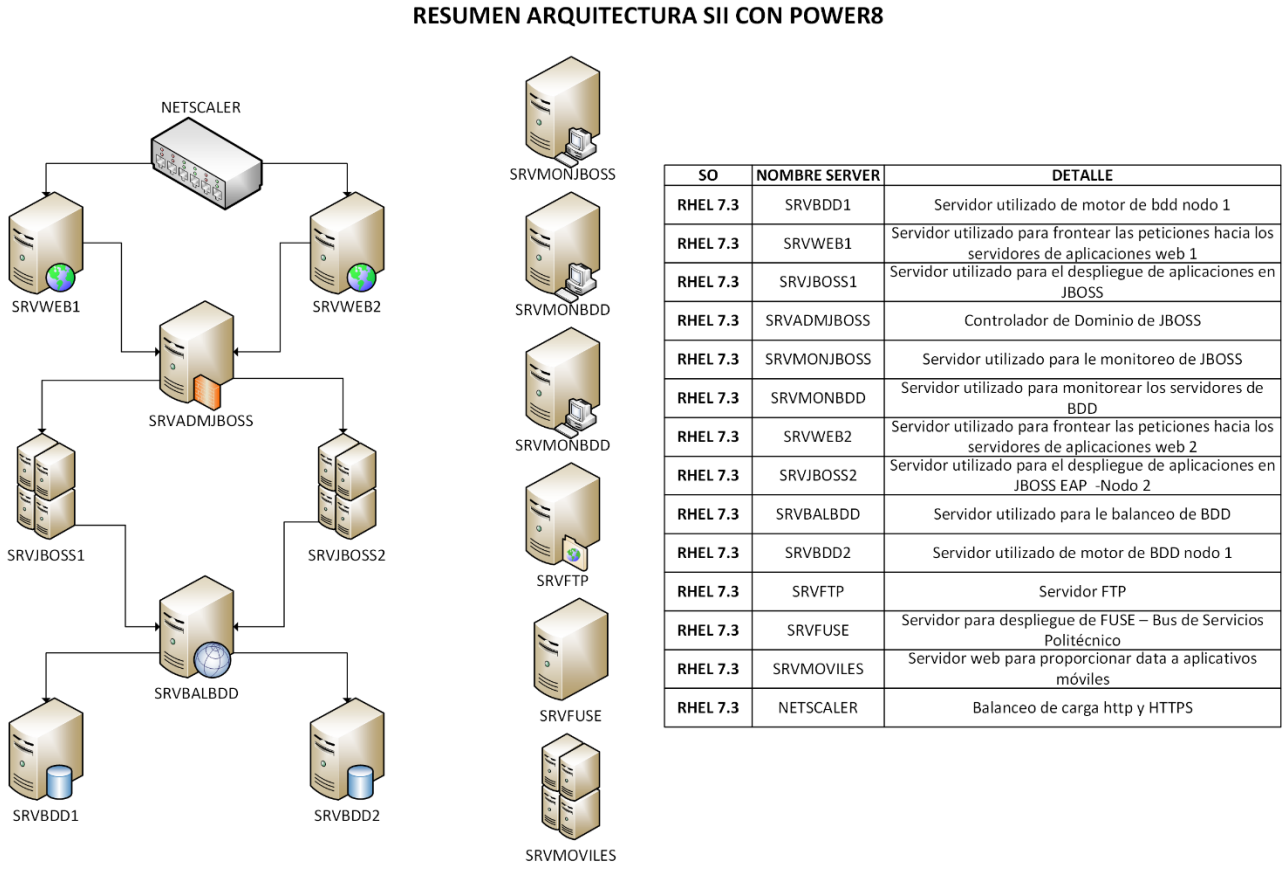
- Modelo: Contiene la lógica del negocio desarrollada totalmente en Java. Aquí se implementan los objetos que representan los datos y se controlan sus transformaciones. El modelo se ve reflejado mediante las clases Impl y VO las

cuales contienen los elementos necesarios para manejar los datos de la aplicación.

- Vista: Contiene las interfaces que se presentan al usuario, desarrollada con páginas JSF, componentes de PrimeFaces y Ajax. La vista se ve reflejada mediante los témplate y las páginas como la index.jsp.
- Controlador: Contiene los eventos del sistema, aquellos que invocan la lógica del negocio conforme a las operaciones que el usuario realice en el sistema. El gestor que reacciona ante los eventos procesa sus acciones y comunica el modelo y la vista es JSF (Java Server Faces). El controlador se ve reflejado en las reglas de navegación contenidas en el fichero faces-config.xml y el servlet faces definido en el fichero web.xml.

Arquitectura Física

Finalmente, la arquitectura física del ambiente de producción del sistema ha sido levantada de la siguiente manera:



SO	NOMBRE SERVER	DETALLE
RHEL 7.3	SRVBDD1	Servidor utilizado de motor de bdd nodo 1
RHEL 7.3	SRVWEB1	Servidor utilizado para frontear las peticiones hacia los servidores de aplicaciones web 1
RHEL 7.3	SRVJBOSS1	Servidor utilizado para el despliegue de aplicaciones en JBOSS
RHEL 7.3	SRVADMJBOSS	Controlador de Dominio de JBOSS
RHEL 7.3	SRVMONJBOSS	Servidor utilizado para le monitoreo de JBOSS
RHEL 7.3	SRVMONBDD	Servidor utilizado para monitorear los servidores de BDD
RHEL 7.3	SRVWEB2	Servidor utilizado para frontear las peticiones hacia los servidores de aplicaciones web 2
RHEL 7.3	SRVJBOSS2	Servidor utilizado para el despliegue de aplicaciones en JBOSS EAP -Nodo 2
RHEL 7.3	SRVBALBDD	Servidor utilizado para le balanceo de BDD
RHEL 7.3	SRVBDD2	Servidor utilizado de motor de BDD nodo 1
RHEL 7.3	SRVFTP	Servidor FTP
RHEL 7.3	SRVFUSE	Servidor para despliegue de FUSE – Bus de Servicios Politécnico
RHEL 7.3	SRVMOVILES	Servidor web para proporcionar data a aplicativos móviles
RHEL 7.3	NETSCALER	Balanceo de carga http y HTTPS

Fig. 29: Arquitectura Física SISLAB

Como esquema adicional tenemos que SISLAB está constituido como lo muestra la Figura 30, en donde se evidencia su estructura de monolito únicamente haciendo referencia a dos esquemas de base de datos, BD1 en donde engloba la mayor cantidad de entidades de SISLAB y BD2 en lo referente a Facturación que si se lo maneja independientemente para todos los sistemas de la DGIP.

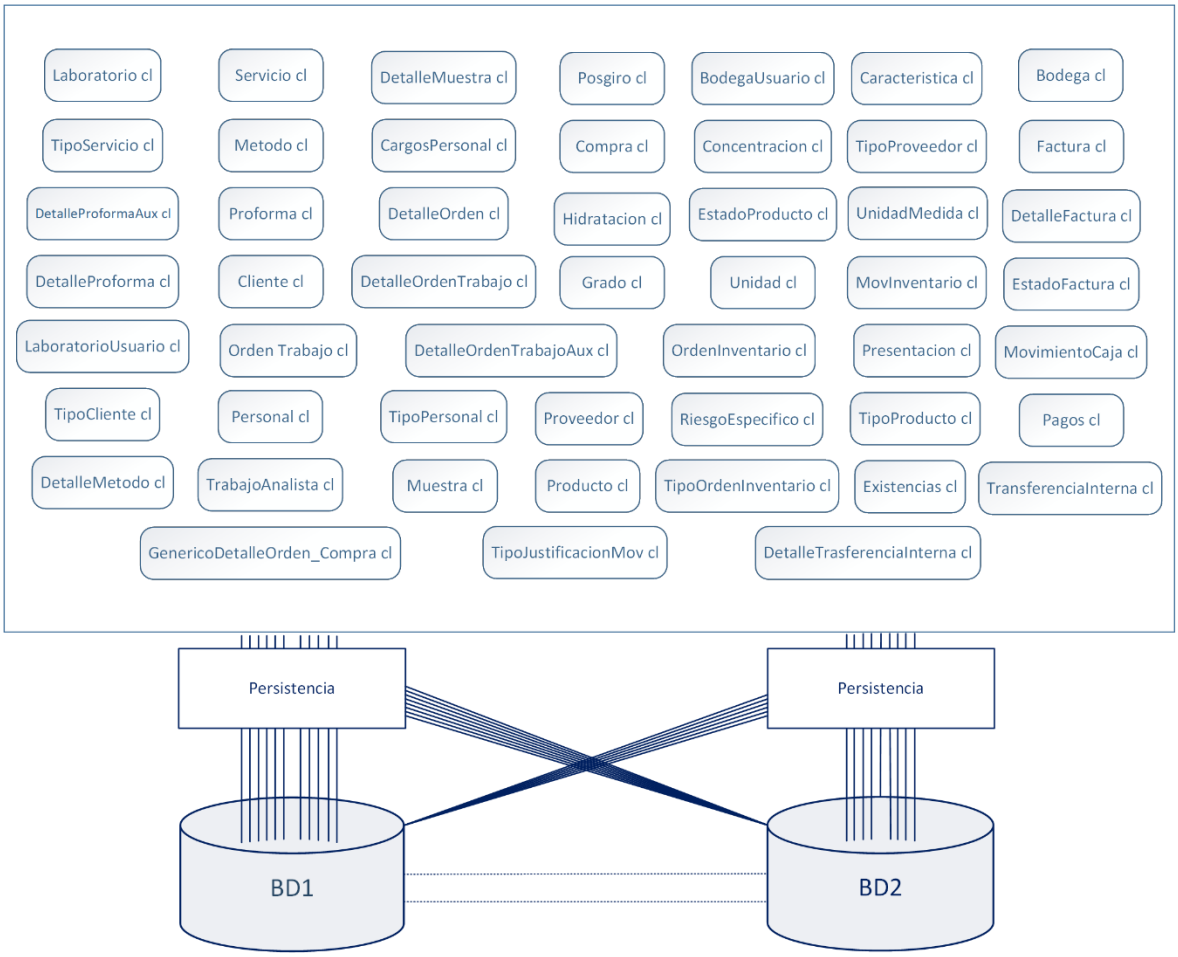


Fig. 30: Relación Capa Negocio y Datos en SISLAB

Cabe acotar que existe un alto grado de dependencia entre las bases de datos debido a que varias entidades del esquema de facturación en BD2 utilizan datos de las entidades de BD1, y simultáneamente algunas tablas dentro del BD1 utilizan datos que se obtienes de la segunda base de datos.

4.2.2. Etapa Dos – Descomposición

OPERACIONES DEL SISTEMA

Modelo de Dominio

Considerando las clases principales que contiene SISLAB, se estructura el modelo de dominio de alto nivel de la siguiente manera:

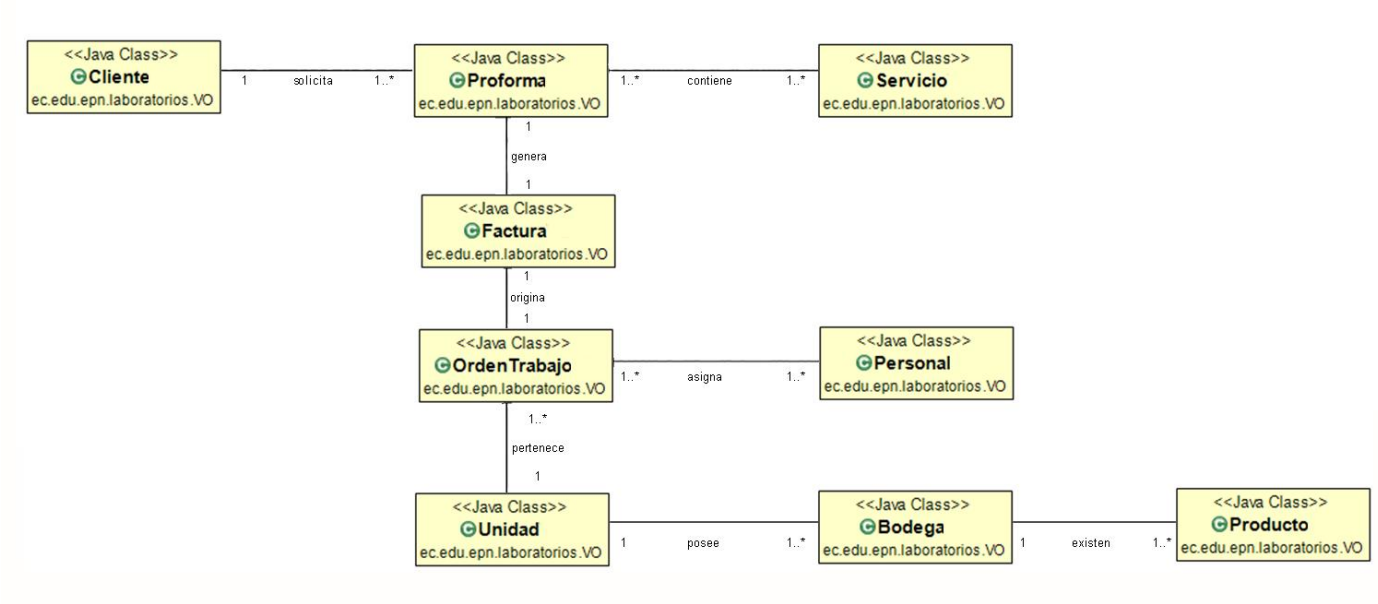


Fig. 31: Modelo de Dominio Actual SISLAB

Funciones del Sistema

Describe el comportamiento del sistema en términos del modelo de dominio.

1	Un cliente solicita una o varias proformas
2	Una o varias proformas contienen uno o varios servicios
3	Una proforma genera una factura.
4	Una factura origina una Orden de Trabajo
5	Una o varias órdenes de trabajo asignan uno o varios personales de trabajo.
6	Una o varias órdenes de trabajo pertenecen a una unidad
7	Una unidad posee una o varias bodegas
8	En una bodega existen uno o varios productos

Tabla 9: Funciones del Sistema Actuales SISLAB

PATRÓN DE DESCOMPOSICIÓN POR LÓGICA DE NEGOCIO

En base a la lógica de negocio y requerimientos funcionales tomados para el sistema, se plantean cuatro ejes principales como microservicios:

1. El primer microservicio y más importante, debido a que contiene el eje principal del sistema, se decir que, a partir de la petición de un cliente, se genera la proforma, que a su vez define el servicio correspondiente y el laboratorio a ser asignado.
2. Un segundo microservicio orientado a las órdenes de trabajo que pueden provenir de una factura o directamente de una proforma, para consiguientemente asignar el personal encargado del trabajo.
3. Se define un tercer microservicio orientado a la gestión de productos, es decir partiendo de la unidad que los solicite verificar las existencias en una bodega.
4. Finalmente, se establece un microservicio definido exclusivamente a facturar, esto debido a que este módulo comparte la misma lógica de negocio para todos los demás sistemas de la DGIP, por lo tanto, es importante poder realizar un aislamiento para reutilización en futuros proyectos.

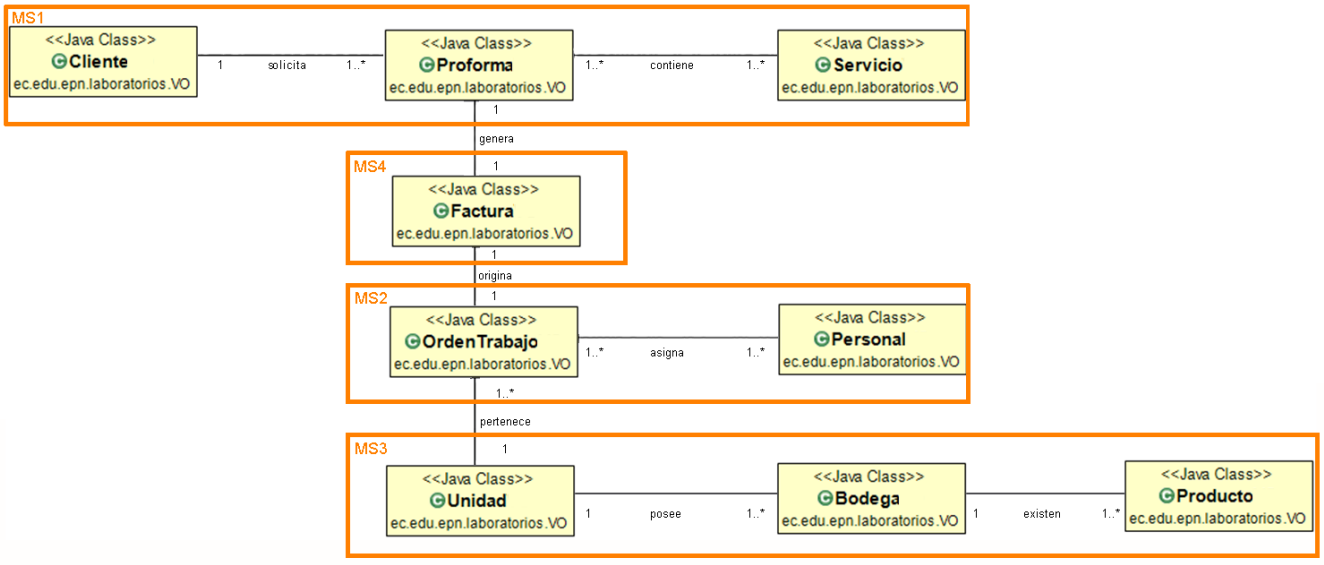


Fig. 32: Diagrama de Modelo de Dominio Dividido SISLAB

A su vez, tomando en cuenta las clases relacionadas al modelo de dominio y sus correspondientes referencias en las tablas de la capa de persistencia, podemos abstraer la primera propuesta para descomponer la base de datos:

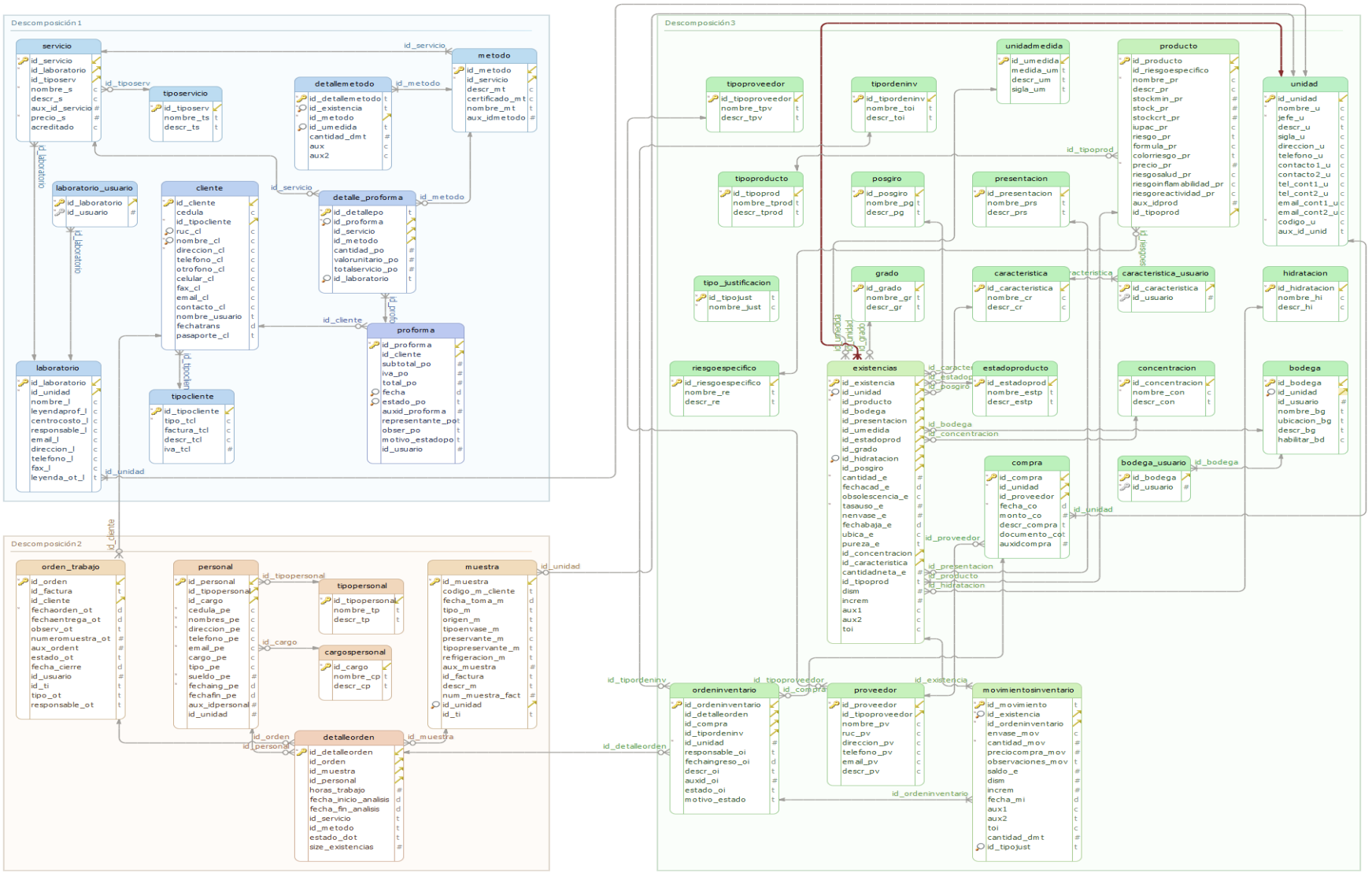


Fig. 33: Primera Propuesta de Descomposición de BD de SISLAB

Sin embargo, se debe analizar si esta propuesta es la óptima o existe una mejor opción viable para descomponer.

Además, se debe aclarar que existe dos contenedores adicionales, uno que posee todas las tablas referentes a facturación y el otro que posee las tablas que no tienen relaciones directas con las demás tablas, como lo son: detalle; saldo_existencia; y control_existencia_metodo.

PATRÓN DE DESCOMPOSICIÓN POR BASE DE DATOS

Tablas Sensibles por número de relaciones

Para analizar la primera propuesta obtenida y realizar una nueva descomposición por base de datos partimos definiendo las tablas sensibles dentro del sistema, para esto observamos el número de relaciones que poseen y de esta manera tenemos:

TABLA	***N.º TABLAS HIJAS***	**N.º TABLAS PADRE**	**N.º TOTAL RELACIONES**
bodega	2	1	3
bodega_usuario	0	1	1
caracteristica	2	0	2
caracteristica_usuario	0	1	1
cargospersonal	1	0	1
cliente	2	1	3
compra	1	2	3
concentracion	1	0	1
control_existencia_metodo	0	0	0
detalle	0	0	0
detalle_proforma	0	3	3
detallemetodo	0	1	1
detalleorden	1	3	4
estadoproducto	1	0	1

existencias	1	11	12
grado	1	0	1
hidratacion	1	0	1
laboratorio	1	1	2
laboratorio_usuario	0	1	1
metodo	2	1	3
movimientosinventario	0	2	2
muestra	1	1	2
norma	0	0	0
orden_trabajo	1	1	2
ordeninventario	1	3	4
personal	1	2	3
posgiro	1	0	1
presentacion	1	0	1
producto	1	2	3
proforma	1	1	2
proveedor	1	1	2
pureza	0	0	0
riesgoespecifico	1	0	1
saldo_existencia	0	0	0
servicio	2	2	4
tipo_justificacion	0	0	0
tipocliente	1	0	1
tipopersonal	1	0	1
tipoproducto	1	0	1
tipoproveedor	1	0	1

tipordeninv	1	0	1
tiposervicio	1	0	1
unidad	5	0	5
unidadmedida	1	0	1

Tabla 10: Relaciones en Entidades de BD de SISLAB

De acuerdo con esto tendríamos como principales tablas sensibles por número de relaciones: detalleorden; existencias; ordeninventario; servicio y unidad.

Tablas Sensibles por acceso de datos

Para poder realizar un análisis de las tablas sensibles por acceso a datos, analizamos el contexto en el cual se desarrollan las actividades del SISLAB. Es decir, las funciones que cumple el aplicativo en los diferentes escenarios que se ejecuta. Como complemento se realiza además un análisis del número y tamaño de los registros existentes en la base de datos, desde su puesta en producción, a través del siguiente script:

bddepn on postgres@PostgreSQL 9.6

```
SELECT stats.relname
           AS Tabla,
       pg_size_pretty(pg_total_relation_size(statsio.relid))
           AS Tamaño,
       stats.n_live_tup
           AS Registros
  FROM pg_catalog.pg_statio_user_tables AS statsio
  JOIN pg_stat_user_tables AS stats
 USING (relname)
 WHERE stats.schemaname = 'Laboratorios'
 ORDER BY registros ASC;
```

Fig. 34: Script para consulta de número y tamaño de registros en SISLAB

Teniendo como resultado:

Tabla	Tamaño	Registros
'caracteristica_usuario'	'8192 bytes'	'0'
'norma'	'8192 bytes'	'0'
'pureza'	'24 kB'	'1'
'tipoproveedor'	'24 kB'	'2'
'tipoproducto'	'24 kB'	'2'

'tipopersonal'	'24 kB'	'3'
'cargospersonal'	'24 kB'	'4'
'tiposervicio'	'40 kB'	'5'
'riesgoespecifico'	'24 kB'	'7'
'estadoproducto'	'24 kB'	'7'
'tipordeninv'	'24 kB'	'8'
'tipo_justificacion'	'24 kB'	'9'
'grado'	'24 kB'	'10'
'tipocliente'	'32 kB'	'10'
'unidadmedida'	'24 kB'	'11'
'bodega'	'24 kB'	'14'
'hidratacion'	'24 kB'	'16'
'posgiro'	'24 kB'	'18'
'presentacion'	'24 kB'	'21'
'unidad'	'80 kB'	'21'
'caracteristica'	'24 kB'	'34'
'proveedor'	'72 kB'	'38'
'laboratorio'	'88 kB'	'53'
'bodega_usuario'	'24 kB'	'54'
'control_existencia_metodo'	'24 kB'	'66'
'detalle'	'80 kB'	'76'
'concentracion'	'64 kB'	'112'
'personal'	'56 kB'	'113'
'compra'	'72 kB'	'176'
'laboratorio_usuario'	'24 kB'	'193'
'metodo'	'160 kB'	'363'
'detallemetodo'	'136 kB'	'1061'
'producto'	'1312 kB'	'1374'
'existencias'	'3512 kB'	'2480'
'orden_trabajo'	'888 kB'	'3488'
'muestra'	'800 kB'	'4002'
'ordeninventario'	'1128 kB'	'5852'

'cliente'	'8192 kB'	'8851'
'servicio'	'9976 kB'	'10430'
'detalleorden'	'1840 kB'	'11318'
'proforma'	'4200 kB'	'16898'
'movimientosinventario'	'4664 kB'	'22280'
'detalle_proforma'	'6800 kB'	'44337'
'saldo_existencia'	'19 MB'	'132605'

Tabla 11: Número y tamaño de registros en entidades de SISLAB

Acorde a la información recopilada se procede a estructurar una clasificación de lectura y escritura intensiva de todas las tablas de la base de datos:

TABLA	LECTURA INTENSIVA	ESCRITURA INTENSIVA
bodega	X	-
bodega_usuario	-	-
caracteristica	X	-
característica_usuario	-	-
cargospersonal	X	-
cliente	X	X
compra	X	X
concentracion	X	X
control_existencia_metodo	X	X
detalle	X	X
detalle_proforma	X	X
detallemetodo	X	X
detalleorden	X	X
estadoproducto	X	-
existencias	X	X
grado	X	-
hidratacion	X	-
laboratorio	X	-

laboratorio_usuario	-	-
metodo	X	X
movimientos_inventario	X	X
muestra	X	X
norma	-	-
orden_trabajo	X	X
ordeninventario	X	X
personal	X	X
posgiro	X	-
presentacion	X	-
producto	X	X
proforma	X	X
proveedor	X	-
pureza	-	-
riesgoespecifico	X	-
saldo_existencia	X	X
servicio	X	X
tipo_justificacion	-	-
tipocliente	-	-
tipopersonal	-	-
tipoproducto	-	-
tipoproveedor	-	-
tipordeninv	-	-
tiposervicio	-	-
unidad	X	-
unidadmedida	X	-

Tabla 12: Lectura y Escritura Intensiva en Entidades de SISLAB

A continuación, se muestra un resumen de la información recopilada:

TABLA	Tablas sensibles por número de relaciones	Tablas sensibles por escritura de datos	Tablas sensibles por lectura de datos
bodega			
bodega_usuario			
caracteristica			
característica_usuario			
cargospersonal			
cliente			
compra			
concentracion			
control_existencia_metodo			
detalle			
detalle_proforma			
detallemetodo			
detalleorden			
estadoproducto			
existencias			
grado			
hidratacion			
laboratorio			
laboratorio_usuario			
metodo			
movimientos_inventario			
muestra			
norma			
orden_trabajo			
ordeninventario			
personal			
posgiro			
presentacion			
producto			
proforma			
proveedor			
pureza			
riesgoespecifico			
saldo_existencia			
servicio			
tipo_justificacion			
tipocliente			
tipopersonal			
tipoproducto			
tipoproveedor			
tipordeninv			
tiposervicio			
unidad			
unidadmedida			

Tabla 13: Resumen de Tablas sensibles en SISLAB

Con esta información podemos establecer una segunda propuesta de descomposición para la base de datos, en donde se resalta una distribución adecuada de las tablas sensibles previamente identificadas:

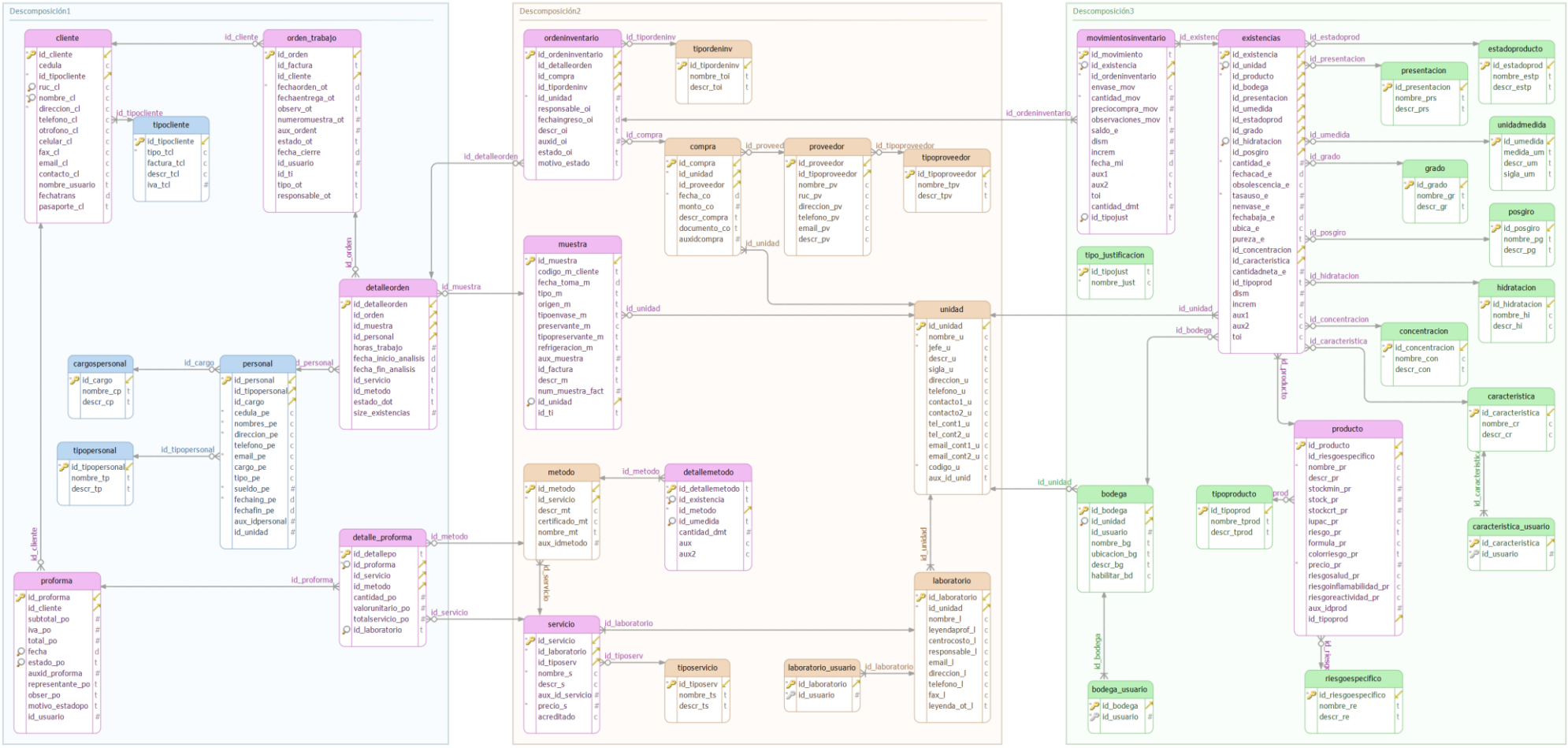


Fig. 35: Segunda Propuesta de Descomposición de BD de SISLAB

Al analizar las dos propuestas se debe verificar que exista la máxima independencia posible entre cada componente, por ende, el número de llamadas de las clases a las tablas de la base de datos debe ser mayor entre el microservicio del modelo de dominio con su base de datos correspondiente (Por ejemplo, MS1 con BD1) y menor con las bases de los otros microservicios (Por ejemplo, MS1 con BD2).

Para poder constatar esto, se estructura una matriz con las llamadas de la capa de persistencia del sistema y las dos propuestas de descomposición de la base de datos:

		Propuesta1						Propuesta2					
		BD1	BD2	BD3	BD4	BD5		BD1	BD2	BD3	BD4	BD5	
MS1	Cliente	2	0	0	0	0		2	0	0	0	0	
	DetalleProforma	5	0	0	0	0		2	3	0	0	0	
	DetalleProformaAux	3	0	0	0	1		0	3	0	0	1	
	DetalleTransferenciaInterna	5	0	0	2	0		2	3	0	2	0	
	DetalleMetodo	2	0	3	0	1		0	2	3	0	1	
	Metodo	4	0	1	0	0		0	5	0	0	0	
	Laboratorio	2	0	1	0	0		0	3	0	0	0	
	LaboratorioUsuario	2	1	1	0	0		1	3	0	0	0	
	Proforma	3	1	1	0	0		4	1	0	0	0	
	Servicio	3	0	1	0	0		0	4	0	0	0	
	TipoCliente	1	0	0	0	0		1	0	0	0	0	
	TipoServicio	1	0	0	0	0		0	1	0	0	0	
		33	2	8	2	2	47	12	28	3	2	2	47
		BD1	BD2	BD3	BD4	BD5		BD1	BD2	BD3	BD4	BD5	
MS2	CargosPersonal	1	0	0	0	0		1	0	0	0	0	
	DetalleMuestra	3	0	0	0	0		2	1	0	0	0	
	DetalleOrden	0	1	0	0	0		1	0	0	0	0	
	DetalleOrdenTrabajo	3	4	0	0	0		3	4	0	0	0	
	DetalleOrdenTrabajoAux	3	2	0	2	1		1	4	0	2	1	
	GenericoDetalleOrden_Compra	0	1	0	0	0		1	0	0	0	0	
	Muestra	2	1	0	2	0		0	3	0	2	0	
	OrdenTrabajo	2	2	1	1	0		4	1	0	1	0	
	Personal	1	3	0	0	0		3	1	0	0	0	
	TipoPersonal	0	1	0	0	0		1	0	0	0	0	
	TrabajoAnalista	2	4	0	0	0		3	3	0	0	0	
		17	19	1	5	1	43	20	17	0	5	1	43
		BD1	BD2	BD3	BD4	BD5		BD1	BD2	BD3	BD4	BD5	
MS3	Bodega	0	0	2	0	0		0	0	2	0	0	
	BodegaUsuario	0	0	3	0	0		0	1	2	0	0	
	Caracteristica	0	0	1	0	0		0	0	1	0	0	
	Compra	0	0	3	0	0		0	3	0	0	0	
	Concentracion	0	0	1	0	0		0	0	1	0	0	
	EstadoProducto	0	0	1	0	0		0	0	1	0	0	
	Existencias	0	0	2	0	0		0	0	2	0	0	
	Grado	0	0	1	0	0		0	0	1	0	0	
	Hidratacion	0	0	1	0	0		0	0	1	0	0	
	MovInventario	2	0	2	0	1		0	2	2	0	1	
	OrdenInventario	0	0	4	0	0		0	2	2	0	0	
	Posgiro	0	0	1	0	0		0	0	1	0	0	
	Presentacion	0	0	1	0	0		0	0	1	0	0	
	Producto	0	0	2	0	0		0	0	2	0	0	
	Proveedor	0	0	2	0	0		0	2	0	0	0	
	RiesgoEspecifico	0	0	1	0	0		0	0	1	0	0	
	TipoJustificacionMov	0	0	1	0	0		0	0	1	0	0	
	TipoOrdenInventario	0	0	1	0	0		0	1	0	0	0	
	TipoProducto	0	0	1	0	0		0	0	1	0	0	
	TipoProveedor	0	0	1	0	0		0	1	0	0	0	
	Unidad	0	0	1	0	0		0	1	0	0	0	
	UnidadMedida	0	0	1	0	0		0	0	1	0	0	
		2	0	34	0	1	37	0	13	23	0	1	37
		BD1	BD2	BD3	BD4	BD5		BD1	BD2	BD3	BD4	BD5	
MS4	Factura	3	0	1	2	0		1	3	0	2	0	
	Detalleactura	2	0	0	2	1		0	2	0	2	1	
	EstadoFactura	0	0	0	1	0		0	0	0	1	0	
	MovimientoCaja	0	0	0	1	0		0	0	0	1	0	
	Pagos	0	0	0	1	0		0	0	0	1	0	
	TransferenciaInterna	3	0	0	2	0		0	3	0	2	0	
		8	0	1	9	1	19	1	8	0	9	1	19

Fig. 36: Llamadas de la capa de negocio a propuestas de descomposición en BD

Se puede observar con color rojo cuando el número de relaciones entre las clases y la base de datos excede de la norma establecida. Es así como se realiza una modificación entre las tablas de la base uno con la base dos para corregir y obtener un modelo más cercano a lo deseado, teniendo como resultado una tercera propuesta:

		Propuesta3					
		BD1	BD2	BD3	BD4	BD5	
MS1	Cliente	2	0	0	0	0	
	DetalleProforma	4	1	0	0	0	
	DetalleProformaAux	2	1	0	0	1	
	DetalleTransferenciaInterna	4	1	0	2	0	
	DetalleMetodo	2	0	3	0	1	
	Metodo	2	3	0	0	0	
	Laboratorio	1	2	0	0	0	
	LaboratorioUsuario	1	3	0	0	0	
	Proforma	3	2	0	0	0	
	Servicio	2	2	0	0	0	
	TipoCliente	1	0	0	0	0	
	TipoServicio	1	0	0	0	0	
		25	15	3	2	2	47
		BD1	BD2	BD3	BD4	BD5	
MS2	CargosPersonal	1	0	0	0	0	
	DetalleMuestra	3	0	0	0	0	
	DetalleOrden	0	1	0	0	0	
	DetalleOrdenTrabajo	2	5	0	0	0	
	DetalleOrdenTrabajoAux	2	3	0	2	1	
	GenericoDetalleOrden_Compra	0	1	0	0	0	
	Muestra	0	3	0	2	0	
	OrdenTrabajo	2	3	0	1	0	
	Personal	0	4	0	0	0	
	TipoPersonal	0	1	0	0	0	
	TrabajoAnalista	2	4	0	0	0	
		12	25	0	5	1	43
		BD1	BD2	BD3	BD4	BD5	
MS3	Bodega	0	0	2	0	0	
	BodegaUsuario	0	1	2	0	0	
	Caracteristica	0	0	1	0	0	
	Compra	0	0	3	0	0	
	Concentracion	0	0	1	0	0	
	EstadoProducto	0	0	1	0	0	
	Existencias	0	0	2	0	0	
	Grado	0	0	1	0	0	
	Hidratacion	0	0	1	0	0	
	MovInventario	2	0	2	0	1	
	OrdenInventario	0	0	4	0	0	
	Posgiro	0	0	1	0	0	
	Presentacion	0	0	1	0	0	
	Producto	0	0	2	0	0	
	Proveedor	0	0	2	0	0	
	RiesgoEspecifico	0	0	1	0	0	
	TipoJustificacionMov	0	0	1	0	0	
	TipoOrdenInventario	0	0	1	0	0	
	TipoProducto	0	0	1	0	0	
	TipoProveedor	0	0	1	0	0	
	Unidad	0	1	0	0	0	
	UnidadMedida	0	0	1	0	0	
		2	2	32	0	1	37
		BD1	BD2	BD3	BD4	BD5	
MS4	Factura	2	2	0	2	0	
	Detalleactura	1	1	0	2	1	
	EstadoFactura	0	0	0	1	0	
	MovimientoCaja	0	0	0	1	0	
	Pagos	0	0	0	1	0	
	TransferenciaInterna	1	2	0	2	0	
		4	5	0	9	1	19

Fig. 37: Llamadas de capa de negocio a tercera propuesta de descomposición en BD

Como mayor referencia se puede observar que el número de relaciones entre “MSn” con “BDn” es más alto con el tercer modelo de descomposición.

Para una mejor asimilación de qué propuesta de descomposición es la óptima, tenemos los siguientes gráficos que simulan las llamadas de la capa de negocio a sus respectivas bases de datos, y en donde se busca que cada Microservicios de la capa de negocio tenga mayor dependencia con su respectiva base y mínima dependencia con las demás.

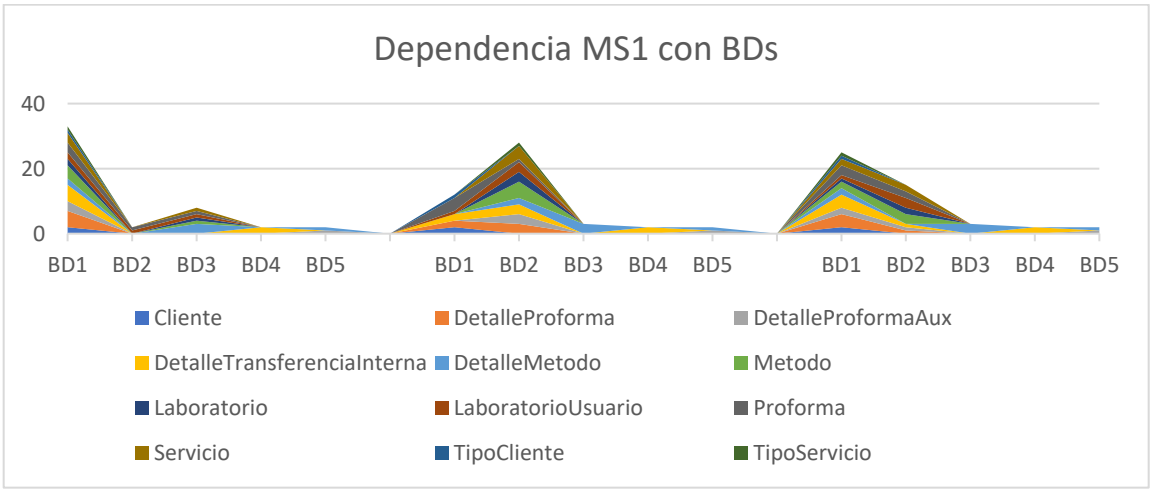


Fig. 38: Dependencia Microservicio 1 de Capa de Negocio con Base de Datos

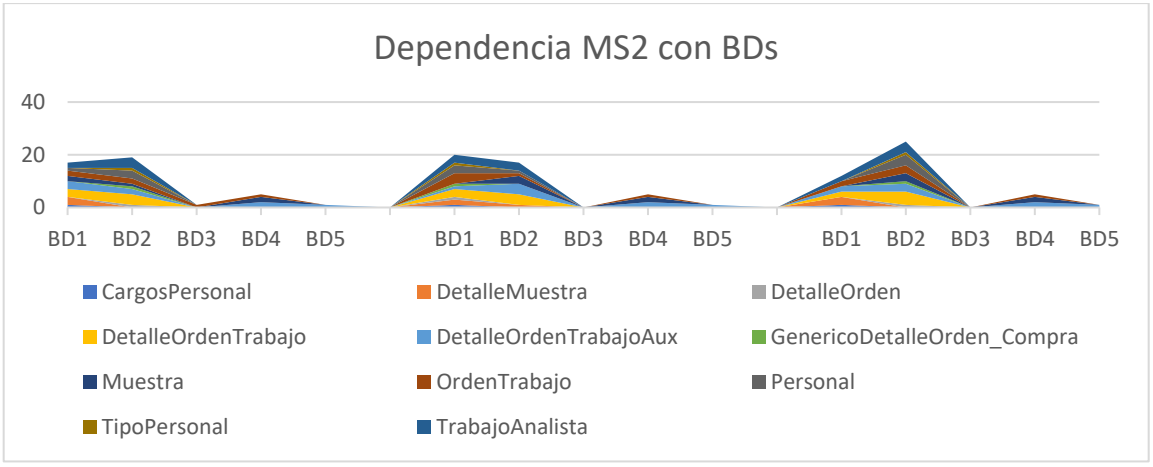


Fig. 39: Dependencia Microservicio 2 de Capa de Negocio con Base de Datos

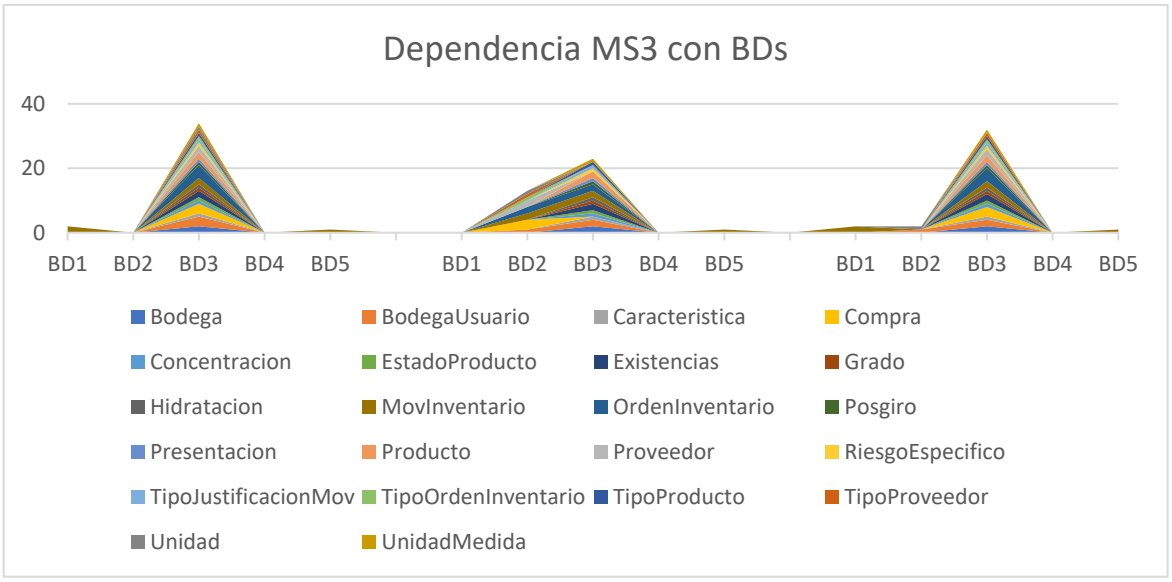


Fig. 40: Dependencia Microservicio 3 de Capa de Negocio con Base de Datos

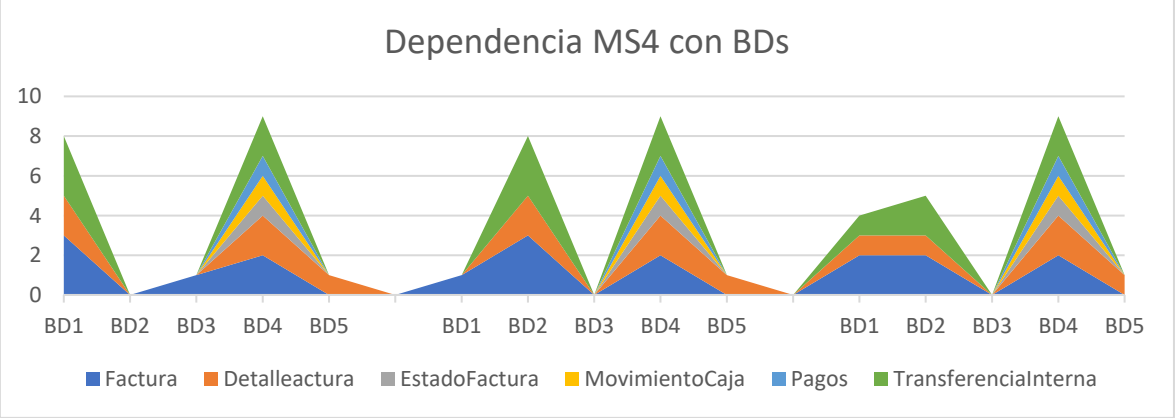


Fig. 41: Dependencia Microservicio 4 de Capa de Negocio con Base de Datos

Así tenemos la descomposición final de la base de datos que se adapta a nuestra descomposición del modelo de dominio:

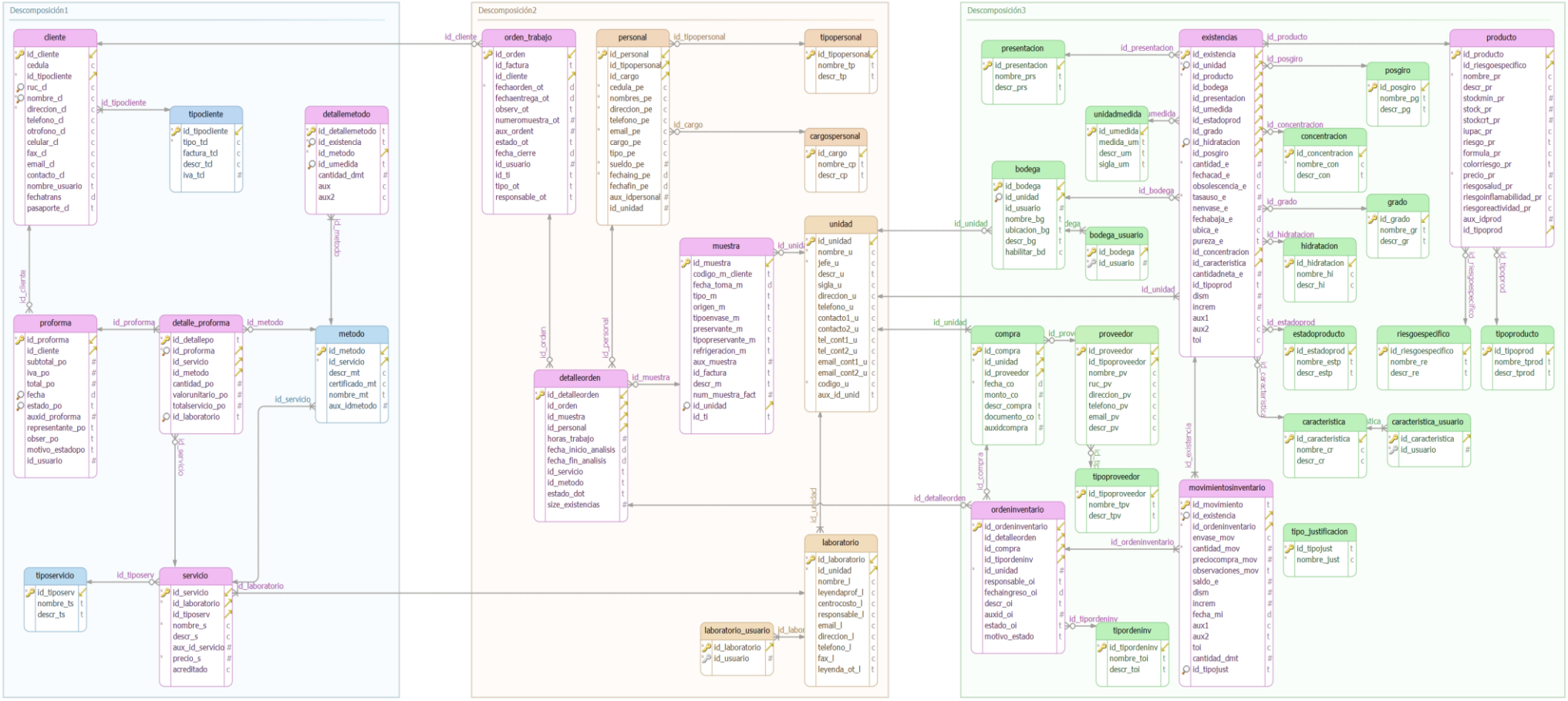


Fig. 42: Descomposición Final de Base de Datos en SISLAB

4.2.3. Etapa Tres - Asociación

En esta etapa como lo define la metodología, buscamos asociar los esquemas propuestos en la etapa de descomposición y acoplarlos dentro del ecosistema previamente definido.

CAPA CUATRO - MICROSERVICIOS

Partimos de la capa cuatro en donde se definen nuestros microservicios previamente descompuestos, teniendo en cuenta que BD4 posee todas las tablas referentes a facturación y BD5 que posee las tablas que no tienen relaciones directas con las demás tablas, como lo son: detalle; saldo_existencia; y control_existencia_metodo. Por su parte cada microservicio, será almacenado en un contenedor que nos facilite su administración y despliegue. Dando como resultado el primer diagrama de relación entre la capa de negocio y la capa de datos en SISLAB.

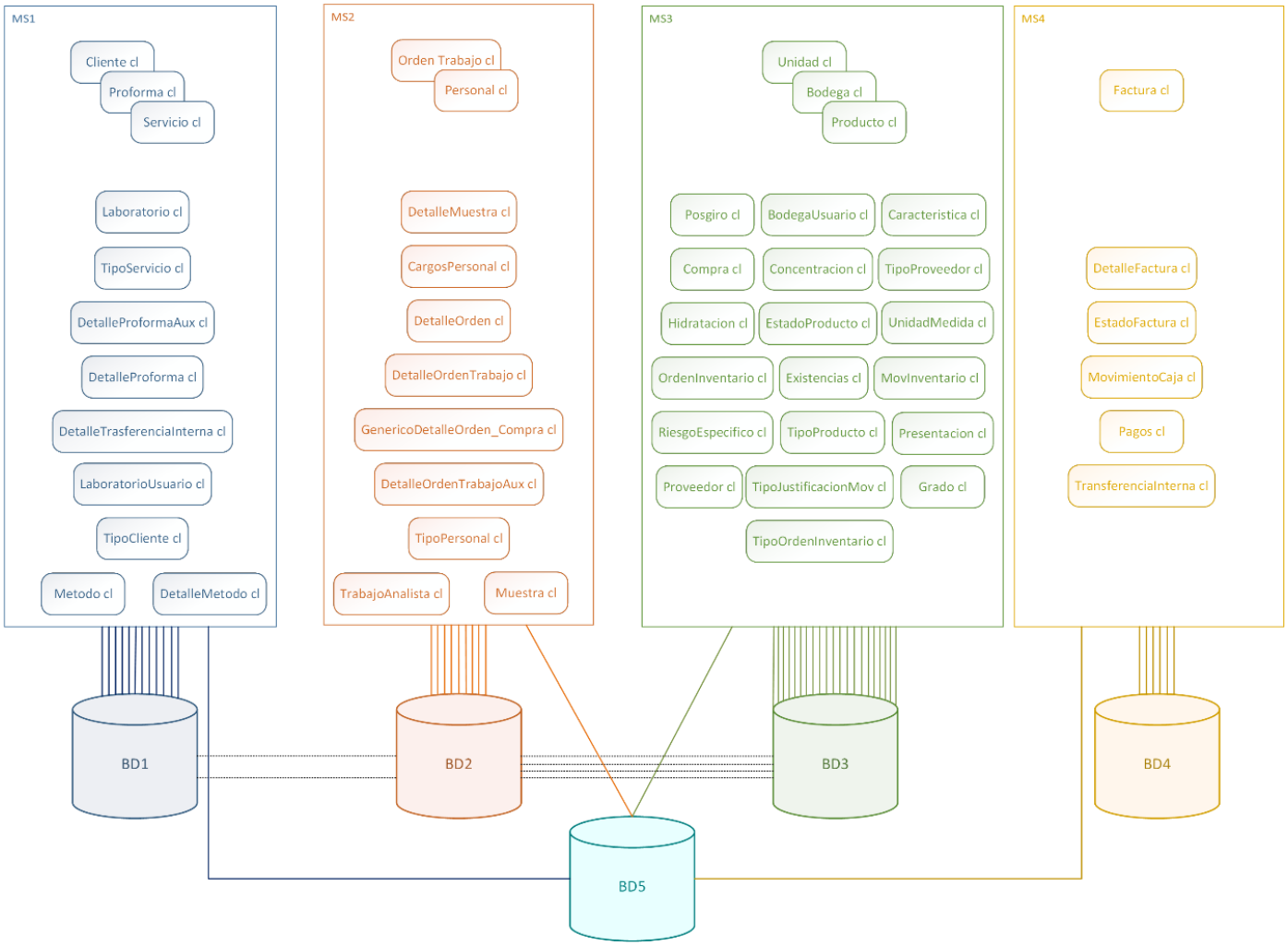


Fig. 43: Relación Capa Negocio y Datos en SISLAB Descompuesto

Para poder analizar de mejor manera elaboramos un diagrama que contenga ya el esquema en conjunto con todos los componentes internos que posera cada microservicio, además de las relaciones que posee entre ellos. Como se observa en la Figura 44, ahora tenemos nuestra primera estructura en la capa cuatro del ecosistema de microservicios.

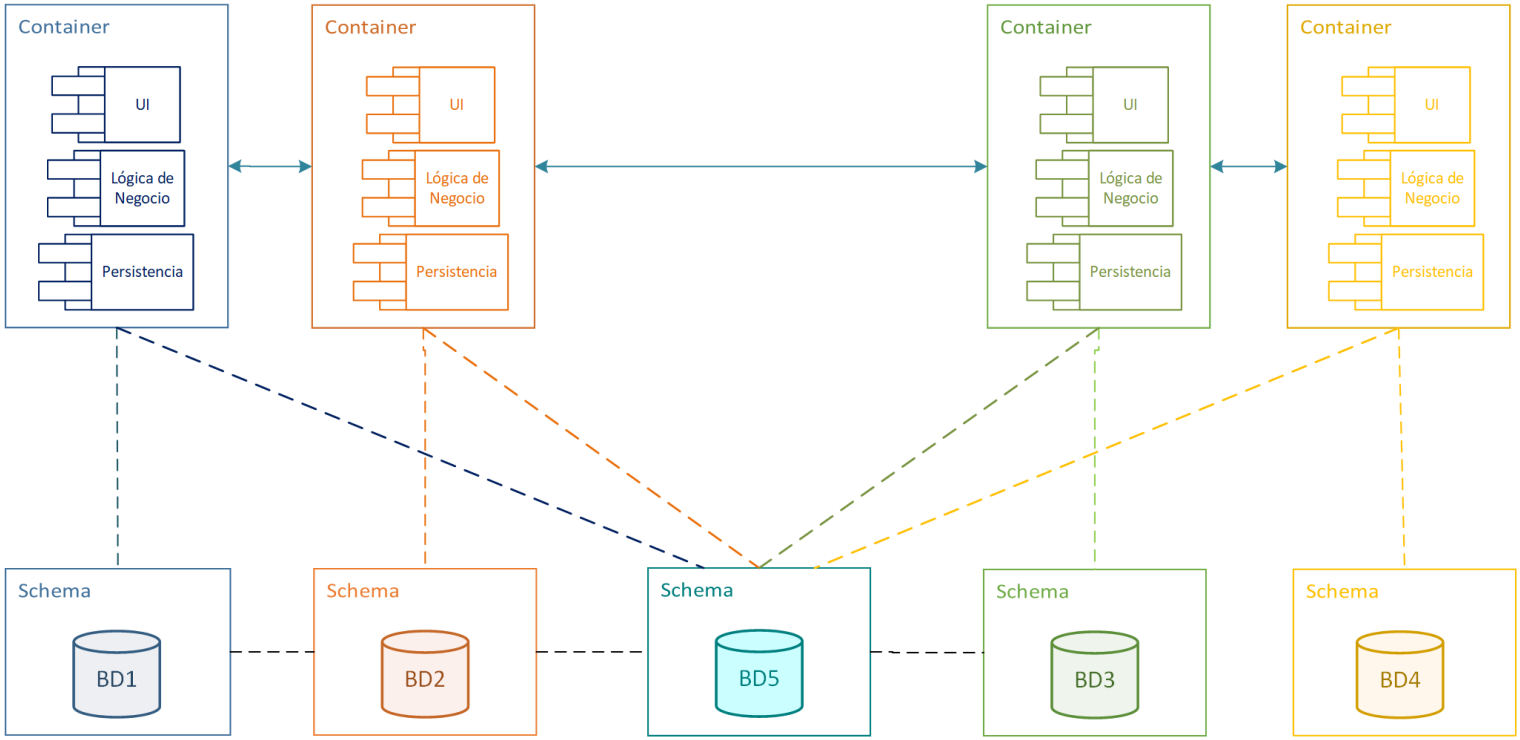


Fig. 44: Estructura SISLAB en capa cuatro de Microservicios

CAPA TRES – PLATAFORMA DE APLICACIÓN

En esta capa procedemos a ubicar los componentes que se consideren necesario para el buen funcionamiento y despliegue de microservicios.

Como se menciona en SMMicro esta capa propuesta inicialmente por Susan Fowler nos permite establecer tres componentes claves para la monitorización: Logging, Dashboards, Alertas.

- Comenzamos con el análisis de la necesidad de un registro de logs para dos de los cuatro microservicios establecidos puesto que mantener almacenado demasiada información a largo plazo se vuelve contraproducente, por ende, se elegirán únicamente a los dos principales, en base a las necesidades de negocio. Estos dos microservicios serian:

  1. MS1 dedicado netamente a cliente, proformas y servicios.
  2. MS4 dado que siempre se debe guardar el registro de facturas por cuestiones del Servicio de Rentas Internas.

- Como segundo punto analizamos la utilización de un dashboard para todos los microservicios, en este caso existen herramientas que nos facilitan su utilización, sin la necesidad de uso innecesario de recursos, por lo cual MS1, MS2, MS3 y MS4 serán monitoreados por igual.

- En el caso de las alertas, no se podrían implementar aun en SISLAB puesto que como se menciona en SMMicro, debe existir un claro registro previo del comportamiento del sistema, esto nos permitirá delimitar cuando un microservicio se está ejecutando de forma normal o peligrosa.

- Como herramientas adicionales y como se menciona en la propuesta metodológica, podemos aplicar un Message Broker ya sea en esta capa o en la de comunicación, para nuestro caso es primordial este componente ya que necesitamos administrar diferentes esquemas de la base de datos para cada microservicio, y, a su vez mantener la transaccionalidad, llevando un registro que nos permita revisar cuando algún evento haya fallado para corregirlo de inmediato.

El esquema en esta capa del ecosistema de microservicios sería de la siguiente manera:

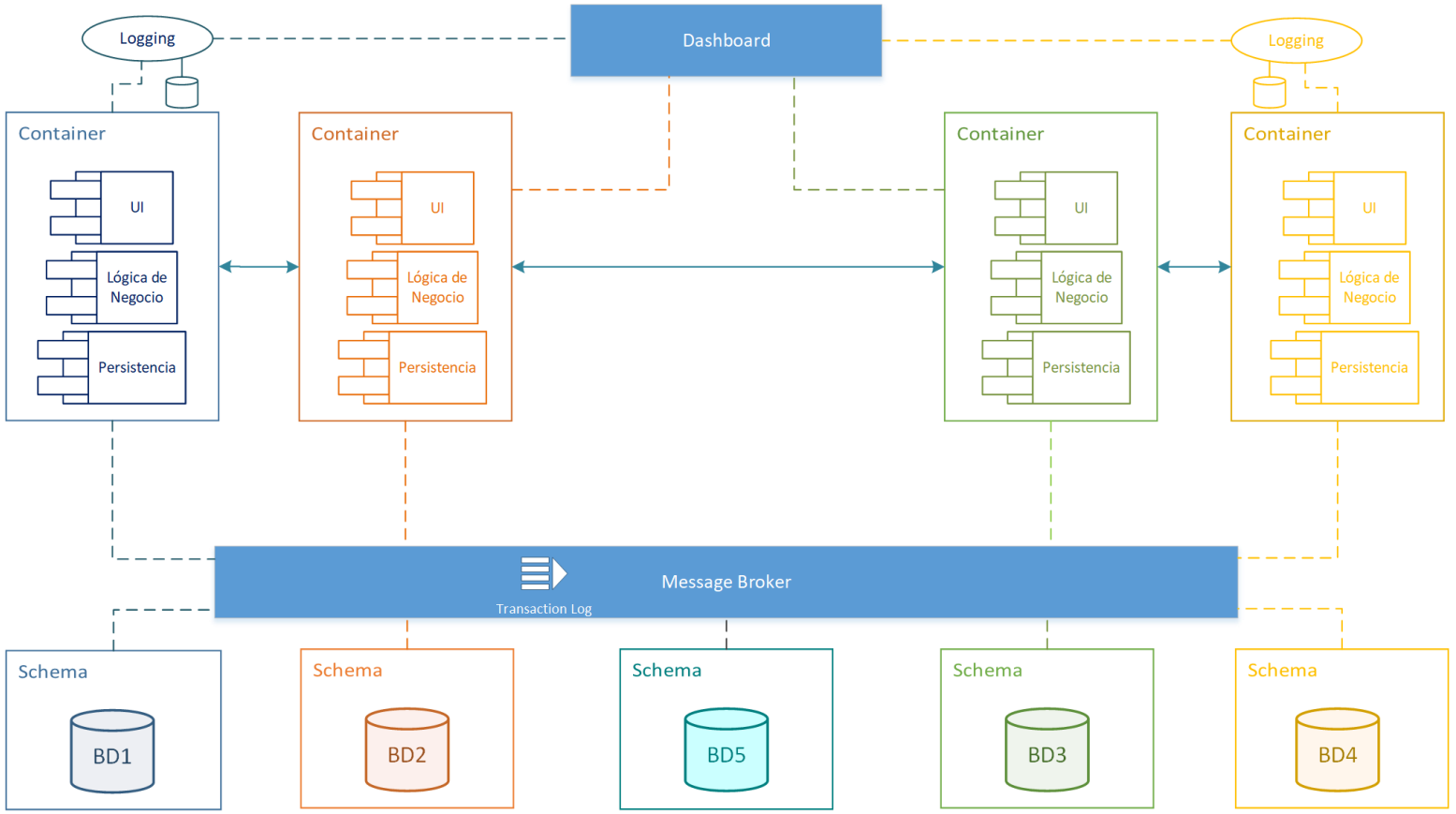


Fig. 45: Estructura SISLAB en capa tres de Microservicios

CAPA DOS – COMUNICACIÓN

En la capa de comunicación analizaremos la factibilidad de implementación de: Discovery Server, Load Balancing, Circuit Breaker, Edge Server. Teniendo en consideración que SISLAB propende a crecer en los próximos años y necesita tener los complementos necesarios para escalar sin disminuir su operatividad.

- El primer punto por analizar es la implementación de un Discovery server o servidor de registro que nos permita reconocer todas las instancias de puertos que posean los microservicios. En nuestro caso lo utilizaremos puesto que se convierte en un componente muy importante para reconocer en qué dirección se encuentran registrados nuestros servicios. A su vez se sugiere utilizar un servidor de configuración que centralice todas las configuraciones adicionales que requiera cada servicio en producción.
- Como Edge Server se propone una implementación de un API Gateway similar al que ya utilizaba el sistema monolito, que facilite un punto de acceso único para los diferentes tipos de clientes que utilizaran el sistema y por ende debe estar acoplado con el servidor de registro.
- El balanceador de carga a este nivel nos facilitara la ubicación, utilización y administración de los servicios que serán consumidos por el usuario. En nuestro estudio de caso se lo implementa en base al escalamiento de múltiples servicios que a futuro puede requerir SISLAB.

Como menciona SMMicro en este punto se debe tener en cuenta la necesidad de protocolos y herramientas de seguridad en la comunicación entre microservicios, ya que puede ser comprometida. Para el SISLAB se recomienda utilizar mecanismos de encriptado y desencriptado con el fin de que los endpoints reconozcan únicamente aquellos paquetes autorizados previamente. Como herramienta adicional se podría utilizar un Web Application Firewall (WAF) [36], ya que existen distribuciones Open Source que facilitan la especificación de los parámetros y reglas que el aplicativo tendrá para casos puntuales, esto es útil ya que, durante la migración se pueden encontrar puntos de fallo concretos que necesiten una regla especifica de mitigación del problema, además el uso de este tipo de firewalls facilita configuraciones precargadas para la prevención de ataques como inyección SQL, inclusión de archivos remotos, violaciones de protocolo http o inyección de código malicioso.

El esquema en esta capa del ecosistema de microservicios sería de la siguiente manera:

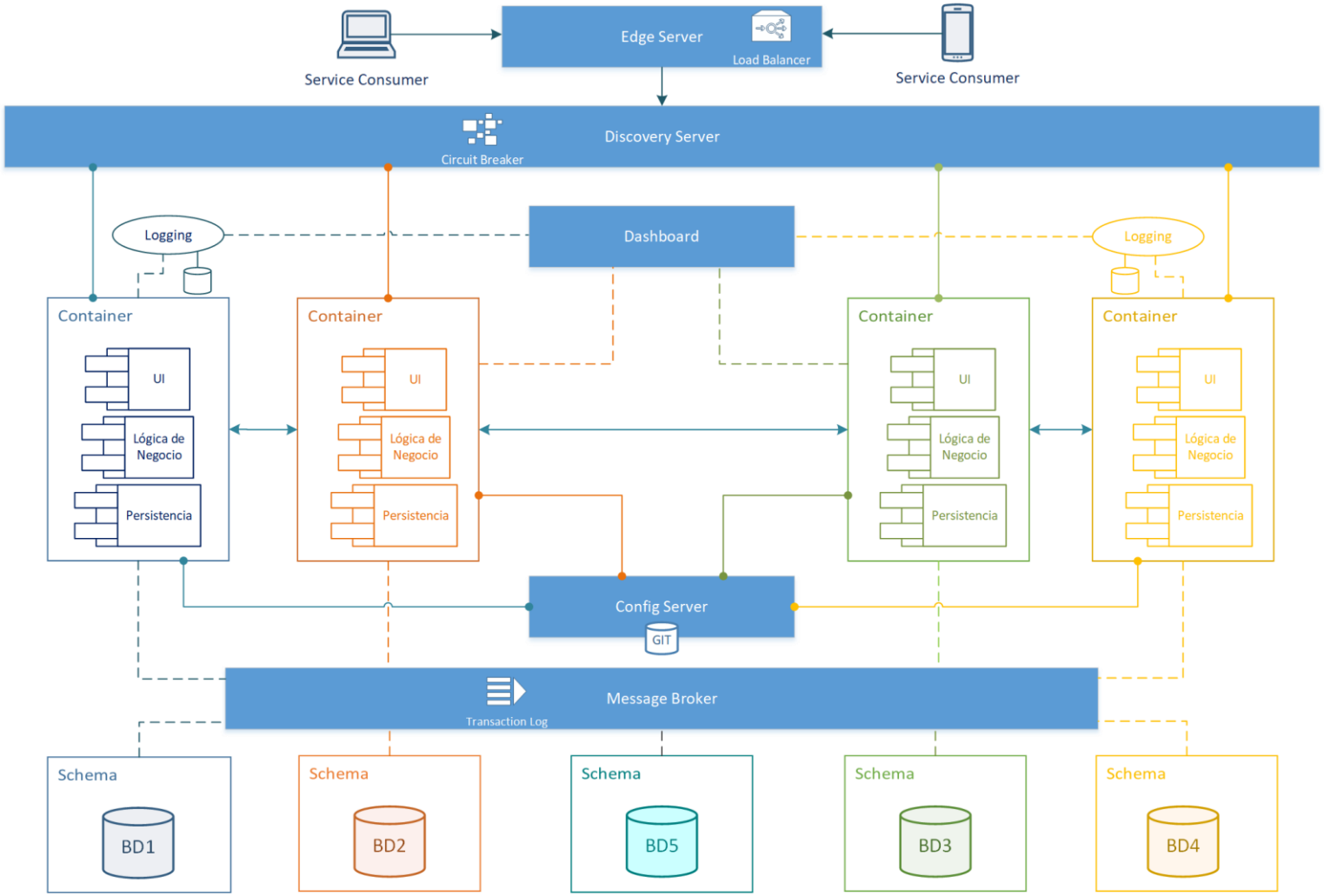


Fig. 46: Estructura SISLAB en capa dos de Microservicios

CAPA UNO – HADWARE

Para la migración a microservicios acorde al entorno encontrado en SISLAB, hemos optado por varias tecnologías IaaS y PaaS que se puedan adaptar y encajar con los servidores físicos que posee la DGIP.

Actualmente ya existen varias soluciones cloud y plataformas como Amazon AWS [37], Istio [38], Microprofile [39] o Microsoft Azure [40], sin embargo, la mayoría de herramienta para ese sistema especifico las vamos a administrar en base al stack tecnológico que nos ofrece Spring Cloud [41] y Netflix OSS [42], ya que ambas se enfocan en la administración de servicios e incluyen una gama de herramientas que necesitamos para nuestro ecosistema de microservicios, así pues:

- Partiendo de los contenedores se propone utilizar Docker [43], ya que nos permite manejar de forma adecuada nuestros microservicios, así como los recursos que consumen e inclusive se los puede utilizar para realizar pruebas en ambientes de desarrollo, prueba y despliegue.
- En cuanto a la base de datos, por el momento ya que los datos no requieren ningún tratamiento especial, podemos seguir utilizando Postgres SQL y dependiendo la disponibilidad en los servidores físicos, realizar una división en diferentes bases de datos o manejar una misma base, pero con varios esquemas para cumplir con la independencia entre microservicios. En este punto para el componente Message Broker que se planteó, se sugiere la utilización de Eventuate [44] para el manejo de eventos y estados que los almacena en el log de transacciones que nos garantiza una transaccionalidad robusta.
- Ya que el sistema originalmente fue desarrollado en Java, *Spring Cloud Config* nos facilita un acoplamiento para soporte de servidor y cliente que a su vez utiliza GIT para almacenamiento de código, y por ende también tiene mecanismos de encriptación y desencriptación en la comunicación.
- De la misma manera *Spring Cloud Netflix* provee integración con Netflix OSS y así nos permite utilizar *Zuul* como el proxi que necesitamos como servidor perimetral, que a su vez se acopla con *Ribbon* como balanceador de carga, *Eureka* como servidor de descubrimiento e *Hystrix* como cortador de circuitos e inclusive como dashboard puesto que esta herramienta como el servidor de registro o descubrimiento nos facilitan mecanismos de monitorización en tiempo real del estado de nuestros microservicios.

El esquema en esta capa del ecosistema de microservicios sería de la siguiente manera:

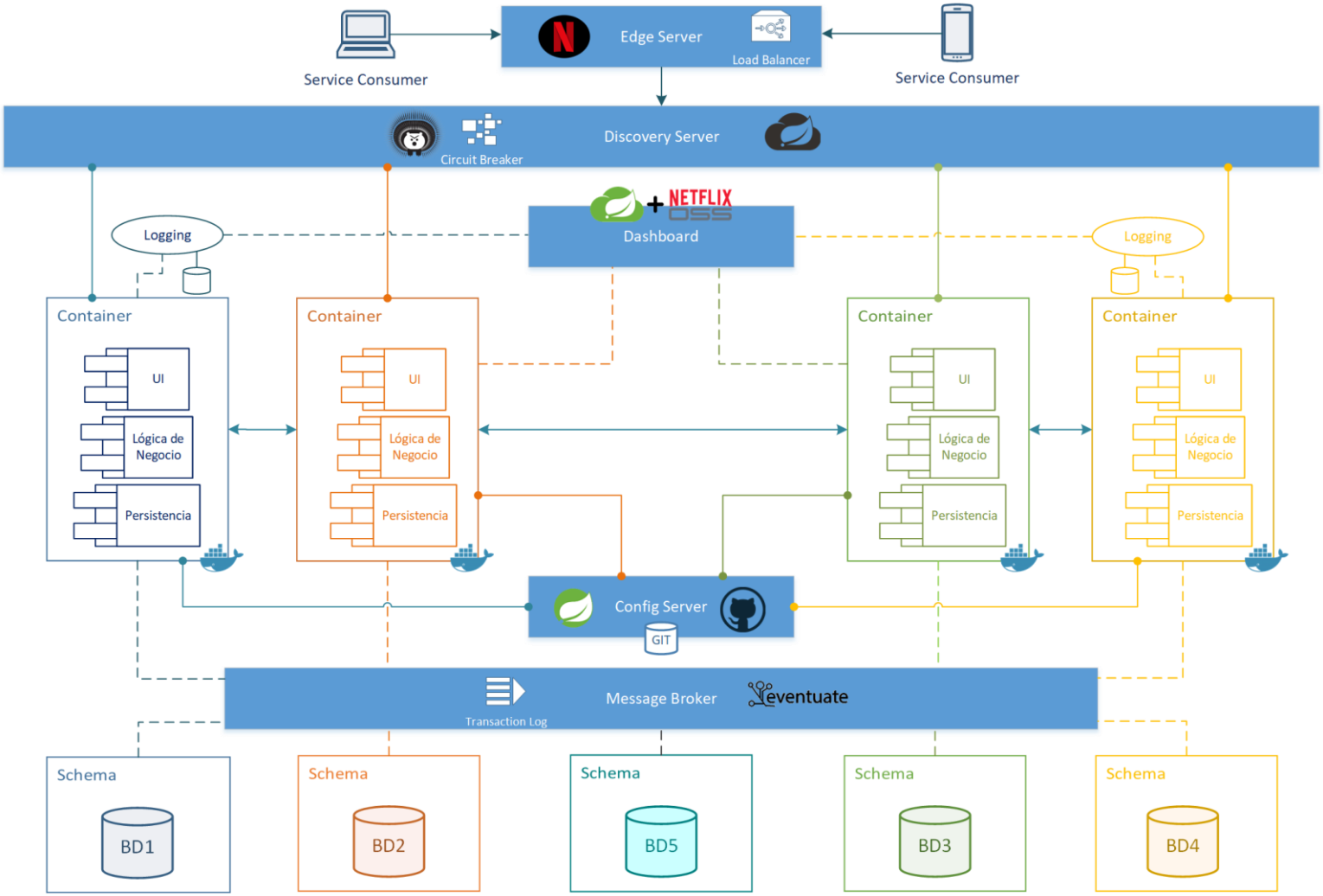


Fig. 47: Estructura SISLAB en capa uno de microservicios

Debemos tener en consideración que el diagrama anterior sería factible si únicamente utilizáramos Cloud Publica, y si el Sistema de Administración de Laboratorios fuera el sistema macro dentro de la EPN. Sin embargo, en la realidad como ya se mencionó, SISLAB es parte del Sistema Integrado de Información, y por ende debemos tener en cuenta el diagrama de la figura 27 que nos muestra cómo se encuentran en producción los servidores de la DGIP, en donde se evidencia que necesitamos tanto tecnología en la nube privada, como publica, y por ende trabajaremos con Cloud Hibrida [45] incluyendo las siguientes observaciones:

- En la DGIP se maneja tecnología basada en la nube de RedHat, por ende, el stack tecnológico para microservicios lo maneja el JBoss Middleware [46], y entrando en concreto a los servidores que actualmente posee la DGIP, SISLAB podría alojarse dividido en contenedores en el servidor *SRVJBOSS1.*
- De igual forma podemos utilizar el bus de aplicaciones empresariales *SRVFUSE* para montar nuestros servicios, así como haciendo uso de OpenShift [47] estableceríamos la administración de los contenedores y utilizaríamos sus funciones de Dashboard y Logging. Cabe mencionar que al utilizar un bus de servicios debemos definir claramente la estructura de las APIs que se conectaran a él, o podemos caer en limitaciones de tipo SOA.
- En cuanto a las bases de datos, podríamos alojar diferentes esquemas en el SRVBDD1 que se encuentra funcional, pero con diferentes esquemas para mantener la independencia de los microservicios. En cuanto a su administración se puede utilizar el servidor SRVBALBDD, u otra opción puede ser montar Eventuate [44], para aprovechar sus funcionalidades con logs de transacciones.
- Se identifica además que el servidor perimetral podría ser omitido, puesto que ya se cuenta con NetScaler [48], SERVWEB1 y SERVWEB2 para la exposición de aplicaciones al cliente, así como su balanceo de carga.
- En lo que respecta a seguridad en la comunicación, RedHat posee una gama de herramientas exclusivas para microservicios [49], de esta manera la EPN podría solicitar un convenio para fortalecer su arquitectura y de este modo explotar los beneficios de Cloud Hibrida.

Así pues, tenemos una asimilación del diagrama de ecosistemas de microservicios de SISLAB, utilizando algunos de los componentes y servidores que ya posee la DGIP para manejar el SII como se muestra en la figura 48.

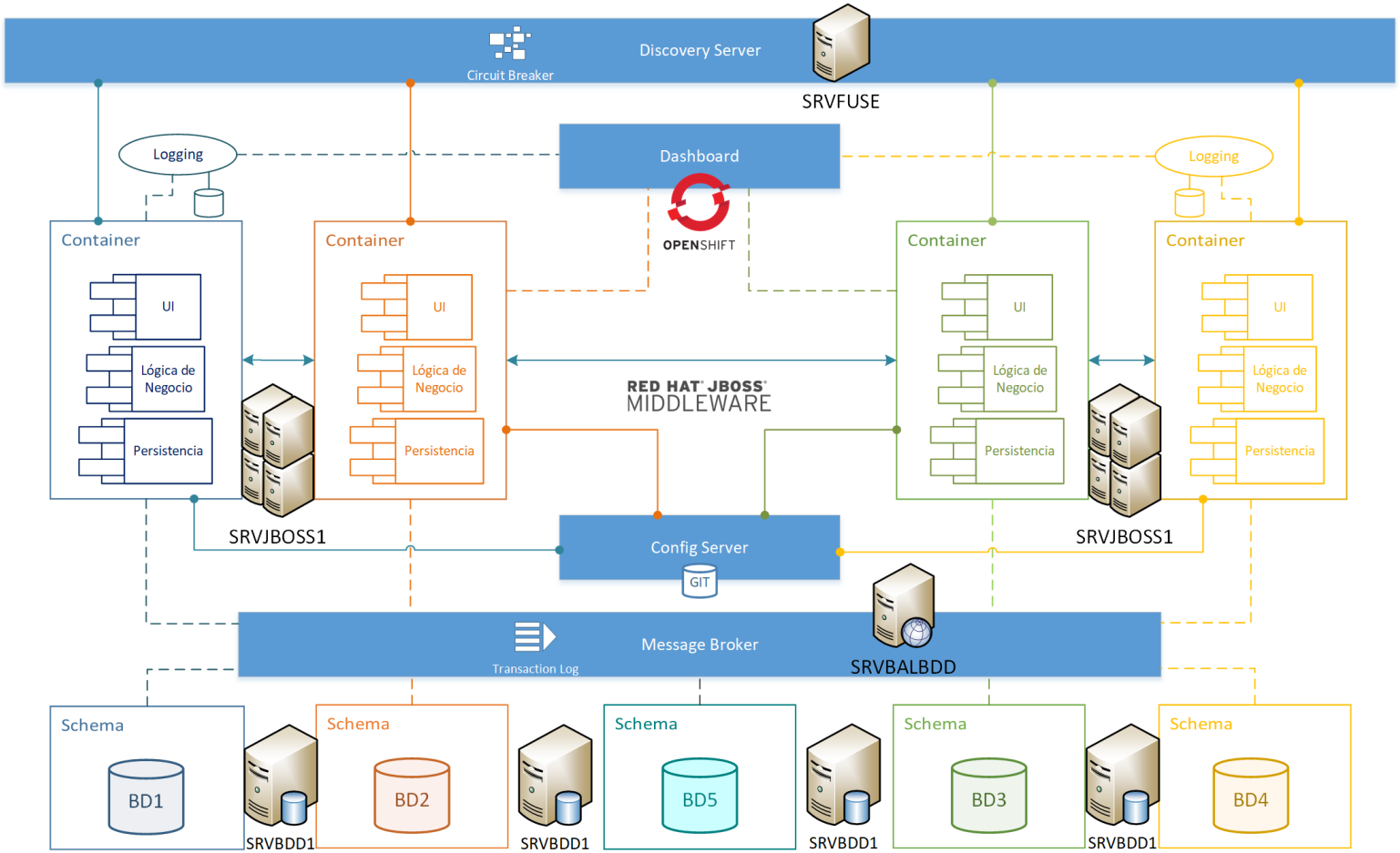


Fig. 48: SISLAB en SII

4.2.4. Etapa Cuatro – Validación

Para poder culminar con la última etapa del modelo procedemos a validar que la arquitectura creada cumple con algunos de los requerimientos que poseen los microservicios. Para esto procedemos a comprobar con las preguntas establecidas en lo que respecta a estabilidad, confiabilidad, escalabilidad y rendimiento. Sin embargo, como menciona la propuesta metodológica se deben determinar algunas consideraciones previas:

Consideraciones para producción

Patrones

Existen varios factores a tomar en cuenta para un despliegue del diagrama construido durante toda esta fase de planificación, como primer punto tenemos que analizar la forma en que realizaremos los cambios sobre el monolito para cumplir con nuestros objetivos. Para esto utilizaremos el patrón de estrangulación de aplicaciones “Strangler Application Pattern” propuesto inicialmente por Martin Fowler en 2014, pero desarrollado a profundidad por Kyle G. Brown [50] en donde consta de tres etapas:

- Transformar: Consiste en crear un sitio paralelo con un servicio que cumpla una funcionalidad en específico. En nuestro caso los cuatro microservicios planteados.
- Coexistir: Consiste en redirigir del sitio existente al nuevo para que la funcionalidad se implemente de forma incremental.
- Eliminar: Consiste en ir eliminando la funcionalidad anterior del monolito (o simplemente deje de mantenerla) a medida que el tráfico se redirecciona a los nuevos microservicios.

Estrategias

En cuanto a estrategias de desarrollo es importante considerar Devops como lo menciona Balalaie en otro de sus artículos [51], como algo intrínseco si se quiere construir microservicios eficientes y robustos, ya que obligatoriamente como se ha mostrado a lo largo de este análisis, no solo implica inclusión de desarrolladores, sino también del personal de infraestructura y todos aquellos que en el departamento de TICs se relacionen con la aplicación desde su concepción hasta su puesta en producción.

Por último, todo esto debe ser englobado en una la línea de desarrollo o implementación estandarizada, para esto Susan Fowler [11] propone primero aplicarlo en un entorno de transición; en segundo lugar, si pasa la fase de etapas, implementarlo en un entorno de producción pequeño, donde se incluya un porcentaje menor del tráfico en producción; y

tercero, si pasa esta fase, podemos extenderlo lentamente a los servidores de producción hasta que se haya desplegado en cada host.

Preguntas de Evaluación

Para culminar con la validación procedemos a contestar las preguntas seleccionadas en SMMicro que fueron tomadas del libro de Susan Fowler [11].

Estabilidad y Confiabilidad

¿El microservicio tiene un repositorio central donde se almacena todo el código?
Si, en nuestro caso se plantea un repositorio en GitHub que a su vez sea administrado con el servidor de configuración de Spring o su equivalente en la familia de productos RedHat.
¿El ecosistema de microservicios tiene una línea de implementación estandarizada?
Si, para la línea de implementación se recomienda utilizar las fases de despliegue en producción de Susan Fowler [11].
¿Qué acceso tiene el entorno de transición a los servicios de producción?
Debido a la arquitectura que mantiene SISLAB dentro de SII, este tiene un Controlador de entrega de aplicaciones (ADS) que gestiona el acceso a los servicios, pero es importante recalcar que se puede incluir dentro del servidor de aplicaciones un servidor perimetral de soporte como Zuul.
¿Las implementaciones para la producción se hacen todas al mismo tiempo, o se implementan incrementalmente?
Se recomienda que sea incrementalmente con el patrón estrangulador ya explicado y sus tres etapas [50], transformar, coexistir y eliminar.
¿Dependencias de microservicios?
Si existe dependencia, sin embargo, como se vio durante el análisis se procuró minimizarlas al máximo tanto en lógica de negocio como en datos.
¿Cómo este microservicio mitiga los fallos de dependencia?
Los fallos de dependencia se los gestiona a través del bus de servicios de aplicaciones que actualmente posee la DGIP. O en su defecto se puede utilizar los servicios cloud de OpenShift [47].
¿Hay copias de seguridad, alternativas, retrocesos o almacenamiento en caché defensivo para cada dependencia?
Existe un log de transacciones que almacena la información referente a datos de cada dependencia y que a su vez nos permite mantener la transaccionalidad al analizarlo si se produjeran fallos.
¿Son confiables los controles de salud al microservicio?

Si, en nuestro caso se utiliza la monitorización de RedHat, en conjunto con los servidores de balanceo de carga y administración de aplicaciones mostrado en la figura 29.
¿Hay interruptores en su lugar para evitar que el tráfico de producción se envíe a hosts y microservicios poco saludables?
Si, las interrupciones serán administradas por Openshift o en su defecto por Hystrix y su monitor de rendimiento que ayudarán para la prevención a futuro.
¿Existen procedimientos para depreciar los puntos finales API de microservicios?
Si, se recomienda utilizar los procedimientos embebidos que posee Spring en su servidor de descubrimiento o registro.

Escalabilidad y Rendimiento

¿Cuáles son los cuellos de botella de recursos de estos microservicios?
En cuanto recursos para el SISLAB no existen cuellos de botella si se implementan las tecnologías cloud propuestas en conjunto con los servidores que posee la DGIP. Para que exista cuello de botella debería existir por lo menos una triplicación del uso de recursos que actualmente utiliza el aplicativo.
¿Escalarán las dependencias con el crecimiento esperado de este microservicio?
Las dependencias de cada microservicio son administradas de tal forma que cada vez que haya una expansión de microservicios, se aminore el impacto entre cada funcionalidad.
¿Se puede enrutar automáticamente el tráfico a otros centros de datos en caso de falla?
Si se podría, sin embargo, no se lo ha incluido como parte de análisis de este modelo. Aunque la herramienta Eventuate permite registrar los eventos que pudieron haber causado un fallo y permitir al administrador corregirlo rápidamente.
¿El microservicio está escrito en un lenguaje de programación que permitirá que el servicio sea escalable y funcional?
Dado que es una migración, se mantiene el lenguaje de origen, en este caso Java, que es escalable y funcional.
¿Hay alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio maneja las solicitudes?
Existe un alto grado de escalabilidad ya que las solicitudes son manejadas por el controlador de acceso de aplicaciones propio de la DGIP, o en su defecto si se opta por cloud se puede implementar el servidor perimetral que su vez posee un balanceador de carga gestionado por Zuul de Netflix OSS.
¿Existe alguna escalabilidad o limitaciones de rendimiento en la forma en que el microservicio procesa las tareas?

Las limitaciones en cuanto a procesamiento de tareas se refieren, dependerá de cómo sean configurado cada contenedor en su entorno de producción. Para esto es primordial el buen uso de OpenShift de RedHat.
¿Este microservicio maneja los datos de una manera escalable y eficiente?
Si, los datos se manejan independientemente para cada servicio y divididos de tal forma que se pueda escalar en caso de expansión.
¿Qué tipo de datos necesita cada microservicio para almacenar?
Los datos en nuestro estudio de caso son en su mayoría simples o básicos, además de ser los mismos para cada microservicio, por ende, es sencillo realizar una revisión del log de transacciones propuesto. No necesitamos por el momento herramientas de tipo Big Data o similares.
¿Cuál es el esquema necesario para sus datos?
Se maneja un esquema relacional que permita almacenar los eventos importantes que cada microservicio realiza.
¿Cada microservicio necesita un mayor rendimiento de lectura o escritura?
Si, como se mostró en varios análisis y tablas, esto depende de los requerimientos que inicialmente tuvo el aplicativo, en donde se agrupo servicios críticos en base a la lógica de negocio y datos.
¿Son de lectura pesada, de escritura pesada o ambas cosas?
Los microservicios de SISLAB propuestos son de lectura intensa, puesto que es utilizada por todos los laboratorios de la EPN. Para mayor detalle se puede visualizar cada entidad en la tabla 12.
¿La base de datos se escala horizontal o verticalmente? ¿Es replicado o particionado?
Escala horizontalmente puesto que existe un acople con el servidor SRVBALBDD que gestiona el aumento de servidores de base de datos que pueden irse incluyendo para optimizar el índice de respuesta. Así mismo no existe replicación, sino más bien diferentes schemas para cada microservicio.
¿Los microservicios usan una base de datos compartida o dedicada?
Utilizan una base de datos compartida ya que la información que manejan no requiere una base de datos dedicada para cada microservicio. Sin embargo, se propone mínimamente trabajar con diferentes esquemas para mantener independencia.
¿El microservicio tiene un único punto de falla?
No posee un único punto de falla, y la mayoría han sido minimizadas al máximo en el análisis de dependencias durante la etapa de descomposición de SMMicro.
¿Se han identificado todos los escenarios de falla y posibles catástrofes del microservicio?
No, puesto que existen posibles fallos que únicamente se los podría detectar en cuanto ya se encuentre en producción el sistema migrado.
¿Cuáles son las fallas comunes en el ecosistema de los microservicios?

Para nuestro caso específico pueden existir fallos referentes a la administración de recursos, ya que si no se cuenta con un cuidado minucioso podemos tener efectos contraproducentes de utilizar cloud hibrida.
¿Cuáles son los escenarios de falla de la capa de hardware que pueden afectar este microservicio?
Dado que SISLAB está conectado y depende en parte de un sistema mucho mayor como el SII, se debe realizar un buen acople de las tecnologías cloud con los servidores propios de la DGIP.
¿Qué fallas en la capa de comunicación y en la capa de aplicación pueden afectar este microservicio?
Los principales fallos pueden ser los de seguridad en la transmisión de información entre microservicios, por ende, debe existir medidas de seguridad que nos facilitan herramientas como el Middleware de RedHat [46].
¿Cómo impactan las fallas y las interrupciones de los microservicios en el negocio?
Existe un impacto menor al que existía en el monolito, dado ahora se implementa mecanismos como cortador de circuitos y monitores de rendimiento. Así pues, para cada microservicio existirá un efecto diferente, considerando que a pesar de que uno falle, los demás seguirán en producción.
¿Hay niveles de falla bien definidos?
No existen niveles de falla bien definidos, ya que esto se lo debe obtener al evaluar incrementalmente el aplicativo en producción con el patrón de estrangulamiento ya explicado [50].

4.3.Resultados y Discusión

Entre los resultados más relevantes tenemos la descomposición final de la base de datos del Sistema de Gestión y Administración de Laboratorios mostrado en la figura 42, acorde a las metodologías de descomposición previamente fundamentadas. De esta manera se busca que cada esquema de la base de datos responda a su correspondiente microservicio de la lógica de negocio, por ende, tenemos una reducción de dependencia con lo visto en la estructura de base de datos inicial de SISLAB. Como punto en consideración, la dependencia existente, únicamente se la puede reducir, mas no mitigar, puesto que, al tratarse de una migración, existen factores de la lógica de negocio que se mantendrán constantes a lo largo del tiempo, y la única forma de cambiarlos es volviendo a rediseñar desde cero el sistema, lo cual no es lo que se busca, ni tampoco realizar cambios que afecten la experiencia de funcionalidad del cliente consumidor.

En cuanto al esquema obtenido de la relación entre la capa de negocio y los datos mostrado en la figura 43, nos permite evidenciar la tentativa de descomposición que agrupa de mejor manera las clases existentes en SISLAB, considerando que ahora se pretende utilizar tres bases de datos para los tres microservicios propuestos, una base de datos para lo referente al módulo de facturación que actualmente ya se lo maneja por separado, y finalmente una quinta base de datos que gestione las tablas aisladas a las funciones del aplicativo, así como futuras implementaciones que necesite expandir el número de entidades o tablas que actualmente posee SISLAB. Este grafico se contrasta claramente con el de la figura 30, en donde teníamos un monolito que gestiona todo con una capa de persistencia robusta y poco adaptable para un escalamiento del software, además que existían también clases obsoletas que fueron creadas para funcionalidades distintas a las que ahora son requeridas por otros usuarios externos a los laboratorios que desean utilizar el software.

Como resultado transcendental también evidenciamos la estructura de SISLAB en las diferentes capas del ecosistema de microservicios, aunque se debe acotar que previo a la capa de hardware ya se posee una estructura importante en donde su implementación dependerá únicamente de la entidad propietaria del aplicativo, en este caso la Dirección de Gestión de Información y Procesos que debe decidir si se propone una implementación netamente basada en cloud como se muestra en la figura 47, o si hace uso de los servidores físicos que dispone y utiliza una implementación como la mostrada en la figura 48 con cloud híbrida.

Entre los resultados también se muestra que existe un aumento considerable del número de componentes para que pueda ser gestionado de manera adecuada SISLAB, sin embargo como se comentó en un inicio, SMMicro no es una receta que se pueda seguir para obtener el mejor resultado con microservicios, la propuesta metodológica es únicamente un punto de referencia que ayude a los encargados de la migración a tener una visión más adecuada de cuáles son los cambio a realizar, es así que muchos componentes serán o no necesarios, acorde al aplicativo analizado. Además, al evaluar nuestro diseño para migración, nos percatamos que si bien se cumple con la mayoría de preguntas que plantea SMMicro en su cuarta etapa, existen algunas consideraciones que se pueden ir mejorando con el paso del tiempo y ya la puesta en producción del sistema. Por ejemplo, aspectos relacionados con seguridad, posibles marcos de trabajo utilizables o incluso cambios que permitan aumentar el rango del alcance. Por ende, se espera que posteriormente se logre añadir más condiciones de valor para mejorar nuestra etapa de evaluación y por ende el esquema en general.

CAPÍTULO 5 - CONCLUSIONES Y RECOMENDACIONES

Una vez analizados los resultados obtenidos, podemos obtener varias conclusiones referentes a la migración. Partiremos haciendo mención de que el objetivo general de este proyecto el cual fue elaborar una propuesta metodológica para la migración de sistemas web con arquitectura monolítica, hacia una arquitectura basada en microservicios, fue realizado y plasmado a través de cuatro etapas, cada una con varios lineamientos propuestos con sus respectivas justificaciones, actividades a realizar, posibles herramientas que pueden utilizar y las pertinentes consideraciones y recomendaciones. Se propone, además, poder encapsular la propuesta metodológica de tal manera que sea fácil de abstraer, mediante un esquema gráfico que se lo denomino Scheme for Migration towards Microservices (SMMicro).

De esta forma para caracterizar la arquitectura monolítica y la arquitectura basada en microservicios, que fue nuestro primer objetivo específico, realizamos una investigación sobre arquitecturas tradicionales monolíticas, en donde claramente se pudo identificar las deficiencias que poseen, en conjunto con varias de las alternativas que han venido evolucionando a lo largo del tiempo para poder llegar a microservicios. Así pues, tenemos que su padre, las Arquitecturas Orientadas a Servicios, tuvieron gran acogida, sin embargo, la flexibilidad que ofrecen los microservicios permite una mejora en cuanto a escalamiento, resiliencia, velocidad, entre otros, acorde a un contexto de utilización.

En cuanto al segundo objetivo específico de determinar los componentes necesarios para una descomposición de microservicios partiendo de una arquitectura monolítica, se realizó un análisis del estado del arte con diferentes artículos y libros en donde se coincidió con varios componentes principales, sin embargo, la implementación de cada uno de estos depende del sistema de estudio de caso, puesto que no todos pueden ser tratados de forma estándar, así como también se encontró que no solo son necesarios ciertos componentes para la arquitectura, sino además un completo ecosistema de microservicios en donde puedan coexistir.

Una vez comprendido esto, para continuar con el cumplimiento de los objetivos específicos, en este caso, definir los lineamientos de transición de una arquitectura monolítica a una descomposición en microservicios, se procedió a delimitar un modelo conceptual estructurado en: comprensión, descomposición, asociación y validación. Cada una de estas justificadas acorde al análisis teórico previo. En donde se tiene a la descomposición como eje principal de los resultados obtenidos, es decir, si la descomposición no sigue un

conjunto de reglas o metodologías afines, se puede caer en errores que se verán identificados al momento de implementar el diseño final obtenido.

Finalmente, para cumplir con nuestro último objetivo específico tenemos que la validación sobre un estudio de caso de la DGIP tomo el rumbo y los resultados esperados, aunque no se pudo identificar el uso total de los componentes propuestos para microservicios, puesto que SISLAB únicamente fue adaptado al alcance de este proyecto, y aún queda la etapa de implementación del diseño obtenido para trabajos futuros, en donde evidentemente se encontraran nuevas observaciones, correcciones y agregaciones a la primera versión de la propuesta metodológica.

Entre otras conclusiones interesantes tenemos el hecho que dentro de la comparativa realizada con el método SACAM se pudo observar claramente la necesidad de migración a un nuevo estilo arquitectónico, así como mitigar la dependencia del stack tecnológico para desarrollo de una aplicación, o implementar estrategias DevOps para cambiar la cultura organizacional de los equipos de trabajo. Todo esto como parte de una transición a microservicios eficiente, aunque con varias implicaciones importantes, como el hecho de que se necesita un mayor grado de componentes para garantizar un despliegue continuo, o la necesidad de protocolos de seguridad más robustos para la comunicación entre servicios, así pues, al realizar la comparativa nos encontramos no solo con ventajas, sino también con dificultades que deben ser tratadas para que una migración de arquitectura no produzca efectos contraproducentes en la lógica de negocio que posee el aplicativo.

Otra de las conclusiones es que este enfoque no es aplicable en todos los sistemas, de hecho, es enfocado en sistemas que crecieron y sobrepasaron sus límites definidos inicialmente, que requieren gran transaccionalidad y resiliencia, pero sobre todo que posean un monolito con estructura bien definida, ya que sin esto no se puede esperar que los resultados sean los deseados.

Ya entrando en el análisis de la propuesta metodológica, se examinaron algunos tipos de descomposición, aunque no todos ya que pueden existir n estrategias, pero todo dependerá del contexto en el cual se los quiera aplicar. Manejar una descomposición por funcionalidad, por madurez, por acceso a datos, o por contexto, siempre denotara que el monolito cumple con ciertas características especiales para su utilización. Una recomendación importante es tener en claro que no manejamos únicamente microservicios, sino además una gama de componentes y herramientas que deben ser implementadas en diferentes capas dentro de un ecosistema.

Otra de las conclusiones importantes que se obtuvo al implementar nuestros lineamientos de la propuesta metodológica sobre SISLAB, corresponde a su primera etapa, en donde se busca que tanto los jefes o líderes de la empresa dueña del aplicativo, como los equipos de trabajo, comprendan las implicaciones de usar microservicios. Aquí se pudo notar que se necesita de varias reuniones e inclusive capacitación para que no exista confusión en cuanto al uso de otras arquitecturas. Además, existe renuencia al cambio, y en la mayoría de los casos es debido a que los sistemas son legados de muchos años y en un entorno en donde no se requiere un aumento considerable de rendimiento, las personas tienden a ser escépticas de los beneficios que podríamos obtener. Aquí se recomienda analizar bien el sistema a trabajar, que como se mencionó, debe cumplir con ciertas características para evaluar si es migrable o si con arquitectura monolítica se obtiene el mejor rendimiento posible para las necesidades del usuario y de la empresa.

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. Como se vio en nuestro estudio de caso, se utilizó primero un patrón de descomposición por lógica de negocio en conjunto con un patrón de descomposición por base de datos puesto que no existe la suficiente información documentada para aplicar los otros tipos de estrategias. Sin embargo, identificar varias propuestas de descomposición es vital para encontrar un equilibrio en las dependencias que posee la capa de negocio con la capa de datos del sistema.

En lo que respecta a la etapa de asociación, se vio un análisis basado en las diferentes capas del ecosistema de microservicios, en donde se enfatiza la selección de los componentes de software necesarios para el funcionamiento adecuado de los cuatro microservicios propuestos, así pues se destaca también que en la capa de hardware se evidencia el trabajo en conjunto con el área de infraestructura de la DGIP, mostrando una vez más la necesidad de un trabajo colaborativo de toda la institución para la producción de una arquitectura distribuida eficiente. 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.

Como última conclusión una arquitectura basada en microservicios nos puede proporcionar múltiples beneficios, teniendo en cuenta varias complicaciones en su implementación. Por ende, nuestra propuesta metodológica a través de SMMicro puede llegar a convertirse en un punto de referencia para facilitar una migración partiendo de un monolito, sin embargo, debe ser validado en más estudios de caso acorde a contextos similares que se tuvo en la selección de SISLAB, e ir puliendo las deficiencias que se encuentren en el camino. 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.

REFERENCIAS BIBLIOGRÁFICAS

[1] L. Carvajal, *Metodología de la investigación: curso general y aplicado*. Faid, 1986.

[2] C. Stoermer, F. Bachmann, and C. Verhoef, "The software architecture comparison analysis method," *Technical Report,* p. 49, 2003.

[3] M. Villamizar *et al.*, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *Computing Colombian Conference (10CCC), 2015 10th*, 2015, pp. 583-590: IEEE.

[4] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to cloud-native architectures using microservices: an experience report," in *European Conference on Service-Oriented and Cloud Computing*, 2015, pp. 201-215: Springer.

[5] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference*, 2016, pp. 318-325: IEEE.

[6] D. Escobar *et al.*, "Towards the understanding and evolution of monolithic applications as microservices," in *Computing Conference (CLEI), 2016 XLII Latin American*, 2016, pp. 1-11: IEEE.

[7] A. Levcovitz, R. Terra, and M. T. Valente, "Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems," *arXiv preprint arXiv:1605.03175,* 2016.

[8] M. Ahmadvand and A. Ibrahim, "Requirements Reconciliation for Scalable and Secure Microservice (De) composition," in *Requirements Engineering Conference Workshops (REW), IEEE International*, 2016, pp. 68-73: IEEE.

[9] J.-P. Gouigoux and D. Tamzalit, "From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture," in *Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on*, 2017, pp. 62-65: IEEE.

[10] M. Mosleh, K. Dalili, and B. Heydari, "Distributed or monolithic? a computational architecture decision framework," *IEEE Systems journal,* 2016.

[11] S. J. Fowler, *Production-Ready Microservices*. United States of America: O'Reilly, 2016.

[12] N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "Microservices: Migration of a mission critical system," *arXiv preprint arXiv:1704.04173,* 2017.

[13] K. Bakshi, "Microservices-based software architecture and approaches," in *Aerospace Conference, 2017 IEEE*, 2017, pp. 1-8: IEEE.

[14] D. Sánchez, "Arquitectura de microservicios," Accessed on: 2017-07Available: http://enmilocalfunciona.io/arquitectura-microservicios-1/

[15] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Pearson Education, 2012.

[16] C. M. University, "Software Architecture Evolution," Accessed on: 2017-08Available: http://www.sei.cmu.edu/architecture/research/previousresearch/evolution.cfm

[17] RedHat, "Logre una arquitectura de microservicios exitosa," vol. 1, p. 4, 2017.

[18] M. Fowler, "Microservices a definition of this new architectural term," Accessed on: 2017-03Available: https://martinfowler.com/articles/microservices.html

[19] A. de Camargo, I. Salvadori, R. d. S. Mello, and F. Siqueira, "An architecture to automate performance tests on microservices," in *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, 2016, pp. 422-429: ACM.

[20] E. Wolff, *Microservices Flexible Software Architecture*. United States: Addison-Wesley, 2017.

[21] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture Aligning Principles, Practices and Culture*. United States: O'Reilly, 2016.

[22] G. Shreelekhya, S. U. Yazhini, and N. Manikandan, "Methods For Evaluating Software Architecture," *International Journal of Pharmacy & Technology,* vol. 8, no. 4, pp. 25720-25733, 2016.

[23] S. Masters, "Using Organizational Business Objectives to Guide a Process Improvement Program," p. 26Accessed on: 2017-03Available: http://cmmiinstitute.com/sites/default/files/resource_asset/2522_Masters.pdf

[24] R. Senthilkumar and T. Arunkumar, "A Survey on Prioritization of Software Quality Attributes," *Indian Journal of Science and Technology,* vol. 9, no. 44, p. 7, 2016.

[25] S. Sverchkov, "Microservices vs. Monolithic Architectures: The Pros, Cons, and Cloud Foundry Examples," p. 15Accessed on: 2017-07Available: https://www.altoros.com/microservices-vs-monolithic-architecture-pros-cons-cloud-foundry-paas.html

[26] C. Richardson, *Microservice Patterns MEAP*. United States of America: Manning Publications Co, 2017.

[27] C. Carneiro and T. Schmelmer, *Microservices from day One*. Florida: Apress, 2016.

[28] Netflix, "A microservices orchestrator," Accessed on: 2017-09Available: https://medium.com/netflix-techblog/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40

[29] M. Fowler, "BoundedContext," Accessed on: 2017-06Available: https://martinfowler.com/bliki/BoundedContext.html

[30] J. Barabas, "IBM - IaaS PaaS SaaS cloud service models," Accessed on: 2017-06Available: https://www.ibm.com/cloud-computing/learn-more/iaas-paas-saas/

[31] C. Kung and A. Soelvberg, "Activity modeling and behavior modeling," in *Proc. of the IFIP WG 8.1 working conference on Information systems design methodologies: improving the practice*, 1986, pp. 145-171: North-Holland Publishing Co.

[32] S. Robinson, G. Arbez, L. G. Birta, A. Tolk, and G. Wagner, "Conceptual modeling: definition, purpose and benefits," in *Proceedings of the 2015 Winter Simulation Conference*, 2015, pp. 2812-2826: IEEE Press.

[33] M. Geers, "Micro Frontends - Extending the microservice idea to frontend development," Accessed on: 2017-08Available: https://micro-frontends.org/

[34] J. Radatz, A. Geraci, and F. Katki, "IEEE standard glossary of software engineering terminology," *IEEE Std,* vol. 610121990, no. 121990, p. 3, 1990.

[35] L. Chacha and E. Viscarra, "Desarrollo e Implantación Del Sistema De Gestión De Análisis Físico-Químico y Microbiológico De Aguas, Suelos y Lodos Para El Centro De Investigación De Control Ambiental (CICAM) De La EPN," Ingeniería en Sistemas Informáticos y de Computación, Facultad de Ingeniería de Sistemas, Escuela Politécnica Nacional, 2010.

[36] S. S. Díaz Méndez, "Firewall de Aplicación Web," p. 8Accessed on: 2017-09Available: http://revista.seguridad.unam.mx/printpdf/2167

[37] Amazon. (2017, 2017-07). *AWS | Cloud Computing - Servicios de informática en la nube*. Available: //aws.amazon.com/es/

[38] Google. (2017, 2017-09). *Istio: An open platform to connect, manage, and secure microservices*. Available: https://istio.io/

[39] E. Foundation. (2017, 2017-08). *Microprofile.io | Optimizing Enterprise Java*. Available: https://microprofile.io/

[40] Microsoft. (2017, 2017-07). *Microservice apps | Microsoft Azure*. Available: https://azure.microsoft.com/en-us/solutions/microservice-applications/

[41] Spring. (2017, 2017-05). *Spring Cloud*. Available: http://projects.spring.io/spring-cloud/

[42] Netflix. (2017, 2017-08). *Netflix Open Source Software Center*. Available: https://netflix.github.io/

[43] Docker. (2017, 2017-04). *Docker Platform*. Available: https://www.docker.com/

[44] Eventuate. (2017, 2017-07). *Event-driven microservices*. Available: http://eventuate.io/whyeventdriven.html

[45] J. Weinman, "Hybrid cloud economics," *IEEE Cloud Computing,* vol. 3, no. 1, pp. 18-22, 2016.

[46] RedHat. (2017, 2017-06). *JBoss Middleware | Innovación más rápida e inteligente*. Available: https://www.redhat.com/es/technologies/jboss-middleware

[47] RedHat. (2017). *OpenShift: Container Application Platform*. Available: https://www.openshift.com/

[48] C. Systems. (2017, 2017-07). *NetScaler ADC - Controlador de entrega de aplicaciones, equilibrador de carga*. Available: https://www.citrix.com/products/netscaler-adc/

[49] J. Falkner, "Migrating Legacy Apps," presented at the Monoliths to Microservices | Red Hat Virtual Event, 2017.

[50] K. Brown "Apply the Strangler Application pattern to microservices applications," *IBM Developer Works,* p. 6, 2017.

[51] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: migration to a cloud-native architecture," *IEEE Software,* vol. 33, no. 3, pp. 42-52, 2016.

ANEXOS

Anexo 1 - Documento sobre requerimientos para estudio de caso entregado a DGIP.

FACULTAD DE INGENIERÍA DE SISTEMAS

CARRERA DE INGENIERÍA DE SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

REQUERIMIENTOS PARA ESTUDIO DE CASO EN PROYECTO DE TITULACIÓN APLICADO A SISTEMA DE LA DIRECCIÓN DE GESTIÓN DE INFORMACIÓN Y PROCESOS DE LA EPN

LÍNEA DE INVESTIGACIÓN:

INGENIERÍA DE SOFTWARE - MEJORAMIENTO DEL PROCESO DE DESARROLLO DE SOFTWARE

TÍTULO DEL TRABAJO DE TITULACIÓN:

PROPUESTA METODOLÓGICA PARA MIGRACIÓN DE SISTEMAS WEB CON ARQUITECTURA MONOLÍTICA HACIA UNA ARQUITECTURA BASADA EN MICROSERVICIOS

JUSTIFICACIÓN PRÁCTICA

La elaboración de una propuesta metodológica, permitirá establecer una guía para la Dirección de Gestión de Información y Procesos (DGIP) de la Escuela Politécnica Nacional, para trasladar sistemas web tradicionales que requieran un alto grado de disponibilidad a sistemas web con arquitectura distribuida. Además, será útil para cualquier empresa que busque desarrollo o migración a sistemas escalables, robustos y no dependientes de una misma tecnología, por ejemplo, una aplicación podría fragmentarse en servicios como gestión de pedidos, gestión de clientes, compras, facturación electrónica, etcétera.

REQUERIMIENTOS

Los siguientes puntos son tomados en cuenta para facilitar la elección de un sistema o parte de uno, que disponga la DGIP para poder ser analizado, y su vez, tomado como ejemplo para la validación de la propuesta metodológica que plantea lineamientos para la migración de sistemas monolíticos a sistemas basados en microservicios.

Como requerimientos generales, se necesita un sistema web, desarrollado preferentemente en un lenguaje de versión libre, orientado a objetos y multiplataforma (Java), con una base de datos relacional, y que posea la mayor parte de la documentación a continuación detallada:

SISTEMA:

Requerimiento	Importancia
Documento con descripción del sistema.	**Alta**
Especificación de requerimientos	
Requerimientos Funcionales	**Alta**

DESARROLLO:

Requerimiento	Importancia
Herramientas utilizadas: entornos de desarrollo integrados, plataformas y tecnologías empleadas en la implementación del Sistema.	Alta
Planificación: Descripción general de la metodología de desarrollo empleada, número de integrantes del equipo y tiempo aproximado de entregables y construcción completa del sistema.	Alta
Descripción de módulos y conexiones entre los mismos.	Alta
Diagramas de Casos de Uso	Alta
Diagramas de Actividades	Media
Diagramas de Clases	Alta
Diagrama de Componentes	Media
Diagrama de Despliegue	Alta

BASE DE DATOS

Requerimiento	Importancia
Modelo Conceptual	Media
Modelo Lógico	Media
Modelo Físico	Alta

Anexo 2 – Checklist de Susan Fowler en Libro "Production Ready Microservices"

Production-Readiness Checklist

This will be a checklist to run over all microservices—manually or in an automated way.

A Production-Ready Service Is Stable and Reliable

- It has a standardlzed development cycle.
- Its code is thoroughly tested through llnt, unit, integration, and end-to-end testing.
- Its test, packaging, build, and release process is completely automated.
- It has a standardized deployment pipeline, containing staging, canary, and production phases.
- Its cllents are known.
- Its dependencies are known, and there are backups, alternatives, fallbacks, and caching in place in case of failures.
- It has stable and reliable routing and dlscovery ln place.

A Production-Ready Service Is Scalable and Performant

- Its qualitative and quantitatlve growth scales are known.
- It uses hardware resources efficiently.
- Its resource bottlenecks and requirements have been identified.
- Capacity planning is automated and performed on a scheduled basis.
- Its dependencies will scale with it.

- It will scale with its clients.
- Its traffic patterns are understood.
- Traffic can be re-routed in case of failures.
- It is written in a programming language that allows it to be scalable and performant.
- It handles and processes tasks in a performant manner.
- It handles and stores data in a scalable and performant way.

A Production-Ready Service Is Fault Tolerant and Prepared for Any Catastrophe

- It has no single point of failure.
- All failure scenarios and possible catastrophes have been identified.
- It is tested for resiliency through code testing, load testing, and chaos testing.
- Failure detection and remediation has been automated.
- There are standardized incident and outage procedures in place within the microservice development team and across the organization.

A Production-Ready Service Is Properly Monitored

- Its key metrics are identified and monitored at the host, infrastructure, and microservice levels.
- It has appropriate logging that accurately reflects the past states of the microservice.
- Its dashboards are easy to interpret and contain all key metrics.
- Its alerts are actionable and are defined by signal-providing thresholds.
- There is a dedicated on-call rotation responsible for monitoring and responding to any incidents and outages.
- There is a clear, well-defined, and standardized on-call procedure in place for handling incidents and outages.

A Production-Ready Service Is Documented and Understood

- It has comprehensive documentation.
- Its documentation is updated regularly.
- Its documentation contains a description of the microservice; an architecture diagram; contact and on-call information; links to important information; an onboarding and development guide; information about the service's request flow(s), endpoints, and dependencies; an on-call runbook; and answers to frequently asked questions.
- It is well understood at the developer, team, and organizational levels.
- It is held to a set of production-readiness standards and meets the associated requirements.
- Its architecture is reviewed and audited frequently.

Anexo 3 – Evaluación de Susan Fowler en Libro "Production Ready Microservices"

Evaluate Your Microservice

To help the reader evaluate the production-readiness of their microservice(s) and microservice ecosystem, Chapters 3–7 conclude with a short list of questions associated with the production-readiness standard discussed. The questions are organized by topic, and correspond to the sections within each chapter. All of the questions from each chapter have been collected here for easy reference.

Stability and Reliability

The Development Cycle

- Does the microservice have a central repository where all code is stored?
- Do developers work in a development environment that accurately reflects the state of production (e.g., that accurately reflects the real world)?
- Are there appropriate lint, unit, integration, and end-to-end tests in place for the microservice?
- Are there code review procedures and policies in place?
- Is the test, packaging, build, and release process automated?

The Deployment Pipeline

- Does the microservice ecosystem have a standardized deployment pipeline?
- Is there a staging phase in the deployment pipeline that is either full or partial staging?
- What access does the staging environment have to production services?

- Is there a canary phase in the deployment pipeline?
- Do deployments run in the canary phase for a period of time that is long enough to catch any failures?
- Does the canary phase accurately host a random sample of production traffic?
- Are the microservice's ports the same for canary and production?
- Are deployments to production done all at the same time, or incrementally rolled out?
- Is there a procedure in place for skipping the staging and canary phases in case of an emergency?

Dependencies

- What are this microservice's dependencies?
- What are its clients?
- How does this microservice mitigate dependency failures?
- Are there backups, alternatives, fallbacks, or defensive caching for each dependency?

Routing and Discovery

- Are health checks to the microservice reliable?
- Do health checks accurately reflect the health of the microservice?
- Are health checks run on a separate channel within the communication layer?
- Are there circuit breakers in place to prevent unhealthy microservices from making requests?
- Are there circuit breakers in place to prevent production traffic from being sent to unhealthy hosts and microservices?

Deprecation and Decommissioning

- Are there procedures in place for decommissioning a microservice?
- Are there procedures in place for deprecating a microservice's API endpoints?

Scalability and Performance

Knowing the Growth Scale

- What is this microservice's qualitative growth scale?
- What is this microservice's quantitative growth scale?

Efficient Use of Resources

- Is the microservice running on dedicated or shared hardware?
- Are any resource abstraction and allocation technologies being used?

Resource Awareness

- What are the microservice's resource requirements (CPU, RAM, etc.)?
- How much traffic can one instance of the microservice handle?
- How much CPU does one instance of the microservice require?
- How much memory does one instance of the microservice require?
- Are there any other resource requirements that are specific to this microservice?
- What are the resource bottlenecks of this microservice?
- Does this microservice need to be scaled vertically, horizontally, or both?

Capacity Planning

- Is capacity planning performed on a scheduled basis?
- What is the lead time for new hardware?
- How often are hardware requests made?
- Are any microservices given priority when hardware requests are made?
- Is capacity planning automated or is it manual?

Dependency Scaling

- What are this microservice's dependencies?
- Are the dependencies scalable and performant?
- Will the dependencies scale with this microservice's expected growth?

- Are dependency owners prepared for this microservice's expected growth?

Traffic Management

- Are the microservice's traffic patterns well understood?
- Are changes to the service scheduled around traffic patterns?
- Are drastic changes in traffic patterns (especially bursts of traffic) handled carefully and appropriately?
- Can traffic be automatically routed to other datacenters in case of failure?

Task Handling and Processing

- Is the microservice written in a programming language that will allow the service to be scalable and performant?
- Are there any scalability or performance limitations in the way the microservice handles requests?
- Are there any scalability or performance limitations in the way the microservice processes tasks?
- Do developers on the microservice team understand how their service processes tasks, how efficiently it processes those tasks, and how the service will perform as the number of tasks and requests increases?

Scalable Data Storage

- Does this microservice handle data in a scalable and performant way?
- What type of data does this microservice need to store?
- What is the schema needed for its data?
- How many transactions are needed and/or made per second?
- Does this microservice need higher read or write performance?
- Is it read-heavy, write-heavy, or both?
- Is this service's database scaled horizontally or vertically? Is it replicated or partitioned?
- Is this microservice using a dedicated or shared database?
- How does the service handle and/or store test data?

Anexo 4– Lineamientos para descomposición de microservicios

Comprensión y Descomposición

	Lineamientos	Justificación	Actividades
Comprensión	Análisis de cambio, en base a lógica de negocio.	Constatar que la institución impulsará un cambio en la forma de estructurar software.	Comprender cambios con director o jefe de proyectos.
			Analizar con equipo de desarrollo posibles implicaciones de migración.
	Estudio de los antecedentes del sistema a migrar.	Comprender el origen de creación del sistema a ser migrado para entender las implicaciones de su evolución en el tiempo.	Extraer información general del aplicativo.
			Extraer información de los Módulos Iniciales del aplicativo.
			Extraer información de la Base de Datos del aplicativo.
	Estudio del estado actual del sistema a migrar.	Identificar las deficiencias actuales del sistema, así como analizar que la migración de arquitectura creará un cambio significativo a futuro.	Extraer información de los Módulos Actuales del aplicativo.
			Extraer información de la Base de Datos del aplicativo.
			Extraer información de la Arquitectura del aplicativo.
Descomposición	Análisis de la información de las operaciones del sistema.	Establecer un punto de referencia para la descomposición	Crear un nuevo esquema que muestre las clases principales del aplicativo.
			Describir el comportamiento del sistema en base al Modelo de Dominio.
	Aplicación de patrones de descomposición en la capa de negocio del aplicativo.	Abstraer una fragmentación adecuada manteniendo de la lógica de negocio del sistema.	Dividir la capa de negocio en base al Modelo de Dominio.
			Definir la primera propuesta de descomposición de capa de negocio.
			Extraer la primera propuesta de descomposición de la base de datos.
	Aplicación de patrones de descomposición en la capa de datos del aplicativo.	Abstraer una fragmentación adecuada manteniendo de la lógica de negocio del sistema.	Definir el número de relaciones padre e hija de tablas.
			Definir el tamaño y número de registros de las tablas.
			Clasificar las tablas en base a lectura y escritura intensiva.
			Abstraer la segunda propuesta de descomposición de la Base de Datos.
	Identificación de una propuesta de descomposición final.	Obtener un diagrama de descomposición que se acople con la lógica de negocio del sistema.	Comparar la independencia entre propuestas de capa de negocio con descomposición en Base de Datos.
			Elegir mejor propuesta de descomposición entre capa de negocio y BD.

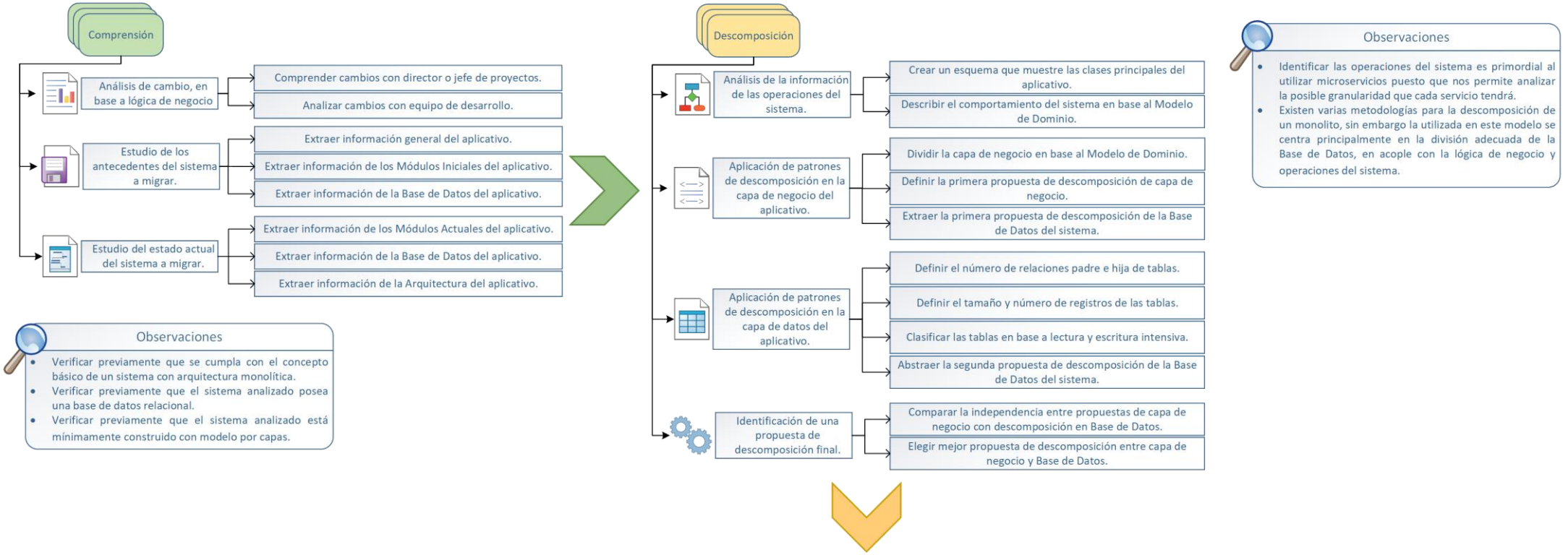
Herramientas	Consideraciones y Recomendaciones
Documentación de planificación de la migración.	No existe un formato específico para estos documentos, existira una variación acorde a la empresa, pero lo escencial es que cumpla con la finalidad propuesta.
Documentación sobre alcance de migración.	
Metodologías de desarrollo.	Es primordial analizar y entender el estado y documentación del sistema a migrar en su origen con relación a la organización. Así como identificar áreas problematicas en su diseño inicial y como esto puede afectar el nuevo cambio de arquitectura.
Requerimientos Funcionales.	
Requerimientos No Funcionales.	
Diagrama Global de Paquetes.	
Clases pertenecientes a paquetes del sistema.	
Esquema Lógico.	
Esquema Físico.	
Diagrama Global de Paquetes.	Se procura identificar transformaciones seguras, es decir cambios que no afectarán el comportamiento existente del aplicativo. Sin embargo, a pesar que el usuario final no distinga una variación considerable en su experiencia, para el sistema como tal, si, puesto que se busca mejorar su rendimiento en backend.
Modelo de Dominio Inicial.	
Esquema Lógico.	
Esquema Físico.	
Patrón de Diseño.	
Arquitectura Física.	
Modelo de clases.	Las operaciones del sistema definidas nos ayudarán a definir posteriormente la granuralidad de los servicios, puesto que las funcionalidades del aplicativo deben perdurar sin cambios.
Modelo de Dominio.	
Tabla con funciones del sistema.	
Lista de posibles microservicios.	La capa de negocio debe mantener la lógica de negocio a pesar de la división del modelo de dominio, se recomienda analizar las funcionalidades del aplicativo en producción para apreciar de mejor manera los posibles microservicios.
Diagrama de Modelo de Dominio Dividido.	
Diagrama de Base de Datos Descompuesto.	
Modelo de Entidades de Base de Datos.	
Tabla con entidades sensibles por número de relaciones.	Es primordial buscar que cada microservicio aisle al máximo sus datos, sin embargo, al tratarse de una base de datos relacional se deben realizar varios posibles escenarios de descomposición puesto que existen varios factores que influiran en el rendimiento final del sistema, estre ellas aquellas entidades estrechamente relacionadas que deberán mantener cierta relación de dependencia.
Querys sobre la Base de Datos.	
Cuadro comparativo de lectura y escritura de entidades.	
Diagrama de Base de Datos Descompuesto.	
Pruebas en ambiente de producción del sistema.	Para seleccionar una descomposición final, se debe analizar las posibles variaciones que tendra la Base de Datos en conjunto con la capa de negocio, sus llamadas, relaciones e instancias. Utilizar un ambiente de producción facilitará la visualización de todos estos factores de comportamiento.
Matriz con llamadas de la capa de persistencia a Base de Datos.	
Diagrama final de descomposición.	

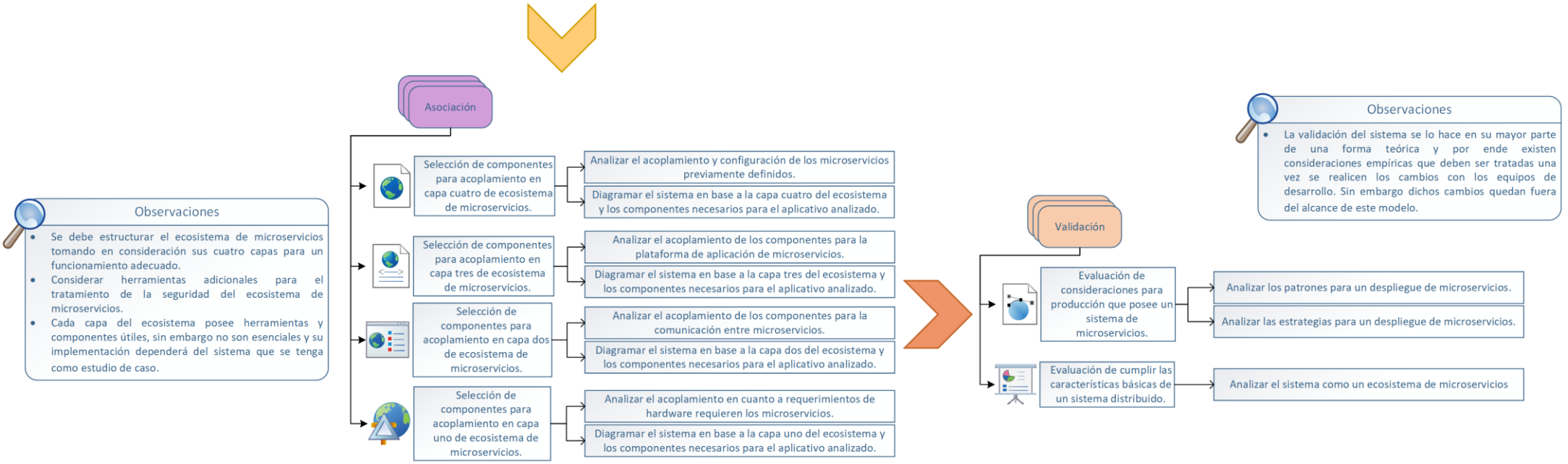
Asociación y Validación

	Lineamientos	Justificación	Actividades
Asociación	Selección de componentes para acoplamiento en capa cuatro de ecosistema de microservicios.	Estructurar los microservicios que en sí facilitaran el núcleo de la nueva arquitectura.	Analizar el acoplamiento y configuración de los microservicios previamente definidos.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
	Selección de componentes para acoplamiento en capa tres de ecosistema de microservicios.	Integrar los componentes que faciliten la configuración y monitorización de los microservicios.	Analizar el acoplamiento de los componentes para la plataforma de aplicación de microservicios.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
	Selección de componentes para acoplamiento en capa dos de ecosistema de microservicios.	Integrar los componenetes que faciliten el despliegue y comunicación de los microservicios.	Analizar el acoplamiento de los componentes para la comunicación entre microservicios.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
	Selección de componentes para acoplamiento en capa uno de ecosistema de microservicios.	Integrar los componentes que faciliten el despliegue y comunicación de los microservicios.	Analizar el acoplamiento en cuanto a requerimientos de hardware requieren los microservicios.
			Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
Validación	Evaluación de consideraciones para producción que posee un sistema de microservicios.	Validar que el aplicativo a migrar puede ser tratado como un sistema basado en microservicios.	Analizar los patrones para un despliegue de microservicios.
			Analizar las estrategias para un despliegue de microservicios.
	Evaluación de cumplir las características básicas de un sistema distribuido.	Comprobar que el sistema analizado puede cumplir los requisitos de un ecosistema de microservicios.	Analizar el sistema como un ecosistema de microservicios

Herramientas	Consideraciones y Recomendaciones
Lista de componentes factibles para integrar en sistema analizado.	Esta sección se centra exclusivamente en los microservicios y cualquier cosa delimitada a ellos, teniendo en cuenta que sus configuraciones se pueden almacenar en el mismo repositorio del microservicio para ser accedidas por las herramientas de las capas inferiores, evitando cambios directos sobre las mismas.
Diagrama de componentes en capa cuatro de ecosistema de microservicios.	
Diagrama de sistema en base a ecosistema de Microservicios.	
Lista de componentes factibles para integrar en sistema analizado.	Encargada de todos los servicios y herramientas internas que son independientes de los microservicios, mismos que deben construirse de tal forma que los equipos de desarrollo no tengan inconvenientes en diseñar, construir o mantener nada que no sea sus propios microservicios.
Diagrama de componentes en capa tres de ecosistema de microservicios.	
Diagrama de sistema en base a ecosistema de Microservicios.	
Lista de componentes factibles para integrar en sistema analizado.	Se centra en la comunicación de microservicios, e interactúa con las otras capas del ecosistema. El tráfico debe ser encaminado apropiadamente a un gran número de aplicaciones diferentes, y luego distribuirse apropiadamente a los servidores que alojan cada microservicio específico.
Diagrama de componentes en capa dos de ecosistema de microservicios.	
Diagrama de sistema en base a ecosistema de Microservicios.	
Lista de componentes factibles para integrar en sistema analizado.	Se enfoca puramente en las herramientas físicas como servidores y equipos de cómputo que se utilizarán para montar el resto de componentes de software. Estos equipos pueden ser propiedad de la empresa o pagar por servicios cloud.
Diagrama de componentes en capa uno de ecosistema de microservicios.	
Diagrama de sistema en base a ecosistema de Microservicios.	
Compendio de patrones utilizables durante la migración.	Si no se valida la posibilidad de utilización de patrones y estrategias para la implementacion de la propuesta en microservicios se puede caer en fallos o errores graves ya estando en producción.
Compendio de estrategias utilizables durante la migración.	
Documento con Preguntas sobre Estabilidad y Confiabilidad.	Un sistema distribuido posee características únicas que deben ser validadas, una vez evaluado esto se procede a evaluarlo como microservicios.
Documento con Preguntas sobre Escalabilidad y Rendimiento.	
Checklist de validación de componentes de sistema distribuido.	

Anexo 5 – Scheme for Migration towards Microservices




Observaciones
Se debe estructurar el ecosistema de microservicios tomando en consideración sus cuatro capas para un funcionamiento adecuado.
Considerar herramientas adicionales para el tratamiento de la seguridad del ecosistema de microservicios.
Cada capa del ecosistema posee herramientas y componentes útiles, sin embargo no son esenciales y su implementación dependerá del sistema que se tenga como estudio de caso.
Asociación
Selección de componentes para acoplamiento en capa cuatro de ecosistema de microservicios.
Analizar el acoplamiento y configuración de los microservicios previamente definidos.
Diagramar el sistema en base a la capa cuatro del ecosistema y los componentes necesarios para el aplicativo analizado.
Selección de componentes para acoplamiento en capa tres de ecosistema de microservicios.
Analizar el acoplamiento de los componentes para la plataforma de aplicación de microservicios.
Diagramar el sistema en base a la capa tres del ecosistema y los componentes necesarios para el aplicativo analizado.
Selección de componentes para acoplamiento en capa dos de ecosistema de microservicios.
Analizar el acoplamiento de los componentes para la comunicación entre microservicios.
Diagramar el sistema en base a la capa dos del ecosistema y los componentes necesarios para el aplicativo analizado.
Selección de componentes para acoplamiento en capa uno de ecosistema de microservicios.
Analizar el acoplamiento en cuanto a requerimientos de hardware requieren los microservicios.
Diagramar el sistema en base a la capa uno del ecosistema y los componentes necesarios para el aplicativo analizado.
Validación
Evaluación de consideraciones para producción que posee un sistema de microservicios.
Analizar los patrones para un despliegue de microservicios.
Analizar las estrategias para un despliegue de microservicios.
Evaluación de cumplir las características básicas de un sistema distribuido.
Analizar el sistema como un ecosistema de microservicios
Observaciones
La validación del sistema se lo hace en su mayor parte de una forma teórica y por ende existen consideraciones empíricas que deben ser tratadas una vez se realicen los cambios con los equipos de desarrollo. Sin embargo dichos cambios quedan fuera del alcance de este modelo.