Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Estudios de Postgrado
Maestría de Tecnologías de la Información y Comunicación

DISEÑO E IMPLEMENTACIÓN DE UNA ARQUITECTURA ESCALABLE BASADA EN MICROSERVICIOS PARA UN SISTEMA DE GESTIÓN DE APRENDIZAJE CON CARACTERÍSTICAS DE RED SOCIAL

Ing. José Manuel De Paz Estrada

Asesorado por la Inga. MSC. María Elizabeth Aldana Díaz

Guatemala, marzo de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UNA ARQUITECTURA ESCALABLE BASADA EN MICROSERVICIOS PARA UN SISTEMA DE GESTIÓN DE APRENDIZAJE CON CARACTERÍSTICAS DE RED SOCIAL

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JOSÉ MANUEL DE PAZ ESTRADA
ASESORADO POR EL INGA. MSC. MARÍA ELIZABETH ALDANA DÍAZ

AL CONFERÍRSELE EL TÍTULO DE

MAESTRO EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

GUATEMALA, MARZO DE 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Marlon Antonio Pérez Turk
EXAMINADORA	Inga. María Elizabeth Aldana Díaz
EXAMINADOR	Ing. Murphy Olympo Paiz Recinos
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO E IMPLEMENTACIÓN DE UNA ARQUITECTURA ESCALABLE BASADA EN MICROSERVICIOS PARA UN SISTEMA DE GESTIÓN DE APRENDIZAJE CON CARACTERÍSTICAS DE RED SOCIAL

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería, Escuela de Estudios de Postgrado, con fecha marzo de 2016.

José Manuel De Paz Estrada

FACULTAD DE
INGENIERÍA - USAC
ESCUELA DE
ESTUDIOS DE POSTGRADO

Escuela de Estudios de Postgrado
Facultad de Ingeniería
Teléfono 2418-9142 / 24188000 Ext. 86226

APT-2017-003

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Postgrado, al Trabajo de Graduación de la Maestría en Artes en Tecnologías de la Información y la Comunicación titulado: ***"DISEÑO E IMPLEMENTACIÓN DE UNA ARQUITECTURA ESCALABLE BASADA EN MICROSERVICIOS PARA UN SISTEMA DE GESTIÓN DE APRENDIZAJE CON CARACTERÍSTICAS DE RED SOCIAL"*** presentado por el Ingeniero en Ciencias y Sistemas **José Manuel De Paz Estrada**, procede a la autorización para la impresión del mismo.

IMPRÍMASE.

"Id y Enseñad a Todos"

Ing. Pedro Antonio Aguilar Polanco
Decano
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, marzo de 2017.

Cc: archivo/la

FACULTAD DE INGENIERÍA - USAC
ESCUELA DE ESTUDIOS DE POSTGRADO

Escuela de Estudios de Postgrado
Facultad de Ingeniería
Teléfono 2418-9142 / 24188000 Ext. 86226

APT-2017-003

El Director de la Escuela de Estudios de Postgrado de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen y dar el visto bueno del revisor y la aprobación del área de Lingüística del Trabajo de Graduación titulado ***"DISEÑO E IMPLEMENTACIÓN DE UNA ARQUITECTURA ESCALABLE BASADA EN MICROSERVICIOS PARA UN SISTEMA DE GESTIÓN DE APRENDIZAJE CON CARACTERÍSTICAS DE RED SOCIAL"*** presentado por el Ingeniero en Ciencias y Sistemas **José Manuel De Paz Estrada,** correspondiente al programa de Maestría en Artes en Tecnologías de la Información y la Comunicación; apruebo y autorizo el mismo.

Atentamente,

"Id y Enseñad a Todos"

UNIVERSIDAD DE SAN CARLOS
ESCUELA DE POSTGRADO
FACULTAD DE INGENIERIA
DE GUATEMALA

MSc. Ing. Murphy Olympo Paiz Recinos
Director
Escuela de Estudios de Postgrado
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Guatemala, marzo de 2017.

Cc: archivo/la

FACULTAD DE
INGENIERÍA - USAC
ESCUELA DE
ESTUDIOS DE POSTGRADO

Escuela de Estudios de Postgrado
Facultad de Ingeniería
Teléfono 2418-9142 / 24188000 Ext. 86226

APT-2017-003

Como Coordinador de la Maestría en Artes en Tecnologías de la Información y la Comunicación del Trabajo de Graduación titulado ***"DISEÑO E IMPLEMENTACIÓN DE UNA ARQUITECTURA ESCALABLE BASADA EN MICROSERVICIOS PARA UN SISTEMA DE GESTIÓN DE APRENDIZAJE CON CARACTERÍSTICAS DE RED SOCIAL"*** presentado por el Ingeniero Ciencias y Sistemas **José Manuel De Paz Estrada,** apruebo y recomiendo la autorización del mismo.

Atentamente,

"Id y Enseñad a Todos"

UNIVERSIDAD DE SAN CARLOS
ESCUELA DE POSTGRADO
FACULTAD DE INGENIERIA
DE GUATEMALA

MSc. Ing. Marlon Antonio Pérez Türk
Coordinador de Maestría
Escuela de Estudios de Postgrado
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Guatemala, marzo de 2017.

Cc: archivo/la

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

RESUMEN

Existen diversos sistemas de gestión de aprendizaje, algunos de uso gratuito y otros que solamente se pueden utilizar realizando un pago por ello; además algunos son software libre, lo que permite que puedan ser modificados.

El problema de estos sistemas de gestión de aprendizaje es que originalmente fueron desarrollados para funcionar sobre una arquitectura monolítica, pues sus componentes están fuertemente acoplados, esto genera el riesgo que si un componente falla o se detiene para realizar un nuevo despliegue, provoca que el resto de componentes del sistema también se detengan, generalizando inactividad en el sistema completo.

Para resolver este problema, la arquitectura propuesta en el presente trabajo se basa en el uso de microservicios, estos microservicios como su nombre lo indica, son servicios pequeños que trabajan juntos y deben estar enfocados en hacer una cosa bien. Estos además deben de tener la capacidad de desplegarse de forma independiente.

Para la construcción del prototipo del sistema de gestión de aprendizaje sobre una arquitectura basada en microservicios, se utiliza un conjunto de patrones arquitectónicos. Estos patrones incluyen el uso de instancias de servicio por contenedor utilizando Docker, el registro y localización de servicios utilizando Consul, la comunicación entre servicios utilizando servicios REST por medio del protocolo HTTP y el patrón arquitectónico de API Gateway utilizando Node.js, como una puerta de enlace a la cual pueden realizar sus peticiones los clientes externos del sistema.

Además, para verificar que la arquitectura propuesta resuelve los problemas que se pueden apreciar al utilizar una arquitectura monolítica, se realizan pruebas de rendimiento para comparar el comportamiento del sistema sobre las dos arquitecturas. Para estas pruebas de rendimiento se realizan pruebas de carga y monitoreo del sistema y al final se puede apreciar que la arquitectura basada en microservicios permite que cuando un servicio se encuentra inactivo, los demás servicios funcionan de forma correcta.

Por otro lado, luego de realizar las pruebas de rendimiento, también se puede apreciar que los requerimientos de hardware y software son mayores al utilizar una arquitectura basada en microservicios, y por lo tanto se debe de realizar un análisis para saber si en verdad es necesario el uso de este paradigma.

PLANTEAMIENTO DEL PROBLEMA Y FORMULACIÓN DE PREGUNTAS ORIENTADORAS

Con la creciente popularidad que han logrado los cursos en línea masivos y abiertos (MOOC por las siglas en inglés de *Masive Online Open Courses*) desde el año 2012, los sistemas de gestión de aprendizaje o por sus siglas en inglés LMS (*Learning Management System*), han sido desplazados a un segundo plano en cuanto se refiere a su uso como medio de aprendizaje en línea utilizado en empresas e instituciones educativas, que han optado por participar en los nuevos sistemas MOOC.

Entre estos sistemas tradicionales se puede encontrar a Moodle y Blackboard Learn, que permiten a los clientes instalar la aplicación en su propio servidor, así como contratar hospedaje en la nube. Estos tienen en común que se basan en una arquitectura monolítica, que, aunque permiten escalar de forma horizontal, es necesario el uso de un nuevo servidor para cada nodo; por su lado Blackboard Learn posee una arquitectura preparada para el uso de balanceo de cargas, mientras que en Moodle y Dockeos es necesario de más trabajo de configuración para lograr los mismos resultados.

Las arquitecturas monolíticas también pueden escalar de forma horizontal, pero realizando réplicas exactas del sistema y balanceando las cargas entre ellas, esto produce que se desperdicien recursos, ya que muchas veces solamente cierta parte del sistema es la que consume muchos recursos y en algún momento dado solamente un módulo requerirá aún más. Por otro lado, a la hora de realizar una actualización en sistemas monolíticos con escalabilidad

horizontal, se debe desplegar y replicar el sistema por completo en el número de nodos que sea necesario, provocando que el proceso sea más complicado.

Otro problema que ocurre debido a una arquitectura monolítica es al momento de ocurrir una falla no recuperable en un módulo del sistema, ya que al ocurrir esto la falla se convierte en una falla general, ya que todos los módulos hacen uso de los mismos recursos. Además, aunque no ocurran fallas, en caso de que un módulo necesite más recursos, éste podrá dejar sin recursos al resto del sistema provocando una falla.

Por otro lado, se puede encontrar a Lore y Edmodo, que han sido concebidos desde su creación para ser utilizados en la nube. Lore, además de ser de uso gratuito, cuenta con muchas características de una red social, pero se ve afectado por la velocidad de carga de cada uno de sus módulos, así como por proveer sus servicios solamente en inglés. Por su lado, Edmodo posee las mismas características sociales y además agrega el rol de “PADRE”, pero su uso está enfocado principalmente en la educación infantil y la versión en español posee mucho contenido solamente en inglés sin traducción, complicando aún más a los usuarios finales a la hora de su configuración.

Una queja por parte de los usuarios hacia los LMS actuales es la poca o ninguna capacidad de interacción con otros sistemas, algunos usuarios se verían muy beneficiados con la capacidad de conectar sus sistemas actuales de calendario o planificadores con su LMS, pero estos al ser poco configurables y actualizables, no permiten generar módulos adicionales de forma rápida que vayan adhiriendo ya sean necesidades o gustos propios de los clientes.

El estudio busca responder las siguientes preguntas:

Pregunta central

¿Cómo implementar un sistema de gestión de aprendizaje con características de red social evitando los problemas de escalabilidad, falta de resiliencia y uso forzado de tecnologías homogéneas de una arquitectura monolítica?

Preguntas auxiliares

- ¿Cómo empacar y realizar los despliegues de las instancias de servicio de forma que se minimice el tiempo de inactividad de los servicios durante el proceso o durante fallas?
- ¿Cómo administrar las instancias de los servicios para escalar horizontalmente un sistema de gestión de aprendizaje?
- ¿Cómo realizar la comunicación interna en un entorno multilenguaje?
- ¿Cómo reducir el tiempo de desarrollo por parte de los clientes externos para acceder a los servicios de forma individual?

OBJETIVOS

Objetivo general

Diseñar e implementar una arquitectura escalable basada en microservicios para un sistema de gestión de aprendizaje con características de red social.

Objetivos específicos

- Implementar un patrón arquitectónico de instancias de servicio por contenedor que provea resiliencia, para minimizar el tiempo de inactividad de los servicios durante un fallo o despliegue de un nuevo servicio.
- Implementar un patrón arquitectónico de registro de servicios que permita dar de alta y de baja a las instancias de los servicios del sistema.
- Implementar un protocolo de comunicación para realizar peticiones entre servicios implementados en diferentes lenguajes de programación.
- Implementar un patrón arquitectónico "*API Gateway"* (Interfaz de programación de aplicaciones como puerta de enlace) para permitir el acceso a los servicios del sistema, por parte de clientes externos.

RESUMEN METODOLÓGICO

Tipo de investigación

Se realizó una investigación tecnológica o también denominada desarrollo, este tipo de investigación tiene como finalidad la invención de artefactos o de procesos con el objeto de ofrecerlos al mercado y obtener un beneficio económico. (Cegarra Sánchez, J., 2004).

Por otro lado, la investigación tecnológica puede observarse como una obtención de conocimientos particulares, tales como:

- Obtención de nuevos productos
- Configuración de nuevos procesos
- Obtención de nuevos artefactos

Desde otra perspectiva, al realizar una investigación tecnológica, realizó una investigación de tipo interactiva, ya que con la solución se pretende sustituir un estado actual de las cosas como lo es la arquitectura monolítica en la cual se basa la mayoría de sistemas de gestión de aprendizaje; por un nuevo estado de cosas deseado, como lo es una arquitectura escalable basada en microservicios para un sistema de gestión de aprendizaje con características de red social.

Se tomó la decisión de realizar el estudio bajo el tipo de investigación tecnológica, ya que se busca presentar una solución tecnológica innovadora, a un problema de la mayoría de sistemas de gestión de aprendizaje actuales. Al hablar sobre una solución tecnológica innovadora; en el caso de este trabajo, se

refiere a desarrollar un sistema, utilizando tecnologías modernas, que proporcione una solución al problema encontrado, además de hacerlo de una forma intuitiva y de fácil acceso para los usuarios finales.

Diseño de la investigación

Se planteó un diseño documental, a pesar que en el año 2015 aún no existía mucha información documentada sobre el tema, fue importante obtener la poca información existente para tomar un precedente sobre lo que se tenía hasta ese momento. Además, se necesitó fundamentar el curso de acciones a tomar para la solución del problema, y el por qué se realizó la solución de una forma u otra, no dejando sin explicar los detalles de la implementación.

Procedimiento Metodológico

Para la construcción del trabajo se realizaron las siguientes fases:

Fase I. Se definieron los requerimientos funcionales utilizando casos de uso, diagramas de secuencia junto con la definición y creación de microservicios que se implementaron en los contenedores de software, para proveer resiliencia y así minimizar el tiempo de actividad de los servicios durante un fallo o despliegue de un nuevo servicio.

Fase II. Se realizó la instalación y configuración del registrador de servicios que permitió dar de alta y de baja a las instancias de los servicios del sistema que se encontraban en cada uno de los contenedores de software definidos y creados en la fase I.

Fase III. Se definieron y crearon los servicios REST del prototipo que se implementaron sobre los contenedores creados en la fase I.

Fase IV. La fase final fue el desarrollo, instalación y configuración del API Gateway, que permite el acceso a los servicios del sistema por parte de clientes externos. Esto permitió realizar las pruebas de rendimiento para determinar el uso de recursos y la disponibilidad del sistema durante el fallo o despliegue de un servicio.

INTRODUCCIÓN

El mundo de la tecnología evoluciona constantemente, debido a las necesidades que surgen día a día, esto ha provocado que vayan apareciendo nuevos retos para las personas relacionadas, tanto a la infraestructura tecnológica como a los desarrolladores de nuevo software. Uno de estos retos se ha generado por la cantidad de personas que cada día se unen a la red de información, provocando la necesidad de diseñar nuevas arquitecturas que permitan escalar de forma óptima, y que además permitan realizar cambios muchas veces drásticos asegurando que el o los sistemas implementados en ellas no queden fuera de línea, o si en caso se queden inactivos, que sea la menor cantidad de tiempo posible.

El área académica es una de las que más se han visto beneficiadas con la llegada de nuevos sistemas de información y desde 1993, con Blackboard implementando el primer sistema de aprendizaje en línea, hasta el año 2012 en que se popularizaron los MOOC (cursos en línea masivos y abiertos) fueron apareciendo muchos otros sistemas como Moodle o Dockeos, pero manteniendo la misma filosofía de ese primer sistema de 1993, sistemas limitados muy parecidos a los foros en línea, que permiten ver anuncios de los cursos y descargar material de los mismos. Estos sistemas además fueron implementándose sobre arquitecturas monolíticas.

A finales de la primera década del siglo XXI, sistemas como Edmodo y Lore se han dado cuenta que esa primera filosofía, a pesar de ser funcional, muchas veces complicaba el uso de los LMS para que fuera utilizado por cualquier tipo de usuario y basándose en el éxito de las redes sociales como

Facebook, fueron implementando funcionalidades de estas redes sociales en sus sistemas LMS. Otra característica importante de estos nuevos sistemas es que se han implementado en la nube y funcionan solamente como un servicio para las instituciones educativas, a diferencia de los sistemas clásicos en que es necesario un servidor dedicado para su instalación.

Cuando se habla de sistemas que brindarán servicio a cualquier institución que se registre en él, se tiene que hablar de alta disponibilidad y de escalabilidad, dos aspectos muy importantes a tomar en cuenta a la hora de implementar cualquier sistema que será accesible en cualquier parte del mundo en que se cuente con una conexión a internet. Una nueva tendencia para atacar estos aspectos son las arquitecturas basadas en microservicios. Estas arquitecturas comienzan a definirse y a mostrar sus beneficios por el año 2014, siendo utilizadas y apoyadas por empresas grandes del internet como lo son Amazon, Google, Netflix y entre otros.

El presente trabajo de graduación propone el diseño de una arquitectura basada en microservicios que proporcione alta disponibilidad, escalabilidad y facilidad de despliegue en un entorno tecnológico heterogéneo para implementar el prototipo de un sistema de gestión de aprendizaje con características propias de una red social.

En el primer capítulo, se muestran antecedentes sobre las arquitecturas clásicas de tres capas utilizadas por los sistemas de gestión de aprendizaje clásicos, así como de la arquitectura basada en microservicios. En el segundo capítulo, se justifica el desarrollo de este trabajo. En el tercer capítulo, se muestran los alcances del trabajo. En el cuarto capítulo, se muestra el marco teórico.

En el quinto capítulo, se presenta el diseño y desarrollo del prototipo de gestión de aprendizaje, que abarca casos de uso, diagramas de secuencia y diagrama entidad relación; además se muestra la implementación del prototipo sobre una arquitectura basada en microservicios utilizando Docker, Consul, servicios REST y Node.js.

Por último, en el sexto capítulo se analizan los resultados de la implementación del sistema que incluyen los requerimientos necesarios de software y hardware para su funcionamiento, así como las pruebas básicas de rendimiento del prototipo del sistema de gestión de aprendizaje sobre la arquitectura propuesta.

1. ANTECEDENTES

1.1. Arquitectura clásica de tres capas utilizada por los sistemas de gestión de aprendizaje clásicos

Como se puede observar en el artículo: *Schools Go Into the 'Cloud' to Embrace the Popularity of Social Media* presentado en mayo del año 2012 en el New York Times, desde 1990 con BlackBoard como la primera compañía que presentaba una plataforma en línea para ayudar a los educadores, hasta el año del artículo, sistemas como Blackboard y Moodle no han cambiado mucho la filosofía con la cual iniciaron, mientras que a partir de la primera década del siglo XXI, redes sociales como Facebook han influenciado al sector para crear sistemas basados en la filosofía de social, tomando por ejemplo a Lore y Edmodo. (Kolesnikov-Jessop, S., 2012).

Además de su filosofía, a pesar de la llegada de los cursos en línea masivos y abiertos (MOOC por sus siglas en inglés), sistemas de gestión de aprendizaje (LMS) como Moodle, Blackboard Learn, Dockeos, Lore y Edmodo siguen siendo extensamente utilizados. Esto se puede comprobar por medio de sus portales con listados de empresas o instituciones que se encuentran utilizándolos.

Moodle por su lado, cuenta con una página de estadísticas (disponible a septiembre de 2015) que muestra cuántas instalaciones de sitios Moodle hay a nivel mundial, al mes de septiembre de 2015, cuenta con 60,550 instalaciones registradas en 223 países. Además, cuenta con mapa de referencia sobre la cantidad de sitios instalada por país, este mapa también permite colocarse sobre

un país en específico para ver detalles sobre las instalaciones sobre el mismo, esto incluye el ver la cantidad de sitios instalados y un listado de sitios públicos que utilizar el LMS.

Como ejemplo, Guatemala cuenta con 145 sitios Moodle registrados, de los cuales se puede observar que este LMS está siendo utilizado en universidades como la Universidad de San Carlos de Guatemala, Universidad Mesoamericana, Universidad Da Vinci de Guatemala, Universidad Mariano Gálvez y otras. También se pueden encontrar varios colegios y empresas que lo utilizan para el manejo de los cursos de capacitación para sus empleados.

Como se puede ver en el sitio antes indicado, al hablar sobre las universidades que utilizan Moodle, el uso del mismo es por Facultad o extensión universitaria, por ejemplo en la Universidad de San Carlos de Guatemala, existe una instalación de Moodle para la Facultad de Arquitectura (http://www.uv-arquitectura.com/) , una para la Escuela de Ciencia Política (http://www.plataformacienciapoliticausac.info/) , en la Universidad Mariano Gálvez, existe una instalación para la Facultad de Ingeniería (http://ingenieriaumg.hol.es/sitio/), y así se puede ir encontrando más ejemplos que muestran cómo es necesaria toda una infraestructura para hospedar cada uno de los sitios mencionados anteriormente.

Por su lado, Moodle, Dockeos y Blackboard Learn trabajan sobre una arquitectura monolítica de tres capas, de las cuales se tienen la capa de acceso a datos, la capa de negocio y la capa de presentación. Mientras que Blackboard ya viene preparado para escalar por medio de replicación de servidores y balanceo de cargas. Para escalar en Moodle y Dockeos, es necesario instalar un nuevo servidor y replicar el servidor en el cual se realizó la primera instalación; luego es necesario instalar un balanceador de cargas y replicar la base de datos

o utilizar la misma base de datos para las dos instalaciones, todo esto de forma manual, ya que no provee una configuración diseñada para la replicación. (Subramanian, P., Zainuddin, N., Alatawi, S., Javabdeh, T. y Hussin, A. R. C., 2014).

Además, estos LMS se encuentran desarrollados en PHP y por lo general implementados bajo un entorno LAMP o WAMP (Linux o Windows, Apache, MySQL, PHP), esto quiere decir que si se necesitan agregar nuevas funcionalidades o realizar cambios para resolver problemas, es necesario que la persona encargada de realizar estas acciones solamente pueda utilizar el lenguaje PHP; este lenguaje al ser interpretado, no podrá mejorar su rendimiento de forma fácil como un lenguaje compilado, a excepción que se utilicen nuevas tecnologías como la HHVM (Hip Hop Virtual Machine) de Facebook que permite al lenguaje PHP funcionar de forma binaria. (Maurer, B., 2015).

1.2. Arquitectura basada en microservicios

Durante los años 2014 y 2015, empresas como Netflix y Amazon han desempeñado un papel fundamental como casos de éxito utilizando arquitecturas basadas en microservicios. Amazon por su parte, desde el año 1994 nació con una filosofía muy similar a lo que hoy se conoce como microservicios, desde el inicio entendieron que los equipos pequeños pueden trabajar de forma más óptima que los equipos grandes, por ello y siendo una empresa con la capacidad para generar las suficientes herramientas para apoyar a sus desarrolladores, crearon los Amazon Web Services, permitiendo a sus equipos encargarse de todo el ciclo de vida de los proyectos, además permitiéndoles contar con infraestructura diseñada especialmente para sus proyectos. (Richardson, C., 2014).

En 1997, Netflix tomando como ejemplo a Amazon, mientras iniciaba sus labores como videoclub, aseguró estructurarse en torno a equipos pequeños que trabajan independientemente, esto para facilitar cambios y que no se entorpecieran entre ellos. Por ello, Netflix se ha convertida en una de las empresas pioneras en microservicios, al formar equipos pequeños independientes, se vieron forzados de cierto modo a crear servicios de la misma forma, esto les ha permitido escalar de buena forma y optimizar sus procesos de despliegue.

Netflix por medio de su sitio web, en el cual son muy activos sus arquitectos, se ha convertido en las principales referencias de libros como: *Building Microservices* y artículos como *Microservices* en donde se ejemplifican los diversos patrones implementados por la empresa. Entre estos patrones, uno de los más comentados es su API Gateway. Además, por medio de su *Open Source Software Center* proveen muchas de sus herramientas de forma gratuita para los desarrolladores en general. (Balalaie, A., Heydarnoori, A. y Jamshidi, P., 2015).

Y así se pueden ir mostrando más ejemplos de cómo otras compañías van poco a poco tomando en cuenta las arquitecturas basadas en microservicios, y creando nuevas tecnologías para mejorarlos, como por ejemplo: Facebook desarrollando Thrift (protocolo de comunicación, cedido a la fundación Apache como software libre) y Google desarrollando GRPC basado en Protobuf (protocolo de comunicación desarrollado por Google como software libre), ofreciendo sus servicios de Google Container Engine y más tarde su orquestador de microservicios Kubernetes.

2. JUSTIFICACIÓN

El presente trabajo sigue una línea de investigación de las tecnologías de la información y la comunicación para el apoyo en la educación, proponiendo un sistema de gestión de aprendizaje sobre una arquitectura basada en microservicios, que sea escalable, tolerante a fallos y que además permita el uso de tecnologías heterogéneas para su desarrollo.

Para inicios del año 2016, se puede observar que en el repositorio público de contenedores Docker: Docker Hub, se pueden encontrar varias opciones de contenedores Moodle, uno de los sistemas de gestión de aprendizaje más utilizados, pero estos siguen funcionando sobre una arquitectura monolítica dentro del contenedor.

Una alternativa para seguir utilizando sistemas como Moodle de forma modular es descargar y modificar su código fuente, dividiéndolos en módulos e instalando cada uno de ellos en distintos contenedores, pero éstos al no estar pensados para este tipo de arquitectura dificultan este tipo de implementación e incrementan la probabilidad de ocurrencia de errores, por lo tanto, se vuelve obligatorio seguir utilizando la arquitectura monolítica para la cual están diseñados.

Otra alternativa para asegurar que el sistema de gestión de aprendizaje cuente con escalabilidad y tolerancia a fallos es utilizar uno de los sistemas que actualmente se encuentran en la nube como lo son: Lore o Edmodo. Estos sistemas han evolucionado de buena forma, pero a pesar de ellos no han sido tan bien recibidos por la comunidad académica.

Por su lado, Lore el día 5 de febrero del año 2016 colocó un anuncio en la página principal de su portal informando sobre el cese de operaciones el 27 de junio del año 2016. Por otro lado, Edmodo ha tenido rechazo por parte de los educadores por carecer de un sistema de monitoreo para estudiantes, por lo cual muchos estudiantes han caído en utilizarlo solamente como una red social interna del establecimiento académico, en lugar de utilizarlo como un sistema de gestión de aprendizaje.

3. ALCANCES

3.1. Investigativos

- Mostrar las diferencias entre las arquitecturas monolíticas y arquitecturas basadas en microservicios utilizando contenedores de software, así como sus ventajas y desventajas.
- Proponer el uso de un registrador de servicios para el descubrimiento de los mismos, por parte de sus clientes, ya sean estos otros servicios internos o externos.
- Dar a conocer el beneficio de utilizar servicios REST para la comunicación interna entre los servicios implementados sobre una arquitectura basada en microservicios.
- Proponer el uso de la programación reactiva para la creación de un API Gateway, que permita exponer servicios internos a los clientes externos sobre una arquitectura basada en microservicios.

3.2. Técnicos

- Diseñar e implementar una arquitectura basada en microservicios utilizando la herramienta de uso gratuito draw.io para el diseño y Docker, para la administración de contendores de software.
- Instalar y configurar un registrador de servicios para su uso por parte de los sistemas del prototipo de sistema de gestión de aprendizaje propuesto.
- Definir e implementar los servicios REST necesarios para la comunicación interna entre los servicios del prototipo de sistema de gestión de aprendizaje propuesto.

- Desarrollo de un API Gateway utilizando Node.js y el paradigma de programación reactiva para acceder a los servicios del sistema.

3.3. Resultados

Prototipo de sistema de gestión de aprendizaje implementado sobre una arquitectura basada en microservicios. El prototipo contará con las siguientes funcionalidades:

- Creación de usuario
- Autenticación en el sistema
- Creación de curso
- Asignación de curso
- Publicación de comentario en muro

Como resultados de esta implementación se espera:

- Implementación de un patrón arquitectónico de instancias de servicio por contenedor que provea resiliencia para minimizar el tiempo de inactividad de los servicios durante un fallo o despliegue de un nuevo servicio.
- Implementación de un patrón arquitectónico de registro de servicios que permita dar de alta y de baja a las instancias de los servicios del sistema.
- Implementación de un protocolo de comunicación para realizar peticiones entre servicios implementados en diferentes lenguajes de programación.
- Implementación de un patrón arquitectónico "*API Gateway"* para permitir el acceso a los servicios del sistema por parte de clientes externos.

4. MARCO TEÓRICO

4.1. Sistema de gestión de aprendizaje

Un sistema de gestión de aprendizaje o LMS por sus siglas en inglés de *Learning Management System*, “es un software instalado en un servidor web que se emplea para administrar, distribuir y controlar las actividades de formación no presencial de una institución u organización” (Watson, W. R., Watson, S. L., 2007).

4.2. Microservicios

El término *Microservice Architecture*, como lo definen James Lewis y Martin Fowler en su artículo: *Microservices*, uno de los artículos más populares sobre el tema en la web, “es una forma particular de diseño de aplicaciones de software como suites de servicios desplegados independientemente”, pero éstos además hacen énfasis, en que “a pesar que no existe una definición precisa de este tipo de arquitectura, hay ciertas características comunes en torno a la organización, la capacidad empresarial, el despliegue automatizado, la inteligencia en los puntos finales y el control descentralizado de los lenguajes y datos”. (Lewis, J., Fowler, M., 2014).

Por otro lado, se puede encontrar la definición por parte de Sam Newman en su libro: *Building Microservices*, “los microservicios son pequeños y autónomos servicios que trabajan juntos”, pero además deben de estar enfocados en hacer una cosa bien.

Estos permiten obtener beneficios clave como lo son el permitir utilizar tecnologías heterogéneas, la resiliencia, una gran capacidad a la hora de escalar, facilidad de despliegue, alineamiento organizacional, componibilidad y se encuentran optimizados para la reemplazabilidad. (Newman, S., 2015).

4.3. Virtualización

La virtualización es una forma de abstraer aplicaciones y sus componentes internos lejos del hardware dándoles soporte y presentando una vista lógica o virtual de estos recursos. (Kusnetzky, D., 2011).

Como su nombre lo indica la virtualización permite crear recursos virtuales, a través de software, estos pueden ser recursos de hardware, de red, de almacenamiento o sistemas operativo, entre otros.

La virtualización de plataformas puede realizarse a nivel completo, esto significa que se virtualiza a todos los componentes de hardware necesarios para instalar un sistema operativo completo dentro del sistema operativo anfitrión sin necesidad de ser modificado. Por otro lado, se tiene la virtualización parcial, que permite virtualizar ciertos componentes de hardware necesarios para ejecutar un sistema operativo diseñado específicamente para el entorno instalado. Y, por último, se tiene la virtualización por sistema operativo o semi-parcial, éste permite ejecutar el sistema operativo sobre el software de virtualización sin crear recursos virtuales de hardware, esto quiere decir que el sistema operativo huésped utilizará los mismos recursos que el sistema operativo anfitrión.

4.4. Contenedor de software

Los contenedores de software son instancias de espacios de usuario aisladas permitidas por solamente un kernel de un sistema operativo, a este método de permitir múltiples instancias en lugar de solamente una, se le conoce como virtualización a nivel de aplicación. (Mouat, A., 2016).

4.4.1. rkt

Rkt es una interfaz de línea de comandos de código abierto para la ejecución de contenedores de software sobre el sistema operativo Linux, al igual que Docker permite automatizar el empaquetado, envío y despliegue de aplicaciones sobre contenedores de software. Una de sus mayores desventajas es la escasa documentación en comparación con Docker y que no se tiene un registro de casos de éxito utilizando la herramienta. (Mocevicius, R., 2015).

4.4.2. Docker

Docker es un motor de contenerización de código abierto, que automatiza el empaquetado, envío, y despliegue aplicaciones de software que son presentadas como contenedores ligeros, portátiles y autosuficientes, que se ejecutarán en prácticamente cualquier lugar. Entre las mayores ventajas de Docker, se puede observar que provee una documentación bien detallada; también se pueden encontrar muchos cursos y tutoriales en línea, así como certificaciones oficiales sobre su aprendizaje. También se cuenta con Docker Hub, que es un repositorio público de imágenes disponibles para Docker, muchas de ellas configuradas por los mismos creadores de los sistemas operativos o herramientas que contienen dichas imágenes, entre estas herramientas se

pueden encontrar desde bases de datos, servidores de aplicaciones, sistemas de gestión de contenido, etc. (Raj, P., Chelladhurai, J. S., Singh, V., 2015).

4.4.2.1. Docker Machine

Herramienta que permite la creación de hosts (anfitriones) para contenedores Docker. Además de la creación hosts, también permite instalar Docker en cada uno de ellos, así como configurar automáticamente el cliente de Docker para comunicarse con él de forma remota. (Holla, S., 2015).

4.4.2.2. Docker Swarm

Herramienta que permite a un grupo de hosts presentarse como un solo host virtual de contenedores Docker, esto permite que cualquier aplicación o herramienta que se comunique con el demonio de Docker pueda utilizar Swarm para escalar transparentemente a múltiples hosts. (Holla, S., 2015).

4.4.2.3. Docker Compose

Es una herramienta para la definición y ejecución de aplicaciones multi-contenedor de Docker. Esto permite que con un simple comando se puedan crear y ejecutar todos los servicios definidos en un archivo de configuración. (Holla, S., 2015).

4.5. Registrador de servicios

Cuando se habla de microservicios, al tratarse de tener múltiples servicios que van a trabajar en conjunto, es necesario una herramienta que permita indicarle a cada cliente de cada servicio cómo localizar las instancias del mismo.

Para ellos es necesario una base de datos de los servicios, sus instancias y sus locaciones y un administrador (registrador de servicios) que permita el registro y dar de baja a los servicios, así como una interfaz que para la consulta de servicios por parte de sus clientes. (Newman, S., 2015).

4.5.1. Netflix Eureka

Entre las herramientas de código abierto desarrolladas por Netflix se puede encontrar Eureka, registrador de servicios utilizado para la plataforma de la misma empresa, es proveída como una librería que permite la instanciación de la herramienta como un servicio REST, entre sus desventajas se encuentra que solamente está disponible para ser utilizada con el lenguaje de programación Java, y es necesario el uso de otras librerías para tener clientes desarrollados en otros lenguajes de programación. Y como ventaja se puede apreciar la compatibilidad que tiene con las otras herramientas de código abierto proveídas por Netflix. (Newman, S., 2015).

4.5.2. Consul

Desarrollada por Hashicorp, Consul es una herramienta basada en servicios REST para el registro de servicios, a diferencia de Eureka se provee clientes para diferentes lenguajes, y para los que no existen clientes específicos, se puede interactuar con ella por medio de su interfaz REST. Además, sus mayores ventajas, provee detección de fallos, integración en entornos distribuidos y almacenamiento de llave/valor. (Newman, S., 2015).

4.6. Comunicación entre microservicios

4.6.1. Colas de mensajes

Como su nombre lo indica, son almacenamientos temporales para mensajes que necesitan compartirse de un sistema a otro, esta comunicación funciona de forma asíncrona; teniendo, por un lado, al cliente que inserta el mensaje a la cola; por el otro; al cliente que saca al mensaje de la cola para procesarlo. Entre los sistemas de colas de mensajes se puede encontrar a RabbitMQ, Apache ActiveMQ, entre otros.

4.6.2. RCP

Por sus siglas del inglés *Remote Procedure Call* o su traducción al español “Llamada a procedimiento remoto” es un protocolo de red para la comunicación de un sistema a otro, que permite a los sistemas despreocuparse de los detalles necesarios para una comunicación exitosa. Estas llamadas funcionan de forma síncrona.

4.6.3. REST

La transferencia de estado representacional originalmente se refería a un conjunto de principios de arquitectura, pero hoy en día ha cambiado para describir cualquier interfaz entre sistemas que utilice directamente HTTP utilizando diferentes formatos como JSON, XML, entre otros.

A los sistemas que siguen los principios REST se les conoce como RESTful. La transferencia de información utilizando REST puede ser síncrona o asíncrona, pero deben de cumplir con ciertas restricciones, las cuales son:

- Cliente-servidor**:** se refiere a la separación de la interfaz de usuario que se encuentra del lado del cliente, con el servidor, que esperará las peticiones del cliente para almacenar y procesar los datos.
- Sin estado (Stateless)**:** indica que el servidor en ningún momento deberá de guardar el estado del cliente, esto quiere decir, que el cliente deberá de enviar toda la información requerida para que el servidor entienda la petición y pueda almacenar o procesar los datos.
- Cacheable: se refiere a que todas las respuestas del servidor a las peticiones de los clientes deber de poder almacenarse en memoria u otro almacenamiento veloz durante un tiempo corto (si se encuentra activado). Esto permite responder de forma más rápida a las peticiones que se realizan de forma frecuente.
- Interfaz uniforme: indica que se debe tener una interfaz uniforme para la comunicación entre todos los componentes. Esto facilita la creación o modificación de código para la realización de peticiones.
- Sistema de capas: los componentes deben de estar separados en capas, esto permite que cada componente pueda acceder solamente a los componentes necesarios, y evita el error a la hora de realizar las peticiones.
- Código-on-Demand: siendo esta la única restricción opcional, el cliente debe tener la capacidad de descargar y ejecutar código proveído por el servidor. (Doglio, F., 2015).

4.7. API Gateway

Al igual que con la necesidad de que exista un registrador de servicios, al utilizar una arquitectura basada en microservicios, surge la necesidad de elegir qué servicios serán de uso interno, así como los servicios que se configurarán para su uso externo. Para ello se utiliza un API de puerta de enlace o “API Gateway” como único punto de entrada para todos los clientes externos, esto

quiere decir que los clientes externos no accederán a los servicios de forma individual; por lo tanto, se podrá manejar de mejor forma la seguridad y la configuración a la hora de exponer los servicios.

4.7.1. Programación reactiva

A la hora de utilizar una puerta de enlace, también se agrega a la plataforma otro punto de posible falla y que puede convertirse en un cuello de botella. La puerta de enlace al recibir peticiones por parte de los clientes, debe de llamar a todos los microservicios necesarios para satisfacer las solicitudes recibidas, si se utiliza un paradigma tradicional, la puerta de enlace necesitaría dejar bloqueado un hilo exclusivamente para satisfacer una petición del cliente, mientras todos los servicios terminan de resolverse, esto en un sistema con muchas peticiones podría provocar la denegación de servicio, que aún se podría empeorar si los microservicios necesarios se tardasen mucho en resolver o si dieran algún error. Para resolver esta clase de problemas, se puede utilizar el paradigma de la programación reactiva, este se basa en la propagación del cambio.

La programación reactiva permite que la puerta de enlace pueda liberar el hilo mientras los microservicios resuelven y retomarlo en cuanto estos finalicen y envíen una respuesta. Este paradigma de programación se basa en cuatro factores importantes que debe de tener todo sistema, estos son: ser modular, escalable, tolerante a fallos y responsivo. (Tsvetinov, N., 2015).

4.7.2. Node.js

Es un entorno en tiempo de ejecución multiplataforma construido sobre el motor JavaScript V8 del navegador Chrome de Google. Maneja un modelo de no

bloqueo manejo por eventos que lo hacen ligero y eficiente. Node.js permite desarrollar bajo el paradigma de programación reactiva por medio de la funcionalidad EventEmitter que permite la suscripción a eventos y actuar sobre ellos de forma asíncrona. (Node.js Foundation, s.f.).

4.7.3. Extensiones Rx

Las extensiones Rx o extensiones reactivas son librerías para el desarrollo de aplicaciones reactivas, creadas inicialmente para .NET por Microsoft, pero liberadas como código libre, llegando a otros entornos de programación por medio de ReactiveX, definida por sus creadores como “una librería para la composición de programas asíncrona y manejada por eventos utilizando secuencias observables”.

ReactiveX extiende el patrón observador para soportar secuencia de datos y/o eventos y agregar operadores que permiten crear secuencias declarativas abstrayendo conceptos de manejo de hilos, sincronización, estructuras de datos concurrentes y entrada y salida sin bloqueos. (Tsvetinov, N., 2015).

4.8. Redis

Es un sistema almacenamiento de datos NoSQL de tipo llave-valor, que funciona como un servidor de persistencia en memoria con posibilidad de copia a disco, permitiendo almacenar información de tipo cadena, listas, hashes, listas ordenadas y otras estructuras de datos.

Redis es principalmente utilizado para el almacenamiento de datos que deben estar disponible en tiempo real, como por ejemplo, datos utilizados en tableros (“dashboards”), datos se sesión de usuario y otros. (Da Silva, M. D., Lopez Tavares, H., 2015).

5. PRESENTACIÓN DE RESULTADOS

5.1. Diseño y desarrollo de prototipo de sistema de gestión de aprendizaje

5.1.1. Funcionalidades del prototipo

Por motivo del presente trabajo, las funcionalidades básicas con que cuenta el prototipo del sistema de gestión de aprendizaje son las siguientes:

- Creación de usuario

Permite la creación de usuarios de tipo estudiante y usuario tipo profesor. Los datos necesarios para la creación de usuario son: el nombre, apellido, dirección de correo electrónico y contraseña.

- Autenticación en el sistema

Permite la identificación de un usuario con el sistema ya sea de tipo estudiante o de tipo profesor. Los datos necesarios para la autenticación de un usuario en el sistema son su dirección de correo electrónico y su contraseña.

- Creación de curso

Permite la creación de cursos en el sistema, estos pueden ser creados solamente por usuarios tipo profesor. Los datos necesarios para la creación de un curso son el nombre y la descripción del mismo.

- Asignación de curso

Permite que un usuario tipo alumno pueda formar parte de un curso, por motivos del prototipo cualquier usuario tipo alumno se puede asignar cualquier curso; sin embargo, para un sistema de gestión de aprendizaje completo, un usuario tipo profesor debería de aprobar la asignación de un alumno.

- Funcionalidades de red social que pueden incluirse en un sistema gestión de aprendizaje

Existen ciertas funcionalidades que se han implementado inicialmente en los sistemas de las redes sociales que puede ser de beneficio implementarlas en un sistema de gestión de aprendizaje, para demostrar una de estas funcionalidades, el sistema contará con la funcionalidad de muro de comentarios.

- Muro de comentarios

Una funcionalidad básica de cualquier red social es el muro de comentarios, este permite que cualquier usuario coloque un comentario o noticia y que cualquier otro usuario pueda visualizarlo ya sea en la página principal del sistema o en algún apartado en especial.

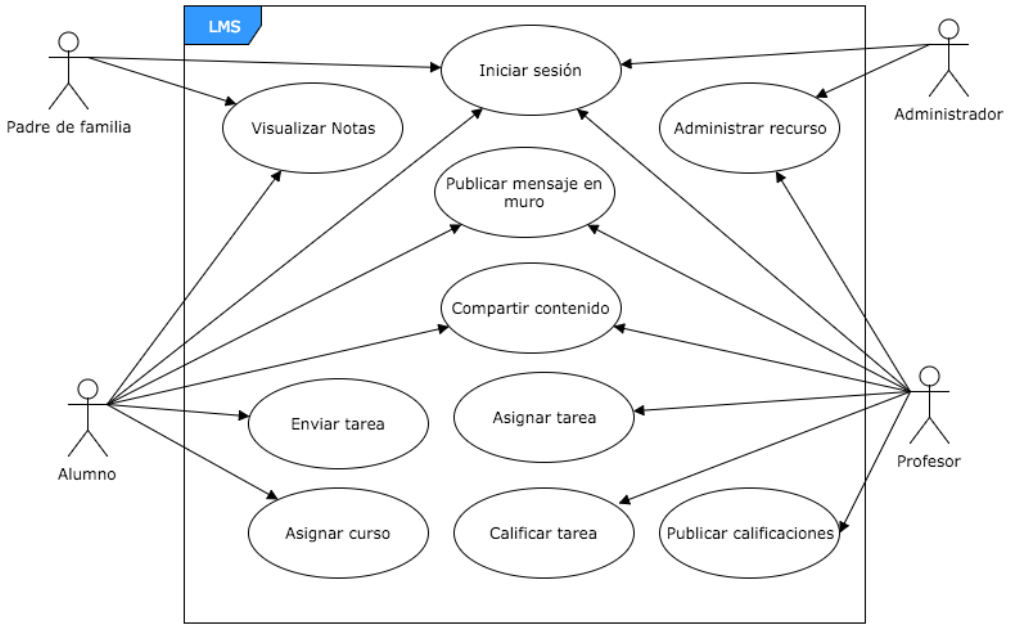
Una forma de implementarlo en un sistema de gestión de aprendizaje es colocando un muro de comentarios en la página principal de cada curso disponible en el sistema. Estos comentarios podrán ser creados, tanto por usuarios tipo profesor como por usuarios tipo alumnos, pero con la diferencia que los comentarios creados por el usuario tipo profesor se deberán de visualizar como comentarios especiales, por la importancia que estos puedan tener para todos los usuarios tipo alumno.

Los comentarios publicados por los usuarios tipo alumno deberán de ser moderados por el usuario tipo profesor a cargo del curso al cual pertenezcan.

5.1.2. Casos de uso

Para efectos del presente trabajo, se detallan solamente los casos de uso de las funcionalidades de un sistema de gestión de aprendizaje presentadas en el apartado anterior “Funcionalidades del prototipo”; sin embargo, en la figura 1, se muestra la totalidad de casos de uso que como mínimo debería de tener un prototipo completo de un sistema de gestión de aprendizaje.

Figura 1. **Diagrama de casos de uso de prototipo de sistema de gestión de aprendizaje**



Fuente: elaboración propia.

5.1.2.1. Especificación de casos de uso de las funcionalidades del prototipo

5.1.2.1.1. Especificación de caso de uso de creación de usuario

A continuación, en la tabla I se presenta el caso de uso de la creación de usuario, ésta solamente cuenta con el nombre, el actor y el flujo principal del caso de uso, ya que la descripción del mismo forma parte del apartado “Funcionalidades del prototipo”.

Tabla I. **Caso de uso de creación de usuario**

Nombre:	Creación de usuario	
Actor:	Alumno, profesor	
Flujo principal:	**Eventos actor**	**Eventos sistema**
	1 Ingresa al sistema de gestión de aprendizaje. 3 Selecciona la opción de crear usuario. 5 Ingresa los datos solicitados por el sistema y presiona continuar.	2 Muestra pantalla con las opciones de crear usuario y autenticarse en el sistema. 4 Solicita nombre, apellido, dirección de correo electrónico y contraseña. 6 Muestra mensaje de usuario creado exitosamente y redirecciona a pantalla principal.

Fuente: elaboración propia.

5.1.2.1.2. Especificación de caso de uso de autenticación de usuario

A continuación, en la tabla II se presenta el caso de uso de la autenticación de usuario, ésta solamente cuenta con el nombre, el actor y el flujo principal del caso de uso, ya que la descripción del mismo forma parte del apartado “Funcionalidades del prototipo”.

Tabla II. **Caso de uso de autenticación de usuario**

Nombre:	Autenticación de usuario	
Actor:	Alumno, profesor	
Flujo principal:	**Eventos actor**	**Eventos sistema**
	1 Ingresa al sistema de gestión de aprendizaje. 3 Selecciona la opción autenticarse en el sistema. 5 Ingresa los datos solicitados por el sistema y presiona continuar.	2 Muestra pantalla principal con las opciones de crear usuario y autenticarse en el sistema. 4 Solicita dirección de correo electrónico y contraseña. 6 Muestra mensaje de bienvenida y redirecciona a pantalla de administración de usuario.

Fuente: elaboración propia.

5.1.2.1.3. Especificación de caso de uso de creación de curso

A continuación, en la tabla III se presenta el caso de uso de la creación de curso, ésta solamente cuenta con el nombre, el actor, el flujo principal del caso de uso y la precondición, ya que la descripción del mismo forma parte del apartado “Funcionalidades del prototipo”.

Tabla III. **Caso de uso de creación de curso**

Nombre:	Creación de curso	
Actor:	Profesor	
Flujo principal:	**Eventos actor**	**Eventos sistema**
	1 Selecciona la opción “Crear nuevo curso” 3 Ingresa los datos solicitados por el sistema.	2 Solicita el nombre y la descripción del curso. 4 Muestra mensaje de curso creado exitosamente y redirecciona a la pantalla principal del curso recién creado.
Precondición:	1 El profesor debe estar autenticado.	

Fuente: elaboración propia.

5.1.2.1.4. Especificación de caso de uso de asignación de curso

A continuación, en la tabla IV se presenta el caso de uso de la asignación de curso, ésta solamente cuenta con el nombre, el actor, el flujo principal del caso de uso y la precondición, ya que la descripción del mismo forma parte del apartado “Funcionalidades del prototipo”.

Tabla IV. **Caso de uso de asignación de curso**

Nombre:	Asignación de curso	
Actor:	Alumno	
Flujo principal:	**Eventos actor**	**Eventos sistema**
	1 El usuario selecciona asignación a curso.	2 El sistema muestra el mensaje de asignación exitosa y redirecciona a la pantalla principal del curso.
Precondición:	1 El alumno debe de estar autenticado. 2 El alumno debe estar en la pantalla de información de curso.	

Fuente: elaboración propia.

5.1.2.1.5. Especificación de caso de uso de publicación de comentario en muro

A continuación, en la tabla V se presenta el caso de uso de la publicación de comentario en muro, esta solamente cuenta con el nombre, el actor, el flujo principal del caso de uso y la precondición, ya que la descripción del mismo forma parte del apartado “Funcionalidades del prototipo”.

Tabla V. **Caso de uso de publicación de comentario en muro**

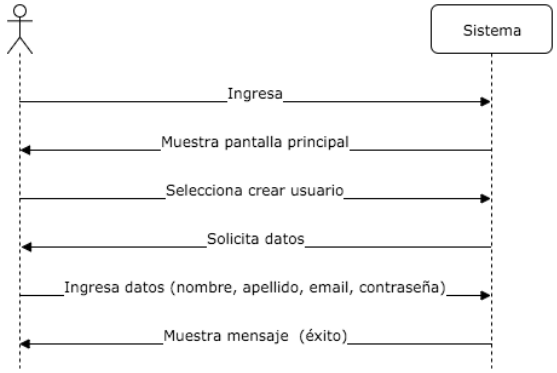
Nombre:	Publicación de comentario en muro	
Actor:	Alumno, profesor	
Flujo principal:	**Eventos actor**	**Eventos sistema**
	1 El alumno o profesor escribe comentario y presionar el botón “publicar”.	2 El sistema coloca el comentario al inicio del muro de comentarios.
Precondición:	1 El alumno o profesor debe estar autenticado. 2 El alumno o profesor debe haber ingresado a un curso.	

Fuente: elaboración propia.

5.1.3. Diagramas de secuencia

Para comprender de mejor forma los cinco casos de uso presentados anteriormente, a continuación, en la figura 2 se puede observar el diagrama de secuencia que corresponde al caso de uso de la creación de usuario.

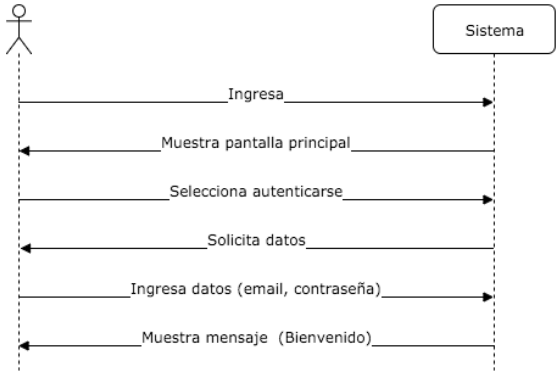
Figura 2. **Diagrama de secuencia de creación de usuario**



Fuente: elaboración propia.

En la figura 3 se puede observar el diagrama de secuencia que corresponde al caso de uso de la autenticación de un usuario con el sistema.

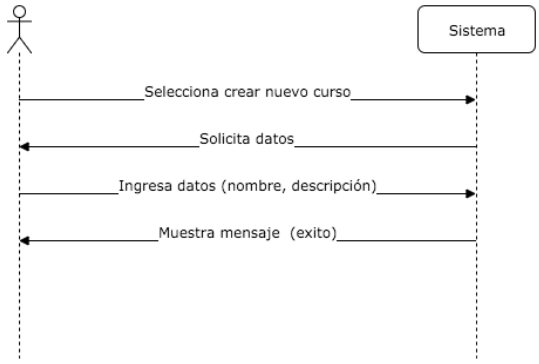
Figura 3. **Diagrama de secuencia de autenticación de usuario**



Fuente: elaboración propia.

En la figura 4 se puede observar el diagrama de secuencia que corresponde a la funcionalidad del caso de uso de creación de curso por medio de un usuario de tipo profesor.

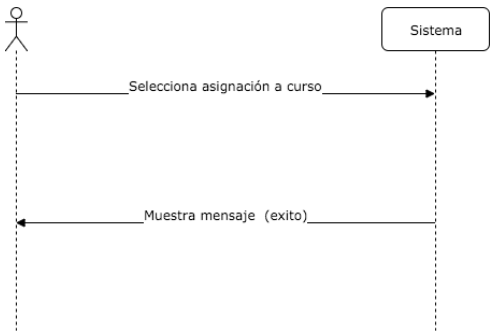
Figura 4. **Diagrama de secuencia de creación de curso**



Fuente: elaboración propia.

En la figura 5 se puede observar el diagrama de secuencia que corresponde a la funcionalidad del caso de uso de la asignación de usuario, en este se puede ver la forma en que un usuario tipo alumno se asigna a un curso para tener la capacidad de ver su contenido.

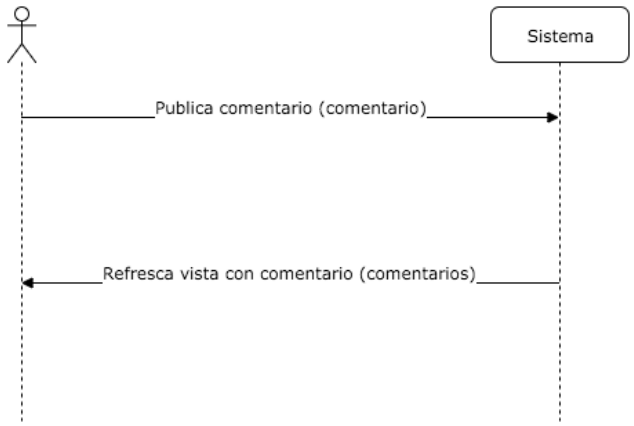
Figura 5. **Diagrama de secuencia de asignación de usuario**



Fuente: elaboración propia.

Por último, en la figura 6 se puede observar el diagrama de secuencia que corresponde al caso de uso de la publicación de comentario en muro.

Figura 6. **Diagrama de secuencia de publicación de comentario en muro**

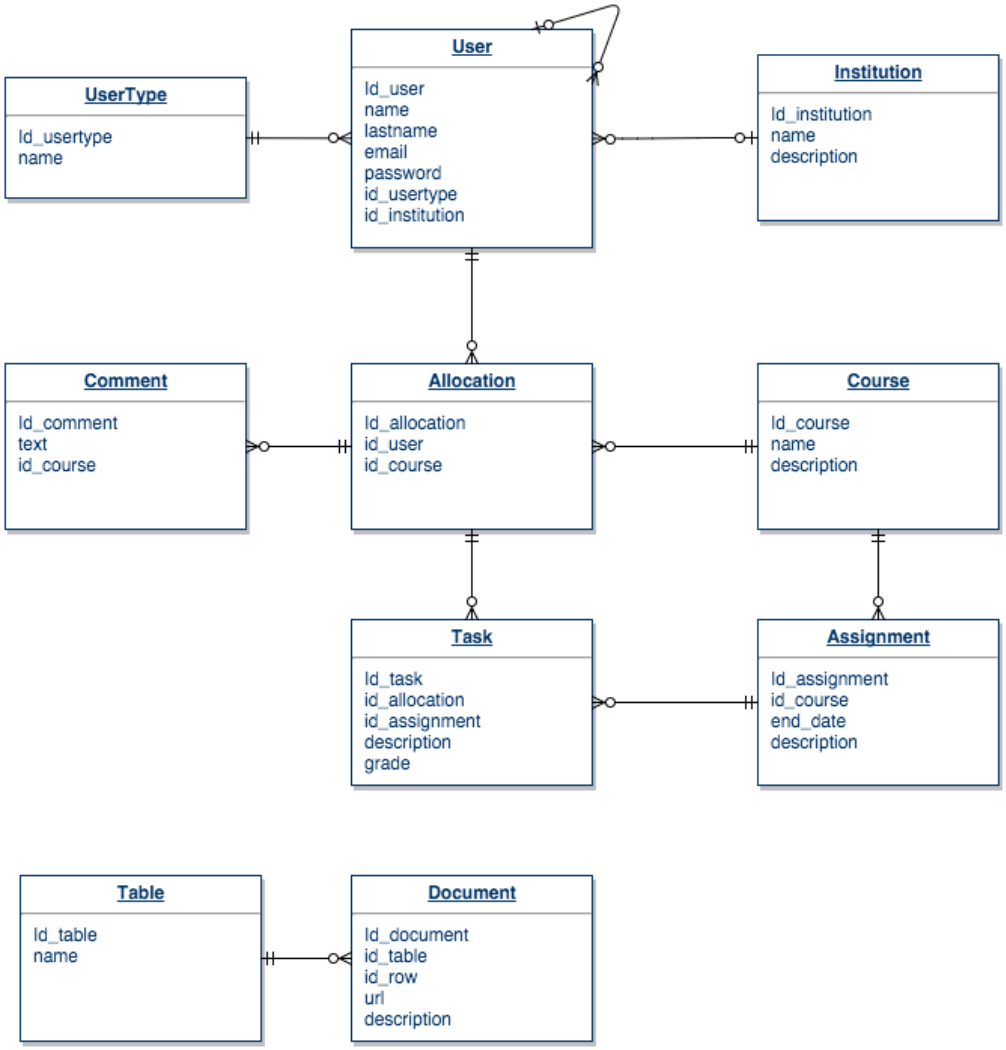


Fuente: elaboración propia.

5.1.4. Diagrama entidad relación

A continuación, la figura 7 muestra el diagrama entidad relación elaborado para el prototipo del sistema de gestión de aprendizaje, en el cual se pueden apreciar todas las tablas necesarias para el prototipo. Este diagrama no solo satisface lo necesario para las funcionalidades que tendrá el prototipo, sino que también contempla las funcionalidades que no son tomadas en cuenta para motivos de este trabajo.

Figura 7. **Diagrama entidad relación**



Fuente: elaboración propia.

5.2. Implementación de prototipo de sistema de gestión de aprendizaje sobre una arquitectura basada en microservicios

Con el auge que ha tomado el paradigma de las arquitecturas basadas en microservicios, día a día va aumentando el número de alternativas disponibles en el mercado de cada uno de los componentes necesarios para construir este tipo de arquitecturas. Por lo tanto, para cada uno de los componentes que se utilizó o se desarrolló para la implementación del prototipo del sistema de gestión de aprendizaje sobre una arquitectura basada en microservicios, fueron tomados en cuenta ciertos aspectos de importancia de acuerdo al componente a evaluar.

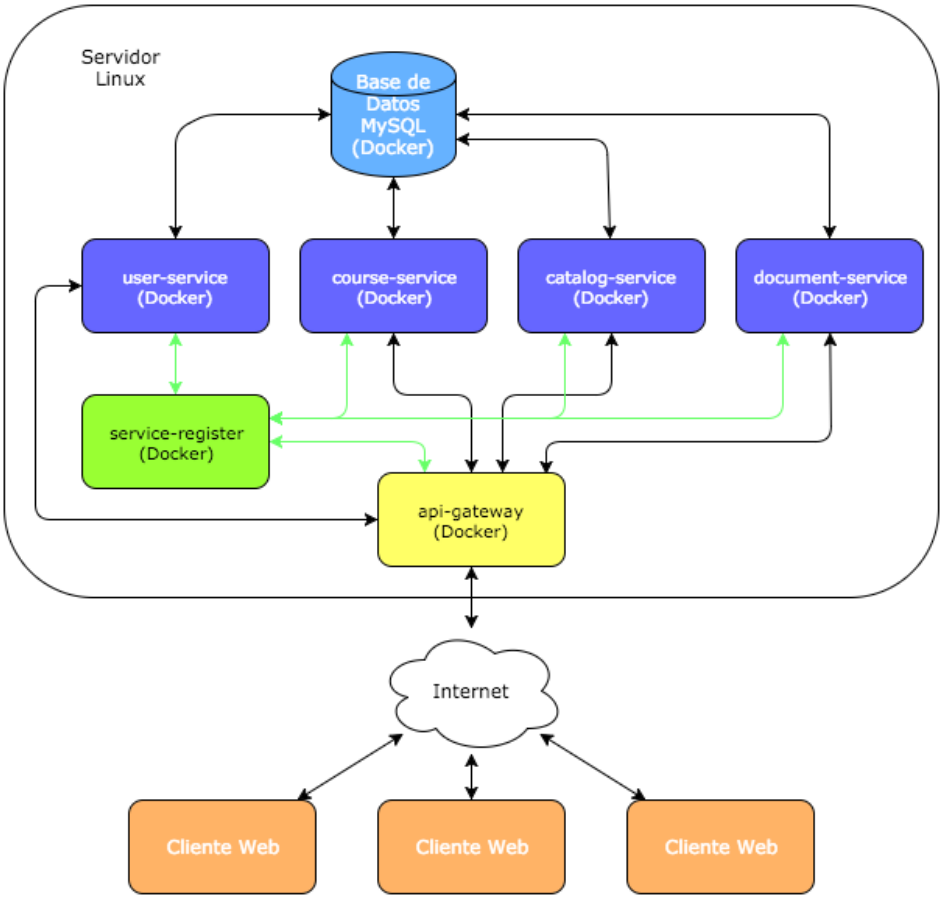
Los componentes básicos que se han definido para la arquitectura propuesta son:

- Contenedores de software
- Registrador de servicios
- API Gateway
- Servicios

5.2.1. Diagrama de arquitectura

La figura 8 muestra el diagrama de la arquitectura propuesto para el sistema de gestión de aprendizaje; sin embargo, el prototipo final de este trabajo solamente cuenta con el servicio de usuario y el servicio de curso, ya que con estos servicios se pueden satisfacer las funcionalidades propuestas en el trabajo.

Figura 8. **Diagrama de arquitectura**



Fuente: elaboración propia.

5.2.2. Implementación de instancias de servicio por contenedor de software utilizando Docker

Para el manejo de los contenedores de software sobre el cual se implementa el prototipo de sistema de gestión de aprendizaje se utilizó Docker. Esto debido a que desde el mes de marzo del año 2013 hasta el año 2016 ha sido el gestor de contenedores de software más importante del mercado.

Algunas de las ventajas de utilizar Docker son:

- Docker es software libre y de uso gratuito, además se cuenta con una gran comunidad y documentación de alto nivel en Internet.
- El repositorio público de imágenes de contenedores Docker Hub, contiene la mayoría de herramientas más utilizadas en el mercado, creadas de forma oficial con la colaboración de los equipos que las desarrollan.
- Alto nivel de integración con diferentes herramientas de infraestructura, tales como: AWS de Amazon, Google Cloud Platform, IBM Bluemix y Microsoft Azure, entre otras.
- Los mayores contribuyentes en su código fuente son: Red Hat, el equipo de Docker, IBM, Google, Cisco Systems y Amadeus IT Group.

En el sitio Docker Hub se cuenta con una gran cantidad de imágenes disponibles para ser descargadas de forma gratuita, para este trabajo se utilizan las siguientes imágenes:

- MySQL: gestor de base de datos software libre, disponible para su descarga de forma gratuita.

- Node.js: permite ejecutar programas desarrollados en JavaScript del lado del servidor, se utiliza para la implementación del API Gateway. Node.js es software libre y está disponible para descargarse de forma gratuita.
- Golang: permite ejecutar programas desarrollados en Go, se utiliza para la implementación de los servicios de usuario y curso. Go es software libre y está disponible para descargarse de forma gratuita.
- Consul: registrador de servicios, software libre, disponible para su descarga de forma gratuita.
- Registrator: permite dar de alta en Consul a los contenedores Docker cuando estos se ponen en ejecución de forma automática. Registrator es software libre y está disponible para descargarse de forma gratuita.
- Cadvisor: permite monitorear los recursos de los contenedores Docker.

En el apartado de cada uno de los componentes se muestra como descargar y ejecutar cada una de las imágenes listadas anteriormente.

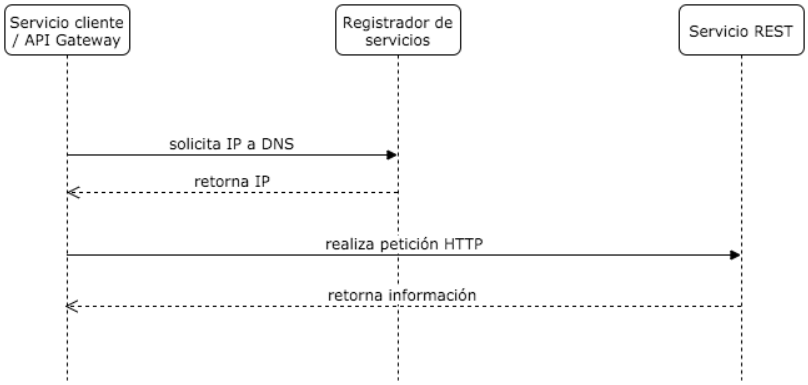
5.2.3. Implementación de Consul como registrador de servicios

Debido a que cada uno de los servicios se encuentra en un contenedor distinto y éstos hacen uso de IPs dinámicas, es necesario contar con un registro de los servicios que forman parte del sistema, y así centralizar el manejo de las rutas internas. Para esta tarea se seleccionó Consul, que es un conjunto de componentes para el descubrimiento y configuración de servicios, éste permite que los clientes puedan descubrir a los servicios por medio del uso de DNS o HTTP.

- Descubrimiento por DNS

Una de las ventajas del uso de Consul es que permite configurarse como un sistema de nombres de dominio (DNS). Como se muestra en la figura 9, el servicio cliente solicita a Consul la dirección IP del servicio al cual requiere hacer una petición y éste retorna la dirección IP, para que el servicio cliente conozca la ubicación a la cual deberá realizar la petición de la información.

Figura 9. **Diagrama de secuencia de solicitud HTTP utilizando DNS para el descubrimiento de servicios**

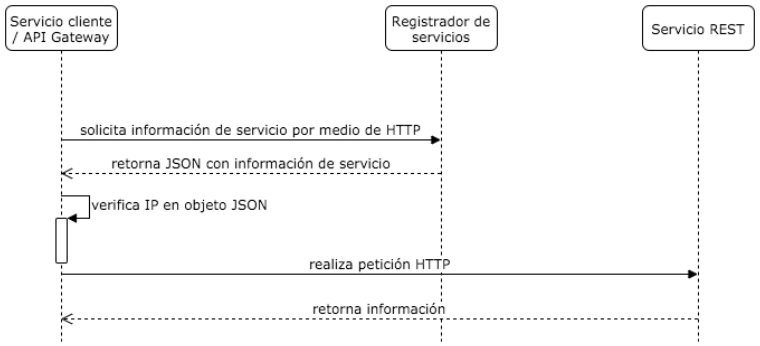


Fuente: elaboración propia.

- Descubrimiento por HTTP

Consul además permite el descubrimiento por medio del protocolo HTTP, como se muestra en la figura 10, el servicio cliente puede solicitar a Consul la información del servicio que quiere consumir y éste retorna un JSON con toda la información necesaria para que el cliente luego pueda verificar la dirección IP a la cual solicitará la información.

Figura 10. **Diagrama de secuencia de solicitud HTTP utilizando el protocolo http para el descubrimiento de servicios**



Fuente: elaboración propia.

La instalación de Consul puede realizarse descargando el binario desde la página web oficial o puede descargarse como una imagen Docker. Para el desarrollo del prototipo del presente trabajo se descargó como una imagen Docker, ya que los servicios estarán implementados sobre este manejador de contenedores de software.

A continuación, se listan los pasos para la descarga y ejecución de Consul utilizando imágenes Docker.

Paso 1: Descargar la imagen utilizando el siguiente comando:

```
docker pull gliderlabs/consul
```

Paso 2: Ejecutar la imagen utilizando el siguiente comando:

```
docker run -p 8400:8400 -p 8500:8500 -p 8600:53/udp -h node1 gliderlabs/consul -server -bootstrap -ui-dir /ui
```

Donde –p permite mapear un puerto interno del contenedor de software hacia un puerto de la máquina anfitrión, en este caso se mapean los puertos 8400 para la interfaz RPC, 8500 para la interfaz HTTP y 8600 para la interfaz DNS.

La etiquieta –h permite configurar el nombre de host, en este caso node1, luego se coloca el nombre de la imagen a utilizar “gliderlabs/consul” y por último, la etiqueta –ui-dir /ui para activar la interfaz de usuario.

Paso 3: ya con los pasos 1 y 2 realizados, se puede verificar la interfaz HTTP con el comando:

```
curl http://localhost:8500/v1/catalog/nodes
```

O se puede verificar la interfaz DNS por medio del comando:

```
dig @0.0.0.0 –p 8600 node1.node.consul
```

- Registro de servicios en Consul por medio de Registrator

Para el registro de los servicios se hace uso de la imagen Registrator, que permite dar de alta a los servicios de forma automática en cuanto éstos se ponen en ejecución.

A continuación, se listan los pasos necesarios para la descarga y ejecución de Registrator:

Paso 1: Descargar la imagen de Registrator utilizando el comando docker pull de la siguiente forma:

```
docker pull gliderlabs/registrator:latest
```

Paso 2: Ejecutar la imagen de Registrator descargada previamente se utiliza el comando docker run de la siguiente forma:

```
$ docker run -d \
   --name=registrator \
   --net=host \
   --volume=/var/run/docker.sock:/tmp/docker.sock \
   gliderlabs/registrator:latest \
     consul://localhost:8500
```

En donde la etiqueta –d indica que la imagen se ejecutará en modo demonio, la etiqueta –name indica que el nombre del contenedor será Registrator, la etiqueta –net indica que se utilizará la misma red que la máquina anfitriona, la etiqueta -volume indica que se creará un volumen virtual para la carpeta indicada, y por último, se indica la imagen que se utilizará junto con la dirección donde se estará ejecutando Consul.

5.2.4. Implementación de servicios REST para la comunicación interna entre servicios

Existen diversas alternativas para la comunicación interna entre los servicios de una arquitectura basada en microservicios, entre ellas se puede encontrar los servicios REST, los sistemas de mensajería de colas, comunicación por medio de RPC, etc. Para la arquitectura propuesta en este documento se utilizarán servicios REST sobre el protocolo HTTP.

La elección de utilizar servicios REST sobre las otras alternativas, se debe a que este tipo de comunicación permite realizar peticiones de forma síncrona y/o asíncrona. Además, permite la comunicación por medio de archivos JSON, que al ser un formato estándar puede ser manejado por cualquier lenguaje de programación, permitiendo así que el sistema pueda ser construido con tecnologías heterogéneas.

Para motivos de demostración del uso de tecnologías heterogéneas, el prototipo está desarrollado utilizando dos lenguajes de programación diferentes. Por un lado, se tiene a Go para la construcción de los servicios de usuario y curso, y por otro lado, se tiene a JavaScript sobre Node.js para la construcción del API Gateway.

Para la creación de servicios REST en Go se utiliza la librería de software libre Goji, ésta permite la creación de servicios de forma fácil e intuitiva. A continuación, en la figura 11 se muestra el código necesario para la creación de un servicio REST que devuelve la frase Hola Mundo utilizando Goji.

Figura 11. **Código fuente de servicio REST "Hola Mundo" utilizando Goji**

```
1   package main
2
3   import (
4       "fmt"
5           "net/http"
6           "goji.io"
7           "goji.io/pat"
8           "golang.org/x/net/context"
9   )
10
11  func holaMundo(ctx context.Context, w http.ResponseWriter, r *http.Request) {
12          fmt.Fprintf(w, "Hola Mundo")
13  }
14
15  func main() {
16      mux := goji.NewMux()
17      mux.HandleFuncC(pat.Get("/"), holaMundo)
18
19      http.ListenAndServe("localhost:8000", mux)
20  }
```

Fuente: elaboración propia.

Como se puede observar en el código presentado en la figura 11, solo es necesario crear una instancia del multiplexor (Mux) de Goji, luego se manda a llamar la función HandleFuncC, enviado como parámetros la ruta en la cual estará disponible el servicio y la función que se ejecutará al llamar al servicio. Por último, se manda a llamar la función ListenAndServe de la librería estándar net/http de Go y se le envía como parámetro la dirección en la cual estará disponible el servicio y el mux de Goji.

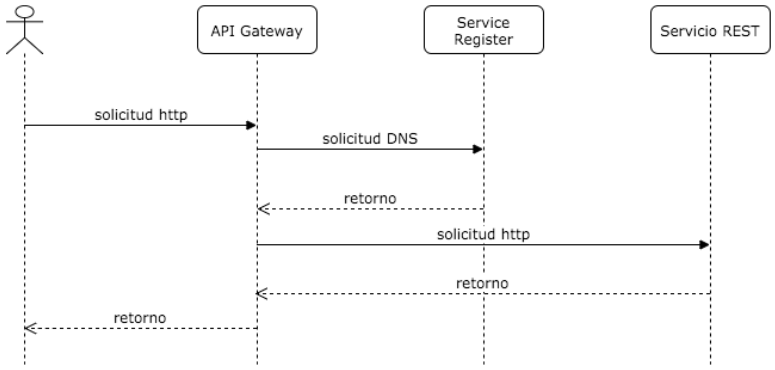
Para la creación de servicios REST del API Gateway utilizando Node.js se utiliza la librería hapi.js. Ésta al igual que Goji, permite la creación de servicios REST de forma fácil e intuitiva.

5.2.5. Implementación de un API Gateway utilizando Node.js para la exposición de servicios REST a clientes externos

A diferencia de los otros componentes arquitectónicos en los cuales se ha tomado la decisión de utilizar herramientas disponibles en el mercado, para el API Gateway se propone el desarrollo de una herramienta que satisfaga las necesidades del prototipo que se desarrolla, esto debido a que el API Gateway es quien expone los servicios a los clientes externos del sistema, por lo tanto es un componente que debe estar desarrollado a la medida ya que para integrarse exitosamente con los servicios internos deberán de realizarse ajustes específicos para cada uno de ellos.

El API Gateway dentro de la arquitectura se presenta como un servicio más, y para acceder a los servicios internos hace uso del Registrador de Servicios como se muestra en la figura 12.

Figura 12. **Diagrama de secuencia de funcionamiento de API Gateway**



Fuente: elaboración propia.

Para el descubrimiento de servicios utilizando Node.js se utiliza la librería Consul, como ejemplo del uso de esta librería, en la figura 13 se muestra el código necesario para solicitar a Consul la dirección a la cual se debe realizar la petición del listado de todos los usuarios del sistema.

Figura 13. **Código fuente en Node.js para solicitar información de servicio a Consul**

```
1   var consul = require('consul')();
2
3   consul.catalog.service.nodes('users', function(err, result) {
4     if (err) throw err;
5     //Utilizar el valor de la variable result
6     //para realizar la petición al servicio de usuarios
7   });
```

Fuente: elaboración propia.

Como se puede observar en el código anterior, se importa la librería Consul por medio de la instrucción “require”, luego se llama a la función “catalog.service.nodes” enviando como parámetros el nombre del servicio que se quiere consultar, en esta caso “users” y la función que se ejecutará al obtener un error o resultado.

Si en dado caso existiera un error, al ejecutarse la función enviada por parámetro, el error se recibirá en la variable “err”, en caso contrario, la información solicitada se recibirá en la variable “result”, en la figura 14 se muestra el contenido de la variable “result” del ejemplo anterior.

Figura 14. **Contenido en formato JSON de la variable “result”**

```
[
  {
    "Node": "node1",
    "Address": "127.0.0.1",
    "ServiceID": "users",
    "ServiceName": "users",
    "ServiceTags": [
      "dev",
      "web"
    ],
    "ServicePort": 80
  }
]

```

Fuente: elaboración propia.

Luego de obtener la información por medio de Consul, se hace uso de la funcionalidad de proxy de la librería h2o2, que forma parte de las librerías de hapi.js para redireccionar la petición al servicio requerido. En la figura 15 se muestra el código de ejemplo para la realización de un servicio HTTP utilizando hapi.js y h2o2 que permite solicitar la lista de los usuarios del sistema.

Figura 15. **Ejemplo de servicio REST utilizando h2o2 para solicitud de servicio de usuarios**

```
server.route({
    method: 'GET',
    path:'/users',
    handler: {
        proxy: {
            host: '127.0.0.1',
            port: '80',
            protocol: 'http'
        }

    }
});

```

Fuente: elaboración propia.

Como se puede observar en el código anterior, al crear una ruta del server, se coloca el objeto proxy, que contiene la información solicitada anteriormente a Consul.

5.2.6. Evaluación del rendimiento del sistema utilizando Apache JMeter

Para evaluar el rendimiento del sistema se hace uso de pruebas de estrés por medio de una aplicación de generación de carga. Además, se debe hacer uso de una herramienta que permita monitorear el desempeño de los servicios del sistema de gestión de aprendizaje. Las herramientas seleccionadas para estas tareas son Apache JMeter para la generación de carga y cadvisor de Google para el monitoreo de los recursos de los contenedores Docker.

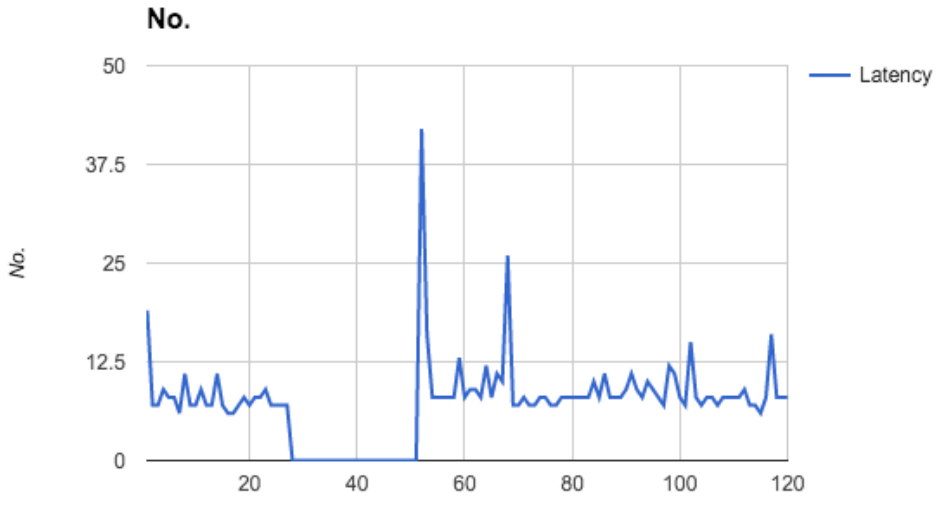
Durante las pruebas de rendimiento se ha tomado en cuenta el servicio que proporciona todos los usuarios registrados y el servicio que proporciona todos los cursos registrados. Apache JMeter se ha configurado para realizar

pruebas con 120 usuarios durante 30 segundos, esto quiere decir que habrá un promedio de 4 solicitudes por segundo, aunque cabe destacar que Apache JMeter realiza las pruebas de forma aleatoria y por lo tanto no siempre se realizarán 4 solicitudes por segundo.

5.2.6.1. Prueba de rendimiento de servicios sobre arquitectura monolítica

En la figura 16, se puede observar el gráfico generado a partir del tiempo que tarda en responder el servidor para cada una de las peticiones realizadas, durante la prueba realizada al servicio de usuarios, sobre una arquitectura monolítica. Durante esta prueba, se detiene el servicio de cursos por 4 segundo, esta acción se realiza para simular una falla o un despliegue por actualización del servicio.

Figura 16. **Gráfico de rendimiento sobre arquitectura monolítica**

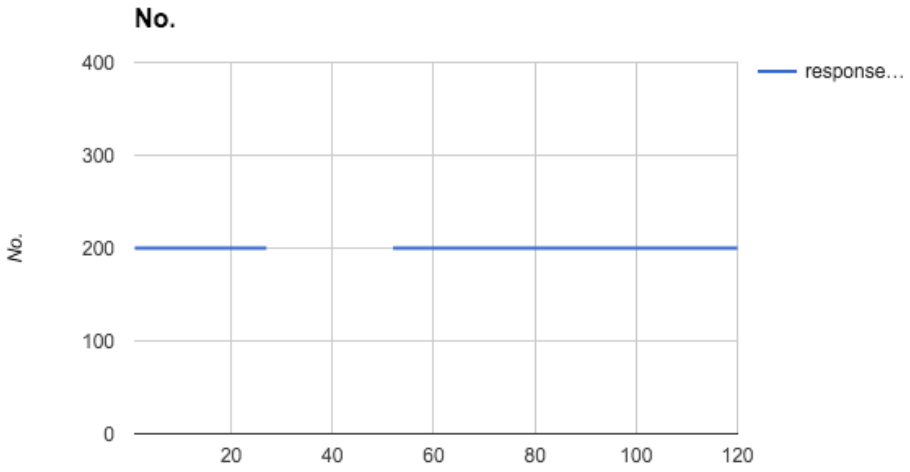


Fuente: elaboración propia.

Como se puede observar se perdieron 24 solicitudes realizadas al servicio de usuarios durante los 4 segundos que se detuvo el servicio de cursos. Además, la primera solicitud realizada después este intervalo, se encuentra por arriba del 300 % del promedio de tiempo de respuesta de las solicitudes realizadas.

Una forma de verlo de mejor forma es por medio de la figura 17, en la cual se puede observar cómo el servidor responde con código exitoso (200) a las peticiones y no responde ningún código durante el tiempo de falla.

Figura 17. **Gráfico de código de respuesta sobre una arquitectura monolítica**

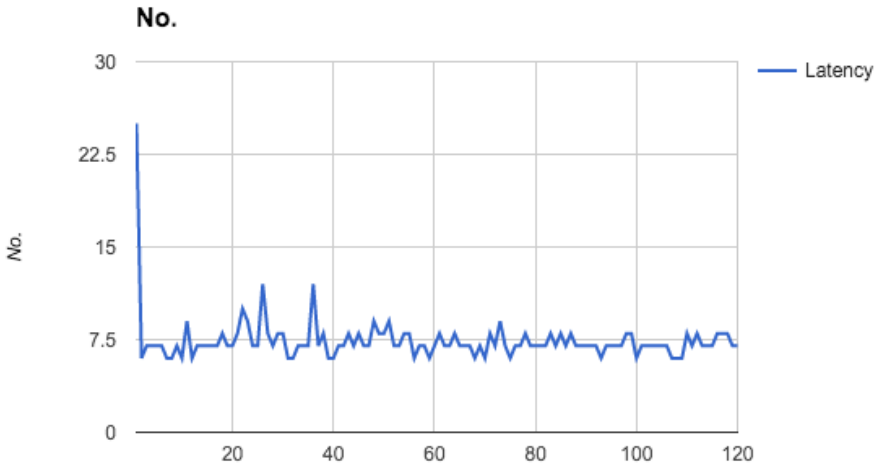


Fuente: elaboración propia.

5.2.6.2. Prueba de rendimiento de servicios sobre arquitectura basada en microservicios

A diferencia de los servicios sobre una arquitectura monolítica, en la figura 18 se puede observar el gráfico generado a partir del tiempo que tarda en responder el servidor a cada de una de las solicitudes realizadas, durante la prueba realizada al servicio de usuarios, cuando éste está implementado sobre una arquitectura basada en microservicios, y se detiene el servicio de cursos.

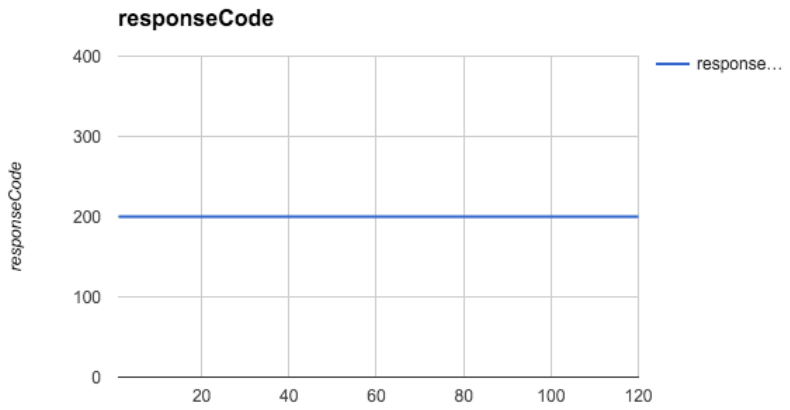
Figura 18. **Gráfico de rendimiento sobre arquitectura basada en microservicios**



Fuente: elaboración propia.

En este caso, la latencia se muestra de una forma más regular y durante el fallo no cambia de patrón. En la figura 19, se puede observar cómo el servidor responde con código 200 a todas las solicitudes realizadas durante la prueba de rendimiento.

Figura 19. **Gráfico de código de respuesta sobre una arquitectura basada en microservicios**



Fuente: elaboración propia.

5.2.6.3. Monitoreo de contenedores por medio de cadvisor

La herramienta cadvisor de Google se instala como un contenedor de software más, por medio del comando docker run, como se puede observar en la figura 20:

Figura 20. **Instalación y ejecución de cadvisor**

```
sudo docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:rw \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  google/cadvisor:latest

```

Fuente: elaboración propia.

5.2.7. Definición de requerimientos de hardware

Conforme al monitoreo realizado por medio de cadvisor a los contenedores durante las pruebas de rendimiento, se determinó que los requerimientos mínimos para el correcto funcionamiento de la arquitectura basada en microservicios son los siguientes:

Tabla VI. **Requerimientos mínimos de hardware**

Imagen y Contenedor	**CPU**	**RAM**	**Espacio en Disco**
Mysql	1 CPU	512 Mb	2 Gb
NodeJs	1 CPU	256 Mb	1.5 Gb
Go	1 CPU	64 Mb	1.5 Gb
Consul	1 CPU	64 Mb	100 Mb
Registrator	1 CPU	64 Mb	100 Mb
cadvisor	1 CPU	64 Mb	350 Mb
TOTAL	1 CPU Compartido	1024 Mb	5.55 Gb

Fuente: elaboración propia.

5.2.8. Definición de requerimientos de software

Con base en el desarrollo realizado durante el presente trabajo, en la tabla VII se pueden observar los requerimientos mínimos necesarios, para el correcto funcionamiento del prototipo del sistema de gestión de aprendizaje sobre una arquitectura basada en microservicios.

Tabla VII. **Requerimientos de software**

Software	**Versión**
Ubuntu	14.4 o superior
Docker	1.8 o superior
Go	1.6 o superior
Node.js	4.2.2 o superior
Consul	0.6 o superior
Registrator	6 o superior

Fuente: elaboración propia.

6. ANÁLISIS Y DISCUSIÓN DE RESULTADOS

6.1. Requerimientos de hardware

Como se pudo observar en la presentación de resultados, los requerimientos mínimos de hardware se obtienen a partir del monitoreo de los recursos utilizados por los contenedores sobre los cuales se encuentran los diferentes servicios del sistema, debe tomarse en cuenta que estos requerimientos mínimos satisfacen de forma correcta una concurrencia de 120 peticiones durante los 30 segundos, que fue la carga máxima que se utilizó durante las pruebas de rendimiento; por lo tanto, en caso de incrementar la concurrencia de usuarios, es necesario incrementar los requerimientos mínimos de hardware necesarios para el correcto funcionamiento del sistema de gestión de aprendizaje.

Un aspecto importante a considerar al implementar un sistema sobre una arquitectura basada en microservicios, es que este tipo de arquitectura implica requerimientos mínimos de hardware mayores a los requerimientos mínimos de hardware que usualmente son necesarios para la implementación de un sistema monolítico, esto se debe a que los servicios al ejecutarse en contenedores individuales, cada uno deberá de poseer sus propios recursos, de forma que todos los recursos necesarios para su funcionamiento se encuentren disponibles en cualquier momento, por lo tanto, estos no pueden ser compartidos con los demás contenedores, ya que si un contenedor fallara provocaría que el recurso compartido no estuviera disponible y así crear una reacción de falla en cadena.

Otro punto a tomar en cuenta, es que los componentes para el registro de servicios y la puerta de enlace API Gateway son componentes adicionales que hay que monitorear constantemente, si ocurriese una falla en estos componentes, se podría detener por completo el funcionamiento del sistema. Además, el número de saltos que se deben realizar para una simple solicitud en una arquitectura monolítica pueden llegar a ser 1 o 2 saltos como máximo, mientras que en una arquitectura basada en microservicios al menos se deben ejecutar los saltos adicionales hacia la puerta de enlace API Gateway y al registrador de servicios, provocando que como mínimo se tengan que realizar de 4 a 6 saltos entre aplicaciones, para retornar la información solicitada.

6.2. Requerimientos de software

Tomando en cuenta el software sobre el cual se construyó el prototipo, todos los componentes utilizados para la implementación del sistema de gestión de aprendizaje con características de red social, sobre una arquitectura basada en microservicios, se pueden encontrar en forma de software libre y están disponibles para su descarga de forma gratuita.

En el caso de la implementación de los contenedores de software, se pudo observar que Docker permitió el despliegue de instancias de los servicios que componen al sistema de una forma aislada, lo cual permitió mantener una disponibilidad del 100 % del servicio de usuarios, en el tiempo en el cual se detuvo el servicio de cursos durante las pruebas de rendimiento.

Un aspecto a tomar en cuenta, es que, aunque se pudo observar cómo el uso de Docker benefició a la implementación del sistema, cabe destacar que en casos, en los cuales el sistema es de mayor tamaño, sería conveniente el uso de un orquestador de contenedores como Docker Swarm o Kubernetes de Google,

ya que estos permiten que muchas de las tareas que se han realizado manualmente durante el desarrollo del prototipo, puedan realizarse de forma automática. Además, estas herramientas permiten el realizar clúster de servidores que permiten realizar réplicas de contenedores y así asegurar más disponibilidad.

En el presente trabajo, no se han tomado en cuenta estas herramientas, pues requieren de muchos recursos y por lo tanto, el costo de mantener el sistema se elevaría en el apartado de infraestructura, además el manejo de estas herramientas requiere de más tiempo para su implementación y mantenimiento, y tomando en cuenta que solamente se ha trabajado un prototipo que además no incluye todas las funcionalidades del sistema de gestión de aprendizaje, estos recursos se estarían desperdiciando, pues aunque no se utilicen éstas herramientas, la arquitectura cumple con los objetivos planteados al inicio del trabajo.

Otro aspecto que no se ha tomado en cuenta durante el desarrollo de este trabajo, es la forma en que se realizó la puesta en marcha inicial de los contenedores sobre los cuales se implementó el sistema, esta acción pudo haberse realizado de forma automatizada y con una simple configuración utilizando la herramienta Docker Compose, la cual permite que en base a un archivo de configuración se puedan crear, ejecutar y enlazar un grupo de contenedores Docker, pero con el objetivo de mostrar cómo utilizar Docker, se prefirió realizar estas tareas de forma manual.

La instalación de los componentes para el registro y descubrimiento de servicios, Consul y Registrator, se realizó utilizando imágenes Docker, Consul por su parte permitió el uso de descubrimiento por medio de DNS y por HTTP, mientras que Registrator permitió registrar los servicios automáticamente,

cuando éstos se pusieron en ejecución sobre los contenedores de Docker. Registrator además permitió dar de baja a los servicios en Consul, cuando los contenedores sobre los cuales se ejecutaban se detenían ya fuera por instrucción ejecutada desde la línea de comandos o por una falla en el contenedor.

Una ventaja de utilizar Consul que no fue aprovechada por la arquitectura propuesta, fue el registro de una dirección de chequeo de salud para los servicios, esta funcionalidad permite que se pueda verificar el estado de los servicios desde la interfaz de usuario de Consul y así detectar posibles problemas con los servicios, antes que estos se conviertan en problemas reales.

En cuanto a la comunicación entre los servicios internos, estos fueron implementados bajo REST, utilizando objetos JSON, esto permitió que las solicitudes entre servicios y sus respuestas fueran ligeras y que no fuera necesario la instalación de un componente extra como un servidor de cola de mensajes o una base de datos NoSQL como Redis.

La ventaja de utilizar el formato JSON es que posee una estructura más simple que otros formatos como XML, y existen librerías para utilizar este formato para la mayoría de lenguajes existentes, lo cual permitió que se pudieran utilizar tecnologías heterogéneas, esto quiere decir que se desarrolló el sistema en diferentes lenguajes de programación y no hubo problemas de compatibilidad.

Por otro lado, para las tecnologías heterogéneas, además del uso de REST, también se tomó en cuenta el beneficio de utilizar los contenedores Docker, pues con ello no es necesario que se instalen sus compiladores o intérpretes en el servidor, solamente es necesario contar con el código fuente, que puede ser compilado y ejecutado o interpretado dentro de un contenedor Docker, esto quiere decir, aunque esto implica que es necesario contar con un contenedor que

permita el funcionamiento en el lenguaje de programación seleccionado, pero, para ello se puede descargar cualquier imagen disponible en Docker Hub o se puede crear una imagen propia a partir de una ya existente.

Por último, el uso de Node.js y en especial de la librería hapi.js con el plugin h2o2 permitió la creación de un API Gateway, al cual pueden realizar sus solicitudes los clientes externos del sistema y éste solicitar a los servicios internos por los datos o acciones correspondientes. Se pudo verificar que con el uso de Node.js se consigue ahorrar recursos de procesamiento, pues su modelo de no bloqueo de entrada y salida permite que se libere el hilo de procesamiento mientras se envía la solicitud a los servicios y se vuelve a tomar en cuanto estos realizar el retorno.

6.3. Pruebas de rendimiento

Con base a las pruebas de rendimiento realizadas, se pudo observar la diferencia en el comportamiento del prototipo del sistema de gestión de aprendizaje, cuando fue implementado sobre una arquitectura monolítica y sobre una arquitectura basada en microservicios.

Para observar estas diferencias de comportamiento, primero se realizó la prueba en el sistema sobre una arquitectura monolítica, y como resultado en la Ilustración 16, se pudo observar, que existió un intervalo de tiempo durante el cual la latencia para obtener una respuesta del servicio de usuarios es igual a 0, esto se debió a los 4 segundos durante los cuales el servicio de cursos no se encontraba disponible, lo cual provocó que todo el sistema de gestión de aprendizaje se detuviera incluyendo al servicio de usuarios.

Esto se pudo entender de mejor forma al ver la figura 17, en la cual se pudo observar cómo, en el mismo intervalo de solicitudes, el servidor no retornó ninguna respuesta, mientras que, para el resto de solicitudes, el servidor retornó el código de solicitud exitosa (200).

A diferencia de los resultados mostrados sobre la arquitectura monolítica, cuando la prueba de rendimiento fue realizada sobre la arquitectura basada en microservicios, los resultados mostrados con la figura 18 y figura 19, se pudo observar cómo el servicio se comportó de una forma tolerante a fallos, ya que retornó los datos de forma exitosa a todas las solicitudes que fueron realizadas sobre él, no importando si otro servicio se encontraba detenido por una actualización o por un fallo en el mismo momento.

Con esto se demuestra que la implementación de una arquitectura basada en microservicios, permite construir sistemas escalables y tolerantes a fallos, esto quiere decir que, si un componente se detiene para realizar una actualización o por una falla, los demás componentes continuarán con su funcionamiento normal.

6.4. Impactos económicos, tecnológicos y sociales

6.4.1. Impacto económico

Los requerimientos de hardware y software definidos a partir de las pruebas de rendimiento realizadas durante la construcción del presente trabajo, demuestran cómo impacta el uso de una arquitectura basada en microservicios en el presupuesto de un proyecto, ya que a pesar que los requerimientos mínimos obligan a contar con un servidor dedicado al proyecto, cuando exista más carga sobre el sistema o se incluyan nuevos módulos, no se obtendrá un impacto

negativo en el rendimiento del sistema, por lo cual no se incrementará el costo en infraestructura.

Además, si es necesario escalar sobre un servicio aislado, no es necesario duplicar todo el sistema, por lo cual también se reducen los costos de infraestructura.

Otro impacto económico que se puede observar es al momento de realizar cambios, o implementar nuevos módulos en el sistema, pues el programador que esté a cargo del desarrollo no tiene necesidad de conocer todos los módulos y por lo tanto, los cambios o actualizaciones pueden desarrollarse de forma más rápida y con menos probabilidad de error, reduciendo así los costos de recurso humano.

6.4.2. Impacto tecnológico

En el ámbito tecnológico, una arquitectura basada en microservicios para un sistema de gestión de aprendizaje con características de red social, demuestra cómo este tipo de arquitecturas puede ser utilizada para cualquier tipo de sistema, no importando la lógica de negocio del sistema, ni el tamaño del mismo.

Por otro lado, se demuestra cómo el uso de los patrones arquitectónicos de instancias de servicio por contenedor, el registrador de servicios y el API Gateway proveen alta disponibilidad, resiliencia y heterogeneidad de tecnologías, características que hacen falta en las arquitecturas planteadas por los sistemas de gestión de aprendizaje más utilizados hasta el año 2016.

6.4.3. Impacto social

Una arquitectura basada en microservicios para un sistema de gestión de aprendizaje con características de red social, impacta de forma positiva en el ámbito social, como resultado se obtiene una alternativa a tomar en cuenta para los colegios, universidades o cualquier otra institución que tenga la necesidad de implementar un sistema en línea para gestionar sus cursos o capacitaciones.

Este tipo de arquitectura apoya la continuidad de los procesos al proveer alta disponibilidad y la posibilidad de realizar actualizaciones sin poner el sistema en estado inactivo.

CONCLUSIONES

1. Se implementó el prototipo de un sistema de gestión de aprendizaje con características de red social sobre una arquitectura basada en microservicios, que permite escalabilidad y una disponibilidad del 100 % durante inactividad o fallos de otros servicios.

2. Se implementó un patrón de instancias de servicio por contenedor utilizando Docker, que permitió reducir el tiempo de inactividad de los servicios durante un fallo o nuevo despliegue del 20 % a 0 %, eliminando por completo el tiempo de inactividad.

3. Se implementó un patrón arquitectónico de registro de servicios por medio de Consul y Registrator, que permite dar de alta y de baja de forma automática, a las instancias de los servicios del sistema.

4. Se implementaron servicios REST sobre el protocolo HTTP para la comunicación entre los diferentes servicios del sistema, estos permiten el uso de diferentes lenguajes de programación y además al utilizar JSON, se logró que las solicitudes y los retornos de los servicios fueran ligeros.

5. Se implementó una puerta de enlace API Gateway utilizando Node.js, hapi.js y h2o2, esto permitió crear un proxy para manejar el acceso a los servicios del sistema por parte de clientes externos, y que se ahorren recursos por su modelo de no bloqueo de entrada y salida.

RECOMENDACIONES

1. Según se pudo observar durante la realización de este trabajo, es necesario realizar previamente al desarrollo, un análisis profundo para determinar si es necesario implementar el sistema sobre una arquitectura basada en microservicios, para sistemas muy pequeños, se puede convertir en algo muy complejo el manejo de la infraestructura.

2. Para la creación y ejecución de contenedores Docker, se recomienda el uso de Docker Compose, esto permite estandarizar los procesos anteriormente mencionados. Además, permite crear configuraciones que pueden ser reutilizadas para crear los diferentes ambientes de desarrollo, pruebas y producción.

3. Se sugiere el uso de herramientas para el monitoreo de los recursos utilizados por los contendores de software, para el caso de contenedores Docker, se aconseja el uso de la herramienta software libre cadvisor de Google, la cual se encuentra disponible de forma gratuita y permite el monitoreo de espacio de disco, uso de CPU y de memoria RAM.

4. Por último, se recomienda el análisis de otras herramientas como Docker Swarm o Kubernetes de Google, que en desarrollo de sistemas muy grandes, permiten realizar de forma automática muchas de las tareas presentes en este trabajo, pero que en sistemas de menos alcance, complican el manejo de la infraestructura.

REFERENCIAS BIBLIOGRÁFICAS

1. Balalaie, A., Heydarnoori, A. y Jamshidi, P. (2015). *Migrating to Cloud-Native Architectures Using Microservices: An Experience Report.* Sharif University of Technology, Tehran, Iran. Department of Computing, Imperial College London, UK. Recuperado de http://arxiv.org/pdf/1507.08217v1.pdf

2. Cegarra Sánchez, J. (2004). *Metodología de la investigación científica y tecnológica*. Madrid España. Díaz de Santos.

3. Da Silva, M. D., Lopez Tavares, H. (2015). *Redis Essentials* (pp. 1). Birmingham, Reino Unido. Pack Publishing.

4. Doglio, F. (2015). *Pro REST API Development with Node.js* (pp. 1-8). La Paz, Canelones, Uruguay. Apress.

5. Holla, S. (2015*). Orchestrating Docker* (pp. 103-105). Bangalore, India. Pack Publishing.

6. Kolesnikov-Jessop, S. (2012). *Schools Go Into the 'Cloud' to Embrace the Popularity of Social Media*. The New York Times. Recuperado de http://www.nytimes.com/2012/05/21/world/asia/schools-go-into-the-cloud-to-embrace-the-popularity-of-social-media.html

7. Kusnetzky, D. (2011). *Virtualization: A manager's guide* (pp. 1-15). Estados Unidos. O'Reilly.

8. Lewis, J., Fowler, M. (2014). *Microservices*. Chicago, Estados Unidos. Martin Fowloer. Recuperado de http://martinfowler.com/articles/microservices.html

9. Maurer, B. (2015). *FAIL as Scale. Queue*. Recuperado de http://dl.acm.org/citation.cfm?id=2839461

10. Mocevicius, R. (2015). *CoreOS Essentials* (pp. 91). Estados Unidos. Pack Publishing.

11. Mouat, A. (2016). *Using Docker* (pp. 3-4). Sebastopol, Estados Unidos. O'Reilly.

12. Newman, S. (2015). *Building Microservices*. Estados Unidos. O'Reilly.

13. Node.js Foundation. (s.f.). *Node.js*. Recuperado de https://nodejs.org/en/

14. Raj, P., Chelladhurai, J. S., Singh, V. (2015). *Learning Docker*. Bangalore, India, Birmingham, Reino Unido. Pack Publishing.

15. Richardson, C. (2014). Microservices: *Decomposing Applications for Deployability and Scalability*. InfoQ. Recuperado de http://www.infoq.com/articles/microservices-intro

16. Subramanian, P., Zainuddin, N., Alatawi, S., Javabdeh, T. y Hussin, A. R. C. (2014). *A Study of Comparison between Moodle and Blackboard based on Case Studies for Better LMS*. Journal Of Information Systems Research And Innovation. Recuperado de

http://seminar.utmspace.edu.my/jisri/download/F_Vol6Feb2014_FullPaper/Pub4_ComparisonBetweenMoodleAndBlackboard.pdf

17. Tsvetinov, N. (2015). *Learning Reactive Programming with Java 8* (pp. 2). Birmingham, Reino Unido. Pack Publishing.

18. Watson, W. R., Watson, S. L. (2007). *An Argument for Clarity: What are Learning Management Systems, What are They Not, and What Should They Become?.* TechTrends (pp. 28-34). Universidad de Indiana, Universidad Purdue, Indiana, Estados Unidos. Recuperado de https://halshs.archives-ouvertes.fr/hal-00692067/document