Aplicación *cloud native* en el contexto de una ingeniería de *software* continua

Zoraida Mamani Rodriguez
zmamanir@unmsm.edu.pe
https://orcid.org/0000-0002-2590-8387
Universidad Nacional Mayor de San Marcos, Perú

Recibido: 28 de marzo del 2024 / Aceptado: 23 de mayo del 2024
doi: https://doi.org/10.26439/interfases2024.n19.7038

RESUMEN. Una aplicación *cloud native* es un tipo de *software* que ha sido diseñado específicamente para ejecutarse en la nube, con enfoque distribuido, elástico, escalado horizontal y compuesto de microservicios con implementación autónoma. Asimismo, se diseñan con arquitecturas web *cloud native*, operan en una plataforma elástica de autoservicio y se caracterizan por su resiliencia y elasticidad. La ingeniería de *software* continua es un proceso que busca articular la ingeniería de requisitos, el desarrollo y las operaciones en un bucle continuo, con una retroalimentación recíproca, con la finalidad de producir un *software* de calidad. En ese contexto, el presente trabajo propone el diseño e implementación de una aplicación *cloud native* en una perspectiva de ingeniería de *software* continua, aplicada al caso de estudio SIGCON. Usa el modelo de servicio *cloud* CaaS, aplica el patrón BFF en la construcción del *software*, realiza contenedorización del *frontend*, *backend* y almacenamiento, y expone los resultados.

PALABRAS CLAVE: *cloud native application* / modelos de servicios *cloud* / patrón BFF / ingeniería de *software* continua

CONTINUOUS SOFTWARE ENGINEERING OF A CLOUD-NATIVE APPLICATION

ABSTRACT. A cloud-native application is a software specifically designed to run in the cloud, focusing on distributed, elastic, horizontally scaled, and microservice-based architecture with autonomous deployment. These applications are designed with cloud-native web architectures, operate on an elastic self-service platform, and stand out because of their resilience and elasticity. Continuous software engineering integrates requirements engineering, development, and operations in a continuous loop with reciprocal feedback to produce quality software. The present work proposes to design and implement a cloud-native application applied to the SIGCON case study from a continuous software engineering perspective. It uses the CaaS cloud service

model, applies the BFF pattern in software construction, containerizes the frontend, backend, and storage, and presents the results.

KEYWORDS: Cloud Native Application / Cloud Service Models / BFF Pattern / Continuous Software Engineering

1. INTRODUCCIÓN

La industria del *software* evoluciona aceleradamente. Nuevos enfoques se orientan a la construcción de *software* nativo de la nube, ejecutándose bajo modelos de computación "sin servidor", con microservicios autónomos desplegados en contenedores y diseñados con arquitecturas web componibles. Estas tendencias tecnológicas resilientes se exponen en la literatura. Por otro lado, existe una alta demanda laboral a nivel global de profesionales idóneos en la construcción de aplicaciones *cloud native*. En consecuencia, corresponde a la academia la formación de recursos humanos con competencias y capacidades en la construcción de *software* bajo estos nuevos enfoques, que les permita asumir los desafíos de la industria. El principal aporte de la investigación se centra en diseñar e implementar una aplicación *cloud native* en el contexto de una ingeniería de *software* continua. Utiliza arquitecturas *cloud* componibles, patrón *cloud* BFF, metodología ágil Scrum, cultura DevOps en el desarrollo colaborativo de aplicaciones escalables, con recursos humanos en proceso formativo, ágiles, rotativos, altamente motivados y enfocados en la automatización moderna de procesos de negocios. Este ecosistema de componentes tecnológicos, humanos, arquitecturas, data, servidores virtualizados y modelos de servicios *cloud,* lo exhibe como innovador en la industria peruana de *software*. Considerando lo expuesto, la presente investigación establece como objetivos: (1) diseñar una arquitectura CNA con una perspectiva de ingeniería de *software* continua; (2) implementar la propuesta en el caso de estudio SIGCON; (3) realizar el despliegue del *software* bajo el modelo de servicio *cloud* CaaS; y (4) evaluar los resultados.

2. MARCO TEÓRICO

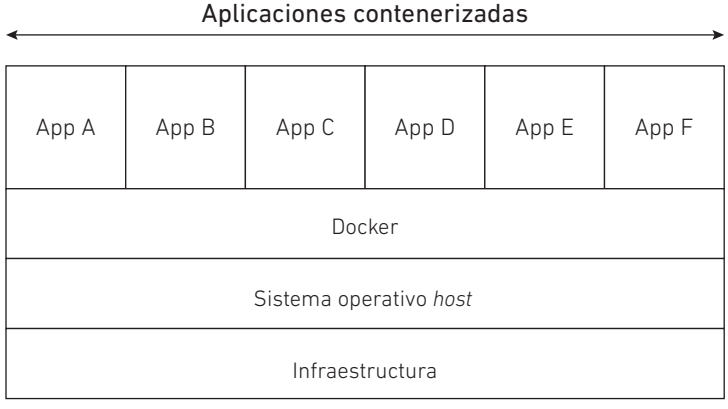
Modelos de servicio *cloud*

El término *cloud* se populariza en el año 2006 con el lanzamiento de un servicio de nube pública de propósito general de parte de Amazon Web Services (AWS). Este tipo de modelo de servicio corresponde a infraestructura como servicio (IaaS), que consiste en proporcionar al consumidor procesamiento, almacenamiento, redes y otros recursos informáticos fundamentales, de manera que dicho consumidor pueda implementar y ejecutar *software* arbitrario, sistemas operativos y aplicaciones. El consumidor no gestiona ni controla la IaaS, pero tiene control sobre el sistema operativo (SO), tiempos de ejecución, escalamiento, código fuente de la aplicación, así como los aspectos relativos a los datos y configuración que residen en ella y al control limitado de componentes de red, como *firewalls*. Posteriormente, en el año 2009 destacan —en el ámbito tecnológico— las plataformas como servicios (PaaS), que ofrecen al consumidor el uso de la IaaS para desplegar sus aplicaciones. Aquí el consumidor no administra ni controla la IaaS, solo tiene control sobre las aplicaciones implementadas y los ajustes de configuración para su funcionamiento (Mell & Grance, 2012). En estos años, en el ámbito de la

comunidad de desarrolladores de *software* se popularizó la plataforma Heroku, debido a que simplificó el proceso de desarrollo y despliegue de las aplicaciones, haciendo más simple, eficiente y competitivo el proceso de despliegue. Como consecuencia de ello, se incrementó la productividad del desarrollo de *software* y se redujeron costos, pues el consumo por el uso de una PaaS es bajo demanda y se disponen de herramientas para monitorear y escalar las aplicaciones. El modelo de *software* como servicio ofrece al consumidor el uso de las aplicaciones del proveedor que se ejecutan en una IaaS. Este modelo es utilizado para distribuir aplicaciones en la nube a los usuarios a través de Internet y se pone a disposición bajo un modelo de pago que puede ser una suscripción o una compra. Hussein et al. (2019) describen un contenedor como una tecnología de virtualización liviana emergente que opera a nivel del SO para encapsular una tarea y sus dependencias de biblioteca para su ejecución. Es posible que diferentes contenedores se ejecuten en un SO, como se aprecia en la Figura 1. El modelo contenedor como servicio (CaaS) surge con la finalidad de resolver problemas de las aplicaciones desarrolladas en un determinado entorno PaaS, limitadas por las especificaciones de ese entorno. CaaS permite el despliegue de la aplicación independiente del entorno PaaS, eliminando los posibles conflictos, barreras o limitaciones que podrían originarse por la convivencia de varios servicios como base de datos, lenguajes de programación, servidores de aplicaciones, entre otros, en un mismo entorno PaaS. Asimismo, este tipo de aplicaciones que se ejecutan en un CaaS son portátiles, toda vez que pueden trasladarse a cualquier otro entorno de ejecución. Una plataforma abierta para ejecutar contenedores de aplicaciones es Docker (Docker, 2024), cuya arquitectura se muestra en la Figura 1.

Figura 1

Modelo de servicio CaaS

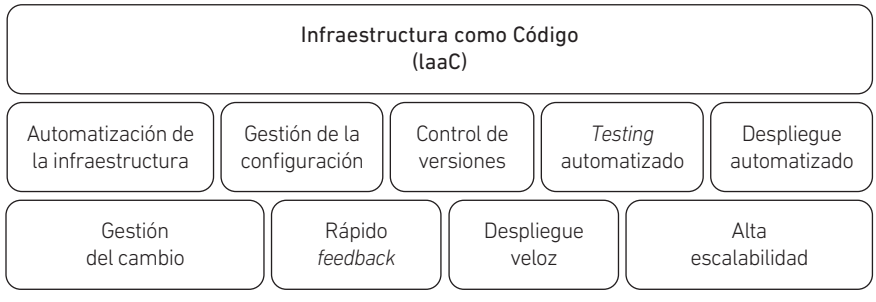


Nota. Tomado de Docker, 2024, *Use containers to build, share and run your applications. https://www.docker.com/resources/what-container/*

El modelo función como servicio (FaaS) utiliza la computación *serverless* o sin servidor. Este tipo de modelo permite que las arquitecturas de microservicios funcionen considerando cada microservicio como una función, la cual es tomada por el proveedor FaaS, que ejecuta y gestiona la función desplegada. Este tipo de modelo es muy utilizado por los desarrolladores de *software*, dado que no tienen que preocuparse por la infraestructura ni por el despliegue, lo que permite la reducción de tiempo y de costos en la construcción de *software*. Entre las ventajas del modelo FaaS está que las tareas de escalamiento, mantenimiento, recuperación ante desastres, así como aspectos de seguridad, son realizadas por el proveedor del servicio FaaS. Como desventajas tenemos la pérdida de control del sistema, el proceso de pruebas (que es más complejo), una dependencia a largo plazo con el proveedor FaaS, además de tener que ceñirse estrictamente a sus requisitos para que la solución *software* funcione correctamente. AWS Lambda es una plataforma FaaS muy popular, seguida por Azure Functions y Google Cloud Functions (Habala et al., 2023). Infraestructura como código (IaaC) es un proceso de gestión de infraestructura de TI que aplica las mejores prácticas: desde el desarrollo de *software* DevOps hasta la gestión de recursos de infraestructura en la nube mediante código o *scripts* como máquinas virtuales (MV), redes, balanceadores de carga, bases de datos y otras aplicaciones en red. A estos *scripts* utilizados en la IaaC se les conoce como *scripts* de configuración como código, los cuales reducen el aprovisionamiento de recursos en la nube (Almuairfi & Alenezi, 2020; Buchanan, 2024). En la Figura 2 se aprecian las funcionalidades que ofrece este proceso de automatización de la infraestructura, gestión de la configuración, control de versiones, automatización de las pruebas, automatización del despliegue, gestión del cambio, rápido *feedback*, despliegue veloz, así como alta escalabilidad.

Figura 2

Funcionalidades de la IaaC



Nota. Tomado de I. Buchanan, 2024, *Infrastructure as code. How infrastructure as code (IaC) manages complex infrastructures*. Atlassian. https://www.atlassian.com/microservices/cloud-computing/infrastructure-as-code

En el trabajo de Duan et al. (2015) se ha realizado una amplia revisión de la literatura sobre el término XaaS y los comprendidos como *as a service*. *Anything* como servicio o cualquier cosa como servicio (XaaS) comprende cualquier herramienta, aplicación o recurso que se suministre a través de la nube mediante suscripción. Este tipo de servicio generalmente se rige por acuerdos del nivel de servicio (SLA) que proveedores y clientes especifican en el contrato. Este mecanismo permite ahorrar costos, ofrece una rápida comercialización, flexibilidad para centrarse en el negocio principal, escalabilidad y confiabilidad. Los modelos de servicios en la nube continúan evolucionando, como se puede apreciar en la Figura 3. Hemos identificado a la fecha hasta veinticuatro categorías.

Figura 3

Modelos de servicio cloud

Acrónimo	Descripción
IaaS	Infraestructura como servicio
PaaS	Plataforma como servicio
SaaS	*Software* como servicio
FaaS	Función como servicio
CaaS	Contenedores como servicio
IaaC	Infraestructura como código
CaC	Configuración como código
XaaS	Cualquier cosa como servicio
AaaS	*Analytics* como servicio
BPaaS	Plataforma *backend* como servicio
MBaaS	*Backend* móvil como servicio
BaaS	*Backup* como servicio
RaaS	*Recovery* como servicio
DRaaS	*Disaster recovery* como servicio
HaaS	*Hardware* como servicio
MaaS	*Monitoring* como servicio
SECaaS	Seguridad como servicio
FWaaS	*Firewall* como servicio
DaaS	*Database* como servicio
NaaS	*Network* como servicio
STaaS	*Storage* como servicio
AIaaS	Inteligencia artificial como servicio
EssS	*Encryption* como servicio
RAaaS	Asignación de recursos como servicio en múltiples proveedores de nube

Aplicación *cloud native*

Una *cloud native application* o aplicación nativa de la nube (CNA) es un sistema distribuido, elástico, con escalado horizontal, compuesto de microservicios que aísla el estado en un mínimo de componentes con estado. La aplicación y cada una de sus unidades de implementación autónoma se diseñan de acuerdo con patrones de diseño centrados en la nube y operan en una plataforma elástica de autoservicio. Así lo definen Kratzke y Quint (2017). Otra definición similar describe a una CNA como un tipo de *software* que ha sido diseñado específicamente para ejecutarse en un entorno de nube. Para ser considerada una CNA, una aplicación *software* debe cumplir con ciertas características, como resiliencia y elasticidad. La resiliencia implica que una CNA debe anticiparse a los fallos y fluctuaciones en el contexto de los recursos de la nube, así como de los servicios de terceros necesarios

para su operatividad. Por su parte, la elasticidad comprende la capacidad de las CNA para ampliar el uso de recursos requerido, evitando el aprovisionamiento excesivo o insuficiente y considerando que la nube es un servicio medido que ofrece autoservicio bajo demanda y que, por tanto, requiere rápida elasticidad (Toffetti et al., 2017). La naturaleza de las CNA es que frecuentemente dependen de servicios de terceros, por lo que se incrementa el riesgo de que estos servicios puedan fallar o presentar insuficiencia en la calidad de su servicio. Las CNA utilizan una pila (*stack* en inglés) de *software* de código abierto para segmentar aplicaciones en microservicios, empaquetar cada microservicio en su propio contenedor y orquestar dinámicamente estos contenedores para optimizar el uso de recursos de la nube.

Ingeniería de *software* continua

La ingeniería de *software* continua (CSE) es un proceso en auge, que busca articular la ingeniería de requisitos, el desarrollo y las operaciones en un bucle continuo, con retroalimentación recíproca, para producir *software* de calidad. La CSE es uno de los principios DevOps que enfatiza que la integración entre el desarrollo de *software* y su distribución operativa debe ser continua. DevOps mejora la colaboración entre las partes interesadas, los equipos de desarrollo y las operaciones. La práctica de la CSE es la entrega rápida, la minimización del tiempo de lanzamiento de nuevas funcionalidades, la mitigación de riesgos, y el impulso de mejoras o refactorizaciones bajo un proceso de entrega continua (Eramo et al., 2024).

3. METODOLOGÍA

En la sección anterior se han revisado los fundamentos de los modelos de servicios de la nube, las CNA y la CSE. En esta sección desarrollaremos la metodología que se aplicará en un caso de estudio. Esta se organiza de la siguiente manera: (A) diseño de la arquitectura CNA en el contexto de una CSE, y (B) implementación de una CNA mediante un caso de estudio.

El caso de estudio de la investigación está orientado a la administración de edificaciones modernas, como son las viviendas multifamiliares. Es una tendencia mundial el crecimiento vertical de las ciudades y la construcción de grandes edificios multifamiliares (Espinoza, 2020; Vega, 2021). Este tipo de predios mantienen áreas comunes, compartidas entre todos los propietarios de las unidades inmobiliarias o departamentos; estos pueden ser parques internos, gimnasios, piscinas, ascensores, salas de niños, salas de cine, zonas de parrillas, entre otros. Dado que estos espacios requieren mantenimiento permanente, se hace necesario contar con un área administrativa que se encargue de todo el proceso de gestión de áreas comunes, emisión y cobranza de recibos de mantenimiento, balance de ingresos y egresos, gestión de visitas, gestión de compras, gestión de recursos humanos, entre otras tareas que implique mantener el predio operativo, de

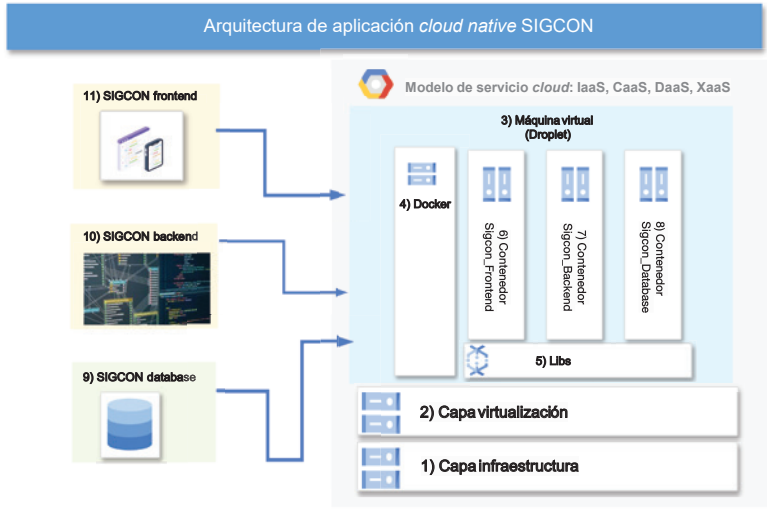
manera que sus propietarios puedan usarlo sin mayores preocupaciones (Casa Grande, 2023). Según lo descrito, existe una alta demanda por contar con *software* que automatice los procesos de gestión de mantenimiento del predio o condominio, y es en ese orden de ideas que proponemos el sistema de gestión de condominios (SIGCON) desde una perspectiva de arquitectura de aplicación *cloud native* en el contexto de una CSE.

A. Diseño de la arquitectura CNA en el contexto de una CSE

En la Figura 4 se presenta la arquitectura del sistema SIGCON. La primera capa (1) corresponde a una IaaS con el proveedor *cloud* Digital Ocean. Se adquirió una MV con 1 GB memoria / 25 GB disco duro con SO Linux Ubuntu 22.10 x64. Esta MV se constituye en la IaaS del proyecto. Sobre esta capa se aplicó el modelo de servicio CaaS, específicamente desplegando tres contenedores (6, 7 y 8 en la Figura 4) gestionados por la herramienta Docker (4) y un conjunto de librerías comunes (5). Se utilizó CaaS debido al uso de *frameworks*; Flask por el lado del *backend*, uso de lenguaje python, angular por el lado del *frontend*, uso del lenguaje typescript, html, css, arquitecturas e interfaces en el desarrollo del sistema SIGCON. Con la finalidad de facilitar su despliegue y evitar posibles conflictos en el uso de puertos y librerías de terceros (también llamadas dependencias), se hizo necesario el aislamiento de la aplicación de su entorno. Por otro lado, la naturaleza de una CNA es su construcción a razón de microservicios y su despliegue en contenedores con fines de resiliencia y elasticidad. En ese orden de ideas, SIGCON fue diseñado bajo un contexto de servicios desacoplados: la clara separación del *backend* de la aplicación (10) con respecto al *frontend* (11), así como el despliegue de la base de datos mediante un modelo DaaS (9), así lo demuestra.

Figura 4

Arquitectura física de la aplicación cloud native *SIGCON*



B. Implementación de una CNA mediante un caso de estudio

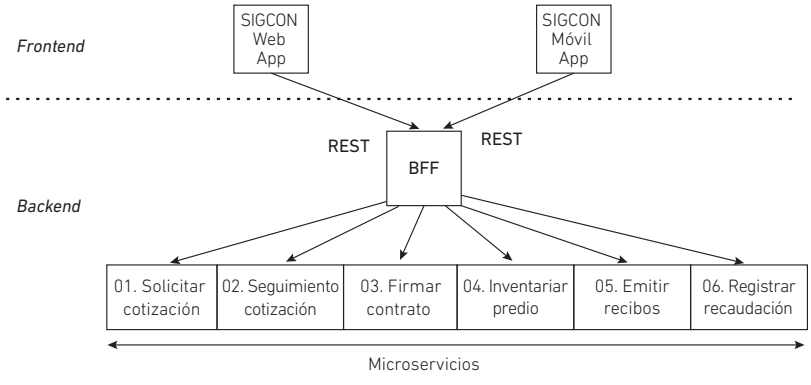
La implementación de la CNA enfocada al caso de estudio SIGCON inicia con el modelado del negocio, identificando cinco procesos clave como casos de uso de negocio (CUN): gestión de servicios (CUN 1.0), gestión de predios (CUN 2.0), gestión del mantenimiento (CUN 3.0), gestión de cobranza (CUN 4.0) y gestión de adquisiciones (CUN 5.0). La fase de requisitos permitió identificar los casos de uso del sistema, a partir de los cuales, aplicando la metodología ágil Scrum, se procedió a formular el *product backlog*, con seis requisitos funcionales fundamentales: 01. solicitar cotización, 02. seguimiento cotización, 03. firmar contrato, 04. inventariar predio, 05. emitir recibos y 06. registrar recaudación.

Arquitectura del software

La arquitectura del *software* del proyecto adopta inicialmente una arquitectura basada en tres capas (capa de presentación, capa de la lógica de aplicaciones y almacenamiento). Luego de aplicar los patrones generales de *software* para la asignación de responsabilidades, se descompone la capa de la lógica de aplicaciones, constituyéndose en una arquitectura multicapas compuesta por estratos verticales o subcapas (capa del dominio, capa de servicios y particiones horizontales), una por cada requisito identificado en la etapa de análisis del proyecto, de acuerdo con Larman (1999). El uso de una arquitectura web moderna conlleva a implementar arquitecturas multicapas utilizando el patrón *backend for frontend* (BFF), proyectándose a la resiliencia, escalamiento y elasticidad que caracteriza a las CNA. Este desacoplamiento corresponde a una división entre la V que reside en el lado del cliente y la M en el lado del servidor, considerando el estilo arquitectónico modelo-vista-controlador. BFF permite utilizar arquitecturas de *software* componibles de manera que el *frontend* y el *backend* evolucionen a diferentes ritmos y escalas. Esto hace posible adaptarse entre las necesidades de aplicaciones nativas, móviles, SPA (*single-page application*) y microservicios (Brown & Woolf, 2016). En la Figura 5 se expone la arquitectura lógica componible del proyecto SIGCON.

Figura 5

Arquitectura lógica del proyecto SIGCON



Desarrollo del frontend

El desarrollo del *frontend* ha sido realizado con el *framework* Angular 16.2.9, el cual permite crear aplicaciones de una sola página (SPA). Las SPA se implementan de forma nativa en los navegadores modernos, utilizan HTML5, CSS3 y JavaScript; almacenan el estado de la página en el cliente y se conectan al *backend* a través de servicios REST. Se conoce así a este enfoque porque todo el código (HTML, CSS y JavaScript) necesario para un conjunto de funcionalidades que pueden corresponder a múltiples pantallas o páginas lógicas, se recuperan como una solicitud de una sola página. JavaScript se encarga de toda la manipulación del modelo de objetos de documento HTML, la navegación de la página y el acceso a los datos del *backend*. Las SPA son utilizadas para aprovechar los principios del diseño responsivo, permitiendo optimizar la experiencia del usuario en cuanto al diseño y tamaño de la pantalla. Los scripts Media CSS se utilizan a menudo para incluir bloques específicos que solo se aplican a determinados tipos de pantalla. Esta técnica permite especificar un conjunto diferente de reglas CSS para tabletas, teléfonos móviles o portátiles, lo que da como resultado pantallas diseñadas y configuradas específicamente para esos dispositivos (Brown & Woolf, 2016). En ese orden de ideas, Angular es un *framework* basado en componentes para crear aplicaciones web escalables; está escrito en TypeScript. Implementa la funcionalidad principal y opcional como un conjunto de bibliotecas de TypeScript que importa a sus aplicaciones. En la Tabla 1 se presentan los principales *frameworks* utilizados en el proyecto.

Tabla 1

Frameworks *del proyecto SIGCON*

Servicios	*Frameworks*
SIGCON_frontend	Angular 16.2.9, bootstrap, nginx, node,js
SIGCON_backend	Python, Flask, marshmallow, psycopg2, SQLAlchemy
SIGCON_database	Postgresql 15

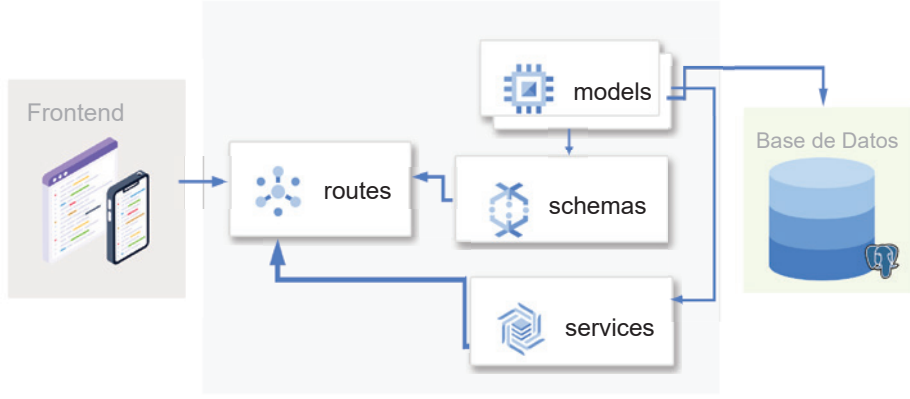
Desarrollo del backend

El *backend* del proyecto, como se aprecia en la Figura 6, se desarrolló utilizando el micro *framework* Flask debido a que permite un desarrollo fácil y rápido, pero con capacidad de escalar a aplicaciones más complejas. Este *framework* utiliza lenguaje de programación Python, de código abierto, no depende de ninguna plataforma, es un lenguaje dinámico, admite programación orientada a objetos y es funcional. En ese contexto, Flask proporciona el código necesario para poner en marcha de forma rápida y segura los servicios web (Flask, s. f.). Asimismo, se utilizaron librerías de terceros para agilizar el desarrollo, como por ejemplo el Flask-Marshmallow, una biblioteca de serialización y

deserialización de objetos, que se integra con Flask-SQLAlchemy —una extensión para Flask que agrega soporte para el mapeador relacional de objetos (ORM) SQLAlchemy—, y permite configurar objetos y patrones comunes para usar esos objetos, como una sesión vinculada a cada solicitud web, modelos y motores (FlaskSQLalchemy, s. f.). Por su parte, el ORM referido brinda el poder y flexibilidad de SQL y proporciona un conjunto completo de patrones de persistencia conocidos a nivel empresarial, diseñados para un acceso eficiente y de alto rendimiento a bases de datos, adaptados a un lenguaje de dominio simple (SQLAlchemy, s. f.). Otra librería de terceros utilizada es psycopg2, que es el adaptador para bases de datos PostgreSQL.

Figura 6

Arquitectura del backend *SIGCON*



Despliegue del proyecto

El proceso de despliegue siguió el diseño de la arquitectura CNA en el contexto de una CSE indicado en la Figura 4. Considerando el enfoque CaaS, se procedió a seleccionar el Docker como herramienta de contenedorización en este proyecto, debido a su simplicidad de uso, además de la documentación y el soporte de una amplia comunidad de desarrolladores que la respaldan. Un contenedor Docker es un paquete de *software* liviano, independiente y ejecutable que encapsula una aplicación y sus dependencias, bibliotecas y archivos de configuración (Merelli et al., 2019). Un contenedor dentro de un entorno virtualizado proporciona un entorno de ejecución consistente y aislado para que las aplicaciones se ejecuten sin problemas en diferentes entornos informáticos. El archivo de configuración Dockerfile para el despliegue del *backend* se indica a continuación: como se aprecia en la sección del código, línea 1, se indica la imagen Docker a utilizar. Una imagen de Docker es un repositorio alojado en Docker Hub, sirve como punto de partida para la mayoría de los desarrolladores de *software*, incluye sistemas operativos

y lenguajes de programación y es confiable dado que tiene pocas o ninguna vulnerabilidad (Docker, 2024). En el presente proyecto se utilizó la imagen Docker Python:3.8, que tiene instalado el SO Linux ubuntu, python:3.8. En la línea 2 se copia el código fuente del *backend* en el directorio predeterminado en ubuntu, carpeta app, la cual se establece como directorio de trabajo en la línea 3. En la línea 4 se instalan las dependencias requeridas para el funcionamiento del *backend*, las cuales se encuentran consignadas en el archivo requirements.txt del proyecto. En la línea 5 se establece el puerto a considerar en el despliegue y, finalmente, en la línea 6, se precisan los comandos para lanzar la ejecución del *backend*.

1. FROM python:3.8
2. COPY ./ /app
3. WORKDIR /app
4. RUN pip install -r requirements.txt
5. EXPOSE 5000
6. CMD [“python3”, “-m” , “flask”, “run”, “--host=0.0.0.0”]

Luego de definir el archivo de configuración Dockerfile, ya se puede crear el contenedor. Para ello se ejecuta el comando: sudo docker build -t sigcon-backend-1.0.0. Esta instrucción dará lectura al Dockerfile y procesará las líneas contenidas allí. Finalmente, para ejecutar el contenedor se lanza la instrucción: sudo docker run -p 5000:5000 sigcon-backend-1.0.0.

Para la contenedorización del *frontend* igualmente se configuró el Dockerfile. Aquí se utilizó la distribución Linux Alpine, como se muestra en la línea 1, se estableció el directorio de trabajo, la carpeta app y en ella se realizó una copia de la carpeta dist que contiene el compilado del *frontend*. En la línea 4 se instala nginx, que es un es un servidor web ligero de alto rendimiento; luego, se requiere copiar el archivo de configuración nginx.conf que contiene los parámetros de configuración. Finalmente, en la línea 6 se copia la carpeta sigcon_frontend que se encuentra en la carpeta dist a la carpeta html.

1. FROM node:16-alpine3.11 AS build
2. WORKDIR /usr/src/app
3. COPY . .
4. FROM nginx:1.17.1-alpine
5. COPY nginx.conf /etc/nginx/nginx.conf
6. COPY --from=build /usr/src/app/dist/sigcon_frontend /usr/share/nginx/html

El procedimiento seguido para crear el contenedor del *backend* es el mismo para el *frontend*: sudo docker build -t sigcon-frontend-1.0.0. Y para ejecutar el contenedor: sudo docker run -p 4200:80 sigcon-frontend-1.0.0..

4. RESULTADOS

El paradigma CNA continúa evolucionando. Hoy ya se habla de ingeniería CNA, cuya implementación aún es compleja, puesto que implica contar con recursos humanos altamente especializados en *cloud native*, modelos de servicios *cloud*, diseño y construcción de arquitecturas *cloud native*, DevOps, integración y entrega continua de *software*, patrones de diseño para enfoques *cloud native*, todo ello complementado al proceso de CSE que deben utilizar los equipos de desarrollo.

El despliegue de los servicios de acuerdo con la arquitectura propuesta se puede apreciar en la Tabla 2. Se trata de contenedores desacoplados, con ejecución independiente. Los *scripts* definidos en la sección Despliegue del proyecto permiten realizar el proceso de integración y entrega continua de *software*.

Tabla 2

Contenedores del proyecto SIGCON

Imagen	Comando	Puertos	Nombre Contenedor	URL
sigcon-back-end-1.0.0	"python3 -m flask ru..."	0.0.0.0:5000->5000/tcp	unruffled_ poitras	http://137.184.120.127:4200/principal
sigcon-fron-tend-1.0.0	"nginx -g 'dae-mon of..."	0.0.0.0:4200->80/tcp	upbeat_tesla	
sigcon-data-base-1.0.0	"postgres 'dae-mon of..."	0.0.0.0:5432->5432/tcp	epic_ brahmagupta	

El utilizar el modelo de servicios CaaS en el proyecto, nos ha permitido disponer del prototipo de la aplicación en la nube a muy bajo costo. Este modelo adopta una infraestructura inmutable, pues aquí los contenedores no se reparan ni modifican. Si uno falla o requiere una actualización, se destruye y se aprovisiona uno nuevo, todo a través de los *scripts* consignados en el archivo Dockerfile.

El prototipo del producto *software* resultante de la investigación se encuentra publicado en la URL http://137.184.120.127:4200/principal.

5. DISCUSIÓN DE LOS RESULTADOS

Concordamos con lo señalado por Stine (2015) y Balalaie et al. (2016) sobre las arquitecturas CNA, las cuales buscan entregar *software* más rápido, con mayor aislamiento, con

tolerancia a fallos y recuperación automática, de manera que permiten un escalamiento horizontal de aplicaciones, e incorporar a la CNA otros entornos (móvil, sistemas heredados), todo ello bajo el enfoque de computación sin servidor.

La parte experimental de nuestra investigación nos permitió coincidir con Kratzke y Quint (2017) en el sentido de que los microservicios no son otra cosa que la descomposición de las funcionalidades de los sistemas tradicionales, conocidos como monolitos. La interacción entre estos es a través de APIs, las cuales siguen el estilo REST con serialización JSON u otro formato. Su despliegue se realiza en plataformas de infraestructuras ágiles de autoservicio, operando de manera autónoma, con escalamiento automatizado, bajo demanda de aplicaciones, gestión del estado, enrutamiento dinámico, equilibrio de carga y monitoreo de métricas.

6. CONCLUSIONES

- La presente investigación propuso el diseño e implementación de una aplicación *cloud native* en el contexto de una CSE. La propuesta usa el modelo de servicio cloud CaaS y aplica el patrón BFF en la construcción del proyecto SIGCON, con la finalidad de contenedorizar el despliegue del *frontend*, *backend* y almacenamiento en entornos virtualizados y desacoplados, con proyección a lograr resiliencia, escalamiento y elasticidad en el tiempo.
- La propuesta puede implementarse en otros dominios de negocios que requieran aplicar arquitecturas web modernas y componibles, utilizando el patrón BFF, modelo de servicio *cloud* CaaS. Estos enfoques permiten afrontar integración y entregas continuas de *software* en contextos de desarrollo acelerados.
- La implementación del proyecto se realizó con la participación de los estudiantes del curso Desarrollo de sistemas web de los semestres académicos 2023-I y 2023-II, de la escuela profesional de Ingeniería de Sistemas de la Universidad Nacional Mayor de San Marcos, y la docente y coordinadora del grupo de investigación Ingeniería web. Se aplicó la metodología ágil Scrum en la gestión del proyecto, la cultura DevOps en la formación de los equipos, con fines de lograr las competencias establecidas en el curso y el perfil de egreso del plan de estudios correspondiente.
- La investigación ha permitido formar recursos humanos según las tendencias actuales en ingeniería de *software*, área en constante evolución, con alta demanda laboral nacional e internacional. Estos profesionales requieren una formación actualizada, idónea y rigurosa, con un producto de valor como resultado.

7. TRABAJOS FUTUROS

De la presente investigación se desprenden algunas sublíneas no cubiertas, como la implementación de los patrones *service discovery* y *circuit breaker*, los cuales pueden ser abordados como trabajos futuros.

REFERENCIAS

Almuairfi, S., & Alenezi, M. (2020). Security controls in infrastructure as code. *Computer fraud & security*, *2020*(10), 13-19. https://doi.org/10.1016/S1361-3723(20)30109-3

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables DevOps: migration to a Cloud-Native architecture. *IEEE Software*, *33*(3), 42-52. https://doi.org/10.1109/MS.2016.64

Brown, K., & Woolf, B. (2016, octubre). *Implementation patterns for microservices architectures*. [presentación de paper]. PLoP'16: Proceedings of the 23rd Conference on pattern languages of programs, 1-35. https://www.hillside.net/plop/2016/papers/proceedings/papers/brown.pdf

Buchanan, I. (2024). *Infrastructure as code. How infrastructure as code (IaC) manages complex infrastructures*. Atlassian. https://www.atlassian.com/microservices/cloud-computing/infrastructure-as-code

Casa Grande. (2023, 12 de mayo). *Reglamento de convivencia de edificios y condominios*. https://www.administracionedificiosperu.com/2020/09/reglamento-de-convivencia-de-edificios.html

Docker (2024). *Use containers to build, share and run your applications.* https://www.docker.com/resources/what-container/

Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N. & Hu, B. (2015). Everything as a service (Xaas) on the cloud: origins, current and future trends. *CLOUD'15: proceedings of the 2045 IEEE 8th International conference on cloud computing*, 621-628. https://doi.org/10.1109/CLOUD.2015.88

Eramo, R., Tucci, M., Di Pompeo, D., Cortellessa, V., Di Marco, A., & Taibi, D. (2024). Architectural support for software performance in continuous software engineering: a systematic mapping study. *Journal of systems and software*, *207*, 111833. https://doi.org/10.1016/j.jss.2023.111833

Espinoza, C. (2020, 22 de febrero). Crecimiento urbano y ciudades del futuro. *El Peruano.* https://elperuano.pe/noticia/90177-crecimiento-urbano-y-ciudades-del-futuro

Flask. (s. f.). *Flask. User's guide.* https://flask.palletsprojects.com/en/3.0.x/

FlaskSQLalchemy. (s. f.). *Flask SQLalchemy. User guide*. https://flask-sqlalchemy.palletsprojects.com/en/3.1.x/

Habala, O., Bobák, M., Šeleng, M., Tran, V., & Hluchý, L. (2023). Architecture of a function-as-a-service application. *Computing and informatics*, *42*(4), 878-895. https://doi.org/10.31577/cai_2023_4_878

Hussein, M. K., Mousa, M. H., & Alqarni, M. A. (2019). A placement architecture for a container as a service (CaaS) in a cloud environment. *Journal of cloud computing, 8*, 7. https://doi.org/10.1186/s13677-019-0131-1

Kratzke, N., & Quint, P-C. (2017). Understanding cloud-native applications after 10 years of cloud computing. A systematic mapping study. *Journal of systems and software*, *126*, 1-16. https://doi.org/10.1016/j.jss.2017.01.001

Larman, C. (1999). *UML y patrones: una introducción al análisis y diseño orientado a objetos*. Prentice Hall.

Mell, P., & Grance, T. (2012). *The NIST definition of cloud computing.* National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145

Merelli, I., Fornari, F., Tordini, F., D'Agostino, D., Aldinucci, M., & Cesini, D. (2019). Exploiting Docker containers over grid computing for a comprehensive study of chromatin conformation in different cell types. *Journal of parallel and distributed computing*, *134,* 116-127. https://doi.org/10.1016/j.jpdc.2019.08.002

SQLAlchemy. (s. f.). *The Python SQL toolkit and object relational mapper*. https://www.sqlalchemy.org/

Stine, M. (2015). *Migrating to cloud-native application architectures*. O'Reilly Media.

Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., & Bohnert, T. (2017). Self-managing cloud-native applications: design, implementation, and experience. *Future generation computer systems*, *72*, 165-179. https://doi.org/10.1016/j.future.2016.09.002

Vega, É. (2021, 5 de mayo). Crecimiento inmobiliario vertical de Lima muestra comportamientos diferenciados. *El Comercio*. https://elcomercio.pe/economia/negocios/crecimiento-inmobiliario-vertical-de-lima-muestra-comportamientos-diferenciados-mercado-inmobiliario-capeco-tinsa-nczenoticia/?ref=ecr