Revista Industrial Data 23(2): 141-149 (2020)
DOI: http://dx.doi.org/10.15381/idata.v23i2.17278

ISSN: 1560-9146 (Impreso) / ISSN: 1810-9993 (Electrónico)
Facultad de Ingeniería Industrial - UNMSM

# Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua

ZORAIDA MAMANI RODRÍGUEZ [1]
LUZ DEL PINO RODRÍGUEZ [2]
JUAN CARLOS GONZALES SUAREZ [3]

RECIBIDO: 13/01/2020 ACEPTADO: 10/08/2020 PUBLICADO: 31/12/2020

**RESUMEN**

Los microservicios se conciben como un estilo arquitectónico enfocado a desarrollar aplicaciones mediante un conjunto de servicios, independientes, escalables, colaborativos, evolutivos, capaces de autoadaptarse a ecosistemas complejos. Por otro lado, DevOps es un paradigma que utiliza un conjunto de principios enfocado en la entrega e integración continua de software, esto implica una nueva cultura para desarrollar y desplegar software en contextos altamente colaborativos y agiles orientados a reducir la brecha que existe entre el desarrollo y las operaciones. Es en este contexto que el presente trabajo propone una Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua y aplica la propuesta mediante un caso de estudio con la participación de equipos de desarrollo conformados por los estudiantes de los cursos de Taller de Construccion de Software y de Sistemas de los semestre académicos: 2018-1, 2018-2, 2019-1, 2019-2 y liderados por los autores de la presente investigación; como resultado de valor se tiene un producto de software constituido por un conjunto de Apps implementado con tecnologías stack lideres bajo un enfoque disruptivo.

**Palabras clave:** Microservicios; Devops; Tecnologías disruptivas.

**ARCHITECTURE BASED ON MICROSERVICES AND DEVOPS FOR CONTINUOUS SOFTWARE ENGINEERING**

**ABSTRACT**

Microservices are conceived as an architectural style focused on developing applications through a set of services, independent, scalable, collaborative, evolutionary, capable of adapting to complex ecosystems. On the other hand, DevOps is a paradigm that uses a set of principles focused on the continuous delivery and integration of software, this implies a new culture to develop and deploy software in highly collaborative and agile contexts aimed at reducing the gap between development and operations. It is in this context that the present work proposes an Architecture based on Microservices and DevOps for continuous software engineering and applies the proposal through a case study with the participation of development teams formed by the students of the Workshop courses of Software and Systems Construction of the academic semesters: 2018-1, 2018-2, 2019-1, 2019-2 and led by the authors of this research; As a result of value there is a software product consisting of a set of Apps implemented with leading stack technologies under a disruptive approach.

**Keywords:** Microservices; Devops; Disruptive Technologies.

## I. INTRODUCCIÓN

Los microservicios se conciben como un estilo arquitectónico enfocado en desarrollar una aplicación mediante un conjunto de servicios, independientes, escalables, colaborativos, evolutivos, capaces de autoadaptarse a ecosistemas complejos. (Richardson, 2019; Bandeira y otros, 2019). Los microservicios surgen con la finalidad de resolver las limitaciones que presentaban los sistemas monolíticos, los cuales se caracterizan por mantener los servicios como una unidad lógica, utilizando los mismos recursos computacionales. Un gran porcentaje de sistemas de información en el mundo se encuentran construidos bajo este enfoque monolítico y en vista de las restricciones que mantiene para evolucionar, conlleva a que las organizaciones apunten a migrar sus ecosistemas a un enfoque moderno, evolutivo e infraestructuras en la nube. REST es el acrónimo de "REpresentational State Transfer", en la Figura 1 se presenta el funcionamiento de REST, 1: el cliente solicita una representacion al servidor, 2: el servicio lanza la petición a la Base de Datos, 3: la Base de Datos retorna la información solicitada, 4: el servicio envia la representación en formato JavaScript Object Notation (JSON) al cliente; el proceso que realiza el servidor es totalmente transparente al cliente. (Kenneth, 2016).

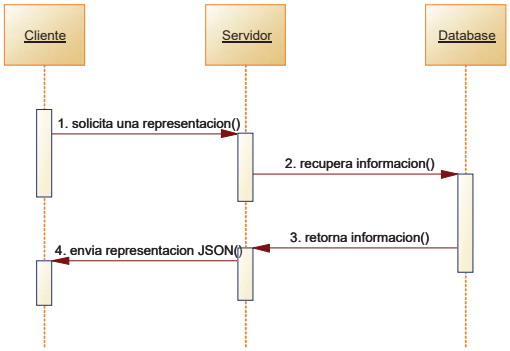


***Figura 1.*** Funcionamiento de REST

Fuente: adaptado de Lange (2016)

1 Docente Asociada de la Facultad de Ingeniería de Sistemas e Informatica de la UNMSM, Lima, Perú. Coordinadora del Grupo de Investigacion *Ingenieria Web*
ORCID: https://orcid.org/0000-0002-2590-8387
E-mail: zmamanir@unmsm.edu.pe

2 Docente Asociada de la Facultad de Ingeniería de Sistemas e Informatica de la UNMSM, Lima, Perú.
ORCID: https://orcid.org/0000-0002-2893-8047
E-mail: ldelpinor@unmsm.edu.pe

3 Docente Principal de la Facultad de Ingeniería de Sistemas e Informatica de la UNMSM, Lima, Perú. Coordinador del Grupo de Investigacion *Investigacion y Desarrollo de Videojuegos Innovadores*
ORCID: https://orcid.org/0000-0002-2309-9133
E-mail: jgonzaless@unmsm.edu.pe

JSON es el formato estándar para transferir la representación del estado en REST, su notación comprende el par: clave-valor, como se puede apreciar en la Figura 2.

```
{
    "codAlumno": "18207001",
    "apePaterno": "AGUILAR",
}
```

***Figura 2.*** Mensaje JSON
Fuente: Elaboracion Propia

RESTful es el acrónimo que se asigna a las API que satisfacen seis restricciones REST como: 1. Client Server: el sistema debe ser realizado por clientes y servidores; los servidores deben tener los recursos que los clientes necesitan, este tipo de separación permite que se pueda contar con distintos tipos de clientes como portales web, aplicaciones mobiles, motores bpm, chatbots entre otros. La separación reduce la complejidad en el servidor y mejora la escalabilidad del cliente. RESTful permite el uso de otros protocolos como FTP, complementariamente al protocolo HTTP. 2. Stateless se refiere a que toda la información sobre la sesión del cliente se mantiene en el cliente y el servidor no tiene que realizar un seguimiento de las sesiones del cliente, esto le permite liberar los recursos en cuanto las solicitudes hayan sido atendidas lo que incrementa el desempeño.3.Cache: su efectividad se asocia a un numero reducido de interacciones cliente-servidor.4.Interface Uniforme: El desacoplamiento entre interfaces y sus implementaciones permite que las interacciones sean simples, esta restriccion se subdivide en cuatro subrestricciones importantes: 4.1.Identificacion de Recursos: Un recurso tiene nombre, un identificador único y un único Identificador de Recurso Universal (URI), la granularidad puede ser muy fina como una entidad del dominio del negocio; o muy gruesa como un proceso de negocio.4.2.Manipulacion de Recursos a través de sus representaciones: Esta restricción consiste en que el servidor debe exponer una representación del estado del recurso mediante formato JSON en el cuerpo de los requerimientos y respuestas HTTP, como por ejemplo en la Figura 3 se puede apreciar un ejemplo de la representación del estado del recurso: alumnoprograma/18207001.

```
{
        "codAlumno": "18207001",
        "apePaterno": "AGUILAR",
        "apeMaterno": "ROMERO",
        "nomAlumno": "JHON HAMILTON",
        "nom_programa": "DOCTORADO EN INGENIERIA DE SISTEMAS E
INFORMATICA"
}
```

***Figura 3.*** Representación del Estado del Recurso
Fuente: Elaboracion Propia

4.3. Mensajes autodescriptivos: Esto significa que cada mensaje de solicitud o respuesta debe incluir suficiente información para que el receptor lo comprenda, entre los métodos HTTP formales a utilizar se tienen: GET, POST, PUT, DELETE. En la Tabla 1 se expone los métodos, las tareas que realizan y la sintaxis para su procesamiento.

**Tabla 1.** *Métodos HTTP*

| Tarea | Método | Ruta |
|---|---|---|
| Crear un alumno | POST | /alumnos |
| Eliminar un alumno | DELETE | /alumnos/{id} |
| Obtener un alumno especifico | GET | /alumnos/{id} |
| Buscar Alumnos | GET | /alumnos |
| Actualizar un Alumno | PUT | /alumnos/{id} |

Fuente: adaptado de Kenneth (2016)

4.4. Hipermedia como el motor del estado de la aplicación: Esta restricción establece el uso de hipermedia como por ejemplo los enlaces para navegar a través de la aplicación. En la Figura 4 se observa que a la representación del estado del recurso: alumnoprograma, se le ha incorporado hipermedia a través de enlaces hacia la representación de los recursos: alumno y pagos respectivamente.

```
{
        "codAlumno": "18207001",
        "apePaterno": "AGUILAR",
        "apeMaterno": "ROMERO",
        "nomAlumno": "JHON HAMILTON",
"_links": {
"self": {
"href": "https://sigapdev-recibos-back.herokuapp.com/alumnos/18207001
},
"pagos": {
"href": "https://sigapdev-recibos-back.herokuapp.com/pagos/18207001"
}
}
}
```

***Figura 4.*** Representación del recurso alumnos con enlace a sus pagos
Fuente: Elaboracion Propia

5. Sistema de Capas: La restricción señala que el cliente solo debe conocer la capa inmediata con la cual interactúa con el servidor, más debe desconocer otras capas que pudieran existir, esta característica permite dotar de seguridad a la comunicación. 6. Código bajo demanda: Es una restricción opcional y consiste en la capacidad del servidor de poder extender la funcionalidad del cliente, enviándole el código a ejecutar, esta característica es similar al funcionamiento de las páginas web, cuando se invoca una página particular, el browser podría

considerar necesario descargar ciertos plugins para su ejecución adecuada.

"Un patron es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos" (Larman, 1999). El patron arquitectura de microservicios cumple la nocion *separately deployed units* esto refiere a que cada componente de la arquitectura se implementa como una unidad separada, lo que permite una implementación mas sencilla con entrega efectiva y optimizada, mayor escalabilidad y desacoplamiento de componentes en la aplicación (Richards, 2015)

Richards (2015) señala que el patron arquitectura de microservicios puede ser implementada mediante el uso de una topología o estilo de implementación, usa dos conceptos clave: componente de servicio y arquitectura distribuida, hace uso de un protocolo de acceso remoto como: JMS, AMQP, REST, SOAP, RMI u otro. Se considera que esta arquitectura ha evolucionado a partir de otras existentes como: aplicaciones monolíticas desarrolladas utilizando el patron de arquitectura en capas y aplicaciones distribuidas desarrolladas a través del patron de arquitectura orientada a servicios (SOA), mientras que las aplicaciones monoliticas implementadas bajo el patron n capas hacían uso de componentes estrechamente acoplados lo que hacia engorroso el mantenimiento, pruebas y escalabilidad de la aplicación, mientras que las aplicaciones desarrolladas bajo el patron SOA ofrece niveles de abstracción, concetividad heterogenea, orquestación de servicios y el alineamiento de TI con los objetivos de negocios, sin embargo es complejo, costoso, ubicuo, difícil de entender e implementar para la mayoría de las aplicaciones. La entrega continua, agilidad del despliegue de aplicaciones son características de su evolución, el patron separa la aplicación en multiples unidades desplegables que pueden ser desarrolladas individualmente, probadas y desplegadas independientemente de otros componenentes de servicios (p. 29).

Actualmente se cuenta con investigaciones relacionadas a esta línea de investigación como el trabajo de López y Spillner (2017) quien propone un método matemático para cuantificar la escalabilidad elástica horizontal de los microservicios. En Richardson y Smith (2016) se desarrolla el proceso de llevar una aplicación monolítica a una aplicación compuesta por microservicios, demuestra el proceso con un caso de estudio basado en un sistema de gestión de taxis utilizando APIs REST. Salza y otros (2017) presentan cCube, una arquitectura de código abierto basado en microservicios utilizado para crear una aplicación de uno o mas algoritmos de clasificación de machine learning evolutivo que puede ser desplegado en la nube con factorización de datos automatico, entrenamiento y filtrado de resultados. Lenarduzzi y Sievi-Korte (2018) presentan un sistema basado en microservicios para seleccionar componentes de Software, los investigadores consideran que la selección de componentes es una de las actividades más importantes en un proceso de desarrollo del sistema y la elección de los correctos es uno de los factores clave. Lenarduzzi, Lomio, Saarimäki y Taibi (2020) realizan una investigacion para determinar si la migración de un sistema monolítico a microservicios disminuye el costo tecnico, el caso de estudio se basó en migrar un proyecto de doce años de vida, se extrajo cinco procesos de negocios del sistema monolitico y se migró a microservicios, analizaron el costo tecnico con SonarQube, complementaron con un estudio cualitativo con los miembros de la compañía para comprender la calidad percibida del sistema; concluye el estudio señalando que la migracion a microservicios ayudo a reducir el costo tecnico a largo plazo. Abdullaha, Iqbal y Erradi ( 2019) proponen un método para descomponer automáticamente una aplicación monolítica en microservicios para mejorar la escalabilidad y el rendimiento de la aplicación. El método utiliza los registros de acceso a la aplicación y un método de aprendizaje automático no supervisado para descomponer automáticamente la aplicación en microservicios asignados a particiones URL que tienen un rendimiento y requisitos de recursos similares.

Por otro lado, DevOps es un paradigma que utiliza un conjunto de principios enfocado en la entrega e integración continua de software, esto implica una nueva cultura para desarrollar y desplegar software en contextos altamente colaborativos y agiles, este paradigma involucra a todos los actores como desarrolladores, profesionales de TI, usuarios, stakeholders orientado a reducir la brecha que existe entre el desarrollo y operaciones (Dittrich y otros, 2018; Jabbari y otros, 2016).

En la Figura N° 5, se grafica el conglomerado de palabras vinculados de alguna manera con las definiciones de Devops y otras áreas del cuerpo del conocimiento de la ingeniería de software como: requisitos, diseño, construcción, pruebas, mantenimiento, gestión de la configuración, calidad, gestión de proyectos, metodologías agiles como: scrum, programación extrema; Cloud Computing e Information Technology Infrastructure Library (ITIL), según lo señala Jabbari y otros (2016).

La adopcion de tecnicas agiles cambio la forma de desarrollar software, permiten percibir resultados de

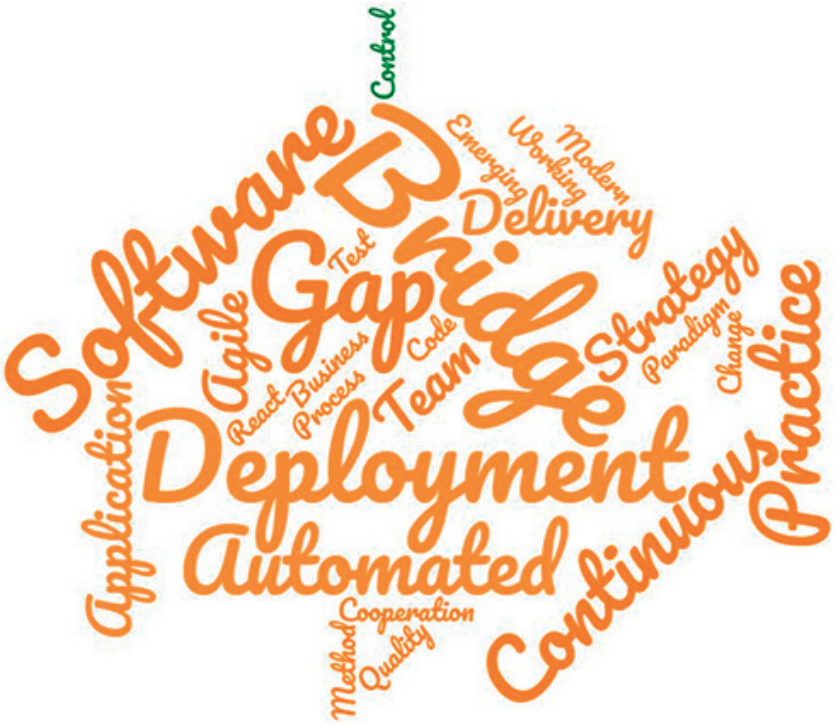


***Figura 5.*** Nube de palabras de términos comunes utilizados en las definiciones de DevOps
Fuente: adaptado de Jabbari y otros (2016)

valor tempranamente en contraste con la tecnicas tradicionales, el ciclo de vida agil comprende ciclos acelerados, la entrega continua se ha convertido en la norma, esta incrementa la tasa de cambios en produccion impactando negativamente a los entornos de produccion si no gestionan el cambio con la misma velocidad que desarrollo. La naturaleza de las aplicaciones de hoy son mucho mas complejas, pueden estar desplegadas en una infraestrutura diversa, pueden abarcar varios idiomas, plataformas y elementos de software mas granulares lo que impide aplicar las parcticas actuales de la gestion de incidentes o gestion de servicios de TI. DevOps aborda las complejidades asociada con el soporte de los cambios dinámicos actuales y ecosistemas de software críticos para el negocio (Craig, Sturm y Pollard, 2017).

La industria del software evoluciona de manera continua, nuevos enfoques disruptivos para su construccion se exhiben en la literatura, las grandes marcas de software las difunden y es responsabilidad de la academia formar recursos humanos con competencias y capacidades que les permitan asumir los desafíos de la industria del software, abordando exitosamente problemas complejos; es en ese contexto que la presente investigación, establece como objetivos: 1) Diseñar una Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua 2) Implementar la Arquitectura propuesta en el proyecto SIGAP 3).Desplegar el software en plataformas Cloud 4) Evaluar los Resultados. El principal aporte de la investigación se centra en la integración de tecnologías emergentes, con capacidad evolutiva como el patron arquitectura de microservicios y devops para el desarrollo colaborativo de un conjunto de aplicaciones escalables, con recursos humanos en proceso formativo, agiles, rotativos, altamente motivados y enfocados en la automatización de procesos de negocios y su aplicación al proyecto SIGAP, este ecosistema de componentes entre tecnológicos, humanos, arquitecturas, bigdata y procesos colaborativos lo exhibe como novedoso en la industria de software peruano y su relevancia para la revista.

## II. METODOLOGIA

En la seccion anterior se ha revisado los fundamentos de microservicios y devops, asimismo se ha identificado recientes antecedentes de investigación sobre las variables de estudio, en esta seccion desarrollaremos la metodología que amerita aplicar, la cual se organiza en:

A. Diseño de la Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua

B. Implementacion de la Arquitectura mediante un Caso de Estudio

C. Resultados de la propuesta

## A. Diseño de la Arquitectura basada en Microservicios y DevOps para ingeniería de software continua

En la Figura 6 se expone la propuesta de un Diseño de Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua, como se puede apreciar hay clara separacion del entorno de desarrollo y de operaciones o produccion, ambas sostenidas por infraestructuras en la nube y devops como una perspectiva transversal que asocia ambos espacios para reducir la brecha entre desarrollo y operaciones mediante la entrega e integracion continua de software, los componentes de la Arquitectura se describen a continuación:

**A nivel de Operaciones:**

**Capa de Datos:** Esta capa contienen la información corporativa en repositorios de datos relacionales o no relacionales, storage para alojamientos de archivos y/o documentos entre otro tipo de contenido.

**Capa de Microservicios**: Esta capa esquematiza el conjunto de de APIs que comprenden el backend o conjuntos de servicios RESTful identificados para brindar soporte a las peticiones procedentes del lado del cliente.

**Capa de Presentación**: Esta capa contiene el conjunto de Apps que comprenden el Frontend con las cuales interactúan los usuarios para el uso de las funcionalidades ofrecidas por las Apps.

El nivel de Desarrollo es una replica aproximada del ambiente de Operaciones o Produccion, se dice que es aproximada debido a se encuentra en constante proceso de elaboración de nuevos productos o servicios o mejoras de los que ya residen en producción.

**Devops:** Es una capa transversal a Desarrollo y Operaciones, contiene el conjunto de principios, y cultura organizacional que impulsa la entrega e integración continua del software, valora el esfuerzo

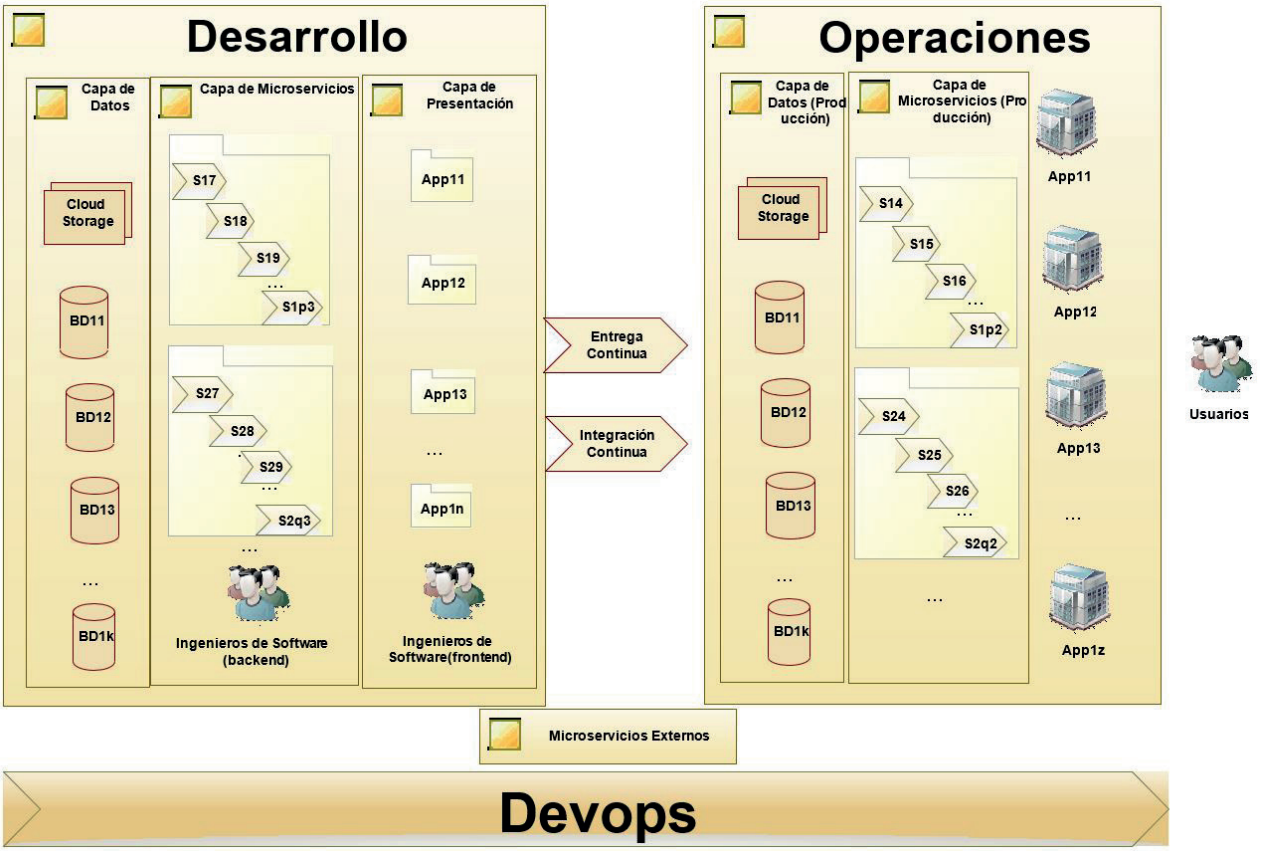


***Figura 6.*** Diseño de la Arquitectura basada en Microservicios y DevOps para ingeniería de software continua (AMDISC)
Fuente: Elaboracion Propia

realizado por los recursos humanos, automatiza los procesos, promueve el trabajo en equipo disciplinado, ordenado, responsable con lo que se reduce la brecha entre estos dos entornos e incrementa la calidad del software. (Anubhav, 2014; Daniels y Davis, 2018; Gheorghiu y otros, 2019)

### B. Implementación de la Arquitectura mediante un Caso de Estudio

El caso de estudio para la implementación de la propuesta se circunscribe al Sistema Integrado de Gestión Académica Administrativa (SIGAP) que contempla cuatro módulos funcionales como: Modulo de Control de Pagos, Modulo de Consulta de Pagos, Modulo de Carga de Recaudaciones, Modulo de Reportes y Estadísticas, Modulo de Legajo Docente. El proyecto SIGAP utiliza el modelo de arquitectura basada en microservicios propuesto y la cultura devops para la entrega e integración continua del software, lo que facilita su escalabilidad. Durante su construcción se ha utilizado el estilo arquitectónico basado en capas con bajo acoplamiento, se tiene una clara separación entre el cliente y el servidor, la implementación de los servicios respeta la filosofía RESTful.

Se aplico devops enfocado en una cultura colaborativa y comprometida por todos los actores del proyecto. Los equipos de desarrolladores estuvieron conformados por los estudiantes de los cursos de Taller de Construcción de Software y Taller de Construcción de Sistemas de las Escuelas Profesionales de Ingenieria de Software e Ingenieria de Sistemas de los semestres académicos 2018-1, 2018-2, 2019-1 y 2019-2 liderados por los autores de la presente investigación.

La filosofía de trabajo consistió en que los equipos debían ser pequeños en no mas de cuatro integrantes, quienes se asignarían un nombre que los identifique durante todo el proceso y se constituya en su marca que los motive para el logro de los sprints, un sprint refiere a un conjunto de tareas puntuales asociadas a un requisito de software que puede ser funcional o no funcional, nuevo requisito, mantenimiento o mejora de un requisito y/o resolución de incidencias. El proyecto fue gestionado aplicando métodos agiles y un proceso de ingeniería de software continua. En https://trello.com/proyectosigap se puede apreciar el productbacklog del proyecto SIGAP, trello es una plataforma colaborativa para gestión de proyectos, utiliza el metodo Kanban; se denomina productbacklog a la lista de requerimientos en un contexto ágil; mientras que en el repositorio publico github se encuentra alojado en la nube el código fuente de los backend y frontend del proyecto SIGAP: https://github.com/proyectosigap, github es una plataforma de desarrollo colaborativo en la nube, utiliza el control de versiones git (Trello, s.f; Tridibesh, 2016; Github, s.f.; Git-scm, 2005) y heroku como infraestructura en la nube para el despliegue y puesta en produccion de las apps del proyecto SIGAP, heroku soporta las tecnologías stack, la configuración de servidores y servicios, facilita la entrega e integracion continua de software, en vista que el despliegue en heroku es automatico, es la forma como los ingenieros de software hoy en dia en todo el mundo desarrollan, comparten, aprenden y trabajan de manera conjunta la construcción de software; Por otro lado backend se refiere a la capa que contiene las especificaciones de los microservicios o la lógica del negocio mientras que frontend es la capa que contiene los componentes del lado del cliente con la cual interactua el usuario final.

En la Tabla 2 se describe las Apps del Proyecto SIGAP, la tecnología stack utilizada, el motor de base de datos que contiene la información del proyecto, el Cloud Storage para el alojamiento de archivos en la nube, la dirección electrónica (URL) de las Apps de los ambientes de desarrollo asi como el nombre de los equipos que participaron durante el ciclo de vida de desarrollo del software. Como se puede apreciar en la Tabla se tiene claramente definido a nivel del cliente o frontend las tecnologías utilizadas, siendo ReactJs la mas usada y Vue.js como segunda opción; mientras que por el lado del servidor o backend no sucede lo mismo, cada App ha utilizado una tecnología distinta: Express.js, Spring Boot, Nuxt.js, Php y Python lo que evidencia la variabilidad en el dominio de tecnologías en los recursos humanos participantes en el Proyecto y se constituye en un valor agregado de los futuros profesionales calificados que les permitirá asumir la amplia cartera de proyectos del sector en lo concerniente a la producción de software a la medida o parametrizada; considerando las necesidades actuales del mercado laboral en materia de Ingenieria de Software son muy diversas y cada día más demandadas, involucra a todas las empresas del sector productivo y de servicios, sean estas públicas o privadas, nacionales o internacionales, como: El sector bancario y Financiero; el sector de transporte y telecomunicaciones; Sector Salud; Educación; Agrícola, Servicios varios, y otros sectores interesados en afianzar la transformación digital en la organización. El Programa Nacional Transversal de Tecnologías de la Información y Comunicación 2016-2021 revela que existe una brecha entre la demanda y la oferta de expertos en tecnologías de

**Tabla 2.** *Apps del Proyecto SIGAP*

| **PROYECTO SIGAP** | | | | | |
|---|---|---|---|---|---|
| **Database**: | Postgresql | | | | |
| **Nombre de App** | **Tecnología Stack** | | **Cloud Storage** | **URL de la App** | **Nombre del Equipo de Desarrollo** |
| | **Front End** | **Back End** | | | |
| SIGAP.Control-Recibos | ReactJs | Express.js (framework for Node.js) | | https://sigapdev-controlrecibos-front.herokuapp.com/ | HighTeam Cobras Winterfell |
| SIGAP.Consulta-Recibos | ReactJs | Spring Boot (framework Java) | | https://sigapdev-consultarecibos-front.herokuapp.com/ | Hard code, Newbee, Stackers, Ctrl+Z Press F, Gaman TichTechPlus, ThinkTEC |
| SIGAP.Legajo-Docente | Vue.js | Nuxt.js | Amazon S3, Cloudera | https://sigapdev-legajodocente-front.herokuapp.com/ | RandhuDevelopers, A-End game |
| SIGAP.Reportes | ReactJs | Php (framework CodeIgniter) | | https://sigapdev-reports-front.herokuapp.com/ | LambdaDev Ghostbusters Team Rock |
| SIGAP.Carga | ReactJs | Python | | https://sigapdev-carga-front.herokuapp.com/ | Alphazero, Dark Side |

Fuente: Elaboración Propia

Información e ingenieria de software en las empresas peruanas y de la región (CONCYTEC, 2016).

### C. Resultados y Discusión

Se propuso una arquitectura basada en Microservicios y DevOps para una ingeniería de software continua con equipos altamente motivados bajo contextos de agilidad permitiendo incrementar la productividad de software, mediante entregas progresivas de software.

Se implementó la Arquitectura en el proyecto SIGAP, Caso de Estudio de la presente investigación, esto ha permitido demostrar la capacidad del desarrollo de software bajo la perspectiva de la cultura Devops, minimizando la brecha entre desarrollo y operaciones, permitiendo una transferencia tecnológica continua de un producto de valor para una unidad organizativa de la institución universitaria como es la Unidad de Posgrado de la Facultad de Ingeniería de Sistemas e Informática.

Se desplegó los Microservicios en Plataformas Cloud o nube, asimismo se consumió otros servicios de acceso abierto como el tipo de cambio del dólar de la Superintendencia Nacional de Aduanas y de Administración Tributaria (SUNAT) permitiendo demostrar que las nuevas tendencias de desarrollo de software debe disponer de un enfoque abierto, colaborativo, independientemente de los lenguajes de programación en los que se encuentren desarrollados, plataformas tecnológicas en las que se encuentren desplegadas, siempre que cumplan con las especificaciones de algún protocolo de acceso remoto para su consumo o invocación.

Se involucró en el proyecto la participación de estudiantes de los cursos de Taller de Construccion de Software y de Sistemas de los semestres académicos 2018-1, 2018-2, 2019-1 y 2019-2 para aplicar y evaluar la nueva cultura DevOps en el desarrollo de software asicomo para desarrollar capacidades acorde a las tendencias tecnológicas disruptivas actuales, considerando que la academia tiene la responsabilidad de formar profesionales que cumplan el perfil de egreso de la carrera que cursan, con las exigencias del mercado laboral en lo concerniente al logro de una de las principales competencias de: desarrollar software bajo un enfoque colaborativo, evolutivo, ágil, con entregas e integración continúa considerando los cambios vertiginosos de las plataformas tecnológicas que evolucionan aceleradamente.

## III. CONCLUSIONES

- La presente investigación propuso una Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua; la propuesta contempla una Capa de Datos, una Capa de Microservicios o Backend y una Capa de Presentacion o Frontend, una Capa de Infraestructura en la Nube y Devops como perspectiva transversal a todas las capas permitiendo la reducción de la brecha entre desarrollo y operaciones con entregas e integración continua del software.
- La Arquitectura se implementó en el Proyecto SIGAP, con la participación de estudiantes de los cursos Taller de Construccion de Software y de Sistemas durante los semestres académicos 2018-1, 2018-2, 2019-1 y 2019-2 permitiendo aplicar y evaluar la cultura DevOps en la formación de equipos agiles y altamente motivados con una formación basada en competencias según el perfil de egreso del plan de estudios.
- La investigación ha permitido realizar transferencia tecnológica continua de un producto de valor para una unidad organizativa de la institución universitaria como es la Unidad de Posgrado de la Facultad de Ingeniería de Sistemas e Informática.
- La propuesta arquitectural es extendible a otros dominios que requieran aplicar nuevas técnicas que les permitan afrontar entregas continuas de software en contextos de desarrollo acelerado que la gestión de incidentes o gestión de servicios de TI no alcanzan cubrir.

## IV. REFERENCIAS BIBLIOGRÁFICAS

[1] Abdullaha, Iqbal y Erradi. (2019). Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software, 151*, 243-257. doi:https://doi.org/10.1016/j.jss.2019.02.031

[2] Anubhav. (2014). *Heroku Cloud Application Development.* USA: O'Reilly Media Inc.

[3] Bandeira y otros. (2019). We Need to Talk about Microservices: an Analysis. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. doi:DOI 10.1109/MSR.2019.00051

[4] CONCYTEC. (2016). *Programa Nacional Transversal de Tecnologías de la Información y Comunicación 2016-2021.* Obtenido de concytec: http://portal.concytec.gob.pe/images/publicaciones/libro_tics_oct.pdf

[5] Craig, Sturm y Pollard. (2017). *Application Performance Management (APM) in the Digital Enterprise.* Elsevier Inc. doi:http://dx.doi.org/10.1016/B978-0-12-804018-8.00010-3

[6] Daniels y Davis. (2018). *What Is DevOps?* USA: O'Reilly Media, Inc.

[7] Dittrich y otros. (2018). Researching Cooperation and Communication in Continuous Software Engineering. *CHASE'18:IEEE/ACM 11th International Workshop on Cooperative*, 87-90. doi:https://doi.org/10.1145/3195836.3195856

[8] Gheorghiu y otros. (2019). *Python for DevOps.* USA: O'Reilly Media, Inc.

[9] Github. (s.f.). *Plataforma de Desarrollo Colaborativo*. Obtenido de Github: https://github.com

[10] Git-scm. (2005). *Sistema de Control de Versiones Distribuido*. Obtenido de Git-scm: https://git-scm.com/

[11] Jabbari y otros. (2016). What is DevOps? A Systematic Mapping Study on Definitions and Practices. *XP '16 Workshops, May 24 2016, Edinburgh, Scotland Uk*. doi:http://dx.doi.org/10.1145/2962695.2962707

[12] Kenneth. (2016). *The Little Book on Rest Services.* Copenhagen.

[13] Larman, C. (1999). *UML y Patrones, Introduccion al analisis y diseño orientado a objetos* (1 ed.). (L. M. Rodriguez, Trad.) US: Pearson.

[14] Lenarduzzi y Sievi-Korte. (2018). Software Components Selection in Microservices-based Systems. *XP '18 Companion, May 21–25, 2018, Porto, Portugal. ACM*. doi:https://doi.org/10.1145/3234152.3234154

[15] Lenarduzzi, Lomio, Saarimäki y Taibi. (2020). Does migrating a monolithic system to microservices decrease the. *The Journal of Systems & Software*, 169. doi:https://doi.org/10.1016/j.jss.2020.110710

[16] López y Spillner. (2017). Towards Quantifiable Boundaries for Elastic Horizontal Scaling. *Proceedings of UCC'17: 10th International Conference on Utility and Cloud Computing Companion,Austin, Texas, USA, December 5–8, 2017 (UCC'17 Companion)*, 35-40. doi:https://doi.org/10.1145/3147234.3148111

[17] Richards, M. (2015). *Software Architecture Patterns,Understanding Common Arquitecture Patterns and When to Use them* (1 ed.). US: O'Reilly Media.

[18] Richardson. (2019). *Microservices Patterns.* U.S: Manning Publicarions.

[19] Richardson y Smith. (2016). *Microservices from design to deployment.* USA: NGINX, Inc.

[20] Salza y otros. (2017). cCube: A Cloud Microservices Architecture for Evolutionary Machine Learning Classification. *Proceedings of GECCO '17 Companion, Berlin, Germany*, 2. doi:http://dx.doi.org/10.1145/3067695.3076089

[21] Trello. (2017). *Plataforma para Gestión de Proyectos Colaborativos*.

[22] Tridibesh. (2016). *Una Guía para el Cuerpo de Conocimiento de Scrum.* Arizona, USA: VMEdu.