UNIVERSIDAD DE ANTIOQUIA

Diseño y desarrollo de una aplicación web para la gestión de datos de los resultados de las ejecuciones de pruebas automatizadas de la empresa Sofka, bajo el marco de trabajo Scrum y utilizando una arquitectura basada en microservicios

Autor

Camilo Posada Angel

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Ingeniería de Sistemas.

Medellín, Colombia

2019

Diseño y desarrollo de una aplicación web para la gestión de datos de los resultados de las ejecuciones de pruebas automatizadas de la empresa Sofka, bajo el marco de trabajo Scrum y utilizando una arquitectura basada en microservicios

Camilo Posada Angel

Informe de práctica como requisito para optar al título de:
Ingeniero de Sistemas.

Asesores (a) o Director(a) o Co- Directores(a).

Manuel Salas
Ingeniero de Sistemas
Automatizador

Jeysson Pérez Gómez
Ingeniero de Sistemas
Especialista en Gerencia Integral

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Ingeniería de sistemas.
Medellín, Colombia
2019

Diseño y desarrollo de una aplicación web para la gestión de datos de los resultados de las ejecuciones de pruebas automatizadas de la empresa Sofka, bajo el marco de trabajo Scrum y utilizando una arquitectura basada en microservicios

Resumen.

Debido a la necesidad de obtener en todo momento las métricas de calidad de los proyectos de software en los que se ve involucrada la empresa Sofka, surgió la necesidad de realizar una plataforma web centralizada donde se pudiera ver el estado del producto de software. Para la realización de esta plataforma se planteó una arquitectura basada en microservicios, lo que permitió un mayor modularidad y una fácil distribución de la carga de trabajo entre los miembros del equipo.

Los microservicios de los que estuve encargado fueron: el módulo de autenticación y registro de usuarios, el api Gateway, el receiver o el microservicio encargado de recibir, mas no de procesar, la información enviada por los robots de automatización y el diseño de la arquitectura de la interfaz de usuario usando el framework angular 6. Todo esto se realizó bajo un marco de trabajo SCRUM bajo un ciclo de vida DevOps lo que permitió estar continuamente integrando las funcionalidades para así poder probar y validar de manera temprana que se estuviera construyendo un producto que en realidad si iba a generar valor a los clientes de Sofka.

Introducción.

Sofka es una empresa de desarrollo de software, donde su filosofía es la construcción de software de manera incremental y funcional bajo el marco de trabajo de las metodologías ágiles, buscando la continua participación y colaboración de los clientes durante todo el proceso de ciclo de vida de desarrollo de la solución tecnológica, es por esto que en el año 2017 se funda el área de QA buscando así garantizar la calidad de los productos construidos.

Debido a que los clientes son cada vez son más exigentes sobre la calidad de los productos, la vieja tendencia en la que las pruebas de software se dejaban para la última etapa está quedando atrás. Se busca entonces que el proceso de pruebas esté alineado a todo el ciclo de vida de desarrollo que proponen los marcos de trabajo de las metodologías ágiles, haciendo que la actividad de pruebas se vuelva algo esencial en el desarrollo de un producto de software (Serna M, 2018). Es debido a esto que nace la necesidad para el área de calidad de Sofka, de desarrollar una plataforma en la cual se consolide toda la información de los resultados de las pruebas, logrando así tener en todo momento disponible las métricas de calidad que ayudan a asegurar que se tiene un producto que satisface todas las necesidades del cliente.

Objetivo general

Diseñar y desarrollar una aplicación web para la gestión de datos de los resultados de las ejecuciones de pruebas automatizadas de la empresa Sofka, bajo el marco de trabajo Scrum y utilizando una arquitectura basada en microservicios.

Objetivos específicos

- Diseño de una arquitectura basada en microservicios y contenedores Docker
- Diseño y desarrollo del módulo que se encarga de recibir los datos enviados por los robots que se encargan de ejecutar las pruebas automatizadas.
- Diseño y desarrollo del módulo de registro y autenticación de usuarios
- Diseño y desarrollo del módulo api Gateway para la comunicación entre los microservicios
- Diseño de la arquitectura de la interfaz de usuario utilizando el Framework Angular 6

Planteamiento del Problema.

Tanto para el área de calidad de software como para el área de desarrollo, las métricas de calidad del producto de software que se está desarrollando son de suma importancia para tomar decisiones sobre el estado actual y el grado de satisfacción que el cliente tiene sobre lo desarrollado. Es por esto por lo que surgió la necesidad de tener una plataforma web centralizada, en la cual se pudieran ver de manera ordena, fácil e intuitiva, en la cual se desplegará la información recolectada por los robots de automatización que realizan pruebas para las aplicaciones en las que es contratada la empresa Sofka.

Se buscó entonces con esto la automatización del proceso de reporte, para que este se fuera generando en tiempo real a medida que los robots de automatización corrieran las pruebas, tanto funcionales como end to end, pudiendo así los gerentes de proyecto, líderes de calidad, scrum masters, etc, entrar a ver el estado actual de la proyecto o producto que se está desarrollando para así asegurar que se cumplen las expectativas tanto del cliente como de usuario final. Se propuso para esto la construcción de una plataforma web centralizada en donde se pueden apreciar en todo momento las métricas de calidad del producto de software.

Marco Teórico

Las pruebas de software hacen una parte importante a la hora de querer entregar un producto de calidad y de asegurar que se cumplen los requisitos y criterios de aceptación que el cliente planteó (ISTQB, 2012). Estas pruebas se clasifican según la forma en que se ejecutan (ISTQB) que pueden ser pruebas manuales o pruebas automáticas. El propósito de ambas es disminuir la probabilidad de presencia de fallos en un ambiente productivo, es por esto que las empresas que quieren garantizar que el producto desarrollado cumple con lo esperado por los usuarios, deben invertir en este proceso de pruebas y aunque estas puedan representar

entre un 30 y un 50% del costo del desarrollo total (Myers, 2004), si algún defecto causa un daño grave en un ambiente productivo el costo de corregirlo suele ser muy elevado, incluso a veces el costo total de la corrección del fallo no puede ser fácilmente medido (ISTQB). Debido a esto las compañías deben tener una vista constante sobre las métricas de calidad del producto de software que se esté desarrollando, usando este insumo como medida de confiabilidad de la organización.

Generalmente el aseguramiento de la calidad se dejaba para la última etapa del ciclo de vida de desarrollo de software, lo cual significaba un gran problema porque generalmente el cliente no quedaba satisfecho con el resultado final, es por esto por lo que la gran mayoría de las empresas de software se están migrando hacia la implementación de metodologías ágiles que apoyan el principio de prueba que dice que estas se deben empezar lo más temprano posible (ISTQB). Teniendo en cuenta que trabajamos entornos que están en un cambio continuo, es precisamente la automatización de procesos lo que puede llegar a disminuir considerablemente los costos asociados a las pruebas (ISTQB), donde la información recolectada por los robots de automatización es usada para tomar decisiones sobre la calidad del producto desarrollado, volviéndose un material de suma importancia para todos los involucrados en el proceso de desarrollo de software.

Es importante también a la hora de desarrollar un software, el hecho de plantear una buena arquitectura inicial, una arquitectura que permita la división clara de trabajo para todo el equipo de desarrollo (Pierre, 2014). Durante los últimos años está cobrando gran fuerza la arquitectura de microservicios, que, al contrario de la arquitectura tradicional de los sistemas monolíticos, la lógica de negocio no está acoplada en una única unidad, de esta manera se descompone el desarrollo en pequeñas unidades donde se busca generar servicios independientes, que sean fácilmente escalables, cumplan una única función y que puedan comunicarse entre sí por medio de una interfaz sencilla. Esta arquitectura de microservicios se acopla fácilmente a las metodologías ágiles, debido a que son flexibles con respecto a reglas de negocio cambiantes (DSONE, 2018) y permiten que el proceso de pruebas se vaya acoplando más fácilmente a todo el proceso de desarrollo.

La necesidad de entregar software de calidad de una manera rápida y eficiente lleva a que se deba tener un fuerte acoplamiento entre el proceso de pruebas y el proceso de desarrollo, para esto, las empresas de software están empezando a adoptar las prácticas como: Integración, entrega y despliegue continuo (M. Shahin, 2017), prácticas que en su conjunto se conoce como DevOps, en las cuales el pilar fundamental es la automatización de pruebas que se ejecutan cada vez que los desarrolladores hacen un cambio en el repositorio al añadir una nueva funcionalidad o corregir un bug. Lo que se busca es que no exista intervención humana a la hora de realizar un cambio a producción teniendo en cuenta de que, si ocurre algún error, el proceso de despliegue a producción no se llevará a cabo.

DevOps es sobre todo un cambio de mentalidad, en donde se busca la colaboración de áreas que tradicionalmente funcionaban por separado, estas son el área de desarrollo y el área de operaciones (Gavilán), permitiendo así una mayor homogeneidad entre los ambientes de desarrollo y los ambientes productivos, permitiendo un desarrollo iterativo en el cual se busca desarrollar el software de manera más ágil y con una mayor calidad que busca siempre dar un incremento de valor para el cliente. Para facilitar estos procesos, es muy importante la creación de diferentes ambientes donde se puedan ir desplegando los cambios de manera controlada, generalmente se manejan tres tipos de ambientes: Ambiente de desarrollo, ambiente de pruebas o calidad y el ambiente productivo que es en donde el cliente final interactúa con la aplicación.

Es necesario entonces el uso de herramientas que permitan la creación de entornos de desarrollo y calidad repetibles y escalables que se asemejen lo mejor posible al ambiente productivo, previniendo así de manera temprana que posibles errores lleguen hasta el usuario final (ISTQB). Es aquí donde se empieza a sacar el mayor provecho de herramientas como Docker, Gradle, Jenkins, Git, Kubernetes. Siendo estas las herramientas más utilizadas hasta el momento para ser acopladas a todo el ciclo de vida d desarrollo y despliegue de las aplicaciones de software (Monus, 2019).

Para que todos los ambientes sean lo más parecido posibles entre sí, se tiene el concepto de infraestructura como código, IaC por sus siglas en inglés, en lo cual se busca en la mayor medida eliminar configuraciones manuales en los servidores (Hummer W) buscando entonces la utilización de scripts para generar la infraestructura donde correrá la aplicación, Scripts que pueden hacer parte del código de la aplicación garantizando así que cada desarrollador, en su ambiente de desarrollo, tenga una infraestructura como la que la aplicación tendrá en su entorno productivo.

Metodología.

Se trabajó bajo el marco de trabajo Scrum, en donde se definieron Sprints de una duración de 4 semanas, en el que cada sprint estuvo constituido por las siguientes ceremonias:

- Sprint planning: Estas reuniones se hacían al comienzo de cada sprint, donde se definían las historias de usuario que se iban a desarrollar.
- Daily Scrum: Reunión que se hacía diariamente, para mostrar los avances que se hubieran tenido el día anterior y que cada integrante del equipo tuviera claro en que se estaba trabajando, así como también dar pronto aviso en caso de que hubiera habido algún inconveniente que pudieran haber afectado el desarrollo normal del sprint.
- Sprint review: Este se realizaba para inspeccionar el incremento realizado durante el sprint.
- Retrospectiva: Servía como un vínculo entre el sprint que había terminado y el nuevo que estaba por comenzar, para hablar de que se había hecho bien y que se podía mejorar.

- Refinamiento: Se realizaba en caso de que algunas de las historias de usuario hubiesen quedado inconclusas.

Resultados y análisis.
Las tecnologías seleccionadas para el desarrollo de la aplicación fueron:

- Node JS (Para el back-end)
- Angular JS (Para el Front-end)
- Neo4j (Para la base de datos)
- Docker (Para la definición de la infraestructura como código)
- Kubernetes (Para los ambientes de calidad y producción)

Una de la razón con más peso para la selección de las tecnologías fue el conocimiento inicial que ya tenía el equipo de trabajo, esto debido a que se requería un rápido prototipado para así lanzar rápidamente un producto mínimo viable. No fueron las únicas tecnologías que se analizaron, también se tuvo en cuenta trabajar el back-end en spring boot, pero esto requería un tiempo extra en realizar un acercamiento hacia esta tecnología, por lo cual se aceptó desde el principio trabajar el back end en node js, precisamente porque permitió tener un rápido prototipado y al ser su naturaleza asíncrona permite trabajar simultáneamente con múltiples usuarios, los cuales se ven representados por los robots de recolección de la información realizando múltiples peticiones al servidor en un corto periodo de tiempo.

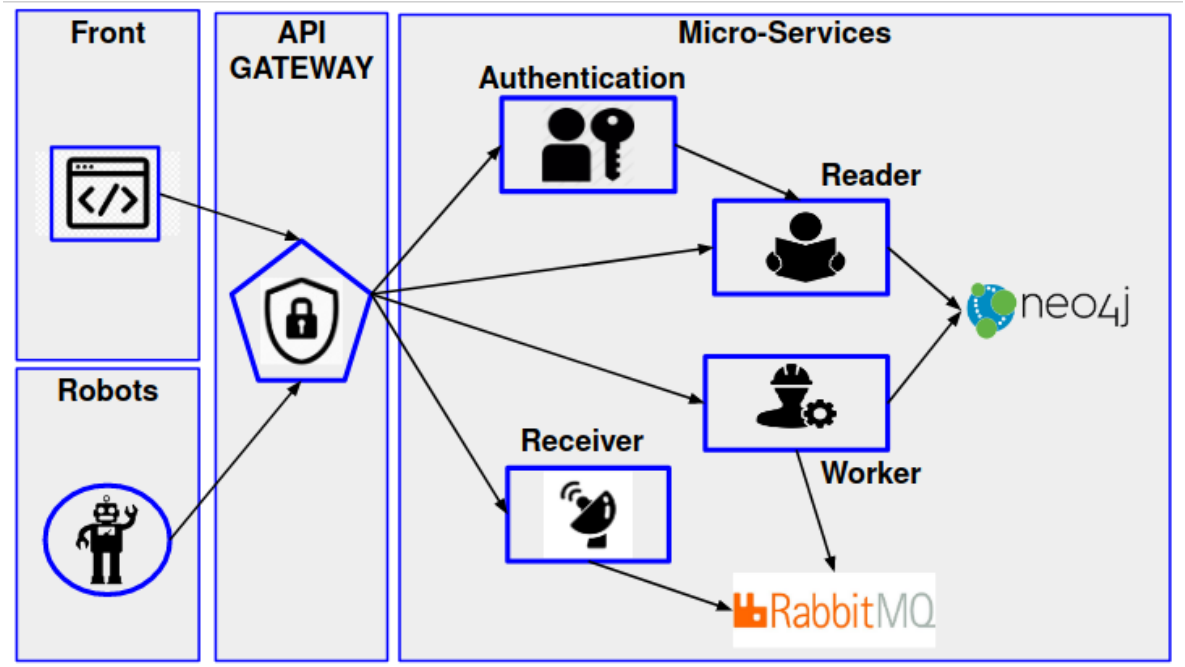

Figura 1. Arquitectura basada en microservicios

La arquitectura planteada, fue una arquitectura basada en microservicios, lo que permitió mucha más modularidad y facilitó la distribución de las tareas a todo el equipo de trabajo, lo que a su vez permitió el rápido despliegue de una versión de pruebas, estos microservicios se trabajaron cada uno como contenedores, para esto se usó Docker para tener la imagen de cada microservicio y Docker compose como orquestador de estos en el ambiente de desarrollo, aquí es donde se empezó a ver la implementación de la infraestructura como código, permitiendo luego, después de tener los microservicios configurados en este script, pasarlos a clusters de kubernetes con la ayuda de una herramienta llamada Kompose. Con esto entonces se tenía el ambiente de calidad, hacia el cual los robots ya podrían mandar información de la ejecución de pruebas automatizadas a través de la red, empezando así a asegurar la calidad del producto de manera temprana. Todo esto permitió fácilmente replicar este ambiente de calidad en producción para así asegurar un despliegue continuo y sin errores ya para el usuario final.

El diseño del módulo que se encarga de recibir los datos enviados por los robots, se diseña como uno de los microservicios llamado Receiver, a este microservicio es al que los robots de automatización de pruebas mandan la información recolectada por las API's en los robots, y este lo que hace es validar que el proyecto hacia el cual se está realizando las pruebas exista y va guardando en formato de texto toda la información que llega en una base de datos en memoria, todo esto para poder manejar el alto flujo de peticiones que pueden llegar en un corto periodo de tiempo. La base de datos en memoria escogida fue RabbitMQ, debido a su sencillez y fácil implementación, así como su buen acoplamiento hacia node JS. Esta es una base de datos en memoria que trabaja por colas, es decir los datos se van almacenando en orden de llegada y ya otro microservicio llega a procesarlos en este orden, el cual sería el consumer o como nosotros lo llamamos el Worker.

Para el módulo de autenticación y registro de usuarios, se expone una interfaz sencilla mediante la cual los usuarios pueden realizar el login, el registro y creación de tokens para el robot de automatización. Hacia este microservicio se comunican el resto de los módulos para validar la identidad de usuario y permitir que realice ciertas acciones. Para esto, se utilizó el método de autenticación por tokens firmadas JWT, es a través de este token que todas las peticiones de los robots de automatización o de la aplicación front-end son validadas, así como también se valida el rol que tiene la persona que está haciendo la petición dentro de la aplicación para temas de los permisos. Los roles que se definieron fueron: Super admin, admin, líder de qa, gerente de proyectos, automatizador y miembro de equipo. De acuerdo con el rol se pueden realizar diferentes acciones dentro de la plataforma web.

El módulo del api Gateway es la pieza central de todos los microservicios, este expone una única url hacia la cual se comunican los clientes, tanto el front-end de la plataforma web como los robots de automatización. Este módulo se puede

entender como la puerta de entrada hacia los demás microservicios, el cual se encarga de redirigir la petición hacia el microservicio que se requiera y de devolver la información desde el microservicio hacia el cliente. Para lograr esto se eligió utilizar la librería de Express Gateway, se escogió esta y no un servicio de Gateway en la nube, porque nos permitía más configuraciones a nivel de código, lo que va con la filosofía que estamos utilizando de tener la infraestructura como código, es una librería que se acopla al Docker compose y utiliza express que es la librería que también se utiliza dentro de la gran mayoría de los microservicios para exponer los endpoints.

Para el front-end el framework escogido fue Angularjs, esto por varios factores: primero porque era el framework con el que el equipo de trabajo ya estaba familiarizado, es decir que no había que esperar a la curva de aprendizaje que podría haber representado elegir otro framework, aparte es un framework muy popular, desarrollado por una empresa que le da gran fiabilidad la cual es Google y es un framework muy utilizado para aplicaciones robustas. Básicamente se planteó un solo módulo, en el cual se fueron dividiendo en servicios que se comunicaban con el back-end por medio de REST, y componentes modulares que cada uno representa un elemento gráfico y funcional de la aplicación web. Se utilizaron también plantillas, más en específico de la plantilla de Admin LTE, la cual es una plantilla gratuita que proporcionaba ya modulo reusables hechos con Bootstrap los cuales se pueden acoplar fácilmente en los componentes creados con Angularjs. Esto permitió empezar a hacer validaciones tempranas de las funcionalidades con los interesados. Para este módulo se pensó también en la internacionalización, para que la plataforma estuviera disponible en varios lenguajes, es por esto por lo que se optó por crear un propio servicio de traducción para así tener más control sobre las traducciones que el proyecto requería que fueran mucho más customizadas a que sencillamente nos tradujera alguna librería de un tercero.

Conclusiones.

- Se ve la importancia de la utilización de un marco de trabajo basado en el agilismo, en nuestro caso SCRUM, ya que esto permite desde las etapas tempranas de la concepción de una idea para un producto de software, que este sea validado por el cliente para así asegurar que se tiene una idea que generará valor.
- El plantear una arquitectura basada en microservicios, hizo posible que la distribución de la carga de trabajo entre los encargados de desarrollar la aplicación fuera más fácil, permitiendo la entrega continua de productos mínimos viables con los que se podía verificar y validar con los interesados la plataforma que se estaba construyendo de manera temprana, permitiendo así ofrecer un producto que sí generó valor a los clientes.

- Es importante entender que tener una arquitectura planteada en microservicios, no significa que se tienen aplicaciones totalmente separadas, sino que hay que tener en cuenta en el diseño de estos, unas interfaces sencillas que permitan la fácil comunicación entre la nube de microservicios, dándole orden al proyecto.
- El tener la oportunidad de aplicar las técnicas más modernas para el desarrollo de software, como lo es el ciclo de DevOps, fue una experiencia grata ya que aquí es donde se nota la importancia de la buena comunicación y el expresar de manera oportuna y adecuada tanto los problemas que se tienen, como las soluciones que se les puedan dar a estos.

Referencias Bibliográficas

CEU IAM. (2018). ¿Por qué todos hablan de metodologías ágiles?. Recuperado de: https://www.ceuiam.com/business-school/por-que-todos-hablan-de-metodologias-agiles--post

De Souza, G (2007). Modelo de maturidade em testes com foco em ambientes de testes heterogêneos. Dissertação (mestrado), Universidade Federal de Pernambuco. Pernambuco, Brasil.

DSONE, (2018), Tools and Techniques for Building Microservices. Recuperado de: https://dzone.com/articles/tools-and-techniques-to-build-microservices

Gavilán Fernández Rubén, DevOps: 6 fases clave para el proceso del desarrollo de software, Recuperado de: https://www.izertis.com/-/blog/devops-6-fases-clave-para-el-proceso-del-desarrollo-de-software

Hummer W., Rosenberg F., Oliveira F., Eilam T. (2013) Testing Idempotence for Infrastructure as Code. In: Eyers D., Schwan K. (eds) Middleware 2013. Middleware 2013. Lecture Notes in Computer Science, vol 8275. Springer, Berlin, Heidelberg

ISTQB (2012). Standard Glossary of terms used in software testing. International Software Testing Qualifications Board. ISTQB: Munich.

M. Shahin, M. Ali Babar, and L. Zhu, (2017), Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. Recuperado de: https://arxiv.org/ftp/arxiv/papers/1703/1703.07019.pdf

Monus Anna, (2019), The 10 best DevOps tools for 2019, Recuperado de: https://raygun.com/blog/best-devops-tools/

Myers, G.J. (2004). The art of software testing. Wiley, second edition

Pierre Bourque, E. Fairley Richard, (2014), Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0

Serna M., Edgar, Martínez M., Raquel, & Tamayo O., Paula Andrea. (2017). Un modelo para determinar la madurez de la automatización de las pruebas del software como área de investigación y desarrollo. Computación y Sistemas, 21(2), 337-352. https://dx.doi.org/10.13053/cys-21-2-2723