Metodología para transformar un software monolítico a un software basado en microservicios en el ámbito web

Methodology to transform a monolithic software into a microservice architecture

César Augusto Jaramillo Acevedo, Juan Pablo Gómez y Jorge Iván Ríos Patiño.
Universidad Tecnológica de Pereira, Colombia
swokosky@utp.edu.co, pablogomez@utp.edu.co,
jirios@utp.edu.co

Resumen — **Este documento presenta una metodología de desarrollo cuyo propósito es proponer procedimientos que permitan transformar un sistema monolítico a una arquitectura basada en microservicios, el documento hace una breve descripción de cada una de sus fases y explica su implementación en una aplicación monolítica de código abierto hasta completar totalmente la transformación a microservicios. La metodología parte del análisis del modelo del negocio de una aplicación monolítica para así proceder a la definición de subdominios, la definición de subdominios funcionales son el punto de partida para el fraccionamiento del monolito y el prototipado de los microservicios. La metodología describe un flujo de construcción de software definido por las etapas de análisis y diseño, implementación, pruebas e integración continua dentro de un ciclo de vida evolutivo que permite realizar una transformación escalonada y controlad. La arquitectura monolítica ha sido el diseño tradicionalmente utilizado para el desarrollo de aplicaciones web desde sus principios, la cual es construida por uno o varios equipos de desarrollos, encargados de trabajar un diseño centralizado, de rápido crecimiento y expuesto a cambios permanentes. A medida que las aplicaciones monolíticas crecen están son susceptibles al desarrollo de anti patrones de diseño que la convierten en sistemas complejos y difíciles de mantener, incrementando no solo los tiempos de mantenimiento sino también los costos de escalabilidad que representan la duplicación de instancias del monolito. La metodología propuesta ofrece un marco de trabajo para que equipos de desarrollo puedan estructurar, planificar y controlar el proceso de transformación de una arquitectura monolítica hacia una arquitectura distribuida. De igual manera la implementación de la metodología permitirá realizar un mejor seguimiento de la integración continua para mejorar el aseguramiento de la calidad de los despliegues en producción de los microservicios, al igual que lograr ser más asertiva la asignación de los recursos humanos disponibles en las actividades del proyecto de acuerdo a las experticias de cada integrante del equipo.**

Palabras Clave - Aplicación distribuida, computación en la nube, desnormalización, desacoplamiento, fraccionamiento, metodología de desarrollo, microservicios, monolito, refactorización, transformación.

Abstract — **This documents presents a development methodology which purpose is propose procedures that allows to transform a monolithic system into an architecture based in microservices, the document offers a description of each stage and explains it's implementation of the methodology in an open source monolith application. The methodology starts from the analysis of the business model of a monolithic application to proceed to the definition of subdomains, the definition of functional subdomains are the starting point for the fractionation of the monolith and the prototyping of the microservices. The methodology describes a flow of software construction defined by the stages of analysis and design, implementation, testing and continuous integration within an evolutionary life cycle that allows a staggered and controlled transformation. The monolithic architecture has been the traditional design used for the development of web applications since its inception, which is built by one or more development teams, responsible for a centralized design, fast growing and exposed to permanent changes. As monolithic applications grow, they are susceptible to the development of anti-design patterns that make it complex and difficult to maintain, increasing not only the maintenance times but also the scalability costs that represent the duplication of instances of the monolith. The proposed methodology provides a framework for development teams to structure, plan and control the process of transforming a monolithic architecture into a distributed architecture. Likewise, the implementation of the methodology will allow a better follow-up of the continuous integration to improve the quality assurance of the deployments in production of micro-services, as well as to be more assertive the allocation of human resources available in the activities of the project according to the expertise of each member of the team.**

Keywords - cloud computing, denormalization, development methodology, decoupling, distributed application, fractionation, microservices, monolith, refactoring, transformation

I. INTRODUCCIÓN

El desarrollo ágil de software tiene como propósito promover un proceso de construcción más flexible y adaptativo a las necesidades del cliente, orientado a entregas las cuales están sujetas a ajustes durante cada fase. Desde la declaración del manifiesto ágil han surgido diversas metodologías que se han popularizado en los últimos años tales como SCRUM y XP. La arquitectura basada en microservicios es un paradigma surgido recientemente el cual propone que los sistemas de software se deben de fragmentar

de tal manera que mejoren su disponibilidad, faciliten el mantenimiento, las modificaciones y la escalabilidad cuando se necesiten aumentar los recursos para un mejor desempeño [1], acercándose así a la filosofía de las metodologías ágiles.
Se entiende como sistema de software monolítico, aquel programa informático que tiene todos sus componentes ejecutandose en un mismo contenedor de aplicaciones y su código fuente se encuentra almacenado una misma base de código, requiere de un gran esfuerzo de sincronización entre los equipos de desarrollo . Las aplicaciones monolíticas de gran tamaño sufren de inconvenientes de disponibilidad, el mantenimiento y la escalabilidad se dificultan a medida que dicho sistema monolito crece, la arquitectura basada en microservicios propone romper una aplicación monolítica en pequeñas aplicaciones con propósitos o funcionalidades específicas de tal manera que lo que se necesite replicar sea lo necesario. En contraste con SOA que tiene como propósito orquestar aplicaciones monolíticas por medio de un protocolo distribuido, la arquitectura microservicios opera a más bajo nivel pretendiendo convertir una aplicación monolítica en una distribuida [2].

II. ANTECEDENTES

Para comenzar a hablar sobre el origen de los microservicios hay que remontarse a los inicios de SOA en el año de 1994 cuando Alexander Pasik un antiguo analista de Gartner acuñó el término SOA antes de que los servicios XML y los servicios web fueran inventados, Pasik creó este nuevo concepto para referirse a un nuevo tipo de arquitectura cliente servidor que había perdido su sentido clásico para referirse a un nuevo tipo de computación distribuida . De acuerdo con Wolff [3] el término servicio en SOA debería de tener las siguientes características:

- Debería de poder implementarse como una pieza individual del dominio
- Debería poderse usar independientemente
- Debería estar disponible en la red
- Cada servicio debería de tener una interfaz, acceder a la interfaz debería ser suficiente para utilizar el servicio.
- El servicio debería poder ser utilizado en diferentes lenguajes de programación y plataformas.
- Para facilitar su uso, el servicio debe de estar registrado en un directorio. Para localizarlo y usar el servicio, los clientes buscan este directorio en tiempo de ejecución
- El Servicio deberia ser de grano fino, de tal manera que permita reducir las dependencias. . Los servicios pequeños pueden implementar funcionalidad de gran utilidad solamente en conjunción con otros servicios. Sin embargo SOA tradicionalmente se enfoca en grandes servicios

III. MARCO TEÓRICO

De acuerdo con James Lewis y Martin Fowler [4] : “el termino microservicios fue discutido en un taller de arquitectos de software cerca a Venecia en Mayo del 2011 para describir un nuevo estilo de arquitectura común que ellos habían recientemente explorado. En mayo del 2012 el mismo grupo decidió escoger microservicios como el nombre más apropiado para esta naciente arquitectura.” Según Newman, los microservicios son componentes de software que permiten modularizar un sistema, cada uno con un propósito diferente y específico deben ser lo suficientemente pequeños de tal manera que sean fáciles de mantener . Newman sugiere que los microservicios deben de ser guiados por los límites del dominio haciendo uso del patrón Model Driven Design el cual fue acuñado por Evans [5] y que se define como un enfoque, con necesidades complejas, que se fundamenta entre una profunda conexión entre la implementación del software y el modelo del núcleo del negocio, el cual provee prácticas y terminologías para la toma de decisiones que tienen como propósito acelerar proyectos de software con modelos complejos.

Los microservicios deben de funcionar como entidades separadas y autónomas, su diseño debe seguir los lineamientos de alta cohesión y bajo acoplamiento, para ello un microservicio no deberá de exponer toda su funcionamiento ya que aumenta la tendencia al acoplamiento entre los consumidores dificultando el mantenimiento de la aplicación cuando se desea hacer una actualización

A. Ventajas de los microservicios

- La gobernanza centralizada establece un estándar para construir una plataforma con la misma tecnología, pero la arquitectura microservicios permite hacer uso de una gobernanza descentralizada facilitando el libre uso de tecnologías para la construcción de cada servicio.
- Capacidad de auto controlar fallos del sistema, a diferencia de los monolitos que necesitan instancias completas de la aplicación el sistema basado en microservicios se puede mantener disponible sin presentar un fallo general.
- Son pequeños y fáciles de mantener ,el escalamiento de los microservicios se hace con los servicios que se necesitan sobre la demanda reduciendo costos de disponibilidad
- El despliegue es menos tedioso que el de una aplicación monolítica ya que se hacen de manera aislada, y en caso de fallo el rollback es más simple de hacer reduciendo el impacto negativo que tendría una aplicación tradicional mal desplegada

IV. METODOLOGÍA

Las aplicaciones monolíticas a medida que crecen son susceptibles al desarrollo de anti patrones de diseño que la convierten en sistemas complejos y difíciles de mantener para

los desarrolladores . Una posible solución a este problema es detener el crecimiento de la arquitectura monolítica de la aplicación y escalarla usando una arquitectura basada en microservicios que soportarán las futuras actualizaciones de la lógica del negocio . El artículo presenta una metodología dividida por etapas que deben completar secuencialmente para transformar un monolito a microservicios: Planeamiento y Factibilidad, Educción y Análisis de Requisitos, Especificación de requisitos, validación de requisitos, diseño y definición de microservicios, desarrollo de los microservicios, pruebas de integración, contratos y punto a punto y despliegue, integración y entrega continua.

Posteriormente el artículo describe un caso de implementación de la metodología, y las conclusiones al respecto.

A. Planeamiento y factibilidad

En la etapa inicial debemos de determinar el nivel de madurez de la aplicación a transformar; de acuerdo con Martin Fowler, pionero de los microservicios, las reglas de negocio debe ser validadas en un monolito primero por los usuarios , y basándonos en la clasificación taxonómica de Daniel Bryant [6], una meso aplicación (2-5 años de antigüedad y de 10-1 KLCF/PA) sería la base elegible para realizar la transformación a una arquitectura de microservicios. También es necesario considerar que exista dominio sobre la propiedad intelectual del código fuente, que se cuente con personal con experticia en programación distribuida y en servicios web Restful. La metodología también incluye una escala para determinar el esfuerzo a partir de la escala desarrollada por Alistar Cockburn que media la productividad, la matriz se representa el esfuerzo que debería hacer un equipo para transformar una aplicación centralizada a una aplicación basada en microservicios, el esfuerzo se representa por un código de colores (azul, verde, amarillo, naranja y rojo) El color azul representa un menor esfuerzo, y el color rojo representa el mayor esfuerzo. Originalmente la matriz de Cockburn representa la formalidad de un proyecto, como la formalidad en un proyecto de software y el esfuerzo son proporcionales se construyó una matriz a semejanza usando la misma tendencia lineal. Ver figura 1.

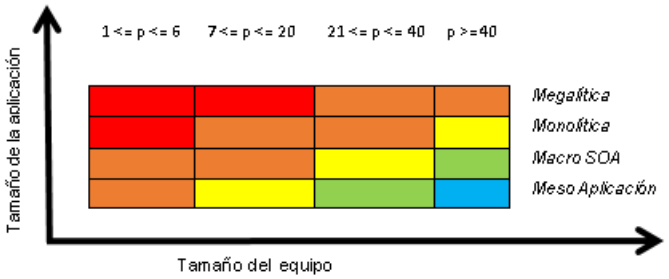

Figure. 1. Adaptación de la escala de Cokcburn para determinar el esfuerzo de la transformación de un monolito a microservicios (Larman 2004,Figura 7.2)

B. Educción y análisis de requisitos

La metodología recomendada como marco de trabajo para la educción y análisis de requisitos de proyectos basados en microservicio, proyectos grandes y equipos pequeños (3 a 9 personas) es scrum junto con el uso de técnicas de otras técnicas como entrevistas, grupos de trabajo y análisis del dominio de tal manera que permita una amplia discusión en la los contextos internos de la aplicación.

C. Especificación de requisitos

La especificación de requisitos se realizará por medio de historias de usuario que definen detalladamente las condiciones iniciales y los criterios de aceptación, una buena práctica para redactar una historia de usuario es haciéndolo por medio de conectores de texto, por ejemplo *"Como un/a .... cuando ... quiero ... porque ..."*. El término "como un/a" se refiere a quien solicita la historia, el "cuando" se refiere a las condiciones que deben cumplirse para llevar acabo la acción, el "quiero" se refiere a la funcionalidad deseada y el "porque" a la justificación de la solicitud de esa funcionalidad. La historia de usuario también se puede complementar con diagramas UML que ilustren mejor los escenarios planteados al igual que prototipos gráficos en caso que se necesitan interfaces de usuario.

Las historias de usuario deben ser validadas antes de transferidas al área de desarrollo, los analistas debe de validar que toda historia de usuario cumpla los siguientes criterios: Independencia, negociable, valiosa al negocio, estimable, pequeña y comprobable tal como lo plantea Bill Wake en su metodología INVEST. Posterior a la validación, cada historia de usuario debe ser estimada en esfuerzo por los stakeholders y priorizada de acuerdo a las necesidades más urgentes del proyecto.

D. Diseño y definición de microservicios

En esta labor nos guiaremos por un enfoque guiado por el diseño DDD el cual propone la definición contextos compuestos por componentes del modelo del negocio que sean consistentes entre ellos, la importancia de los contextos es que cada uno de ellos será candidato para implementarlos como un microservicio. Para definir los contextos comenzaremos a analizar la documentación técnica existente en el monolito como diagramas UML, manuales referencias de alguna API existente y documentación del código fuente. A partir del anterior análisis procederemos a inferir los nombres de los contextos y sus modelos internos para posteriormente diagramarlos, a continuación se muestra un ejemplo de diagrama de contextos que se construye a partir de la documentación existente.

Luego de haber definido los contextos, fragmentaremos el código fuente de acuerdo a la organización de los contextos, moviendo los archivos fuentes a nuevos proyectos que sean representativos de cada contexto. También necesitaremos modificar la manera como se generar los archivo distribuibles de tal manera que no generemos un solo instalador para toda la

aplicación sino un instalador por cada servicio y de esta manera pueda ser desplegado y escalado independientemente. En cuanto al código fuente fragmentado en el monolito todos los servicios SOAP deberán exponer sus funcionalidades por medio de nuevas interfaces rest y las interfaces gráficas deberán conectarse por medio de nuevas interfaces rest para su interacción con la lógica del negocio [7]

E.Desarrollo de los microservicios

La elaboración de un microservicio no comprende solamente la extirpación de un fragmento de código o de alguna de sus tablas de la base de datos sino también del manejo del resto de conceptos necesarios para que la arquitectura sea implementable como lo son el descubrimiento y registro de servicios, la configuración centralizada, resilencia a los fallos, el manejo del balanceo de carga y monitoreo, entre otros. En esta fase la metodología da unas pautas de como debería preparase el ambiente de los microservicios para que tenga una ejecución adecuada durante la coreografía, ejecución autónoma de cada servicio, junto con los otros servicios, a continuación se presenta un resumen de las mismas.

F. Pruebas de integración, contratos y punto a punto

La arquitectura de microservicios facilita refinamiento de los límites de la lógica de negocio lo cual permite aislar las pruebas de unidad haciéndolas más simples y fáciles de entender y mantener .

G. Despliegue, integración y entrega continua

Las aplicaciones basadas en microservicios al tener una naturaleza distribuida pueden ser instaladas en máquinas virtuales o contenedores (máquinas virtuales más simplificadas). Docker es un manejador de contenedores que facilita la creación, aislamiento y administración de contenedores y los deja listos para usar, lo que nos permite simular un entorno distribuido y validar el correcto funcionamiento de la aplicación para corregir errores de comunicación encontrados entre los microservicios.

H. Implementación de la metodología

La aplicación que se escogió para la implementación de la metodología es un software monolítico, de código abierto, escrito en java y MVC Spring Framework , que expone servicios restful tipo CRUD y se conecta a una base de datos Mysql localmente donde almacena la información asociada a un blog (autores, contraseñas, publicaciones y posts) . El código fuente está disponible en su propio repositorio de github .

Aunque este aplicativo no supera las 500 líneas de código por artefacto se escogió para aprovechar su simplicidad e ilustrar la manera cómo se puede transformar un monolito a microservicios. Debido a que es un caso de estudio pequeño se omitieron las etapas correspondientes a la prefactibilidad, educción y especificación de requisitos donde las historias de usuario se reducirían a convertir los contexto del monolito a microservicios como veremos a continuación. Por lo tanto nuestro punto de partida será la ejecución de la etapa diseño de microservicios donde delimitaremos los contextos encontrados en el monolito a partir del análisis de la descripción en github y de la los resultados de la inspección del código fuente. A continuación se muestra un ejemplo de un diagrama de contextos producto de un análisis de la documentación existente. Ver figura 2.

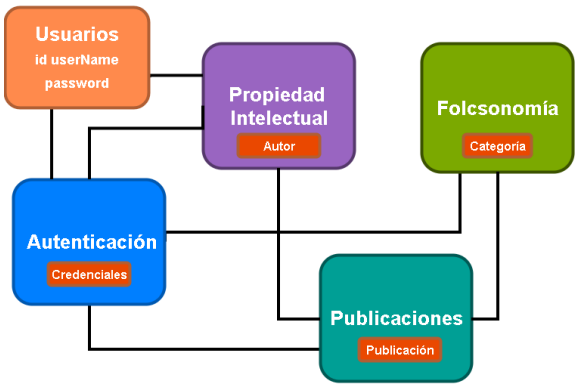

Figure. 2. Contextos existentes en el monolito

Posteriormente, para la transformación del monolito se delimitaron 5 fases, en cada una de ellas se implementaran las etapas de desarrollo e integración continua con la intención de hacer una migración evolutiva para ayudar a la didáctica de la labor [8]. A continuación se describen las fases:

1) Fase I Extracción de microservicios de autenticación y propiedad intelectual: En esta fase se desacoplaron los modelos credenciales y los autores, ambos modelos compartían propiedades sin embargo cada uno se separó en bases de códigos diferentes a dos nuevos microservicios. El código de ambos microservicios se extrajo al igual que se partió la tabla autores en dos nuevas tablas las cuales movieron a dos nuevas base de datos para los servicios autenticación y propiedad intelectual, ambos microservicios continuaron comunicándose con el monolito para soportar el resto de funcionalidades del sistema. También se crearon nuevos servicios que hacen parte de la infraestructura del ecosistema como servicio de registro y descubrimiento por medio de un servidor Eureka donde tanto por medio del cual el monolito y los dos nuevos microservicios se comunican. Se configuró un servicio de configuración centralizada haciendo uso del Spring Cloud Config, Service y de Gitstack como servidor git, un servicio de monitoreo y soporte de cortocircuitos utilizando las bibliotecas de Hystrix, y se creo un servicio de bitácora por medio de mensajería haciendo uso del servidor RabbitMQ y se utilizó como balanceador de carga del cliente las librerías de Ribbon para las llamadas entre microservicios, los resultados de la primera fase pueden ser encontrados en el repositorio de github.

2) Fase II Extracción de microservicios de publicaciones: En este fase se partió por segunda vez el monolito para extraer el servicio de publicaciones el cuál se incorporó a la colección de microservicios, se movió la tabla publicaciones a una nueva base de datos en el microservicio y se integró con el resto de servicios de infraestructura, los resultados de la primera fase pueden ser encontrados en su propio repositorio de github.

3) Fase III Extracción de microservicio de folcsonomía y eliminación de monolito: Finalmente el monolito transformó su lógica restante a un cuarto microservicio y desapareciendo en su totalidad. Para este microservicio se creo nueva base de datos la cual albergará la tabla categorías. También se añadió soporte del spring cloud bus de tal manera que las llamadas de tipo inserción, actualización y borrado se manejan por medio de mensajes a través de RabbitMQ desacoplando de esta manera las llamadas de persistencia entre los microservicios que tengan base de datos, los resultados de la tercera fase pueden ser encontrados en su propio repositorio de github.

4) Fase IV Despliegue de microservicios en docker: Se automatizó el despliegue de los microservicios en contenedores haciendo más sencilla su instalación, simulando un ambiente similar al de producción, puesto que cada microservicio se instaló en un contenedor independiente, los resultados de la primera fase pueden ser encontrados en su propio repositorio de github.

5) Fase V Entrega continua y despliegue de los artefactos de software hasta su salida a producción: En esta fase se automatizó la sincronización de código, la promoción y la publicación de los artefactos de software en los diferentes ambientes de ejecución y para así preparar los diferentes tipos de ambientes para desarrollo, pruebas de errores, validación de requerimientos y producción, para cada uno de ellos se asoció una carpeta compartida en la red donde se desplegaban los artefactos..

V. ANÁLISIS DE RESULTADOS

La transformación de una arquitectura monolítica a microservicios puede ayudar el desarrollo de una aplicación web en las siguientes formas:

TABLE 1. VERIFICACIÓN DE INDICADORES DE HIPÓTESIS

Forma	Razón
Semejanza entre la estructura de la organización y la aplicación web	El software se adapta a las necesidades del negocio y no al contrario
Fragmentando el código fuente legado haciéndolo más legible	Reducir tiempos de mantenimiento del software
Disminuyendo el acoplamiento entre los componentes	Facilitad la escalabilidad de la aplicación y simplifica el mantenimiento
Ofreciendo tareas automatizadas de integración y de entrega continua	Automatización de tareas que reducen los tiempos de implementación y ayudar al aseguramiento de la calidad
Previniendo fallos de sistema no manejados y haciendo diagnósticos de rendimiento en tiempo real.	Aumenta la confiabilidad de la aplicación

V. CONCLUSIONES

La arquitectura microservicios plantea un cambio de paradigma sobre cómo se están actualizando los sistemas legados en la actualidad ya que al proponer una transformación hacia una arquitectura distribuida la aplicación reduce el acoplamiento interno entre componentes simplificando la escalabilidad de los mismos y facilitando la lectura de la base de código fuente que se reorganiza por contextos lo que disminuye los tiempos de reparación de errores o de actualizaciones del sistema.

Sin embargo todas estas bondades necesitan de ciertas condiciones para que puedan ser aprovechadas, entre ellas están una considerable madurez de la aplicación, dominio de la propiedad intelectual del código fuente y un equipo de desarrolladores cualificados en aplicaciones distribuidas.

De igual forma la transformación a una arquitectura de microservicios representa un gran reto puesto que incrementa la complejidad del sistema volcado hacia una comunicación distribuida lo que hace un poco más dispendioso la realización de pruebas de desarrollo y el monitoreo del sistema, a favor de un aumento de la cohesión, disminución del acoplamiento pero con mayor facilidad para escalar sobre la nube, automatización en la entrega de artefactos y los despliegues y mejor control en los fallos generales del sistema en producción.

REFERENCIAS BIBLIOGRÁFICAS

[1] Newman Sam. Building microservices. Primera edición. Sebastopol,California. O'Reilly Media Inc, 2015. ISBN 978-1-4919-5035-7, pp. 2-6

[2] Krueger Ken, 2015, Microservices with Spring Cloud disponible en https://www.udemy.com/microservices-with-spring-cloud/learn/v4/t/lecture/2953328?start=0 [Recuperado 8 de Mayo 2017]

[3] Wolff Eberhard, Microservices: Flexible Software Architecture, Primera edición. Crawfordsville,Indiana. Addison-Wesley, 2016. ISBN 978-0-134-60241-7

[4] Lewis James & Fowler Martin, 2014, Microservices. Imagen], disponible en http://martinfowler.com/articles/microservices.html [Recuperado Recuperado 27 Junio 2016]

[5] Evans Eric. Domain-driven design. Primera edición, Boston, Massachusetts. Addison-Wesley, 2004. ISBN 0-321-12521-5

[6] Bryant Daniel, 2015, Microservice Maturity Model Proposal, disponible en https://taidevcouk.files.wordpress.com/2015/02/microservice-maturity-model-proposal-daniel-bryant-danielbryantuk.pdf [Recuperado 8 de Mayo 2017]
[7] Brown Kyle,2016, Refactoring to microservices, disponible en https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-1/ [Recuperado 08 Mayo 2017]
[8] Caso estudio : restful-blog-microservices, 2017, Disponible en: https://github.com/juanwalker/restful-blog-microservices-phase1 [Recuperado 8 de Mayo 2017]