Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017 Gestión de las TICs para la Investigación y la Colaboración, San José, del XX al XX de julio de 2017

Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web

Daniel López[a], Edgar Maya[b]

[a] Asamblea Nacional del Ecuador, Coordinación General de Tecnologías de la Información y Comunicación, Piedrahita 5-21 y Av. Gran Colombia, Pichincha, Ecuador
daniel.lopez@asambleanacional.gob.ec

[b] Universidad Técnica del Norte, Instituto de Posgrados, Av. 17 de Julio 5-21 Gral. José María Cordova, Imbabura, Ecuador
eamaya@utn.edu.ec

Resumen. Actualmente, el proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software tradicional o monolítica que ha sido adoptada del lenguaje de programación utilizado, la plataforma o de la experiencia del personal del área de desarrollo; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento, escalabilidad y entregas. El objetivo del presente estudio fue identificar las tecnologías, metodología y arquitectura que utiliza la CGTIC para el desarrollo de aplicaciones web y la correspondiente identificación de las tecnologías existentes para el desarrollo e implementación de microservicios, utilizando como base de la investigación un enfoque cualitativo, con un tipo de investigación descriptiva y diseño documental. Se empleó la técnica de grupo focal aplicado a los funcionarios del área de desarrollo de software de la CGTIC, revisión bibliográfica de arquitectura de microservicios. Como avance de la investigación, el análisis ha permitido identificar el estado del arte respecto a microservicios y su implementación así como la identificación de los requisitos y necesidades relativos al desarrollo de aplicaciones web y como satisfacerlas mediante el diseño de una arquitectura de software.

Palabras Clave: Microservicios, arquitectura de software, aplicaciones web.

Eje temático: Infraestructura y desarrollo de software.

1 Introducción

En la actualidad, a nivel empresarial y tanto en el sector privado como público se realiza desarrollo de software para suplir las necesidades de automatización de procesos internos, este desarrollo ha seguido las tendencias impuestas por la plataforma, lenguaje de programación o por la experiencia del área de desarrollo, lo cual deviene en la implantación de sistemas de construcción tradicional o monolítico.

El proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software monolítica,

misma que ha sido adoptada por el uso de un lenguaje de programación específico para construcción de aplicaciones web empresariales; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento, escalabilidad y entregas [1]

1.1 Problema

El presente trabajo de investigación, parte de un análisis de la problemática existente con el desarrollo de aplicaciones web de la CGTIC, dicho desarrollo hace uso de una arquitectura monolítica la cual incide en diferentes aspectos tanto tecnológicos y administrativos. Las incidencias más evidentes se pueden observar al aplicar procesos de mantenimiento en sistemas complejos, sea por una petición de cambio, nueva funcionalidad o corrección de un fallo, en los cuales, la resolución de un problema o cambio simple implica el redespliegue de toda la aplicación debido a que se tiene todas las funcionalidades en un único paquete incrementando los riesgos de fallos. De igual forma, por el tiempo que toma la implementación de un cambio o nueva funcionalidad, se presenta resistencia al cambio en el usuario final, y en consecuencia las actualizaciones son menos frecuentes porque requieren de un mayor esfuerzo y coordinación de los grupos de desarrollo y la realización de pruebas más extensas.

En aspectos de calidad, surgen complicaciones en la escalabilidad ya que puede requerirse escalar un módulo específico, pero, por el aspecto monolítico es necesario escalar la aplicación en su totalidad. De igual manera sucede con la resilencia, al ocurrir un fallo por caída o sobrecarga en una parte de la aplicación, se pierden todas sus funcionalidades; si bien este último aspecto puede ser subsanado mediante replicación o clusters, también incrementa las dificultades de coordinación, configuración y eleva los costos de equipamiento y esfuerzo.

Otra problemática a resaltar, es la incidencia que produce el mantenimiento aplicaciones en el normal desenvolvimiento de las actividades de los usuarios; en este sentido, una de las áreas más críticas es el Plenario de la ANE, el cual hace uso de software para registrar las votaciones de los legisladores en la creación o reforma de leyes de afectación nacional, por lo que resulta difícil cambiar, agregar o actualizarlo. Adicionalmente, algunos de los sistemas se encuentran impedidos de actualización que es de vital importancia por su nivel de criticidad y afectación, esto se debe una vez más por estar construida bajo el esquema monolítico.

Por los aspectos descritos se concluye que existe la necesidad de definir una nueva arquitectura de software con un enfoque de vanguardia que facilite el desarrollo de nuevas aplicaciones para las diferentes unidades organizacionales de la ANE. Bajo esta nueva perspectiva, se pretende obtener arquitectura de software flexible que permita el mantenimiento de las aplicaciones evitando la interrupción de actividades del personal que las usa.

2 Antecedentes

En la Universidad de Aarhus de Dinamarca [2], evalúa diferentes tácticas de disponibilidad y mantenibilidad en la construcción de un prototipo con arquitectura de microservicios, en el proceso, concluye con una revisión de las características de los microservicios enunciadas por Fowler y Lewis [3]en cuatro criterios principales que definen a un microservicio:

Enfoque en las capacidades de negocio. El proceso de desarrollo de software se vuelve más flexible bajo el enfoque de microservicios, cada microservicio está compuesto por su propia lógica de negocios, interfaz de usuario, base de datos y funcionalidad de comunicación con otros servicios, por lo tanto, en el equipo de desarrollo que cada miembro necesita experiencia en desarrollo de backend, frontend, y bases de datos, permitiéndoles agregar nuevas características rápidamente sin arriesgar la estabilidad y el funcionamiento de todo el sistema.

Independencia de los servicios autónomos. Cada servicio contiene su propia lógica de negocio y se despliega por separado. Esto hace posible cambiar y extender gradualmente con nuevas características sin afectar el resto del sistema. La propiedad autónoma permite la flexibilidad en la selección de la tecnología más adecuada para un servicio específico.

Gestión descentralizada de datos. Todos los servicios tienen su propia base de datos en esquemas pequeños y simples, que se ven por separado. Los datos están desacoplados, lo que requiere mucha gestión y pruebas para garantizar que un sistema nunca actualice o elimine los datos en un servicio sin actualizar o eliminar los datos correspondientes en otros servicios que contienen los mismos datos o conjunto de referencias.

Tolerancia a fallos. Los sistemas de microservicio consisten en múltiples unidades pequeñas que pueden fallar; Estas unidades están ligeramente acopladas y pueden ser restauradas automáticamente. lo que resulta en un sistema más estable y robusto.

Por su parte Borčin [4] en su trabajo de investigación, realiza un análisis de los atributos de calidad de la arquitectura de microservicios profundizando en los retos de su implementación:

Interfaces y comunicación. Cada microservicio debe exponer alguna interfase, para algunas tecnologías no equipadas con una especificación, esto puede representar serios problemas incluso REST que es utilizado en la arquitectura de microservicio para la comunicación no tiene una interfaz bien definida.

Transacciones y bloqueos. Se necesita un conjunto completamente nuevo de operaciones para una transacción que comprueba la comunicación de microservicios. se debe asegurar de que ningún microservicio utilice flujos cerrados o recursos bloqueados y que no modifique los datos que otros servicios están utilizando. Estos bloqueos podrían provocar la suspensión de todo el sistema.

Programación coreográfica. El paradigma de programación coreográfica puede ayudar en la creación de sistemas libres de bloqueos, sin errores de comunicación. La programación coreográfica puede llegar a ser más importante con el desarrollo ulterior de la arquitectura del microservicio, ya que proporcionan un terreno sólido para una formalización de la comunicación.

Puntos de entrada. La seguridad es un problema importante cuando se trata de la arquitectura de microservicios y los sistemas distribuidos en general. En aplicaciones monolíticas se puede asegurar fácilmente la seguridad de toda la aplicación, ya que toda la comunicación pasa a través de los puntos de entrada que posee cada módulo que lo compone. En contraste, en microservicios, garantizar la seguridad se vuelve mucho más difícil. Toda la aplicación se compone de un microservicio independiente (lenguaje y máquina) que expone sus interfaces y el núcleo de la aplicación para la comunicación exterior, proporcionando nuevos puntos de entrada para los intrusos.

Red compleja. Una extensa red de comunicación puede ser fácilmente expuesta a ataques, ya que es difícil administrar cuando la aplicación consta de muchos microservicios y esos servicios envían miles de mensajes. El monitoreo de esta compleja red es también muy difícil.

Concluye que, la arquitectura de microservicios traerá nuevos problemas y desafíos relacionados no sólo con la comunicación y su estructura distribuida. Actualmente sólo se puede decir que los microservicios todavía están en una etapa inicial y aún es desconocido para muchos desarrolladores, pero están ganando mucha atención y un sitio en el mercado, especialmente entre una medianas y pequeñas empresas.

2.1 Microservicios

Es un enfoque para el desarrollo de una aplicación única como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y mecanismos ligeros de comunicación, a menudo un recurso de una interfaz de programación de aplicaciones (API) sobre protocolo de transferencia de hipertexto (HTTP). Estos servicios están construidos alrededor de las capacidades del negocio y con independencia de despliegue e implementación totalmente automatizada. Existe un mínimo de gestión centralizada de estos servicios, los que pueden estar escritos en lenguajes de programación diferentes y utilizar diferentes tecnologías de almacenamiento de datos [3].

El término microservicios no es relativamente nuevo, este estilo arquitectural fue acuñado por Martin Fowler en un taller de arquitectos de software como una descripción del nuevo campo que los participantes estaban explorando. No existe una definición en concreto para microservicio, sin embargo, una aproximación que la realiza [5] lo define como: “Pequeños servicios autónomos que trabajan juntos”.

Una arquitectura de microservicios promueve el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares y auto-contenidas, lo cual difiere de la forma tradicional o monolítico [6].

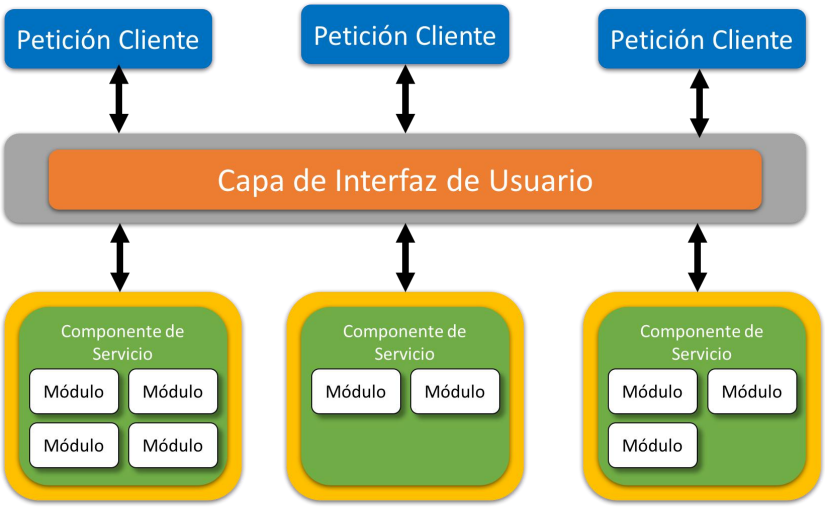

Fig. 1. Patrón básico de arquitectura de Microservicios [7]

Una de las ventajas de utilizar microservicios es la capacidad de publicar una aplicación grande como un conjunto de pequeñas aplicaciones (microservicios) que se pueden desarrollar, desplegar, escalar, manejar y visualizar de forma independiente. Los microservicios permiten a las empresas gestionar las aplicaciones de código base grande usando una metodología más práctica donde las mejoras incrementales son ejecutadas por pequeños equipos en bases de código y despliegues independientes. La agilidad, reducción de costes y la escalabilidad granular, traen algunos retos de los sistemas distribuidos y las prácticas de gestión de los equipos de desarrollo que deben ser considerados. [8]

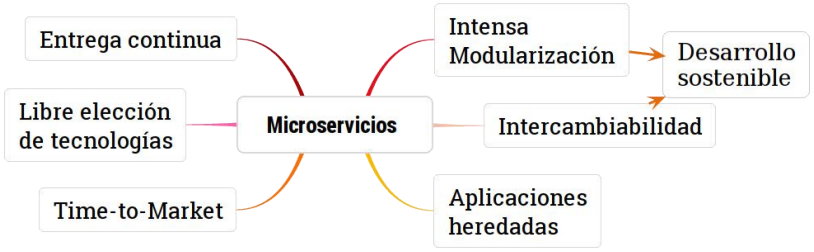

Fig. 1. Beneficios de los microservicios

Topologías. Según Richards [7], existen muchas maneras de implementar un patrón de arquitectura de microservicios, pero se destacan tres topologías principales que son las más comunes y populares: basada en API REST (Transferencia de Estado Representacional del inglés Representational State Transfer), basada en aplicaciones REST y la centralizada de mensajería.

- Topología basada en API REST es útil para sitios web que exponen servicios individuales pequeños y autónomos a través de algún tipo de API. Consta de componentes de servicio de grano fino (de ahí el nombre microservicios) que contienen uno o dos módulos que realizan funciones empresariales específicas independientes del resto de los servicios. En esta topología, estos componentes de servicio de grano fino se acceden normalmente mediante una interfaz basada en REST implementada a través de una capa de API basada en la Web desplegada separadamente. Algunos ejemplos de esta topología incluyen algunos de los servicios web REST basados en la nube como los usados por Yahoo, Google y Amazon.
- Topología basada en aplicaciones REST difiere del enfoque basado en API REST en que las solicitudes de cliente se reciben a través de pantallas de aplicaciones empresariales tradicionales basadas en web o en clientes pesados (fat-client) en lugar de una simple capa de API. La capa de interfaz de usuario de la aplicación se despliega como una aplicación web separada que accede de forma remota a componentes de servicio desplegados por separado (funcionalidad empresarial) a través de simples interfaces basadas en REST. Otra diferencia se encuentra en que los componentes de servicio tienden a ser más grandes, de grano más grueso y representan una pequeña porción de la aplicación empresarial general en lugar de granos finos, servicios de acción simple. Esta topología es común para las aplicaciones empresariales pequeñas y medianas que tienen un grado relativamente bajo de complejidad.
- La topología de mensajería centralizada, es similar a la basada en REST, excepto que en lugar de usar REST para acceso remoto, esta utiliza un intermediario de mensajes centralizado ligero (por ejemplo, ActiveMQ, HornetQ, etc.). Es importante considerar que no se debe confundir con el patrón SOA o considerarlo "SOA-Lite". El agente de mensajes ligeros que se encuentra en esta topología no realiza ninguna orquestación, transformación o enrutamiento complejo, es sólo un transporte ligero para acceder a los componentes de servicio remotos. Esta topología se encuentra frecuentemente en aplicaciones de negocios más grandes o aplicaciones que requieren un control más sofisticado sobre la capa de transporte entre la interfaz de usuario y los componentes de servicio. Entre los beneficios que aporta frente a las anteriores es que son mecanismos avanzados de colas, mensajería asíncrona, monitoreo, manejo de errores y mejor balanceo de carga y escalabilidad. El único punto de fallo y los problemas de cuello de botella arquitectónico generalmente asociados con un intermediario centralizado se abordan a través de la agrupación de intermediarios y de la federación de intermediarios (dividir una única instancia de intermediario en múltiples instancias de intermediario para dividir la carga de procesamiento de mensajes en función de las áreas funcionales del sistema).

Aplicación monolítica. En una aplicación monolítica toda la lógica se ejecuta en un único servidor de aplicaciones. Las aplicaciones monolíticas típicas son grandes y construidas por múltiples equipos, requiriendo una orquestación cuidadosa del despliegue para cada cambio. También se consideran aplicaciones monolíticas cuando

existen múltiples servicios de API que proporcionan la lógica de negocio, toda la capa de presentación es una sola aplicación web grande. En ambos casos, la arquitectura de microservicios puede proporcionar una alternativa.

Tabla 1. Arquitecturas monolíticas vs. microservicios [9]

Categoría	**Arquitectura Monolítica**	**Arquitectura de microservicios**
Código	Una base de código única para toda la aplicación.	Múltiples bases de código. Cada microservicio tiene su propia base de código.
Comprensibilidad	A menudo confuso y difícil de mantener.	Mayor facilidad de lectura y mucho más fácil de mantener.
Despliegue	Implementaciones complejas con ventanas de mantenimiento y paradas programadas.	Despliegue sencillo ya que cada microservicio se puede implementar de forma individual, con un tiempo de inactividad mínimo, si no es cero.
Lenguaje	Típicamente totalmente desarrollado en un lenguaje de programación.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.
Escalamiento	Requiere escalar la aplicación entera, aunque los cuellos de botella estén localizados.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.

2.2 Servicios web.

Los servicios Web son componentes de software ligeramente acoplados entregados a través de tecnologías estándar de Internet. Es decir, los servicios Web son aplicaciones de negocio autodescriptivas y modulares que exponen la lógica empresarial como servicios a través de Internet a través de la interfaz programable y donde el protocolo de Internet (IP) puede ser utilizado para encontrar e invocar esos servicios. Un servicio web es el elemento clave en la integración de diferentes sistemas de información, ya que los sistemas de información pueden basarse en diferentes plataformas, lenguajes de programación y tecnologías [10].

SOAP. La implementación de servicios web por protocolo simple de acceso a objetos (SOAP) se desarrolló como una alternativa al estándar CORBA (Common Object Request Broker Architecture). Para garantizar el transporte de datos en SOAP, se utilizan protocolos como HTTP, SMTP, entre otros, en formato XML. En este enfoque, un proveedor de servicios publica una descripción del servicio o una interfaz

para el registro de servicios, por lo que el solicitante del servicio puede encontrar una instancia de servicio correcta y utilizarla. Algunos problemas de rendimiento en SOAP se producen al formar el mensaje SOAP ya que agrega un encabezado adicional y partes al cuerpo al mensaje. Los servicios Web basados en SOAP incluyen una variedad de estándares, tales como WSDL, WSBPEL, WS-Security, WS-Addressing Estas normas fueron desarrolladas por organizaciones de normalización, como W3C y OASIS [11]

REST. Un servicio REST se define como una agregación de diferentes recursos que pueden ser alcanzados desde un identificador universal de recurso (URI) base. Un recurso representa a una entidad del mundo real cuyo estado está expuesto y puede cambiarse accediendo a un URI. Una Representación es la descripción de los mensajes enviados o recibidos de un Recurso en términos de un lenguaje tecnológico. Actualmente XML y JSON son los idiomas más populares para describir estos mensajes [12]

2.3 Integración continua

La integración continua (CI) es una práctica en el desarrollo de software en la que los desarrolladores hacen integraciones automáticas de un proyecto lo más a menudo posible (generalmente a diario) con el propósito de detectar fallos lo antes posible. Esta práctica de integración comprende la compilación y ejecución de pruebas de todo un proyecto.

Prácticas de integración continua. Para la aplicación de prácticas de CI se requiere que cada vez que se agregue una nueva parte al sistema, también se creen casos de prueba automáticos para cubrir todo el sistema incluyendo las partes recién agregadas. De igual forma, se requiere que el software sea probado y construido automáticamente y una retroalimentación inmediata sobre los nuevos códigos integrados hacia los desarrolladores. De acuerdo con Hamdan y Alramouni [13], estas prácticas son:

- Integrar el código con frecuencia. Este núcleo de la integración continua. No se debe esperar más de un día para enviar código al repositorio de código compartido.
- No integrar código “roto”. Código roto es un código que contiene cualquier tipo de error cuando se incluye en una construcción de CI.
- Corregir las compilaciones no funcionales de inmediato. Una compilación no funcional puede ser un error en la compilación, en la base de datos o en la implementación. Es cualquier cosa que impide que la compilación de informes de éxito.
- Escribir pruebas automatizadas. Las pruebas deben ser automatizadas para que funcionen en un sistema de CI. También debe cubrir todo el código fuente.
- Todas las pruebas e inspecciones deben aprobarse. Para que una compilación se apruebe, el 100% de las pruebas automatizadas deben pasar con éxito. Este es el criterio más importante de CI en cuanto a la calidad del software.

- Ejecutar compilaciones privadas. Las herramientas de CI permiten a los desarrolladores tener una copia del software del repositorio de código compartido localmente en sus estaciones de trabajo. Esto les proporciona la capacidad de tener una compilación de integración reciente localmente antes de integrarla al servidor de generación de integración principal para asegurar que no falle.

2.4 Contenedor de aplicaciones

Un contenedor de aplicaciones se basa en la virtualización de sistemas operativos usando los contenedores Linux [14], precisamente es un motor de contenedor de código abierto, que automatiza el empacado, entrega y despliegue de cualquier aplicación de software, se presenta como contenedores livianos, portátiles y autosuficientes, que se ejecutarán prácticamente en cualquier lugar. Un contenedor es un cubo de software que comprende todo lo necesario para ejecutar el software de forma independiente. Puede haber varios contenedores en una sola máquina y los contenedores están completamente aislados entre sí y también de la máquina anfitriona [15]

La palabra contenedor se define como "un objeto para contener/resguardar o el transporte de algo". La idea detrás de los contenedores de "software" es similar. Son imágenes aisladas e inmutables que proporcionan funcionalidad diseñada en la mayoría de los casos para ser accesibles sólo a través de sus APIs. Es una solución para hacer que el software funcione de forma fiable y en (casi) cualquier entorno. No importa dónde se estén ejecutando (computador portátil, servidor de pruebas o de producción, centro de datos, etc.), el resultado siempre debe ser el mismo [16].

3 Metodología

Como base de la investigación se usó un enfoque cualitativo que permitió descubrir o afinar las preguntas de investigación sobre las necesidades del diseño de una arquitectura de software. Se aplica un tipo de investigación descriptiva [17] para la contextualización y descripción del entorno de aplicación de una arquitectura en el desarrollo de aplicaciones web. El diseño documental [18] utilizado para la revisión bibliográfica sobre arquitectura de microservicios. Se empleó la técnica de grupo focal [19] aplicado a los funcionarios del área de desarrollo de software de la CGTIC para determinar el estado de situación actual sobre proceso de software aplicado.

3.1 Análisis Cualitativo

Recolección de datos. El instrumento de reunión de expertos fue desarrollado tomando en cuenta los antecedentes del entorno, tales como: Tecnología en uso, documentación disponible, experiencia del personal.

La reunión de expertos tuvo dos enfoques principales:

- Desarrollo de aplicaciones
- Arquitectura de software

Por cada pregunta se definió posibles respuestas esperadas como guía del conductor de acuerdo su ámbito relacionado.

Resultados. Con respecto al desarrollo de software y con base en la información recolectada de la aplicación del instrumento de investigación se concluye que:

El proceso de desarrollo de software utiliza una metodología ágil debido al carácter cambiante de los sistemas en ejecución, estas a su vez, constituyen en su mayoría los orientados a la web, con una clara tendencia al desarrollo en entornos móviles. Existe un uso mayoritario de aplicaciones a nivel interno con alta criticidad mismas que no cuentan con servicios alternativos en presencia de fallos o eventos fortuitos que permitan la continuidad del servicio.

El producto de software que se genera carece de principios de modularidad debido al uso de la arquitectura implícita de la plataforma de desarrollo utilizada. Como resultado las aplicaciones adquieren un carácter monolítico embebiendo en una sola unidad ejecutable todas sus funcionalidades lo que incide tanto en su desarrollo como mantenimiento.

En consecuencia y bajo el criterio unificado obtenido, se concluye que es de alta importancia emplear una arquitectura de software para el desarrollo de software que permita una estandarización de las aplicaciones y brinde una mejor calidad a los productos de software.

3.2 Proceso de arquitectura de software

El proceso de diseño de la arquitectura es una extensión del proceso general de diseño de ingeniería. El diseño de la arquitectura se centra en la descomposición de un sistema en componentes y las interacciones entre los componentes para satisfacer los requisitos funcionales y no funcionales. Un sistema de software puede ser visto como una jerarquía de las decisiones de diseño (reglas también llamado diseño o contratos). Cada nivel de la jerarquía tiene un conjunto de reglas de diseño que de alguna manera se une o conecta los componentes en ese nivel. La salida principal del proceso de diseño de la arquitectura es una descripción arquitectónica. El proceso describe los siguientes pasos generales [20]:

- El establecimiento de requisitos, mediante el análisis de los controladores del negocio, el contexto del sistema, y los factores que los interesados del sistema estimen crítico para el éxito.
- La definición de una arquitectura, mediante el desarrollo de estructuras arquitectónicas y estrategias de coordinación que satisfagan los requisitos.
- La evaluación de la arquitectura, mediante la determinación de cuándo y qué métodos de evaluación de la arquitectura son apropiados, la realización de las evaluaciones, y la aplicación de los resultados para mejorar el desarrollo de la arquitectura.

- La documentación de la arquitectura, con el suficiente detalle y en una forma fácilmente accesible para los desarrolladores y otras partes interesadas.
- El análisis de la arquitectura, para el rendimiento del sistema, la seguridad.
- Implementación, de la arquitectura definida a través del desarrollo de un prototipo.

4 Resultados

La investigación se encuentra en etapa de desarrollo de la propuesta y posteriormente su aplicación mediante la implementación de un prototipo bajo la arquitectura propuesta que demuestre su uso, por lo que aún no se ha obtenido resultados.

5 Conclusiones

- El actual desarrollo de software bajo la metodología aplicada se contrapone a la arquitectura utilizada, lo cual dificulta su correcta aplicación.
- El mantenimiento o cambio en las funcionalidades de las aplicaciones representa un problema debido a su naturaleza monolítica.
- Existe una clara intención de establecer una nueva arquitectura de desarrollo de aplicaciones acorde a la utilización de nuevas tecnologías.

Agradecimientos

Este trabajo forma parte de la Tesis Arquitectura de Software Basada en Microservicios para Desarrollo de Aplicaciones Web de la Asamblea Nacional.

Referencias

1. Carneiro, C.; Schmelmer, T.: Microservices From Day One. Apress. Berkeley, CA. (2016).
2. Nielsen, D. Investigate availability and maintainability within a microservice architecture (Tesis Doctoral). Master's thesis, Aarhus University. (2015).
3. Fowler, M.; Lewis, J. Microservices. Viittattu, (2014)
4. Borčin T. Service activity monitoring for SilverWare. Masaryk University. (2017)
5. Newman, S. Building Microservices. O'Reilly Media, Inc. (2015).
6. Wolff, E. Microservices: Flexible Software Architectures. CreateSpace Independent Publishing Platform. (2016).
7. Richards, M. Software Architecture Patterns. O'Reilly Media, Inc. (2015).
8. Villamizar, M., Garces, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., y Gil, S. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. Computing Colombian Conference (10CCC), 2015 10th (pp. 583–590). (2015).

9. Daya, S., Van Duy, N., Eati, K., Ferreira, C., Glozic D., Gucer, V., Vennam R. Microservices fron Theory to Practice IBM Corporation.(2015).
10. Wagh, K., Thool, R. A comparative study of soap vs rest web services provisioning techniques for mobile host. Journal of Information Engineering and Applications, 2(5), 12–16. (2012).
11. Tihomirovs, J., Grabis, J. Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics. Information Technology and Management Science, 19(1). (2016)
12. Valverde, F., Pastor, O. Dealing with REST services in model-driven web engineering methods. V Jornadas Científico -Técnicas en Servicios Web y SOA, JSWEB, 243–250. (2009).
13. Hamdan, S., Alramouni, S. A Quality Framework for Software Continuous Integration. Procedia Manufacturing, 3. (2015).
14. Kratzke, N. About microservices, containers and their underestimated impact on network performance. Proceedings of Cloud Computing, 2015, 165–169. (2015).
15. Raj, P., Chelladhurai, J. S., Singh, V. Learning Docker: Optimize the power of Docker to run your applications quickly and easily. Birmingham Mumbai: Packt Publishing. (2015).
16. Farcic, V., The DevOps 2.0 Toolkit. packtpub. (2016).
17. Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P.. Metodología de la investigación. 5ta Edición La Habana: Editorial Félix Varela, 2. (2010)
18. Galeano, M. E. Diseño de proyectos en la investigación cualitativa. Universidad Eafit. (2004).
19. Álvarez-Gayou, J. L. Cómo hacer investigación cualitativa. Fundamentos y metodología. Colección Paidós Educador. México: Paidós Mexicana. (2003).
20. Babar, M., Brown, A., Mistrík, I., Agile Software Architecture: Aligning Agile Processes and Software Architectures. Newnes, (2013).