Dom. Cien., ISSN: 2477-8818
Vol. 6, núm. 2, abril-junio 2020, pp. 763-781

DOI: http://dx.doi.org/10.23857/dc.v6i2.1193

Ciencias tecnicas y aplicadas

Artículo de investigación

Migración de un monolito a una arquitectura basada en microservicios, caso de estudio sistema “kbus”

Migration of a monolith to a microservices-based architecture, case study "kbus" system

Migração de um monólito para uma arquitetura baseada em microsserviços, sistema "kbus" de estudo de caso

Yeferson Torres-Berru [I]
ymtorres@tecnologicoloja.edu.ec
https://orcid.org/0000-0003-3784-3493

Jeferson Camacho-Macas [II]
jeferson.camacho@kradac.com
https://orcid.org/0000-0002-2479-7320

John Solano-Cabrera [III]
jpsolano@tecnologicoloja.edu.ec
https://orcid.org/0000-0002-3479-1491

Luis Fernando León-Pinzón [IV]
fernando.leon@tecnologicoloja.edu.ec
https://orcid.org/0000-0001-8342-0673

Correspondencia: ymtorres@tecnologicoloja.edu.ec

***Recibido:** 29 de febrero de 2020 ***Aceptado:** 30 de marzo de 2020 * **Publicado:** 16 de abril de 2020

[I] Máster Universitario en Ingeniería de Software y Sistemas Informáticos, Ingeniero en Sistemas, Docente en el Instituto Superior Tecnológico Loja, Loja, Ecuador.

[II] Ingeniero en Electrónica y Telecomunicaciones, Kradac Cia Ltda. Gerente Sistema Kbus, Loja, Ecuador.

[III] Ingeniero en Sistemas, Docente en el Instituto Superior Tecnológico Loja, Loja, Ecuador.

[IV] Ingeniero en Informática, Docente en el Instituto Superior Tecnológico Loja, Loja, Ecuador.

Resumen

El sistema de monitoreo y control de flota Kbus está presente en 15 ciudades del Ecuador y 2 de Colombia, maneja diariamente alrededor de 5 millones de datos los cuales en mayor medida provienen de los equipos de rastreo GPS, además recibe 200 mil peticiones al día de los diversos clientes, sea desde la web o desde las aplicaciones móviles. Para realizar la migración se obtuvo el modelo tecnológico del monolito, los recursos disponibles, el número de peticiones diarias, etc. Gracias a esta información fue posible la definición de los microservicios, la comunicación entre ellos y su agrupación, lo que permitió obtener una nueva Arquitectura. El monolito previo se encontraba en Java, sin embargo, para la migración se usó NodeJs bajo un contenedor Docker, para la implementación se trazó un plan de trabajo mediante Sprints basados en la metodología SCRUM. Finalmente se realizó la comparativa de rendimiento entre las 2 arquitecturas, obteniendo mejores resultados en tiempo de carga y de actualización con la arquitectura de microservicios.

Palabras clave: Arquitectura de software; cloud computing; microservicios; monolitos, NodeJS.

Abstract

The Kbus fleet monitoring and control system is present in 15 cities in Ecuador and 2 in Colombia, handles about 5 million data daily, most of which comes from GPS tracking equipment, and also receives 200,000 requests a day from various customers, either from the web or from mobile applications. To perform the migration, we obtained the technological model of the monolith, the available resources, the number of daily requests, etc. Thanks to this information it was possible to define the microservices, the communication between them and their grouping, which allowed to obtain a new Architecture. The previous monolith was in Java, however, for the migration NodeJs were used under a Docker container, for the implementation a work plan was drawn up using Sprints based on the SCRUM methodology. Finally, the performance comparison between the two architectures was carried out, obtaining better results in loading time and updating with the microservices architecture.

Keywords: Software arquitecture; cloud computing; microservices; monolites; NodeJs.

Resumo

O sistema de monitoramento e controle de frota da Kbus está presente em 15 cidades do Equador e 2 na Colômbia, gerencia cerca de 5 milhões de dados diariamente, a maioria proveniente de equipamentos de rastreamento GPS, e também recebe 200.000 solicitações por dia de os vários clientes, da Web ou de aplicativos móveis. Para realizar a migração, foram obtidos o modelo tecnológico do monólito, os recursos disponíveis, o número de solicitações diárias etc. Graças a essas informações, foi possível definir os microsserviços, a comunicação entre eles e seu agrupamento, o que permitiu obter uma nova arquitetura. O monólito anterior estava em Java, no entanto, para os NodeJs de migração foram usados em um contêiner do Docker, para a implementação foi elaborado um plano de trabalho usando Sprints com base na metodologia SCRUM. Por fim, foi feita uma comparação de desempenho entre as duas arquiteturas, obtendo melhores resultados no tempo de carregamento e atualização com a arquitetura de microsserviços.
Palavras-chave: Arquitetura de software; computação em nuvem; microsserviços; monólitos, NodeJS.

Introducción

La arquitectura de microservicios es una alternativa a la arquitectura monolítica en el desarrollo de software [1] [2], con objetivo de mejorar la disponibilidad de los sistemas y solucionar algunos problemas que se presentan en especial cuando se realizan actualizaciones [3] [4].

Los microservicios permiten la elasticidad rápida y automatizada. Los mecanismos de tolerancia a fallos logran que los errores de los microservicios individuales no afecten a otros servicios gracias al aislamiento existente entre ellos. "Dado que los servicios pueden fallar en cualquier momento es importante ser capaz de detectar los errores rápidamente y, de la misma forma, reiniciarlos, por lo que es esencial para el éxito plantear ajustes para un monitoreo avanzado que evalúen variables de rendimiento, el número de solicitudes por segundo" [5] [6].

Actualmente el sistema de control, monitoreo satelital y gestión de recaudo, para flotas de transporte "Kbus" utiliza una arquitectura monolítica en los servicios que presta a las 5 aplicaciones tanto web como móviles en cada uno de los subsistemas que lo componen. Al utilizar este tipo de arquitectura limita la posibilidad de modificar un módulo sin afectar a los demás causado molestias a los usuarios y afectando la disponibilidad del sistema, así como su escalabilidad, para solucionar

este inconveniente se ve oportuno migrar a una arquitectura basada en microservicios con el propósito de tener independencia entre los módulos que componen el sistema.

La mayoría de los monolitos son demasiado grandes para ser migrados hacia otra arquitectura en un solo paso, por lo que esta migración debe llevarse a cabo gradualmente, en varias etapas, también llamados incrementos donde cada incremento es liberado en producción [6]. Para realizar la propuesta de migración a una arquitectura basada en microservicios y lograrlo con éxito, se Identificaron los límites de negocio [7] [8], se definieron los módulos esenciales y se evaluaron las actualizaciones que sufren, para posteriormente adaptarla a una nueva arquitectura basada en microservicios, para ello se planteó la metodología a seguir para llevar a cabo la migración y desarrollar un cronograma de implementación.

“Microservicios” (MS) es un término al que varios autores definen de manera diferente: un estilo arquitectónico, un enfoque, una tecnología, una arquitectura. Sin embargo, en lo que todos coinciden es que los microservicios representan una mejora y alternativa a las aplicaciones con arquitecturas tradicionales (n-capas) [9].

Para el desarrollo de los diferentes servicios que intervienen en el ejemplo se ha utilizado la tecnología Node JS que permite la programación del lado servidor apoyándose en JavaScript y ofrece una configuración del apartado entrada/salida del servidor orientado a eventos, lo que facilita mucho el despliegue global de nuestra aplicación. Además, Node JS permite la construcción, el despliegue y la escalabilidad de servicios web de manera sencilla que son características imprescindibles en el desarrollo de arquitecturas basadas en microservicios [10].

Docker es una plataforma “Open Source” destinada a la construcción y el despliegue de aplicaciones en sistemas distribuidos de un modo en gran parte automatizado [11].

Docker utiliza un sistema de virtualización ligera para la puesta en escena de sus máquinas virtuales (denominadas contenedores). Esto significa, que mientras una máquina virtual tiene un sistema operativo completo, una memoria propia y emula los recursos del sistema operativo original, múltiples contenedores Docker pueden correr a la vez apoyándose sobre un único “motor Docker” sin necesidad de virtualizar un sistema operativo nuevo, con el ahorro de recursos que esto supone [12].

Materiales y métodos

La migración de una aplicación con determinada arquitectura hacia otra requiere una serie de pasos ordenados dentro de una ruta de migración que garantice el éxito de este proceso, esto con el fin de adaptar la aplicación web actual a los procesos de negocio actuales del sistema [13].

La transición no puede realizarse de manera automática, sino que es necesario especificar modelos que involucren pasar primeramente por un enfoque orientado a servicio [14].

Para realizar la migración de una arquitectura monolítica a una basada en Microservicios para el sistema KBUS, se realizó las siguientes actividades:

Conocer el estado actual del sistema Kbus

Permitió tener una mejor perspectiva del problema, conociendo de cerca sus fortalezas y debilidades, será posible encontrar una solución para el beneficio del sistema y de los usuarios a quien va destinado el servicio.

Definir los módulos esenciales

El sistema Kbus cuenta con tres módulos esenciales de reportes, administración y alarmas.

Módulo de Reportes: Permite generar reportes con información para el dueño de la unidad como número de vueltas realizadas, dinero recaudado, por parte de las cooperativas filtrado de Buses, estaciones e itinerarios, etc.

Módulo de Administración: Permite el control y gestión de las flotas de buses.

Módulo de Alarmas: Existen tres tipos de alarmas estas pueden ser por medio de un mail, mensaje de texto o una alerta emitida del sistema. Estas notificaciones son enviadas a los diferentes usuarios (administrador, cooperativa o dueño de la unidad) en los siguientes casos: exceso de velocidad, retrasos en la ruta, adelantos en las rutas, desvió de ruta, no cumplimiento de la ruta, sobrepaso de multas, etc.

Adaptar a una nueva arquitectura basada en microservicios

Con los servicios previamente identificados, cada uno con un grado de independencia y desplegados de forma automática, se define un mecanismo ligero de comunicación para la interacción entre ellos, teniendo como resultado que cada componente se tratará como una unidad independiente y única, lo que permite modificar, realizar actualizaciones sobre un componente sin

afectar a los demás, garantizando así la disponibilidad del sistema Kbus. Vale recalcar que dentro de la arquitectura monolitica la interrupción del sistema Kbus por actualizaciones es muy frecuente lo que involucra molestias a los usuarios, pérdida de tiempo y dinero.

Implementar los microservicios

Comparativa arquitectura monolítica vs microservicios

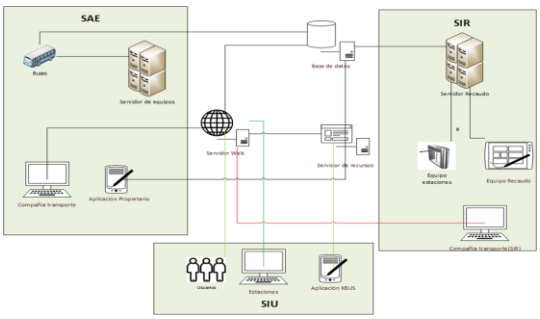
En esta fase se realizara la comparativa entre la arquitectura monolítica y la de microservicios esto se realizó en rendimiento (tiempo de carga) y estrés para lo cual se usó Jmeter además se midió el tiempo de puesta en producción de un cambio en el sistema (actualización).

Resultados y discusión

Estos dos apartados suelen aparecer juntos en muchos trabajos. No debemos confundir esta discusión o análisis con la obtención de conclusiones, algo que depende tanto de los resultados y de su análisis como del marco teórico y de los objetivos.

Estado Actual Kbus

Figura 1. Arquitectura KBUS



En la figura 1 se observa el estado actual del sistema KBUS y sus subsistemas, se muestra la implementación física del mismo, así como las iteraciones con las flotas de buses y los usuarios finales a los cuales se presta el servicio de los cuales rescatamos los siguientes componentes de software.

Sistema SAE WEB: Sistema mediante el cual acceden las compañías de transporte, entidades de control y propietarios y pueden ver la información de operación de los vehículos además de los usuarios administradores pueden crear nuevas empresas, buses, rutas.

Aplicación Propietario: Aplicación móvil disponible en Android e IOS, mediante la cual el propietario de un vehiculo puede revisar la operación y el recaudo del mismo.

Sistema SIR WEB: Sistema mediante el cual acceden las compañías de transporte, entidades de control y propietarios y pueden ver la información de los ingresos generados por sus vehículos.

Sistema SIU WEB: Sistema mediante el cual se conectan las terminales ubicadas en las estaciones de abordaje de pasajeros y en las cuales se muestra los tiempos de llegada de los vehículos.

Aplicación KBUS SIU: Aplicación móvil disponible en Android e IOS mediante la cual los pasajeros pueden revisar tiempos de llegada de buses, rutas, horarios hacer denuncias y sugerencias.

Servicios actuales

Se realizó la clasificación de servicios por subsistema, detallando su utilización, en la plataforma desde la cual es llamada, los parámetros de entrada, la respuesta obtenida y la prioridad para el negocio de cada uno de ellos (Figura 1), para ello se tendrán en cuenta los servicios que son consultas y reportes debido a que los servicios administrativos serán clasificados de manera general pues en todos se realizan las operaciones básicas del CRUD (Create, Read, Update, Delete).

Figura 2. Ejemplo se servicios actuales

SAE

Nombre Recurso	Utilidad	Parámetros de entrada	Parámetros de Salida	Acceso
Información Vehículo	Permite obtener información del vehículo por medio del registro municipal o placa.	valor (Pudiendo ser placa o registro municipal), idCiudad (String)	Devuelve un objeto JSON con la información del vehículo.	*Sistema web *App Propietario
Vehículos empresa	Permite obtener los vehículos de una empresa	idEmpresa (int)	Devuelve un json con la lista de vehículos	*Sistema web
Posición vehículo	Permite obtener información del vehículo y su posición actual. La	idVehiculo (int)	Devuelve un objeto JSON con la información	*Sistema web *App Propietario

En total en el sistema Kbus se usan 80 servicios de consultas y reportes que son usados en el SAE, SIR, SIU y en el sistema Web, aplicación móvil de propietarios y aplicación para usuarios.

Además de los servicios administrativos CRUD para las Ciudades, Empresas, Rutas, Vehículos, Puntos de Control, Estaciones, Multas, Tarjetas, Puntos de Venta, Horarios, Propietarios que suman un total de 44 recursos. Siendo un total de 124 recursos a migrar.

Identificación de candidatos monolitos

Para la identificación de candidatos a monolitos las opciones a elegir son el subsistema al que pertenecen, su relación con el negocio o simplemente el módulo del que forman parte. Según el diagrama de despliegue el modelo actual consta de un solo monolito de manera que si tenemos que realizar algún cambio en alguno de los servicios antes mencionados no tendríamos funcionalidad en ninguno de los subsistemas antes mencionados (Figura 3).

Figura 3. Características para identificación de monolitos

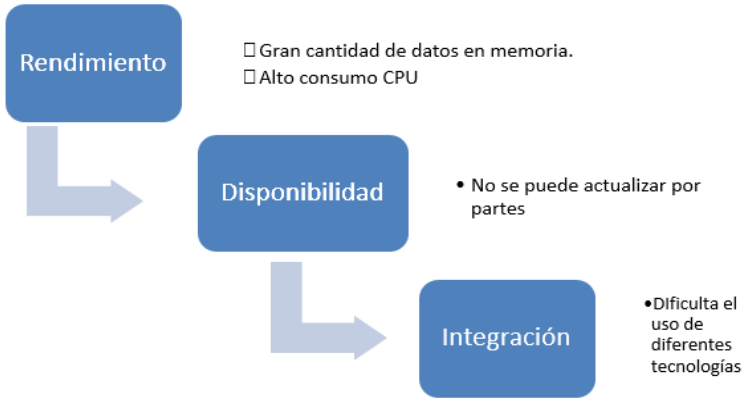
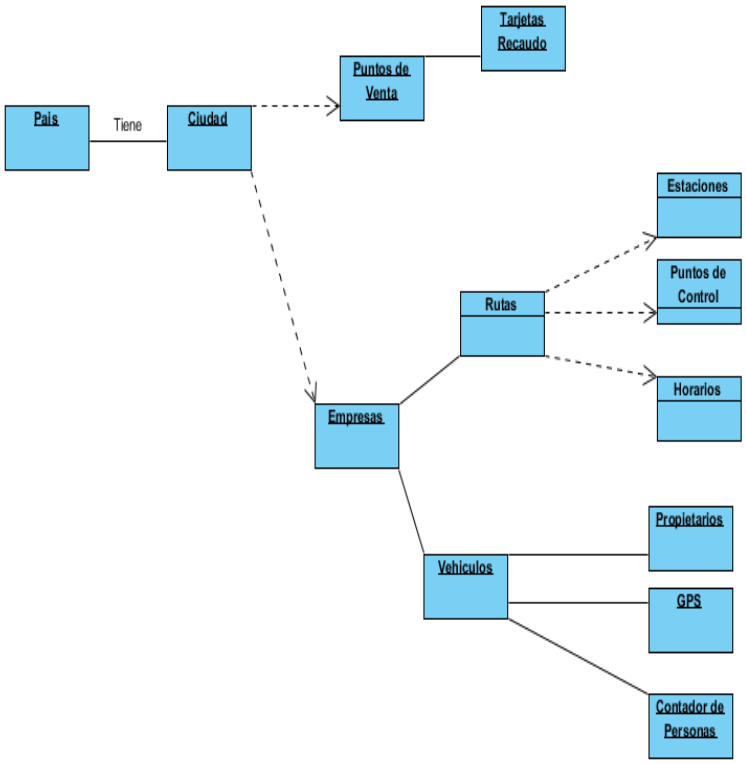


Diagrama de objetos

Para tener un mejor nivel de abstracción se clasifican los objetos en ayudando a obtener un mejor modelo para de los microservicios (Figura 4).

Figura 4. Diagrama de Objetos



A continuación, se detalla los componentes del sistema KBUS en base a su jerarquía dentro del sistema.

Ciudad: Lugar donde se encuentra instalado el sistema de control y monitoreo KBUS.

Empresa: Compañía de transporte Urbano que contrata KBUS o alguno de sus subsistemas SAE, SIR o SIU.

Ruta Recorrido que es realizado por el bus el mismo que debe ser cumplido en los tiempos preestablecidos.

Estación Lugar donde ingresan los pasajeros que se movilizan por el sistema de transporte urbano.

Punto de control Lugar de la ruta donde el bus debe llegar en un tiempo determinado con el fin de controlar el cumplimiento de frecuencias.

Horarios Horas en las que está activo el servicio de transporte, estas pueden variar dependiendo el día de la semana o feriados etc.

Propietarios Personas que adquieren los vehículos de las empresas que forman parte de KBUS.

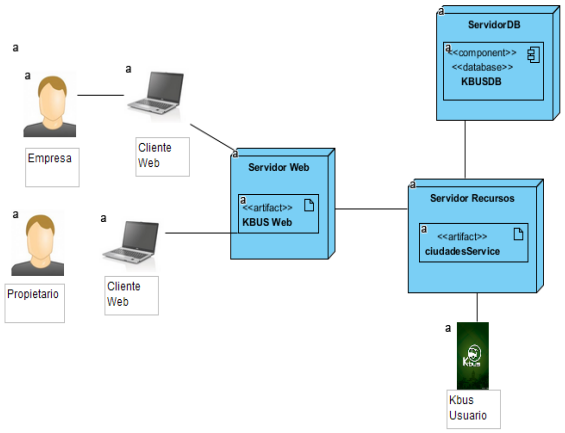
GPS Dispositivo tecnológico instalado en los vehículos el cual envía la información de rastreo en tiempo real al servidor KBUS.

Diseño de los microservicios

Para el diseño de los microservicios tendremos en cuenta los subsistemas, así como también los objetos principales los cuales se van a tomar en cuenta para poder considerar un monolito.

Ejemplo de Monolito

Figura 5. Diagrama de despliegue ciudades Service



El micro servicio ciudadesService cuenta con cuatro servicios de reportes además de los servicios de gestión de ciudades, turnos y estaciones este servicio interactúa con el sistema web SAE y SIR además de la aplicación móvil SIU como se puede observar en el siguiente diagrama de despliegue (Figura 5).

MicroServicios

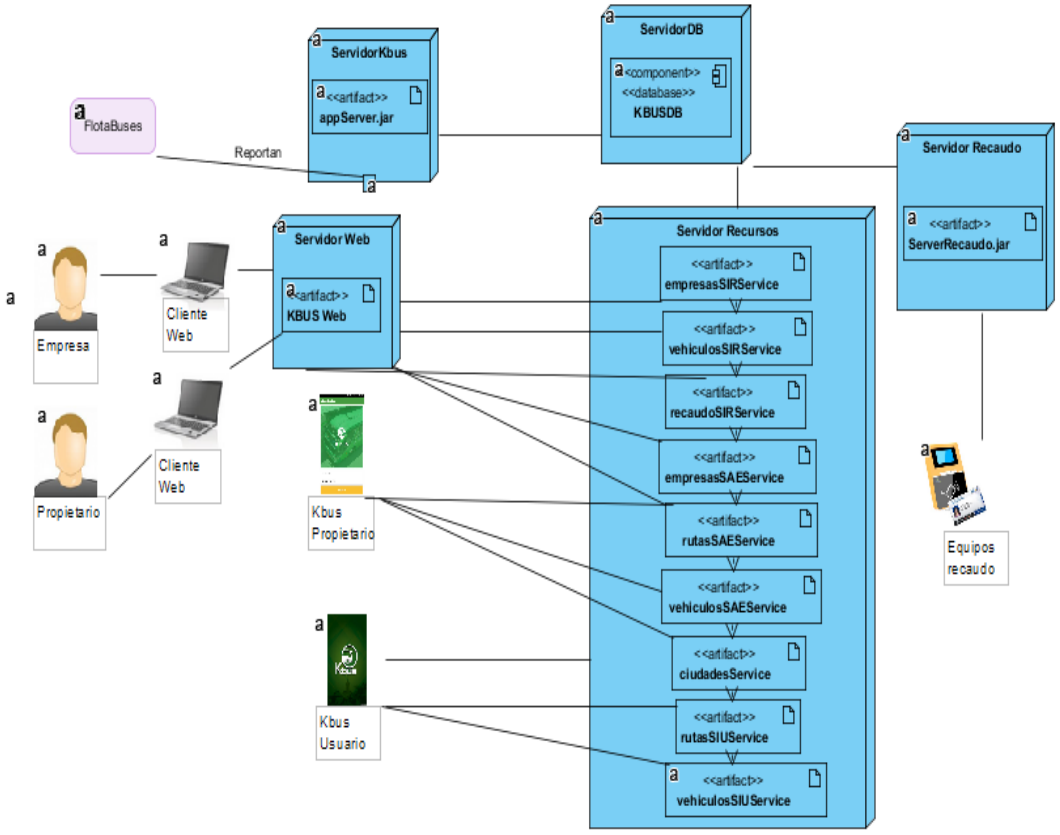
Por lo tanto como se observa en la Figura 6, tendremos 10 microservicios distribuidos de la siguiente manera 3 para el SAE, 2 para el SIU, 3 para el SIR y uno que es compartido por los 3 subsistemas.

Figura 6. Valoración de los microservicios

	SubSistema	Objeto	Microservicio
1	SAE SIR SIU	Ciudad	ciudadesService
2	SAE	Empresa	empresasSAEService
3	SIR	Empresa	empresasSIRService
4	SAE	Ruta	rutasSAEService
5	SIU	Ruta	rutasSIUService
6	SAE	Vehículo	vehiculosSAEService
7	SIU	Vehículo	vehiculosSIUService
8	SIR	Vehículo	vehiculosSIRService
9	SIR	PuntosVenta, Tarjetas	recaudoSIRService

Arquitectura de microservicios

Figura 7. Arquitectura de Microservicios



Planificación de riesgos

En esta sección se muestran los riesgos y su plan de mitigación debido a que la migración engloba un sin número de riesgos a considerar, es necesario tener en cuenta que un plan de contingencia puede mitigar el impacto, frente a cualquier percance.

Cada uno de los riesgos, tendrán su respectiva descripción, nivel de riesgo y plan de acción en caso de presentarse

Migración de microservicios

Se detalla cada una de las actividades para la migración de los microservicios.

Implementación. Proceso de codificación de la solución en el lenguaje de programación seleccionado en la fase anterior y en base a las definiciones de entradas y salidas para cada microservicio.

Pruebas Fase en la que se realiza el test tanto a nivel de código como de funcionalidades de los microservicios, en esta fase se utilizara herramientas como SOAPUI y POSTMAN para verificar que se encuentren realizados los controles, y la información esta verificada correctamente.

Despliegue Puesta en marcha del servicio usando un contenedor de aplicaciones como DOCKER aquí se realizarán pruebas de estrés.

Migración Proceso en el cual se cambia acceso a los recursos ya sea en el sistema web o en las aplicaciones móviles.

Recursos usados

Los recursos usados para realizar la migración, dentro del personal del sistema KBUS son los siguientes (Tabla 1).

Tabla 1. Recursos disponibles

Nombre del recurso	Tipo	Etiqueta de material	Iniciales	Grupo	Capacidad máxima
Programador 1	Trabajo		P1		100%
Programador 2	Trabajo		P2		100%
Control de Calidad	Trabajo		QA1		100%
Programador móvil	Trabajo		P2		100%

A continuación, se detalla el perfil laboral de los recursos disponibles:

Programador 1: Programador con conocimientos en Node Js, construcción de los microservicios, así como el despliegue de los mismos en el contenedor Docker.

Programador 2: Programador con conocimientos en Node Js, Angular y EXTJS, apoyo construcción de los microservicios cambio en el consumo de los servicios actuales a los microservicios.

Programador móvil: Programador con conocimientos en Ionic y Android, consumo de recursos a los microservicios.

Control de Calidad: Trabajara de la mano con el equipo de desarrollo probando los nuevos recursos, así como los aplicativos finales luego del cambio a microservicios.

Figura 8. Visión general de los recursos

Planificación de riesgos

La migración engloba un sin número de riesgos a considerar, es necesario tener en cuenta que un plan de contingencia puede mitigar el impacto, frente a cualquier percance. Cada uno de los riesgos se evaluó en base a una descripción, nivel de riesgo y plan de acción en caso de presentarse.

Comparativa arquitectura monolítica vs microservicios

En la presente sección se muestra la comparativa en tiempos de carga entre la arquitectura monolítica y la de microservicios, basado en el trabajo realizado en base a la herramienta

POSTMAN además se midió el tiempo de despliegue al realizar la actualización de un recurso con la arquitectura monolítica y en la de microservicios (Tabla 2).

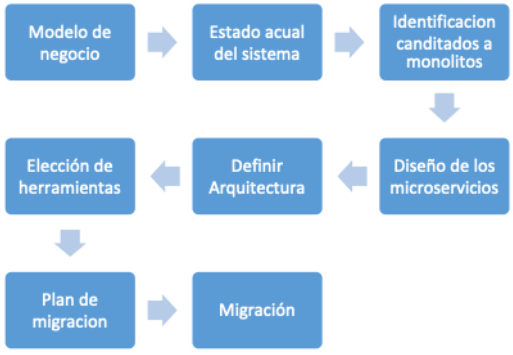
Tabla 2. Comparativa Arquitectura Monolítica vs Microservicios

Aplicativo	Tiempo de carga Monolítica	Tiempo de carga Microservicios	Tiempo de actualización Monolítica	Tiempo de Actualización Microservicios
SAE Web	2.1 segundos	0.9 segundos	3 minutos	1 minuto
Propietario	1.2 segundos	0.5 segundos	5 minutos	1 minuto
SIR Web	2.0 segundos	0.8 segundos	5 minutos	1 minuto
SIU Web	1.5 segundos	0.6 segundos	4 minutos	1 minuto
SIU App	1.9 segundos	0.2 segundos	3 minutos	1 minuto

Conclusiones

Se planteó una metodología a seguir para llevar a cabo la migración, la cual contiene los siguientes pasos que para mejor comprensión se detallaran en el siguiente diagrama.

Figura 9. Diagrama de pasos necesarios para la migración



Se desarrolló la implementación en base a los recursos disponibles en la empresa, el perfil de cada uno y las actividades a realizar, cronograma final de implementación y los resultados de la migración realizada así como el despliegue de la nueva arquitectura.

Además, se determinó que.

Es vital tener una documentación clara del sistema implementado con el objetivo de enfocar la migración hacia un enfoque correcto y poder priorizar los servicios más importantes para el modelo de negocio, de manera que la migración cumpla con su objetivo principal que es mejorar el rendimiento del sistema para los clientes.

La comunicación entre los microservicios puede resultar compleja debido a la información que debe ser distribuida a cada uno de estos y a las llamadas de los servicios REST para procesar los datos enviados.

Tener la independencia de cada uno de los servicios permite hacer su mantenimiento y desarrollo mucho más simple, pues el enfoque esta puesto en esta tarea y en ese contexto en específico. Adicionalmente permite cambiar técnicamente todo el sistema, permitiendo escoger diferentes lenguajes y herramientas que pueden resolver mejor un determinado problema.

El monitoreo y las pruebas para asegurar el correcto funcionamiento de los microservicios se torna complejo debido a la cantidad concebida de los mismos, ya que todos estos módulos tienen un comportamiento dinámico.

Referencias

1. F. Bucshmann, K. Henney, and D. C. Schmidt, Pattern-oriented software architecture. Chichester: Wiley, 1996.
2. P. C. Clements, "A survey of architecture description languages" Procedings of the 8th International Workshop on Software Specification and Design, Schloss Velen, 1996, pp. 16-25.
3. R. Monroe et al, Stylized architecture, design patterns and objects Carnegie Mellon University 1997.
4. Microsoft,«Layered application.,» 2004. [En línea]. Available: http://msdn.microsoft.com /architecture/patterns/default.aspx?pull=/libra. [Último acceso: 20 05 2017].

5. C. Reynoso, Introducción a la Arquitectura de Software, Universidad de Buenos Aires 2004.
6. M. Villamizar et al., "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud" 2015 10th Computing Colombian Conference (10CCC), Bogota, 2015, pp. 583-590.
7. X. Feng et al , «REST : An Alternative to RPC for Web Services Architecture» 2009.
8. A.Manso, «SOA y Web Services» 2011. [En línea]. Available: http://soawebs.blogspot.com.co/2011/01/soa-y-web-services.html. [Último acceso: 19 05 2017].
9. IBM,Microservices 2007 .[Enlínea]. Available: https://www.ibm.com/developerworks/ssa/webservices/ newto/ws-websvc.html. [Último acceso: 21 05 2017].
10. S. Snepe, D. Namiot, «On micro-services architecture.» International Journal of Open Information Technologies, pp. 24-28, 2014.
11. C. de la Torre et al , «Microsoft Azure: Azure Service Fabric y la arquitectura de microservicios» Microsoft Magazine, 2015.
12. W. Hasselbring, «Microservices for Scalability.» de Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering. 2016
13. N. Dragoni et al, «Microservices : yesterday , today , and tomorrow.» 2016 [En línea]. Available: https://doi.org/10.13140/RG.2.1.3257.4961.
14. RedHat, «CONSIGA UNA ARQUITECTURA DE MICROSERVICIOS EXITOSA» 2016.
15. H. Knoche, «Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices,» In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering. 2016 , pp. 121-124.

References

1. F. Bucshmann, K. Henney, and D. C. Schmidt, Pattern-oriented software architecture. Chichester: Wiley, 1996.
2. P. C. Clements, "A survey of architecture description languages" Procedings of the 8th International Workshop on Software Specification and Design, Schloss Velen, 1996, pp. 16-25.

3. R. Monroe et al, Stylized architecture, design patterns and objects Carnegie Mellon University 1997.
4. Microsoft, "Layered application.," 2004. [Online]. Available: http://msdn.microsoft.com /architecture/patterns/default.aspx?pull=/libra. [Last access: 20 May 2017].
5. C. Reynoso, Introduction to Software Architecture, University of Buenos Aires 2004.
6. M. Villamizar et al., "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud" 2015 10th Computing Colombian Conference (10CCC), Bogota, 2015, pp. 583-590.
7. X. Feng et al, "REST : An Alternative to RPC for Web Services Architecture" 2009.
8. A.Manso, «SOA and Web Services» 2011. [Online]. Available: http: //soawebs.blogspot. com.co/2011/01/soa-y-web-services.html. [Last access: 19 May 2017].
9. IBM, Microservices 2007. [Online]. Available: https://www.ibm.com/developerworks/ssa / webservices / newto / ws-websvc.html. [Last access: 21 May 2017].
10. S. Snepe, D. Namiot, "On micro-services architecture." International Journal of Open Information Technologies, pp. 24-28, 2014.
11. C. de la Torre et al, «Microsoft Azure: Azure Service Fabric and the architecture of microservices» Microsoft Magazine, 2015.
12. W. Hasselbring, "Microservices for Scalability." from Proceedings of the 7th ACM / SPEC on International Conference on Performance Engineering. 2016
13. N. Dragoni et al, "Microservices: yesterday, today, and tomorrow." 2016 [Online]. Available: https://doi.org/10.13140/RG.2.1.3257.4961.
14. RedHat, «GET A SUCCESSFUL MICROSERVICE ARCHITECTURE» 2016.
15. H. Knoche, "Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices," In Proceedings of the 7th ACM / SPEC on International Conference on Performance Engineering. 2016, pp. 121-124.

Referências

1. F. Bucshmann, K. Henney e D. C. Schmidt, arquitetura de software orientada a padrões. Chichester: Wiley, 1996.

2. P. C. Clements, "Um levantamento das linguagens de descrição da arquitetura" Procedimentos do 8º Workshop Internacional sobre Especificação e Design de Software, Schloss Velen, 1996, pp. 16-25.
3. R. Monroe et al., Arquitetura estilizada, padrões de design e objetos Carnegie Mellon University 1997.
4. Microsoft, "Aplicativo em camadas.", 2004. [Online]. Disponível: http://msdn.microsoft.com /architecture/patterns/default.aspx?pull=/libra. [Último acesso: 20 de maio de 2017].
5. C. Reynoso, Introdução à Arquitetura de Software, Universidade de Buenos Aires 2004.
6. M. Villamizar et al., "Avaliando o padrão de arquitetura monolítica e de microsserviço para implantar aplicativos da Web na nuvem" 2015 10ª Conferência Colombiana de Computação (10CCC), Bogotá, 2015, pp. 583-590.
7. X. Feng et al, "REST: uma alternativa ao RPC para arquitetura de serviços da Web" 2009.
8. A.Manso, «SOA e Serviços da Web» 2011. [Online]. Disponível: http: //soawebs.blogspot. com.co/2011/01/soa-y-web-services.html. [Último acesso: 19 de maio de 2017].
9. IBM, Microservices 2007. [Online]. Disponível: https://www.ibm.com/developerworks/ssa / webservices / newto / ws-websvc.html. [Último acesso: 21 de maio de 2017].
10. S. Snepe, D. Namiot, "Na arquitetura de microsserviços". Jornal Internacional de Tecnologias de Informação Abertas, pp. 24-28, 2014.
11. C. de la Torre et al, «Microsoft Azure: Azure Service Fabric e a arquitetura dos microsserviços» Microsoft Magazine, 2015.
12. W. Hasselbring, "Microsserviços para escalabilidade". dos Anais da 7ª ACM / SPEC na Conferência Internacional de Engenharia de Desempenho. 2016
13. N. Dragoni et al, "Microsserviços: ontem, hoje e amanhã". 2016 [online]. Disponível: https://doi.org/10.13140/RG.2.1.3257.4961.
14. RedHat, «OBTENHA UMA ARQUITETURA DE MICROSERVIÇOS DE SUCESSO» 2016.
15. H. Knoche, "Sustentando o desempenho do tempo de execução enquanto moderniza de forma incremental o software monolítico transacional para microsserviços", Nos Anais do

7º ACM / SPEC na Conferência Internacional sobre Engenharia de Desempenho. 2016, pp. 121-124.