RESUMEN

ARQUITECTURA DE BACK END CON AMAZON WEB SERVICES (AWS) PARA SISTEMAS ESCOLARES

por

Rossemery Elizabeth Brañes Vilchez

Asesor: Saulo Hernández Osoria

RESUMEN DE TESIS DE MAESTRÍA

Universidad de Montemorelos

Facultad de Ingeniería y Tecnología

Título: ARQUITECTURA DE BACK END CON AMAZON WEB SERVICES PARA SISTEMAS ESCOLARES

Investigador: Rossemery Elizabeth Brañes Vilchez

Asesor principal: Saulo Hernández Osoria, Maestría en Ciencias Tecnología Informática

Fecha de culminación: Mayo de 2019

Problema

Las instituciones educativas no cuentan con un sistema que pueda ser flexible a sus necesidades, ajustándose a procesos internos que sean propios de ellos, como son las instituciones educativas cristianas. Las arquitecturas Back End que se están utilizando en los sistemas que manejan no cuentan con una plataforma escalable y flexible, impidiendo que se tenga un desempeño óptimo.

Método

Para el desarrollo de la arquitectura Back End propuesta, se optó por utilizar los servicios de Amazon Web Services (AWS), después de haber hecho una comparación con los otros competidores que ofrecen servicios similares. Se realizaron distintas

pruebas que confirman la flexibilidad y escalabilidad de los servicios, reforzando la arquitectura con los servicios que brinda AWS, brindando una seguridad optima en la nube con respecto a los datos. Todo ello conlleva a una arquitectura robusta que pueda ser compatible con muchas aplicaciones y programas.

Resultados

Para generar los resultados, se preparó todo el ambiente para la arquitectura Back End en Amazon Web Services, teniendo diversos escenarios. Entre los más relevantes son las siguientes: se mantuvo activo el servicio REST, para acceder a la información con un servicio; como se está trabajando con microservicios por este medio, la comunicación es más rápida que un XML. La flexibilidad y escalabilidad de las instancias y servicios de AWS es admirable, dividiendo la carga de trabajo en otras instancias que se crearon; de este modo se redujo el tiempo de respuesta. La seguridad de la arquitectura propuesta es óptima, ya que gracias a los servicios que brinda AWS se puede hacer una infraestructura impenetrable, evitando filtraciones de información o actividades sospechosas.

Conclusiones

La arquitectura Back End que se propuso con AWS tiene un alto nivel de escalabilidad y flexibilidad; para adaptarse a diversos entornos y cargas de trabajo, se tienen estándares y protocolos de seguridad para alertar al administrador en caso de que haya algo fuera de lugar. Por último, la robustez de esta arquitectura se consolida con las características antes mencionadas y la amplia gama de servicios que ofrece AWS para ser utilizado en el ambiente creado.

Universidad de Montemorelos

Facultad de Ingeniería y Tecnología

ARQUITECTURA DE BACK END CON AMAZON WEB SERVICES (AWS) PARA SISTEMAS ESCOLARES

Tesis
presentada en cumplimiento parcial
de los requisitos para el grado de
Maestría en Ciencias Computacionales

por

Rossemery Elizabeth Brañes Vilchez

Mayo de 2019

ARQUITECTURA DE BACK END CON AMAZON WEB SERVICES (AWS) PARA SISTEMAS ESCOLARES

Proyecto

presentado en cumplimiento parcial de
los requisitos para el grado de
Maestría en Ciencias
Computacionales

por

Rossemery Elizabeth Brañes Vilchez

APROBADO POR LA COMISIÓN:

M.C. Saulo Hernández Osoria
Asesor principal

Mtro. Rudy Oswaldo Dzul Ramírez
Asesor externo

M.C. Alejandro Walterio García Mendoza
Miembro

Dr. Ramón Andrés Díaz Valladares
Director de Posgrado e Investigación

M.C. Daniel Arturo Gutiérrez Colorado
Miembro

02 de Mayo 2019
Fecha de aprobación

DEDICATORIA

Primeramente, dedico este proyecto a Dios, porque me acompañó en toda esta travesía desde el primer día que llegué a este país. “Pon en manos del Señor todas tus obras, y tus proyectos se cumplirán” (Proverbios 16:3).

A mis padres, Olga Vilchez Gago y Eliud Brañes Ruíz, por brindarme su apoyo incondicional; a pesar de la distancia, siempre estuvieron pendientes de mí.

A mi hermano menor, Iván Brañes Vilchez, que me motiva a seguir adelante y me brinda su afecto a la distancia.

A todas mis amistades que me apoyaron con sus palabras de ánimo y reconfortaron mi ser cuando lo necesitaba. Una vez más puedo comprender que la amistad es sin fronteras.

TABLA DE CONTENIDO

LISTA DE FIGURAS

RECONOCIMIENTOS

Al único Dios, porque sin él nada soy. Reconozco su grandeza y misericordia que tuvo conmigo en cada aspecto de mi vida. Dedico mi vida a su servicio y puedo ver sus bendiciones en mi ser.

Al maestro Saulo Hernández Osoria, mi asesor principal, por su paciencia, afecto, consejería y orientación en todo el tiempo que duró esta investigación.

Al maestro Daniel Gutiérrez Colorado, coordinador del posgrado, por su constante preocupación y apoyo para terminar esta investigación.

Al ingeniero Kenny Aguirre Orosco, amigo de la universidad, que me apoyó con sus sugerencias y discernimiento con respeto al tema de investigación.

A mi querida iglesia “Huaycán Central” (Perú), en la que sus feligreses preguntaban acerca de este proyecto y que cuándo volveré a estar con ellos; motivándome a seguir adelante.

CAPÍTULO I

DIMENSIÓN DEL PROBLEMA

Planteamiento del problema

El problema que afrontan muchas instituciones educativas es la necesidad de un software que supla sus necesidades para facilitar el trabajo administrativo y educativo. Existen numerosos aplicativos que tienen diversas herramientas y funciones, pero no cuentan con una arquitectura Back End que sea flexible para soportar completamente las necesidades de la institución, donde el performance es inferior a los recursos que se requieren, la flexibilidad y robustez no se adaptan a la arquitectura, impidiendo que se tengan un desempeño óptimo. Enfrentando restricciones, con las aplicaciones que actualmente utilizan, al no ser flexibles en ciertos procesos propios de la institución, ya que cada institución educativa tiene procesos particulares como son los de las instituciones educativas cristianas.

En muchos de estos sistemas, las arquitecturas Back End no están propiamente desarrolladas para el servicio que ofrecen; muchos de ellos son adaptados internamente para su funcionamiento parcial, dejando de lado la arquitectura de software, olvidando que esta es la guía principal y responsable directo del éxito del proyecto (Meliá, 2007). Se podría decir que es el cimiento de vital importancia para el desarrollo de un software, agregando características que pueden dar mayor robustez a la arquitectura, como la flexibilidad, la escalabilidad y la seguridad (Tahuiton Mora, 2011).

Declaración del problema

El problema a investigar en este estudio fue el siguiente:

Este proyecto surge de la necesidad de instituciones educativas que necesitan una plataforma que se amolde a su entorno de trabajo. Existen diversos software que están basados en una arquitectura de software cliente-servidor, que no posee una eficiente escalabilidad y flexibilidad, impidiendo amoldarse a otras plataformas de trabajo e interactuar con tecnologías modernas.

Al conocer la realidad de estas instituciones educativas, surge la necesidad de desarrollar una nueva arquitectura de software que les permita interactuar con diferentes aplicaciones, acoplándose a los servicios y ajustándose a los requerimientos actuales y futuros de las instituciones.

Justificación

El proyecto surge de la necesidad de que las instituciones educativas tengan una plataforma que pueda soportar sistemas informáticos para sus labores administrativas; de este modo, se optimizarían los procesos, generando eficiencia entre los usuarios que requieren la información.

Es por ello que el proyecto pretende apoyar y servir ante esta necesidad de las instituciones educativas, brindando una plataforma de desarrollo. Esta arquitectura de Back End debe ser robusta y adecuada para soportar una gran cantidad de información y transacciones en tiempo real; además, debe tener seguridad ante las personas maliciosas que puedan robar o manipular la información y ser escalable con la capacidad de incrementar el proyecto a grandes proporciones, según se requieran los recursos.

Objetivos

Para la presente investigación se plantearon los siguientes objetivos:

Objetivo general

El objetivo general del estudio fue el siguiente:

Proponer una arquitectura de software tan robusta que soporte el desarrollo de diversos tipos de aplicaciones; que sea segura para proteger la información por medio de esquemas de seguridad; que sea flexible y escalable para apoyar el desarrollo continuo; de esta forma, será la base para el desarrollo de un sistema escolar para instituciones educativas.

Objetivos específicos

Para cumplir el objetivo general, fue necesario desglosarlo en los siguientes objetivos específicos:

1. Definir una arquitectura Back End para sistemas escolares.
2. Conocer qué tipo de arquitectura de software se utilizará.
3. Conocer los beneficios de Amazon Web Services (AWS).
4. Seleccionar los servicios que se utilizarán de AWS.
5. Utilizar las tecnologías y servicios de AWS en la arquitectura propuesta.

Hipótesis

En esta investigación se plantea la hipótesis siguiente:

Es posible definir una arquitectura de Back End robusta, flexible y segura, utilizando AWS como plataforma de soporte para sistemas escolares.

Limitaciones

Algunas limitaciones de esta investigación fueron las siguientes:

Este proyecto no cuenta con un presupuesto; por ese motivo, se utilizó la versión gratuita de Amazon Web Services (AWS) que es de un año. Se tuvieron ciertas restricciones, como la cantidad de servidores en la nube (EC2), la transferencia de datos, el límite de espacio en la base de datos, el rastreo de registros, entre otros. Como la facturación es mensual, esto permite que el tiempo de usar un servicio se restablezca cada inicio de mes. Por ejemplo, se tienen 750 horas al mes para el uso de la base de datos (RDS) y si se llevan 600 horas, al empezar un nuevo mes, este restablece el valor a 750 horas.

Delimitaciones

Se presentan algunas delimitaciones en esta investigación:

Este trabajo de tesis pretendió proponer una arquitectura Back End de software enfocado a sistemas escolares con respecto a la administración de instituciones educativas, mas no se desarrolló el software completo. Se trata de una propuesta, que será evaluada y según su aceptación se definirá el desarrollo posteriormente.

Definición de conceptos

A continuación, se definen algunos de los términos utilizados en esta investigación, que son los siguientes:

API: es la sigla en inglés de Application Programming Interface, que es interfaz de programación de aplicaciones, definiéndolo como un conjunto de tareas rutinarias para el acceso de funciones de un software (Definición De, 2017).

Arquitectura: es una técnica de diseñar y construir todo tipo de espacios en estructuras, satisfaciendo las necesidades de un individuo (Braude, 2000).

AWS: es la sigla de Amazon Web Services, que es una plataforma de servicios en la nube que ofrece potencia de cómputo.

*Back End: e*s la parte posterior del sistema, donde yacen los algoritmos y el código bruto del sistema (Garfias Suárez, 2012).

*Bifurcación: e*n términos informáticos, recibe este nombre la acción de crear un proyecto en otra dirección que el proyecto principal.

CPU: es la sigla en inglés de central processing unit, que es la unidad de procesamiento central, siendo esta la parte central de toda computadora, ya que es la que cumple la tarea de procesamiento de todas las funciones, así como también de almacenamiento de la información (Definición De, 2017).

Escalabilidad: es la habilidad de reaccionar antes una situación nueva, donde pueda adaptarse sin perder la calidad, teniendo un crecimiento continuo.

Flexibilidad: es la capacidad que tiene un objeto o persona para amoldarse sin el riesgo de que pueda romperse.

Framework: es un ambiente laboral que facilita el desarrollo del trabajo.

Front End: es la parte externa del sistema; se podría decir una interfaz gráfica del sistema, donde puede interactuar con el usuario (Gunnulfsen, 2013).

Hardware: se refiere al conjunto de los componentes que conforman la parte material, externa y física de una computadora (Real Academia Española, 2017).

HTML: es la sigla en inglés de Hipertext Markup Language, que es lenguaje de marcas de hipertexto, siendo un lenguaje que se utiliza fundamentalmente en el desarrollo de páginas web (Definición De, 2017).

IaaS: es la sigla en inglés de Infrastructure as a Service; es una infraestructura que tiene la función de proporcionar acceso a recursos informáticos situados en un entorno virtualizado a través de una conexión pública (Ayyapazham y Velautham, 2017).

IEEE: es la sigla en inglés de The Institute of Electrical and Electronics Engineers, que es Instituto de Ingeniería Eléctrica y Electrónica. Es una organización mundial dedicada a estandarizar y normalizar las áreas técnicas (IEEE, 2000).

Microservicios: son pequeños servicios que se ejecutan de forma autónoma y comunicándose entre sí (Daza Corredor, Parra Peña y Espinosa Rodríguez, 2015).

PaaS: es la sigla en inglés de Platform as a Service, que es la plataforma como servicios, que pertenece a una categoría de servicios en la nube, que proporciona una plataforma y entorno para permitir a los desarrolladores crear aplicaciones y servicios que funcionen a través de internet (Jia bin, Shi wei y Wei chuan, 2018).

Prototipo: puede definirse como un modelo o tipo de alguna creación; frecuentemente se nombra así al primer desarrollo final, referenciándolo para futuros modelos de la misma categoría.

Robustez: cualidad de robusto, que hace referencia a vigoroso, fuerte.

SaaS: es la sigla en inglés de Software as a Service, que es el software como servicio, que describe cualquier servicio en la nube en el que los consumidores pueden acceder a aplicaciones de software a través de internet (Czarnul, 2013).

Servicio: es un recurso que representa las tareas y funcionalidades de entidades, como proveedores o la misma persona.

Servidor: es un objeto que sirve para almacenar información y conectarse por medio de la red (Real Academia Española, 2017).

SOAP: es la sigla en inglés de Simple Object Access Protocol, que es un protocolo estándar de intercambio de mensajes por redes de computadoras (Stankevicius Affiliation, 2013).

Software: es el equipamiento lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes tangibles, que son llamados hardware (Real Academia Española, 2017).

Web: vocablos en inglés como red o malla, que es utilizado en el contorno tecnológico para llamar a una red informática (Definición De, 2017).

Web Services: que traducido sería servicio web, que es una tecnología que utiliza protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

XML: es la sigla en inglés de Extensible Markup Language, que es el Lenguaje de Marcas Extensibles, y se trata de un lenguaje que utiliza características para decir algo en otro lenguaje.

CAPÍTULO II

MARCO TEÓRICO

Arquitectura de software

Durante décadas, los ingenieros de software han creado sus propios diseños, empezando de cero o reutilizando diseños de otros. Hoy en día, esa situación ha cambiado, al surgir una nueva disciplina conocida como arquitectura de software, que utiliza un lenguaje común a todos los ingenieros de software para detallar diseños de alto y bajo nivel (Braude, 2000).

De acuerdo con The Institute of Electrical and Electronics Engineers (IEEE, 2000), la arquitectura de software es la organización fundamental de un sistema complementado con sus componentes, con las relaciones entre ellos, con el ambiente, así como con los principios que orientan su diseño y evolución. La arquitectura de software es también conocida como arquitectura lógica.

La arquitectura se ocupa fundamentalmente de la plataforma de un sistema en sus elementos constitutivos y sus interrelaciones para lograr un propósito determinado (Sangwan, 2014).

Considerando los enunciados anteriores, se puede afirmar que cada sistema posee una arquitectura que se diseñó para alcanzar su óptimo funcionamiento, cumpliendo con las expectativas para las cuales fue creado. Por otro lado, los sistemas no solo se limitan al hardware; es necesario también incluir a personas, instalaciones,

software, políticas y documentos, ya que todos estos elementos son requeridos para producir el resultado esperado e incorporar los diferentes atributos de calidad. El diseño de la arquitectura se centra en la descomposición de un sistema en componentes y sus interacciones, para satisfacer los requisitos funcionales y no funcionales.

Recogiendo lo más importante, una arquitectura de software es la estructura de un programa que también abarca el flujo de datos, definiendo las interacciones entre todos los elementos. La arquitectura de software se compone de un modelo legible de la estructura del sistema y de las relaciones entre sus componentes.

Por ello, si la arquitectura está correctamente descrita, permite evidenciar los siguientes puntos:

1. La documentación de la estructura y propiedades del sistema en un lenguaje claro para todas las partes implicadas en el desarrollo del mismo.

2. La evaluación de las decisiones de diseño desde las etapas más tempranas posibles del proceso de desarrollo y durante la evolución del mismo.

3. La reutilización de modelos y componentes de software y el uso de componentes comerciales.

4. El prototipado evolutivo y el crecimiento incremental.

Por otro lado, se tienen los patrones arquitectónicos que brindan solución a problemas y definen un esquema, capturando elementos esenciales para una arquitectura de software. Diversas arquitecturas pueden utilizar el mismo patrón que describe un problema en particular y brinda su solución.

De acuerdo con Jiménez Torres, Tello Borja y Ríos Patiño (2014), estos patrones, se pueden categorizar o agrupar en la composición de patrones, las historias con

patrones, las secuencias y, por último, las colecciones. A continuación se hace una descripción de cada uno de ellos.

1. La composición de patrones**:** este se enfoca en solucionar un problema en específico que no impacta en la arquitectura de software; por ello no necesariamente tienen que interactuar los patrones entre sí.

2. Historias de patrones**:** este es un reporte o diario que muestra al usuario las soluciones aplicadas en los diversos problemas presentados, en qué orden se aplicaron los patrones y el tiempo que llevó la solución.

3. Secuencias de patrones**:** este considera un histórico de soluciones que puede generalizar un procedimiento a otros sistemas que comparten estructuras similares, revisando las restricciones y el diseño que tengan para su aplicación.

4. Colección de patrones**:** en este tópico se categorizan los patrones de forma personalizada; dicho de otra manera, es como un catálogo que no impacta a la arquitectura, pero brinda el conocimiento a los usuarios de sus funcionalidades.

Se expresa que la arquitectura de software está íntimamente relacionada con patrones arquitectónicos; se diría que el conjunto de estos patrones forma un marco de referencia para crear la arquitectura de software, que es indispensable en la creación de un producto de software; es por ello que no se debe escatimar ni pasar por alto el más mínimo detalle para minorizar los riesgos (Suárez y Gutiérrez, 2016).

Esto permite a todo el equipo de desarrollo tener una línea base para trabajar, compartiendo las restricciones que tenga el proyecto y los objetivos a alcanzar (Meaurio y Schmieder, 2013). Por lo tanto, es necesario realizar evaluaciones periódicas a la arquitectura del software que se está proponiendo, haciendo pruebas, generando diversos

escenarios que puedan surgir; revisando su robustez y escalabilidad ante los escenarios propuestos (Burgos Coronel, 2015).

Meliá (2007) trabajó con aplicaciones web, presentando la problemática de que estas eran cada vez más complejas en estructura, contenido, interfaz y comportamiento. Presentó la carencia de aspectos arquitectónicos para capturar la distribución de los componentes, al faltar la trazabilidad de modelos para métodos funcionales hasta la implementación. Propuso un proceso de desarrollo de arquitectura de software para aplicaciones web. Meliá resalta la definición de la arquitectura de software como responsable directo para guiar el diseño y evolución mediante el uso de patrones arquitectónicos.

Sin embargo, la evolución del software y el cambio continuo de las tecnologías ha generado la necesidad de modificar una arquitectura según las necesidades que se presenten; para ello se puede utilizar la ingeniería inversa como una solución que permita recuperar parte de la arquitectura de software realizada.

Este es uno de los mayores retos para la arquitectura, ya que a su vez se tiene un enfoque de ingeniería inversa que permite reconstruir vistas arquitectónicas, plasmando los conceptos de propiedades fundamentales a procesos relacionados y su evolución. Considerando una opinión arquitectónica, se puede expresar que dentro de un marco de trabajo se establecen los convenios para la construcción, interpretación y utilización de vistas arquitectónicas para enfatizar los intereses definidos de un sistema. Es decir, en esta etapa se define la arquitectura de software del sistema, teniendo en cuenta las necesidades de ser adaptable a los intereses que se definieron para su construcción (Monroy, Arciniegas y Rodríguez, 2016).

Tahuiton Mora (2011) expresa que la arquitectura de software es un punto clave para el desarrollo del software, que tiene como objetivo desarrollar sistemas de software inteligibles, robustos, estructurados, que logren la reusabilidad y la escalabilidad.

Las arquitecturas hibridas son arquitecturas donde se han ido integrando nuevos componentes de diferente naturaleza y orígenes, como desarrollo de terceras personas, código abierto, componentes gratuitos, servicios web, entre otros. Este tipo de enfoque se ha generado por personas oportunistas y estrategias organizacionales que, en su momento, fueron una solución inmediata al problema con una mínima inversión. Sin embargo, tuvieron ciertos problemas con la selección, adaptación e integración de los componentes requeridos para la arquitectura del sistema, porque esperaban que la estructura de esta arquitectura fuera genérica y funcionara adecuadamente con los componentes que se iban a integrar (Carvallo Vega y Franch Gutiérrez, 2009).

Componentes

La arquitectura de software tiene diversos componentes que le permiten generar el ambiente adecuado para el desarrollo del software, con los servicios que estarán en contacto; está compuesta por clientes, servidores, bases de datos, tuberías y filtros, repositorios, Front End y Back End. A continuación, se expone cada uno.

Servidor

Se llama servidor a un equipo (máquina física integrada en una red), que ofrece su servicio a los programas (clientes) que llegan en múltiples peticiones, encargándose de almacenar archivos y sirviendo de información donde se solicita (Castedo et al.,

2008). Algunos servidores pueden ser una simple computadora; otros, unos ordenadores muy potentes.

Existen tipos de servidores para servicios web, de archivos, correos electrónicos, bases de datos, proxy, DNS, entre otros.

Cliente

El cliente puede ser un usuario que interactúa con el Front End del sistema, enviando peticiones para obtener información; o se le puede llamar cliente a ciertos programas que interactúan con el servidor, solicitando respuesta para continuar con un proceso determinado. Este tipo de “clientes” se programan para ejecutarse en un determinando tiempo o se activan según sigan una cadena de secuencia (Duarte Vega, 2016).

Base de datos

Se puede definir la base de datos (BD) como un almacén (servidor), donde se permite guardar la información de forma organizada para poder utilizarla. Dicho de otra manera, es un conjunto de información relacionada que se encuentra de manera estructurada y agrupada.

La BD se compone de tablas que tienen columnas y filas donde se almacena toda la información.

Una BD debe contar con las siguientes características: tener accesos concurrentes para los usuarios, integridad en los datos, seguridad de acceso y auditoría continua, respaldo de la BD por medio de backups para recuperar la información en cualquier momento e independencia lógica y física de los datos (Díaz Rodríguez, 2015).

Tuberías y filtros

Tuberías es una cadena de procesos que están conectados de tal forma que la salida de cada elemento es la entrada del próximo proceso como una cadena; es decir, cada elemento es un eslabón; este se conecta con el anterior y el nuevo con el anterior; así sucesivamente, formando una cadena de procesos que permite la comunicación y sincronización de los mismos. Esto es aplicado en la transformación de los datos de entrada que, mediante una serie de operaciones, son convertidos y transformados en datos de salida.

Los filtros son componentes que trasmiten datos por medio de las tuberías, trabajando de manera independiente; por este motivo, los filtros no necesitan conocer el funcionamiento de los procesos; solo se preocupan de la entrada y salida de los componentes (Matos Arias y Silega Martínez, 2013).

Repositorios

Los repositorios son espacios centralizados, donde se encuentran los archivos informáticos de manera organizada que contienen un conjunto de datos o software; es decir, son servidores centralizados donde se almacenan paquetes de información, con la finalidad de proveer recursos directos para suplir alguna petición.

Front End

Todo proyecto de software tiene dos divisiones muy marcadas que están estrechamente relacionadas con el funcionamiento ideal del software (Front End y Back End). La Figura 1 muestra la idea grafica de la diferencia entre ellos.

El Front End es la parte frontal del software; dicho de otra manera, es un conjunto de páginas que podrían estar presentados en la Web, móvil o aplicación escritorio GUI (Graphical User Interface); el mismo que interactúa directamente con el usuario final a través de un ordenador, celular u otro dispositivo, por medio de formularios y ventanas emergentes, entre otros (Gunnulfsen, 2013).

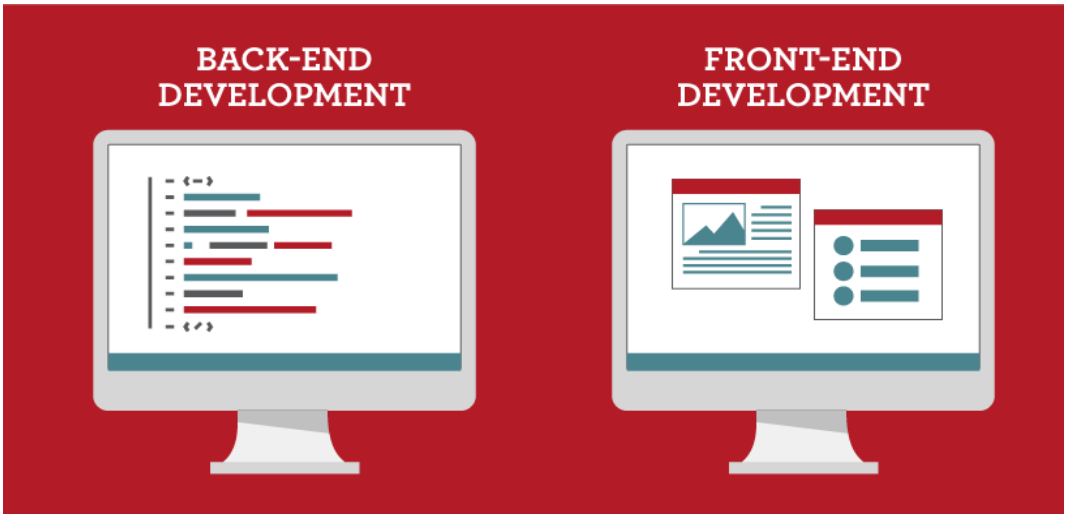


Figura 1. Back End y Front End development (Engard, 2017).

Para el Front End se elabora el diseño de todas las funciones que realiza el software: pestañas, formularios, consultas, reportes, entre otros, con el fin de mostrar el contenido e información solicitada en una interfaz creativa, dinámica y fácil de entender para el usuario. Para ello se realizan bocetos con la creación de la interfaz para escoger el indicado; posteriormente, el boceto final se transforma en prototipos, teniendo en cuenta la gama de colores, tipografía, iconografía, calidad de imágenes y estructura del sitio.

Los lenguajes más usados del Front End son HTML, CSS y JavaScript; de ellos se generaron diversos frameworks que ayudan al desarrollador a aplicar diversos estilos de acuerdo con su necesidad, teniendo una variedad de librerías en qué apoyarse; entre ellos resaltan React, Redux, Angular, Bootstrap, Foundation, entre otros.

Como tipo de herramientas de diseño gráfico se encuentran el Ilustrator y Photoshop. Estas son muy utilizadas para la creación de los prototipos que va a tener el software. Muchas personas afirman que el Front End es más importante que el Back End, ya que la primera impresión es la que cuenta. Si fuera así, solo mostraría de manera gráfica las funciones que realizará el software, pero no funcionaría, porque sería como un “cascaron” sin funcionalidad alguna; por ese motivo, tiene que estar complementado con el Back End, que es la parte interna y funcional del sistema para tener un óptimo desempeño (Valdivia Caballero, 2016).

Back End

Se podía decir que el Back End es el lado opuesto al Front End. Se encarga de la parte lógica, parte no visible para el cliente, ya que interactúa estrechamente con el desarrollador, dando funcionabilidad a los elementos que se definieron en el Front End.

El Back End trabaja directamente con lenguajes de programación. Cada uno de estos lenguajes requieren de una lógica, que a su vez se encarga de optimizar los recursos, revisar la seguridad, los estándares de calidad y la accesibilidad, entre otros (Garfias Suárez, 2012). El usuario no contempla el Back End, ya que solo interactúa con el Front End, pero existe un código que está funcionando internamente para mostrarle al usuario lo que requiere, como una búsqueda, un reporte, una modificación de datos, entre otros.

Hay diversos lenguajes de programación que se pueden utilizar en el Back End. Los más utilizados son Java, C#, Ruby, PHP y Python; cada uno de estos cuentan con diferentes frameworks. Las herramientas que se utilizan en un ambiente de desarrollo pueden ser compiladores, editores de código, depuradores para revisión de errores, seguridad, gestores de bases de datos, entre otros.

Algunas personas piensan que el Front End es más importante que el Back End y viceversa. Ambos tienen funcionalidades distintas, pero cada uno está estrechamente relacionado con el otro. Es por ello que la comunicación entre estas dos áreas es vital y se deben tener conocimientos básicos de ambas partes para poder acoplar las funcionalidades del software o sitio web (Garfias Suárez, 2012).

Galicia García, Ortega Ginés y Curioca Varela (2015), en Tehuacán, Puebla, desarrollaron un Back End que tuviera la capacidad de adecuarse a diferentes Front End de diversos portales web, permitiendo crear estructuras y personalizar campos para las secciones web. Para ello, se realizó un muestreo no probabilístico de diferentes sitios web en la región, según su clasificación. Para el desarrollo del Back End se utilizó la tecnología LAMP (Linux, Apache, Mysql y PHP), teniendo en cuenta la compatibilidad con la mayoría de los desarrollos web y cumpliendo con los estándares de seguridad, flexibilidad y adaptabilidad. Como resultado, se implementó en ocho portales web, que se encuentran funcionando en diferentes ámbitos, como en gobiernos municipales, servicios y venta de productos, portales de congresos, escuelas y PYMES.

Tipos de arquitectura

En esta sección se enlistan los tipos de arquitectura de software, describiendo sus características principales, y sus ventajas y desventajas, que se explican a continuación.

Arquitectura monolítica

La arquitectura monolítica es un modelo tradicional unificado para el diseño de un programa de software. Monolítico significa compuesto todo de una pieza. El software monolítico está diseñado para ser autónomo; los componentes del programa están interconectados e interdependientes en lugar de estar débilmente acoplados, como es el caso de los programas de software modulares (Salazar Hernández, 2017).

Las aplicaciones monolíticas se organizan en grupos funcionales, incluyendo todos los aspectos respecto del proyecto, del procesamiento y del almacenamiento de la información. Esta estructura es de los primeros sistemas operativos instituidos fundamentalmente por un solo programa compuesto por un conjunto de rutinas entrelazadas, de tal forma que cada una puede llamar a cualquier otra. Puede ser implementada en una computadora central o mainframe; desde una terminal de servicio se puede monitorear y recibir los resultados que devuelva (López Hinojosa, 2017).

Entre sus características se encuentran las siguientes: realiza una construcción de componentes compilados al mismo tiempo; no cuenta con privilegios y protecciones para ingresar a rutinas o utilizar recursos de la computadora, es desarrollada a la medida del usuario; por este motivo es eficiente y rápida en la ejecución y se construye el programa final a base de módulos compilados separadamente, que se unen a través de un linker.

La ventaja de esta arquitectura es que es muy eficiente, ya que se producen pocos cambios en el contexto. Y entre las desventajas resaltan la carencia de flexibilidad para soportar diferentes ambientes de trabajo o tipos de aplicaciones. Son más complejas a la hora de depurar errores (Gómez Fermín y Moreno Poggio, 2014)

Arquitectura cliente-servidor

La arquitectura cliente-servidor comprende que un cliente realiza peticiones a un programa o servidor, esperando una respuesta; esto se puede aplicar en un sistema operativo multiuso distribuido con una red de computadoras; además, este tipo de arquitectura es el soporte de la mayor parte de la comunicación por redes y que comprende las bases sobre las que están construidos los algoritmos distribuidos (Duarte Vega, 2016).

Las características que presenta esta arquitectura son las siguientes: utiliza protocolos asimétricos, donde el servidor se limita a escuchar, en espera de que el cliente inicie una solicitud; el servidor ofrece sus recursos (lógicos y físicos) a una cantidad variable y diversa de clientes que serán utilizados por estos. Estos servicios estarán encapsulados, para ocultar a los clientes los detalles de su implementación; el que se encuentren centralizados los servidores, en efecto, facilita la integridad y el mantenimiento de los datos y programas; los sistemas se encuentran débilmente acoplados, interactuando por envío de mensajes y con facilidad de realizar la escalabilidad, añadiendo nuevos clientes a la infraestructura (escalabilidad horizontal) o incrementando la potencia del servidor o servidores (escalabilidad vertical) (Castedo et al., 2008).

Entre las ventajas que tiene esta arquitectura es la facilidad de integración entre sistemas diferentes y el poder compartir información. Además, favorece el uso de interfaces gráficas interactivas. Su estructura modular facilita la integración de nuevas tecnologías y el crecimiento de la infraestructura, aprovechando la escalabilidad en las soluciones.

No obstante, las desventajas que presenta esta arquitectura son las dificultades

en realizar el mantenimiento, ya que implica diversas interacciones con distintos proveedores, obstaculizando el diagnóstico de fallas; es por ello que hay que tener estrategias para el manejo de errores y para mantener la consistencia de los datos.

Otra desventaja es que no cuenta con las suficientes herramientas para la administración y el ajuste de los sistemas. El desempeño (performance) puede presentarse por congestión en la red y dificultad de tráfico de datos, entre otros (Gómez Fermín y Moreno Poggio, 2014).

Arquitectura orientada a servicios (SOA)

La Arquitectura Orientada a Servicios utiliza los servicios para dar soporte a los requisitos de negocio, permitiendo crear sistemas escalables, que apoyan en el rendimiento y reducen los costos, basándose en las tecnologías de la información (TI); de este modo se mejoraron los procesos del negocio y optimización del tiempo (Zhang et al., 2018).

El núcleo de esta arquitectura está determinado por componentes independientes, facilitando su mantenimiento si alguno de ellos deje de funcionar, que se comunican por medio de mensajes (Daza Corredor et al., 2015).

Entre sus características se mencionan las siguientes: la interacción que tiene con los servicios es desacoplada, ya que cada servicio SOA es independiente y puede ser reemplazado sin perjudicar la aplicación; los servicios son autónomos, ya que pueden ser desarrollados, instalados y darles mantenimiento de forma independiente y los servicios utilizan protocolos (HTTP), formatos y estándares de seguridad basados en políticas (Camacho, Chamorro, Sanabria, Caicedo y García, 2017).

Las ventajas de esta arquitectura son las siguientes: permite la reutilización de los servicios comunes por medio de interfaces, incrementando oportunidades de negocios y reduciendo costos; como los servicios son autónomos, esto provee desacople y abstracción, facilita la integración con otros sistemas, las aplicaciones son más productivas y flexibles para el desarrollo, las aplicaciones son más seguras y manejables, minimizando el riesgo de inactividad y pérdida de datos y se pueden ejecutar muchas instancias en un mismo servicio en otros servidores al mismo tiempo; de este modo aumenta la escalabilidad y disponibilidad de la arquitectura (Arias Orizondo y Estrada Senti, 2013).

Por otro lado, entre sus desventajas, se considera que los servicios intercambian mensajes para realizar tareas específicas y pueden llegar millones de mensajes, teniendo un gran desafío en gestionar una gran cantidad de servicios. A su vez, el costo de inversión es muy alto y requiere un cambio organizacional; este es un gran desafío para la organización.

Duarte Vega (2016) propuso una arquitectura altamente escalable para ser implementada con un servicio web, utilizando las tecnologías gratuitas de Microsoft. Para esta propuesta, utilizó la combinación de patrones arquitectónicos con la finalidad de obtener facilidad en el desarrollo, sencillez y adaptabilidad a las tecnologías que se utilizaron. La arquitectura de software propuesta hereda las características relevantes de la arquitectura de software orientada a servicios (SOA), Bus de Servicios Empresariales (ESB) y Microservicios; que puede ser consultada por el cliente a través de aplicaciones móviles, portales u otros servicios web (ver Figura 2).

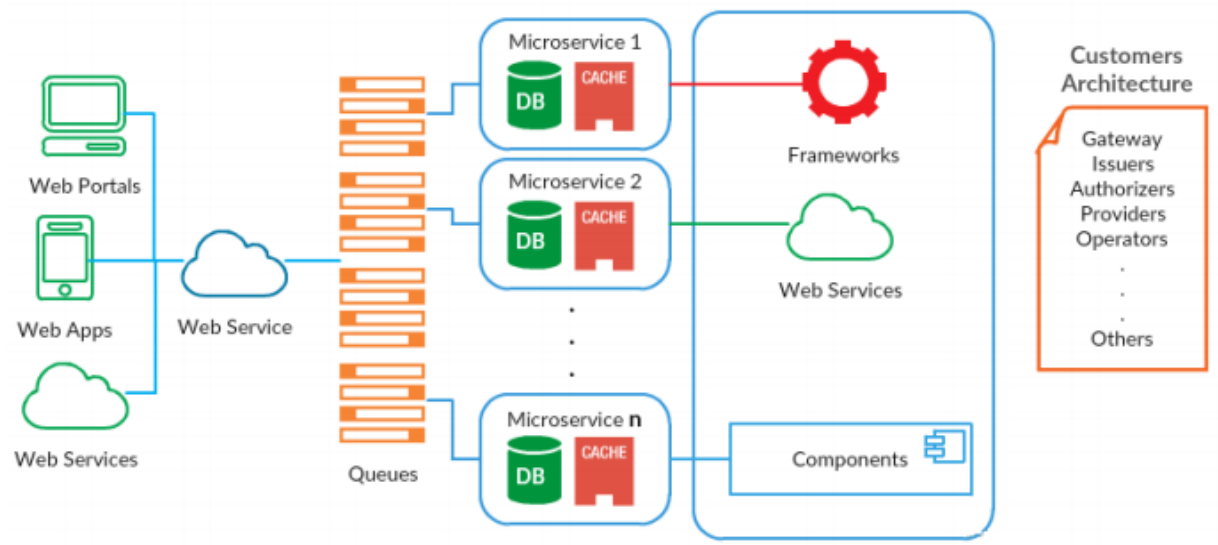


Figura 2. Arquitectura propuesta para un servicio web completo (Duarte, 2016).

Arquitectura de microservicios

Para poder comprender la arquitectura de microservicios, se tiene que definir qué son los microservicios**.** Como su nombre propiamente lo dice, son pequeños servicios en los que cada uno se ejecuta de forma autónoma y se comunica entre sí. Gracias a su sencilla escalabilidad, este método de arquitectura se considera especialmente adecuado cuando se procura la compatibilidad con un amplio sector de diferentes plataformas (web, móvil, entre otras). Básicamente, el desarrollo de un proyecto que se base en este método conforma una aplicación o herramienta mediante la relación de diversos servicios independientes que se despliegan según se vayan necesitando.

Los microservicios permiten responder las demandas con mayor rapidez, ya que el desarrollo y la adaptación son más ágiles para las aplicaciones, siendo un enfoque

diferente y nuevo con respecto al desarrollo de software tradicional. Actualmente, distintas partes del equipo de desarrollo pueden trabajar simultáneamente en el mismo proyecto y entregar un valor a los clientes de inmediato (Gadge y Kotwani, 2018). Se puede definir que la arquitectura de microservicios consiste en construir una arquitectura con un conjunto de servicios autónomos y pequeños, que se ejecutan independientemente, comunicándose con mecanismos ligeros (López Hinojosa, 2017).

Las características que tiene esta arquitectura son las siguientes: trabaja con servicios pequeños e independientes, acoplándose de forma flexible; cada servicio es un código base independiente, que puede administrarse por un equipo de desarrollo. Los servicios pueden implementarse de manera independiente, los servicios son los responsables de conservar sus propios datos o estado externo, los servicios se comunican entre sí mediante API bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios y no es necesario que estos compartan la misma pila de tecnología, las bibliotecas o los marcos de trabajo (Salazar Hernández, 2017).

Esta tecnología puede ser utilizada en aplicaciones de mayor tamaño que requieran alta velocidad de publicación y escalabilidad inmediata cuando necesite utilizar aplicaciones con dominios complejos apoyándose en pequeños equipos de desarrollo.

A continuación, se detallan las ventajas de esta arquitectura: cuenta con implementaciones independientes, ya que los equipos de trabajo pueden centrarse en un servicio, esto permite que el código base sea fácil de entender y de este modo, nuevas personas que ingresen al desarrollo puedan comprender rápidamente el proyecto. Cuenta con un aislamiento de errores; esto es muy valioso, visto que, si un servicio

deja de funcionar, otros servicios pueden apoyarlo para no paralizar la aplicación, los equipos del proyecto pueden elegir qué tecnología se adapta mejor al servicio y los servicios pueden escalar de manera independiente (Duarte Vega, 2016).

Una gran desventaja de esta arquitectura es que no se puede trabajar en varios ambientes simultáneamente, lo que complica el trabajo a los desarrolladores y arquitectos con respecto al software en construcción.

López Hinojosa (2017) menciona un caso de la coordinación general de tecnologías de la información y comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) que contaba con una arquitectura monolítica que permitía solventar diversas necesidades, generado varios procesos para resolverlos. Sin embargo, por avances de la tecnología, contando con nuevos patrones de diseño, versiones de plataformas, entre otros, los sistemas han quedado sin uso por no ser flexible a los cambios.

De tal manera que, lo que antes era una solución, ahora se convierte en un problema continuo por el mantenimiento que necesita para su funcionamiento, generando trabajo adicional para resolver o unir algún proceso de la arquitectura monolítica con alguna nueva versión de una aplicación. Al no contar con una arquitectura definida, los desarrolladores generan aplicaciones bajo su criterio y experiencia.

Frente a esta problemática, se propuso al CGTIC utilizar una arquitectura basada en microservicios para el desarrollo de aplicaciones web, utilizando nuevas tecnologías. Las actividades de la arquitectura se encuentran en un proceso incremental; de este modo, se permite adecuar el trabajo en las actividades de arquitecturas individuales con los cambios de los requisitos.

Suryotrisongko, Puji Jayanto y Tjahyanto (2017) realizaron una investigación para el desarrollo de una aplicación de servicio público de quejas basado en la aplicación web, utilizando la arquitectura de microservicio Spring boot. Esta arquitectura se realizó para dividir la funcionalidad de la aplicación en pequeñas partes, adecuándolas para ser utilizadas por los microservicios en diferentes procesos interconectados; de este modo se convirtió en una aplicación con un proceso comercial completo. La gran ventaja de este proyecto es que la arquitectura permite agregar más microservicios sin afectar a otros procesos.

La aplicación se implementó en la plataforma de la nube a la cual se puede acceder desde cualquier navegador. Se dividió el Front End en cuatro interfaces para los usuarios, que son el administrador, la unidad de trabajo del gobierno, el reclamo que utiliza el ciudadano y el vendedor. Cada uno de estos se asocia con los microservicios como se observan en la Figura 3.

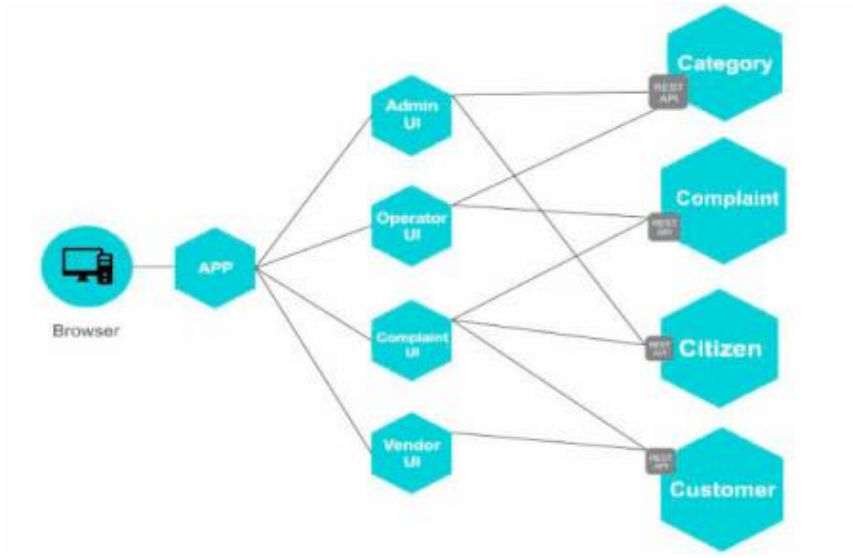


Figura 3. Arquitectura de microservicios (Suryotrisongko et al., 2017).

Salazar Hernández (2017) realizó una arquitectura con microservicios, combinada con la virtualización por sistema operativo para compararla con una arquitectura monolítica, evaluando las ventajas y desventajas de trabajar con este enfoque. Para ello, se evaluaron diversos escenarios, revisando si el tipo de arquitectura propuesta se adaptaba a las pruebas del usuario. Como resultado, se obtuvo una implementación eficiente con el uso de contenedores al momento de implementar los patrones de arquitectura de microservicios.

Arquitectura web services

La arquitectura web services puede ser traducida como servicios web; es un componente al que se puede acceder mediante protocolos web estándar, utilizando XML para el intercambio de información (Antunes y Vieira, 2016). El término web services describe una forma estandarizada de integrar aplicaciones web mediante el uso de XML, SOAP, WSDL y UDDI sobre los protocolos de la Internet. XML es usado para describir los datos, SOAP se ocupa para la transferencia de los datos, WSDL se emplea para describir los servicios disponibles y UDDI es utilizado para conocer cuáles son los servicios disponibles.

Web services tiene como principal función permitir la comunicación entre las empresas y sus clientes. Así mismo, permite a las organizaciones intercambiar datos sin necesidad de conocer los detalles de sus respectivos sistemas de información (Dahan, El Hindi y Ghoneim, 2017).

Los web services permiten la comunicación entre aplicaciones, ya sean de diferentes orígenes o versiones sin necesidad de utilizar programas costosos, porque la comunicación se hace mediante XML. Además, no están ligados a ningún sistema

operativo o lenguaje de programación. Por ejemplo, un programa escrito en Java puede conversar con otro escrito en Pearl; aplicaciones Windows pueden conversar con aplicaciones Unix. Sin embargo, los web services no necesitan usar browsers (navegador de internet) ni el lenguaje de especificación HTML.

Las características de un web services son las siguiente: el servicio debe ser accesible por medio de la web, utilizando protocolos y estándares como HTTP, codificando los mensajes y teniendo un estándar para que un cliente pueda utilizar el servicio; cada servicio web debe contener una descripción de sí mismo, para conocer su funcionalidad y se tiene que encontrar un método que permita reconocer un web services de forma automática para su conexión (Tian, Wang, He, Sun y Tian, 2017).

Los servicios web pueden implementarse de varias formas. En este sentido, se pueden distinguir dos tipos de servicios web que son los denominados servicios web grandes (big web services), que son los de SOAP y los de REST. Se explican a continuación.

Simple Object Access Protocol (SOAP). Se utilizan mensajes XML para intercomunicarse con estándares SOAP; este lenguaje (XML) define la arquitectura y formato de los mensajes. Dichos sistemas normalmente contienen una descripción legible por la máquina de la descripción de las operaciones ofrecidas por el servicio, escrita en Web Services Description Language (WSDL), que es un lenguaje basado en XML para definir las interfaces sintácticamente (Stankevicius, 2013).

El formato de mensaje SOAP y el lenguaje de definición de interfaces WSDL se ha extendido bastante y existen diversas herramientas de desarrollo, como Netbeans, Eclipse, JeatBrains, que son útiles para el desarrollo de aplicaciones de servicios Web.

El diseño de un servicio basado en SOAP debe establecer un contrato formal para describir la interfaz que ofrece el servicio web. WSDL puede utilizarse para describir los detalles del contrato, incluyendo mensajes, operaciones y la localización del servicio web (De la Cruz Caicedo, Bolaños Bastidas, Ordóñez Ante y Corrales Muñoz, 2014).

Representational State Transfer (REST). Estos servicios son adecuados para escenarios básicos de integración y pruebas ad hoc. Además, suelen integrarse mejor con HTTP que los servicios basado en SOAP, ya que no requieren mensajes XML o definiciones del servicio en forma de fichero WSDL (Tihomirovs y Grabis, 2016). Los servicios web REST utilizan estándares muy conocidos, como HTTP, SML, URI, MIME, y tienen una infraestructura ligera, permitiendo a los servicios que se construyan utilizando herramientas de forma mínima. De este modo, el desarrollo de servicios REST es económico y tiene muy pocos obstáculos para su adopción (Meliá, 2007).

Encapsulamiento de aplicaciones

A continuación, se mencionará algunos tipos de encapsulamiento de aplicaciones.

Contenedores

Un contenedor es un ambiente aislado y portable que permite encapsular aplicaciones con recursos que necesita para su funcionamiento. Esta arquitectura se centra en diseñar la virtualización del hardware y parte de la plataforma del software, haciendo referencia a una virtualización ligera. Debido a que los contenedores comparten un solo kernel del sistema operativo, pueden reducir los costos de licencias del sistema operativo, aumentar el rendimiento, eliminar los recursos de memoria y procesador, necesarios para ejecutar múltiples versiones del sistema operativo. El

grupo de máquinas físicas o virtuales donde los contenedores son ejecutados son llamados clúster. En la construcción de un proyecto, existen diversos ambientes como desarrollo, prueba y producción, donde el software tiene que estar interactuando directamente con estos; así mismo, deben tener un óptimo rendimiento en cada ambiente. Los contenedores apoyan en este proceso, encapsulando la aplicación en un paquete liviano y único. La tecnología de contenedores puede beneficiar a los entornos de nube, permitiendo a las aplicaciones moverse entre las nubes. Los contenedores se pueden acomodar a diferentes plataformas, como Google, Amazon y Microsoft, que ofrecen el servicio de red pública, facilitando el uso de contenedores privados en la nube, la mayoría de veces para aplicaciones de nube híbridas (Brogi, Neri y Soldani, 2018).

Para la implementación de una arquitectura de microservicios, se requiere una gran demanda de recursos y esto puede ser aplacado por la agilidad de los contenedores que hacen más sencillo el trabajo. Aunque son independientes uno del otro, ambos se complementan perfectamente.

Docker

Docker es una tecnología para crear contenedores ligeros y portables para las aplicaciones software que pueden ejecutarse en cualquier máquina con docker, sin tener que preocuparse de las versiones del software que ya están instaladas en las máquinas y tiene los elementos necesarios para que funcione la aplicación sin ser compatible. Así mismo, ofrece un modelo de implementación basado en imágenes. Esto permite compartir una aplicación o un conjunto de servicios, con todas sus dependencias, en varios entornos. Docker también automatiza la implementación de la aplicación dentro del entorno del contenedor.

Un contenedor de Docker funciona utilizando el sistema operativo que tiene la máquina en la que se ejecuta el contenedor, mientras que una máquina virtual necesita instalar un sistema operativo para funcionar (Suhartanto et al., 2017).

Entre sus beneficios cabe mencionar que un contenedor Docker no contiene todo un sistema completo, sino únicamente aquellas librerías, archivos y configuraciones necesarios para desplegar las funcionalidades que contiene; esto lo convierte en autosuficiente. La ligereza y peso que tiene un contenedor Docker es inferior a un sistema virtual se puede desplegar en cualquier otro sistema, siendo muy portable.

Servicios en la nube

En el transcurso de los últimos años han ido naciendo nuevas ideas tecnológicas y dentro de ellas están los servicios en la nube. Cuando se trabaja este tipo de aplicaciones, se tiene que definir lo que se va a abarcar, ya que el concepto nube es muy amplio y existen distintas formas de hacerlo permitiendo una mayor flexibilidad y sencillez a la hora de desplegar las aplicaciones o mantenerlas (Luna Encalada y Castillo Sequera, 2017). Existen tres distintas formas para utilizar la nube. A continuación, se describen:

Software as a Service (SaaS)

El SaaS se trata de cualquier servicio basado en la web, donde se puede acceder mediante un navegador, sin tener que ir al propio software. Se tiene poco control de este tipo de servicio, ya que solo es de consulta. Se tienen como ejemplos de SaaS el webmail de Gmail, Elastic Compute Cloud (EC2), Amazon Elastic, Dropbox, entre otros (Czarnul, 2013).

Platform as a service (PaaS)

PaaS es una plataforma es propia para el área de desarrollo, en la que se empiezan a crear las aplicaciones que se ejecutarán en la nube, evitando preocuparse por la infraestructura, ya que es propio de esta.

Este modelo reduce la complejidad a la hora de desplegar y mantener aplicaciones, ya que las soluciones PaaS gestionan automáticamente la escalabilidad, usando más recursos si fuera necesario. Los desarrolladores tienen que preocuparse de que sus aplicaciones estén optimizadas para consumir menos recursos (número de peticiones, escrituras en disco, espacio requerido, tiempo de proceso); todo ello sin entrar al nivel de máquina (Khalid y Shahbaz, 2017).

Entre los ejemplos se pueden mencionar Google App Engine, Microsof Azure y Force.com, entre otros (Jia bin et al., 2018).

Infrastructure as a service (IaaS)

Iaas brinda acceso a recursos informáticos igual que los otros servicios cloud (SaaS y PaaS), con la diferencia que ofrece una infraestructura de procesamiento.

Con la utilización de las IaaS hay más control que con PaaS, aunque a cambio de eso, se tendrá que encargar de la gestión de infraestructura. Un ejemplo perfecto es el proporcionado por Amazon Web Services (AWS), que provee una serie de servicios como Elastic Compute Cloud (EC2) que permite manejar máquinas virtuales en la nube. Se puede elegir qué tipo de sistema operativo se desea utilizar, como Linux o Windows, así como la capacidad de memoria o procesador de cada una de sus máquinas virtuales (Ayyapazham y Velautham, 2017).

Proveedores cloud

Dentro del ámbito empresarial, diariamente se está decidiendo optar por nuevas formas de trabajo, como la migración en la nube, iniciando con la computación en esta, que es la corriente principal de la era digital.

Hay diversos proveedores que ofrecen estos servicios y se han ido posicionando en el mercado gracias a la satisfacción de los usuarios.

En RightScale (2018), se realizó una encuesta a 997 múltiples industrias y compañías, donde el porcentaje de encuestados utilizaron aplicaciones en la nube, exponiendo un óptimo resultado a favor de AWS, con un 68% en el año 2018 y sin quedarse atrás Azure, con un 58% como se observa en la Figura 4 (Digman, 2018).

Loten (2018) expuso un artículo en el que realizó una entrevista a los directores de información de las grandes empresas públicas por medio de encuestas; mencionaron que utilizan AWS en un 56% seguidamente de un 49% de Microsoft Azure.

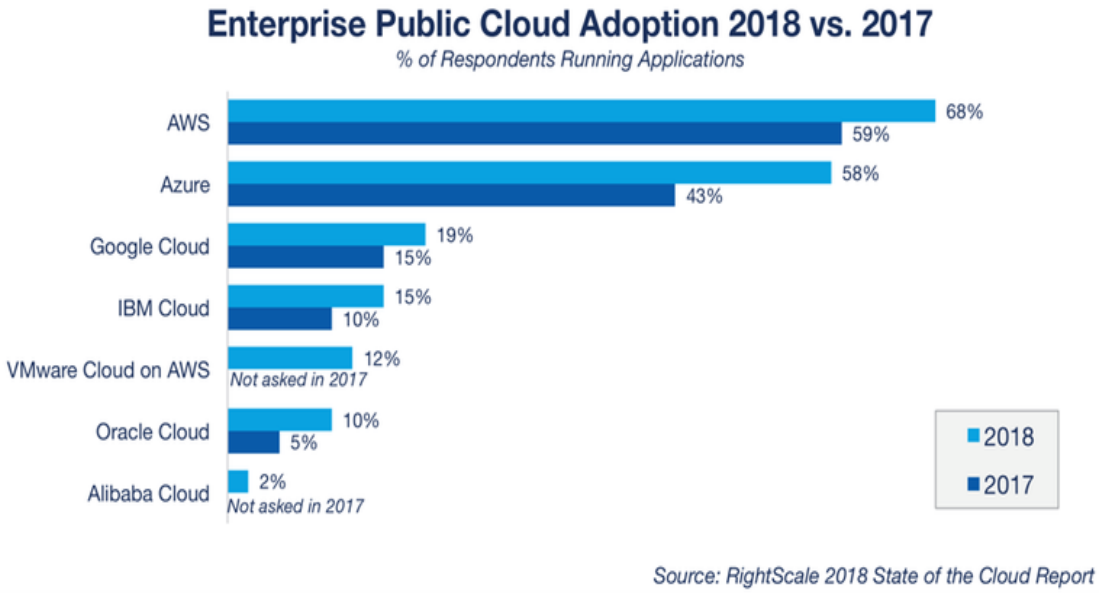


Figura 4. Adopción de la nube pública empresarial (RightScale, 2018).

Amazon Web Services (AWS)

AWS es una plataforma que domina en el mundo de las TI. Entre los principales beneficios para elegir AWS están los siguiente: flexibilidad rigurosa, aplicaciones rápidas y sencillas, escalabilidad de la solución, mantenimiento de datos seguros y alcance masivo de las operaciones; este último punto es una de las razones más sobresalientes de AWS. Al tener una gran variedad de servicios, se considera la mayor red de centro de datos; por otro lado, brinda opciones para soportar otras plataformas.

El precio es el punto débil de AWS; sin embargo, la organización está reduciendo costos para sus usuarios, sin afectar la calidad del producto. Con respecto a la nube híbrida, es un área donde AWS no es tan soportable.

Microsoft Azure

Microsoft Azure es uno de los fuertes competidores de AWS, brindando servicios como beneficios en la nube, teniendo un crecimiento acelerado por la popularidad de sus productos anteriores, como Windows, siendo una plataforma leal para los consumidores de Microsoft, generando descuentos atractivos para los servicios en la nube de Azure, apoyando en la integración rápida con otras organizaciones. Azure se integra adecuadamente con los sistemas primarios de Microsoft, como System Center, Windows Server y Active Directory, ya que es un escenario muy común para industrias como los sectores financieros, organizaciones empresariales y grandes empresas (Mora Rodríguez, 2016). Azure puede ser algo restrictivo con respecto a soportar otras plataformas o si desea ejecutar otra cosa que no sea Windows Server, en comparación con AWS. Otra desventaja es al momento de expandirse, cuando la empresa busca una solución más compleja en la nube.

Google Cloud Platform (GCP)

GCP es una plataforma más para la computadora en la nube, donde permite a los desarrolladores probar, construir e implementar aplicaciones de todo tipo. GCP, tiene una variedad de servicios para aplicaciones móviles empresariales, como App Engine, que crea aplicaciones de forma ágil, realizando un nivel alto de computación, redes, almacenamiento y base de datos. No obstante, Google tiene menos servicios con respecto a sus competidores y cuenta con todos los requisitos para desarrollar proyectos en aplicaciones móviles (Olcina Valero, 2017).

En el Apéndice A se muestra un cuadro donde se comparan las características, beneficios, ventajas y desventajas que tienen estos servicios en la nube.

Infraestructura de Amazon Web Services (AWS)

Frente a un mundo tecnológico, donde se dice que la seguridad en la nube es mejor que cualquier infraestructura física, se ha incrementado su uso para almacenar todo tipo de archivos y poder acceder a ellos desde cualquier lugar por medio de la internet. Se requiere de una plataforma para soportar la cantidad de almacenamiento y transacción de toda esta información.

Ante esta demanda, surge Amazon Web Services, ofreciendo la mayor cantidad posible de herramientas y servicios, para crear un entorno de computación en la nube (ver Figura 5).

Los servicios de Amazon Web Services están preparados para trabajar con pequeñas, medianas o grandes corporaciones, ya sea que esté buscando un sitio web de marketing, o de almacenamiento o de comercio electrónico. Con este beneficio, los

clientes se sienten motivados a crecer e innovar de acuerdo con su proyección institucional e invertir en la nube.

Amazon Web Services realiza el cifrado de datos en reposo y tránsito e implementa funciones de seguridad para administrar la infraestructura de TI de las empresas. Puede crear diversas instancias según lo necesite y estas son escalables (Fusaro, Patil, Gafni, Wall y Tonellato, 2011).

Se puede decir que AWS es una plataforma que contiene una colección de servicios en la nube y ofrece potencia de cómputo e infraestructura, como almacenamiento de bases de datos, entrega de contenido y otras funcionalidades para ayudar a las empresas a escalar y crecer (Axelrod, 2015).

Amazon tiene un largo trayecto en el mundo digital, consolidándose como una de las más grandes en este tipo de mercado; sin embargo, han ido emergiendo otros competidores, como Microsoft y su nube Azure.

Los servicios de Amazon Web Services tienen presencia en más de treinta zonas de disponibilidad dentro de 16 regiones geográficas en el mundo; de este modo brinda un acceso a la información que es inmediato (tiempo real).

Entre las empresas que están trabajando con esta tecnología, están las siguientes: Pinterest, Netflix, Reddit, Foursquare. La plataforma y servicios que utilizan estas grandes organizaciones son las mismas que ofrece AWS para cualquier usuario que desee trabajar en su nube, brindando una plataforma de alto nivel.

Algunos beneficios de AWS son una amplia compatibilidad con Content Management System (CMS) y su plataforma de desarrollo. AWS puede usar cualquier CMS que desee, incluidos WordPress, Drupal, Joomla y más.

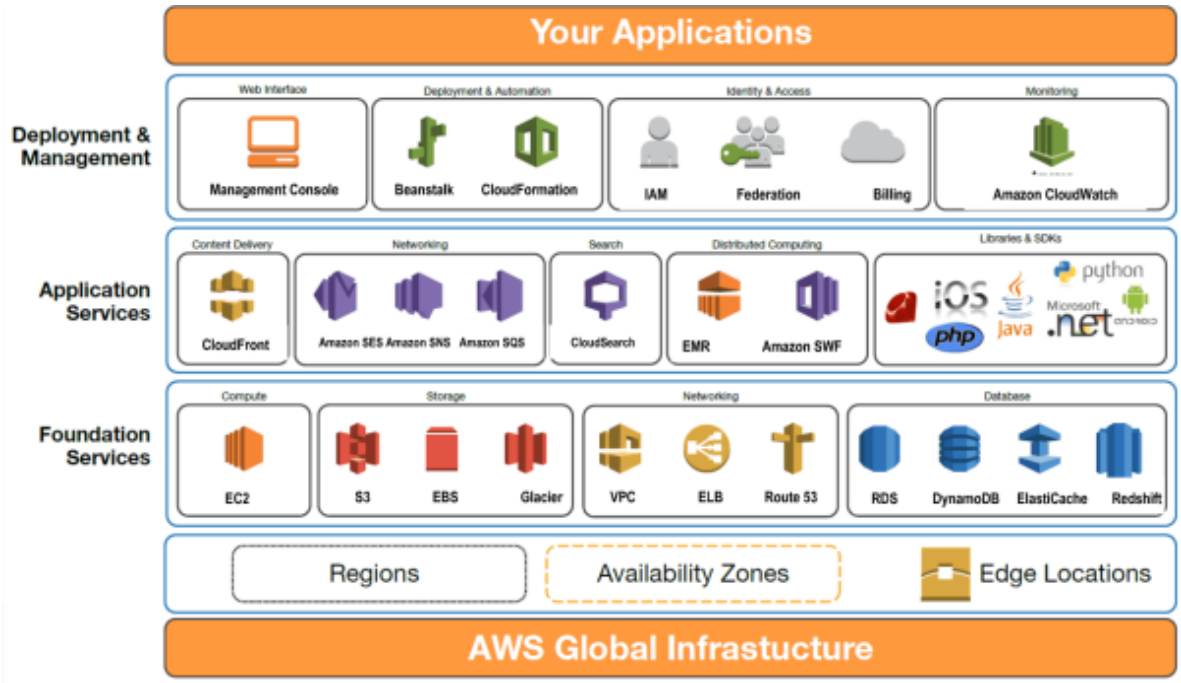


Figura 5. Plataforma del Amazon Web Servicie (Westcon-Comstor, 2018).

AWS también admite y proporciona Software Development Kit (SDK) para plataformas populares como Java, Ruby, PHP, Node.js y .Net. Con AWS se puede tener un centro de datos que hospede el sitio web en cualquier geografía del mundo que se elija con solo unos pocos clics, y se pueden desarrollar y reducir dinámicamente los recursos; según la organización vaya creciendo, se van ampliando poco a poco los recursos de un sitio web, a la vez que se incrementa el número de visitas. El tráfico de un sitio web o algunos servicios puede fluctuar mucho, desde “horas muertas” (sin usuarios), hasta horas pico de tráfico (demanda de usuarios); debido a ello, muchos usuarios tienen inconvenientes al momento de utilizarlo y se generan molestias por el tiempo de espera para la respuesta solicitada. Con una infraestructura de AWS, previa configuración, este inconveniente se soluciona con una escalación automática para satisfacer las necesidades de los usuarios y modelos de precios flexibles: AWS solo

cobra por los recursos que se usan sin costos iniciales ni contratos a largo plazo; así mismo, tiene opciones de alojamiento web que ofrecen precios de pago por uso o precios fijos mensuales (Cliff, 2017).

Servicios de AWS

Amazon cuenta con diversos productos y servicios que brinda al usuario de acuerdo a sus necesidades, divididos en áreas como cómputo, almacenamiento, base de datos, redes, entrega de contenido, herramientas para desarrolladores, entre otros.

De esta manera, apoya a los administradores de TI, a los de sistemas y a los desarrolladores a gestionar y monitorear fácilmente sus recursos de infraestructura híbrida. Brindando estos servicios completamente ordenados, puede aprovisionar, configurar y administrar sus recursos de AWS a una escala automática. Además, puede monitorizar los logs y las métricas de la infraestructura con paneles de control y alarmas en tiempo real. Así mismo, ayuda a monitorizar, supervisar, controlar y garantizar la conformidad y la seguridad (Rojas Albarracín, Páramo Fonseca y Hernández Merchán, 2017).

Amazon Elastic Compute Cloud (Amazon EC2)

Amazon EC2 es un servicio web ofrecido en un entorno informático de nube segura y de tamaño modificable (computación en la nube). En otras palabras, puede decirse que es como una “máquina virtual” en la nube, que ofrece a los usuarios contar con entornos de desarrollo, testeo y gestión de aplicaciones.

En Amazon EC2 se pueden utilizar tantos servidores virtuales como se necesiten en cuestión de minutos, permitiendo escalar hacia arriba o abajo según el control

de tráfico y amoldándose a las necesidades. De este modo, se elimina la necesidad de invertir en la adquisición de un hardware, permitiendo pagar solo por la capacidad que se utiliza (Bhardwaj, Jain y Jain, 2010).

Entre las características del Amazon EC2 se pueden mencionar las siguientes: (a) los entornos informáticos virtuales son conocidos como instancias, (b) las plantillas preconfiguradas para las instancias son conocidas como imágenes de máquina de Amazon (AMI), conteniendo fragmentos que necesita el servidor (como el sistema operativo), (c) las configuraciones como CPU, memoria, almacenamiento, redes de las instancias son conocidas como tipos de instancias, (d) para el inicio de sesión por primera vez en las instancias, al momento de crearlas se proporcionan un par de claves (AWS almacena la clave pública y el cliente guarda la clave privada), (e) los volúmenes donde se almacenan los datos temporales que se eliminan cuando la instancia se detiene o termina, son conocidos como volúmenes de almacén de instancias, (f) para los volúmenes de almacenamiento persistente de datos, se usa Amazon Elastic Block Store (EBS), que es conocido como volúmenes, (g) las ubicaciones físicas que se utilizan para los recursos como instancias y volúmenes de Amazon EBS son conocidas como regiones y zonas de disponibilidad, (h) se tiene un firewall que permite definir los protocolos, puertos y rangos de direcciones IP y se puede alcanzar a las instancias mediante el uso de grupos de seguridad, (i) las direcciones IPv4 en la nube dinámica son conocidas como direcciones IP elásticas, (j) los metadatos, que permiten asignar los recursos en Amazon EC2, son conocidos como etiquetas y (k) las redes virtuales que están aisladas del resto de la nube de AWS y pueden conectarse a su propia red son conocidas como nubes privadas virtuales (Jiménez Domingo, 2013).

Los beneficios que resaltan de Amazon EC2 son los siguientes:

1. El cómputo a escala de web elástica, con Amazon EC2, puede disminuir o aumentar cargando una o miles de instancias en el servidor de forma simultánea. Con el apoyo de Auto Scaling de Amazon EC2 se puede hacer que una instancia aumente o disminuya automáticamente, conservando su configuración según se necesite.

2. Con Amazon EC2 se tiene el control total de las instancias desde su acceso raíz. Se puede detener una instancia y mantener sus datos en la partición; posteriormente, reiniciar la misma instancia por medio de las API; esto se realiza de manera remota.

3. Los servicios de alojamiento en la nube son flexibles; se puede elegir entre una variedad de tipos de instancia, paquetes de software y sistemas operativos, permitiendo seleccionar la configuración de memoria, el CPU y el almacenamiento de la instancia.

4. Amazon EC2 tiene una integración total con los otros servicios que ofrece AWS para dar una segura y óptima solución, procesando consultas y almacenamiento por medio de una variedad de aplicaciones.

5. El entorno de Amazon EC2 es muy confiable para las instancias que se ejecutan en los centros de datos y su infraestructura de red está acreditada por Amazon, comprometiéndose a tener una disponibilidad completa en todas las regiones de Amazon EC2.

6. La seguridad es la mayor prioridad de Amazon Web Services, proporcionando una arquitectura de red en conjunto con otras aplicaciones para proporcionar una red sólida y segura para los recursos informáticos; de este modo, satisface a los usuarios más exigentes en Amazon EC2 (Kulkarni, Sutar y Gambhir, 2012).

La computación en la nube ofrece diversas ventajas ante el despliegue y ejecución de las aplicaciones, facilitando su gestión, reduciendo el gasto de recursos como

equipos de cómputo y consumo de energía; sin embargo, la decisión de migrar a un alojamiento en la nube es muy complejo, debiendo evaluar las facilidades ofrecidas por los proveedores. En cuanto al factor económico, este influye mucho en la toma de decisiones (Álvarez, Hernández, Fabra y Ezpeleta, 2018).

Amazon Machine Image (Amazon AMI)

Como su propio nombre lo dice, Amazon AMI es una imagen que contiene las mismas configuraciones y características del EC2; se podría decir que es como una copia fiel de la instancia que se seleccionó.

El sistema de archivos que tiene Amazon AMI está cifrado, comprimido y dividido en partes que son cargadas por AWS para ser almacenadas con un nombre, versión, tipo de arquitectura, identificación del kernel y clave de descifrado.

Estas imágenes se pueden tomar como plantillas que interiormente tienen la configuración del sistema operativo, determinando el entorno que utilizará el usuario. Esto permite disparar muchas instancias cuando se requiere, evitando tener que configurar cada vez la creación de cada instancia, facilitando el trabajo y ahorrando tiempo en la duplicidad de la información. Así mismo, estas son configurables para tener un mayor control y restringirlas en caso de no ser necesarias (Caballer Fernández, 2014).

Este sistema brinda una lista de beneficios que son los siguientes: (a) una sólida integración con AWS, ya que es uno de sus servicios importantes de AWS donde se puede configurar, disparar y mover un EC2 en cualquier momento, (b) para su seguridad, se divide en dos vertientes; una es la limitación de los accesos y la otra es la reducción de las debilidades del propio software. Cada Amazon AMI está cifrada y protegida con una clave, para evitar exponer los datos a amenazas de seguridad y (c)

por medio de las constantes actualizaciones, Amazon AMI se refuerza cada vez más en temas de integración con AWS y seguridad. A su vez, están sujetas a las nuevas políticas que AWS puede cambiar, evitando errores garrafales.

Amazon Relational Database Service (Amazon RDS)

Amazon RDS es otro servicio web que apoya a una base de datos en la configuración, funcionamiento y escalado; todo esto en la nube. Se puede decir que es un entorno donde las bases de datos se encuentran asiladas y están en ejecución. Dentro de una instancia, el usuario pude crear una o varias bases de datos, utilizando herramientas o aplicaciones para su mantenimiento y gestión. Para el acceso a una base de datos que se encuentra en una instancia de AWS, se tiene que utilizar una aplicación de SQL estándar, ya que Amazon RDS no admite el acceso directo en su interfaz de host. Amazon RDS soporta diversas bases de datos como Oracle, PostgreSQL, MySQL y SQL Server, entre otros (Salazar Cárdenas, 2014).

Tradicionalmente, cuando se compraba un servidor físico (CPU), contenía todas las características y componentes necesarios (memoria, almacenamiento, procesador, entre otros), para sobrellevar la carga de trabajo que se tenía contemplado. Pero, al incrementar la carga y transacción de datos, este servidor no era suficiente para responder de manera eficiente a las tareas solicitadas, teniendo que adquirir un nuevo equipo o agregando nuevos componentes para reforzar el servidor.

Con Amazon RDS todos estos componentes se encuentran divididos, facilitando la escalabilidad de ellos en forma independiente. Si se requiere más memoria y menos almacenamiento, se pueden asignar estos fácilmente, gracias a un escalamiento automático que es configurable. Existen cuatro instancias de Amazon RDS, llamadas T3, T2,

M5 y M4, que cuentan con servicios básicos hasta los más complejos, según se considere la necesidad y el precio.

Entre las características de Amazon RDS cabe mencionar las siguientes:

1. El rendimiento de las instancias dentro de Amazon RDS es ampliable, optimizando el rendimiento del CPU en la nube. Eso dependerá del volumen de tráfico que se da cuando se esté solicitando algún recurso en un mismo tiempo. Por ejemplo, en un sistema financiero sería el estado de cuenta de los clientes.

2. Para que un almacenamiento sea duradero, Amazon RDS provee un servicio a nivel de bloque, suministrando tres tipos de volúmenes para mejorar la calidad ante las necesidades de cargas de trabajo con respecto a las bases de datos. La primera es para uso general, con el respaldo de SSD (Solid State Drive) para almacenar los datos como en memoria flash. Así mismo, esta es muy apropiada para una amplia carga de trabajo en la base de datos. La segunda es IOPS (Input/Output Operations per second), que ofrece un trabajo en tiempo real (baja latencia); por otro lado, están diseñadas para cargas de trabajo intensivas, ya sea de entrada o de salida de peticiones. El tercero son volúmenes magnéticos, utilizados para cargas de trabajo de poca frecuencia (más pequeños).

3. Amazon RDS tiene instancias optimizadas para EBS (Elastic Block Store), que consisten en bloques de almacenamiento para ser utilizados por Amazon EC2 para proteger errores de cualquier componente, y IOPS provisionadas; de este modo, se obtiene una latencia menor que puede llegar a milisegundos.

Los beneficios que brinda Amazon RDS son los siguientes:

1. Facilidad de poder crear y utilizar una instancia en cuestión de minutos. Teniendo la base de datos en la nube, se tiene el control de aplicar actualizaciones a instancias de la base de datos.

2. La escalabilidad es uno de los puntos fuertes de Amazon Web Services y sus servicios lo heredan, optando por recursos informáticos cuando los necesiten, como ampliar o reducir la memoria según sea la situación. Las réplicas de lectura, que son parte del escalado, se realizan cuando el tráfico es demandante; en otras palabras, se crean replicas (copias) de la instancia de la base de datos para que pueda abastecer el incremento de peticiones, facilitando la lectura de los datos. Esto incrementa el rendimiento de la base de datos.

3. Las copias de la base de datos se generan continuamente (cada 5 minutos); esto quiere decir que se puede restablecer una instancia en cualquier momento.

4. Amazon RDS cifra o encripta sus bases de datos por medio de claves de Amazon Key Management Service (KMS); de este modo, maneja la seguridad de la información permitiendo que solo un usuario, con los accesos necesarios, pueda acceder a la nube.

5. Amazon RDS cuenta con más de cincuenta métricas de componentes CPU, memoria, almacenamiento, entre otros, para monitorear los problemas de la base de datos evaluando su rendimiento.

A su vez, Amazon RDS puede notificar por medio de un email o mensaje de texto móvil los cambios, alertas, errores, sugerencias o eventos que sean importantes en la base de datos (Henríquez, Del Vecchio, y Peternina, 2015).

Amazon Elastic Load Balancing (Amazon ELB)

Amazon ELB tiene la función principal de distribuir el tráfico automáticamente de peticiones entrantes por medio de varias rutas como instancias de Amazon EC2 y a través de contenedores y direcciones IP (Guabtni, Ranjan y Rabhi, 2013). Por ejemplo, un sitio web que tiene diversos servicios en AWS, cuando los usuarios desean recibir información de varios servicios, ELB distribuye las entradas y las redirecciona al servicio que se solicite. No obstante, el ELB tiene un constante monitoreo sobre las instancias y el tráfico entre ellas (Unda, Corall y Marcillo, 2013)

Existen tres tipos de balanceadores de carga que ofrecen servicios específicos, de acuerdo con la demanda de trabajo. El balanceador de carga de aplicaciones se encarga del equilibrio de tráfico HTTP y HTTPS, brindando un apoyo a las arquitecturas modernas que incluyen contenedores y microservicios. El balanceador de carga de red, adecuado para el tráfico de TCP (protocolo de control de transmisión) y TLS (seguridad de capa de transporte). Además, este balanceador tiene la capacidad de controlar miles de solicitudes por segundo, manteniendo una latencia ultrabaja. Así mismo, este balanceador controla los patrones de tráfico volátiles e inesperados, optimizándolos. El balanceador de carga clásico, como su propio nombre lo dice, suministra equilibrio de carga de manera básica a diversas instancias de Amazon EC2, funcionando a nivel de solicitud y conexión. Este balanceador está diseñado para las aplicaciones que se crearon en la red de EC2-Classic.

Entre sus características destacan las siguientes: (a) ELB distribuye el tráfico instintivamente a sus destinos finales, generando la alta disponibilidad de acceso a la información en todo momento; (b) por medio de su continuo monitoreo entre las instancias,

ELB puede detectar los problemas que hubiera en alguno de ellos. Si hubiera peticiones para dicha “instancia defectuosa”, ELB direcciona el tráfico de esta a otra instancia que pueda atender la solicitud mientras se repone la instancia dañada; de este modo se evita un cuello de botella, (c) la seguridad es vital en todo Amazon; por ello se requiere crear y administrar diversos grupos de seguridad y uno de ellos debe ser para los balanceadores; (d) ELB contiene un servicio integrado de administración y descifrado SSL/TLS, ofreciendo la administración flexible, centralizada con respecto a los parámetros SSL y eliminando el arduo trabajo para el CPU y (e) se puede integrar con Amazon CloudWatch, que permite tener un monitoreo del rendimiento de las instancias y aplicaciones en tiempo real.

Los beneficios que ofrece ELB son los siguientes: (a) elasticidad, ya que tiene la capacidad de controlar cambios de madera rápida dentro de los patrones de tráfico en la red. Como tiene un Auto Scaling, puede satisfacer los niveles de carga de las aplicaciones sin que se realice manualmente por el usuario; (b) flexibilidad, ya que ELB puede utilizar direcciones IP, permitiendo direccionar el destino de las peticiones; (c) se pueden monitorear las aplicaciones en tiempo real por medio de registros, métricas y rastreo de las solicitudes de Amazon CloudWatch, permitiendo la identificación de problemas y optimizando el tiempo de respuesta y (d) tiene un control del equilibrio de cargas hibridas, que pueden ser de manera local y en la nube (AWS); utilizando el mismo balanceador de carga, proporciona la transmisión y migración de las aplicaciones locales a AWS.

Amazon CloudWatch

Amazon CloudWatch es un servicio de monitorización de los recursos y aplicaciones que se ejecutan en infraestructuras locales, híbridos y de AWS, para evitar

cambios repentinos que afectan el rendimiento del sistema, tomando acciones de optimización en los recursos y logrando una vista unificada del estado actual de los servicios de AWS (González García, 2017). Se puede utilizar CloudWatch para configurar alarmas, revisión de registros y métricas, resolución de errores por la toma de acciones automatizada y la búsqueda de información para la optimización de las aplicaciones; con ello, se busca poder continuar con su ejecución, evitando el margen de error (Arias Preciado, Mariaca Lira y Parodi Mendoza, 2011).

Amazon CloudWatch cuenta con las siguientes características:

1. Recopila y almacena los registros de recursos, servicios y aplicaciones rápidamente. Amazon CloudWatch puede recopilar métricas establecidas de muchos servicios de AWS, sin la necesidad de que el usuario se involucre directamente. A su vez, cuenta con métricas personalizadas que permiten observar el rendimiento y la solución de errores de las instancias.

2. Amazon CloudWatch cuenta con paneles que permiten elaborar gráficos de las aplicaciones que están siento monitoreadas. Con ello se puede tener una rápida y visual situación del problema, permitiendo configurar alarmas según los niveles de prioridad, brindando notificaciones y toma de acciones según se haya programado. Como las alarmas son en tiempo real, se basan en eventos y métricas, reducen el tiempo de inactividad y el impacto en la organización.

3. Por medio del Auto Scaling, en CloudWatch se pueden configurar reglas para desencadenar acciones automatizadas cuando concuerden con algún evento en especial, permitiendo la rápida respuesta ante los cambios.

4. CloudWatch cuenta con un análisis de los registros de forma minuciosa, creando un log de historial de todos los registros que son procesados para ser consultados por el administrador.

5. Para mayor seguridad, se cifran los datos en tránsito y reposo; solo puede tener acceso el usuario administrador para revisar estos logs (Agrawal, Wiktorski y Rong, 2017).

Entre los beneficios de Amazon CloudWatch destacan los siguientes: (a) monitoriza los recursos de Amazon Web Services de manera práctica y sencilla. Como se puede conectar con más servicios, automáticamente se publican métricas que muestran un detalle a cada segundo; (b) por medio de los logs, se pueden visualizar rápidamente los problemas que presentan las aplicaciones o visualizar un patrón recurrente que puede ser riesgoso; para ello se puede tomar medidas correctivas a tiempo y (c) como utiliza el Auto Scaling, pueden detectar recursos que no se estén utilizados para finalizarlos; de este modo, apoyar la reducción de costos.

Amazon Virtual Private Cloud (Amazon VPC)

Amazon VPC son redes privadas virtuales en AWS, que permiten tener un espacio en la nube con un dominio propio, donde se pueden almacenar, gestionar y resolver estados críticos en menor tiempo (Chiriboga Mogollón, 2014). Así mismo, puede utilizar tanto IPv4 como IPv6 para obtener acceso a los recursos y a las aplicaciones de manera segura y sencilla (Huitrón Toledo, Zapata Nogales y Zaragoza López, 2017).

Las características que tiene Amazon VPC son las siguientes: (a) tiene una conexión directa con redes públicas (Internet), permitiendo disparar instancias para enviar/recibir tráfico en el internet; (b) tiene instancias de subred privada y permite el acceso a

internet sin mostrar la dirección IP, por medio de un gateway NAT (Network Address Translation) para una subred pública; (c) el tráfico para las instancias VPC puede dirigirse al centro de datos de la compañía por la conexión VPN (Virtual Private Network) cifrada; (d) permite tener una conexión privada entre otras VPC dentro de AWS; (e) se puede acceder de modo privado a otros servicios de AWS sin tener la necesidad de utilizar un Gateway o proxy, entre otros y (f) permite sintetizar la arquitectura interna de la red de los diversos servicios de Amazon VPC.

Los beneficios de Amazon VPC son los siguientes: (a) seguridad, por medio de los grupos de seguridad y las listas de control que filtran el tráfico de entrada/salida de las instancias de la subred. Se tiene el control de direccionar las instancias para que se puedan ejecutar en un hardware dedicado para atender a un proceso que es crítico; (b) se puede crear una VPC fácilmente por medio del administrador de AWS; a su vez subredes, grupos de seguridad, rangos de IP y tablas de ruteo, todo en un mismo lugar y (c) adquiere los beneficios de escalabilidad de AWS, ajustando sus recursos de manera automática, según el tamaño de la instancia y ahorrando costos (Carrillo Ordoñez, 2013).

CAPÍTULO III

METODOLOGÍA

Introducción

En esta sección se describe la arquitectura Back End propuesta como plataforma para sistemas escolares. Como se explicó en el capítulo II, sobre los proveedores que brindan el servicio en la nube, revisando sus características y sus fortalezas y debilidades, para este proyecto se optó por la utilización de la plataforma de Amazon Web Services (AWS), por las razones siguientes:

1. AWS tiene una interfaz sencilla e intuitiva para los clientes; su facilidad de uso y diseño permite que puedan hospedar de forma rápida y segura alguna aplicación. Posee documentación para obtener acceso a sus plataformas de hospedaje en las aplicaciones de AWS.

2. Los servicios que ofrece AWS son de diversos tipos. Permitiendo el apoyo de la arquitectura que se va a proponer y pensando en el desarrollo del software, las creaciones de instancias se realizan en segundos (son rápidas), optimizando tiempos.

3. Se puede crear a comodidad del cliente el ambiente de trabajo, ya que AWS permite seleccionar el sistema operativo, el lenguaje de programación, las base de datos y las plataformas de aplicaciones web, entre otros servicios. Cuando se dispone a desarrollar el proyecto, se tendrá un abanico de opciones para elegir la forma más viable de desarrollo.

4. AWS ofrece la flexibilidad ante los cambios que se requieren dentro del proyecto, ya sean de desarrollo, pruebas, producción, configuración, entre otros.

5. Los proyectos que estén dentro de AWS contarán con una infraestructura informática global, permitiendo tener una escalabilidad total. Por medio de Auto Scaling y Elastic Load Balancing, la aplicación podrá reducirse o ampliarse según se requiera, ya que se tiene el acceso a los recursos de almacenamiento e informáticos. Esto dará soporte cuando se esté utilizando un servicio demandante y se necesite atender todas las peticiones sin dejar de funcionar.

6. AWS brinda la seguridad y protección para los proyectos por medio de una arquitectura de red y un centro de datos de alta seguridad que protegen la privacidad de los clientes. De este modo, se puede escalar en la nube con la seguridad de que AWS está preparada para proteger los datos.

Arquitectura propuesta

Para la arquitectura Back End de sistemas escolares, se optó por utilizar la infraestructura cloud Amazon Web Services. El proyecto se desarrolló en un ambiente on-premise o en la nube, donde se comunica con el sitio de producción que se encuentra en la nube de AWS, por medio de Jenkins y GitHub.

La arquitectura propuesta (ver Figura 6) está diseñada para el ambiente de producción del proyecto, ya que se pretende atender a diversos usuarios en diferentes puntos de acceso, donde el tráfico de datos se elevará a ciertas horas. Es ahí, donde AWS, con los servicios que brinda, ayudará a escalar rápidamente para atender la demanda de peticiones y responder inmediatamente, evitando tiempos muertos del sistema, así como retraso en la información y peculiares que alguna vez se han presentado.

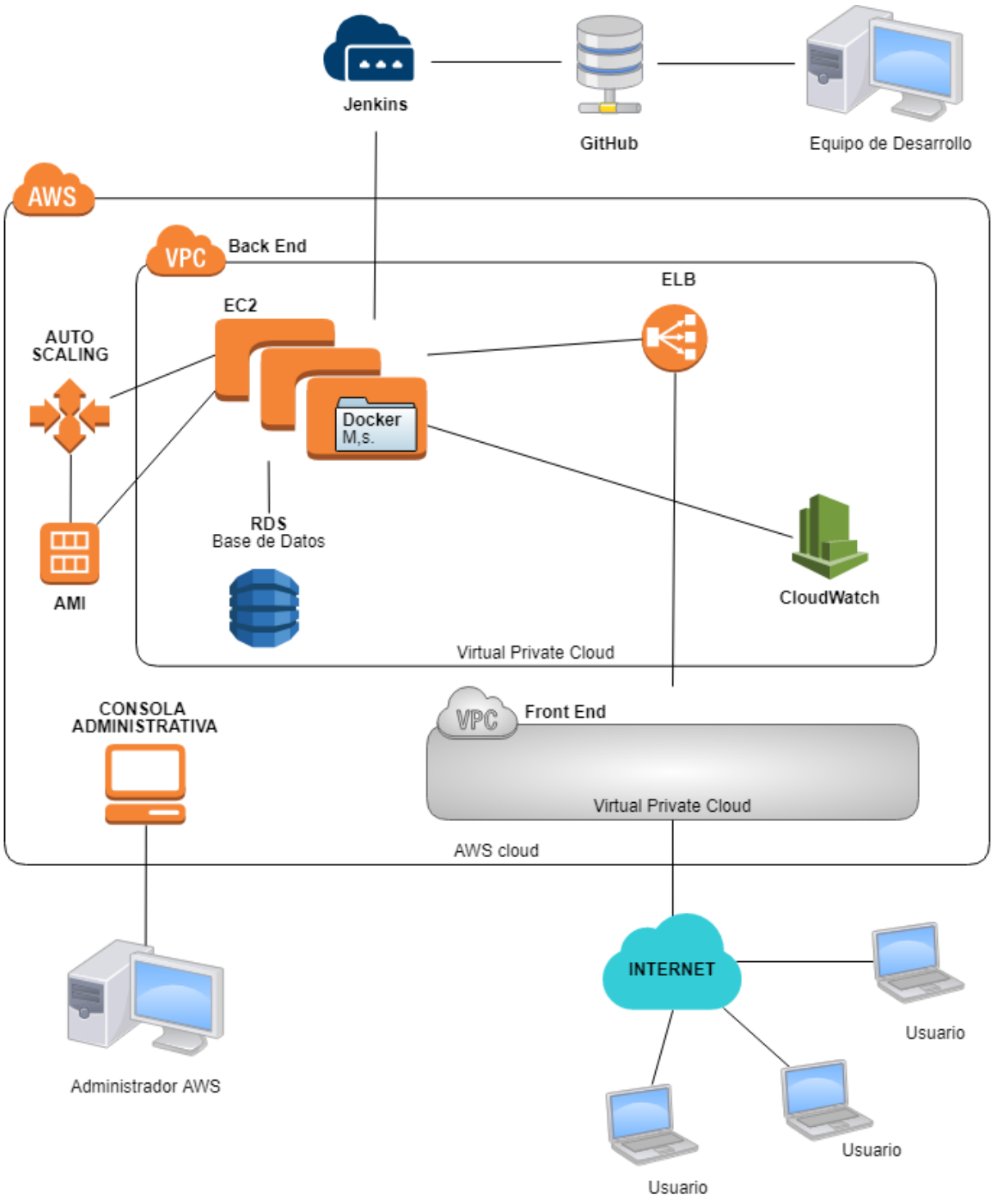


Figura 6. Arquitectura Back End propuesta.

Como se expone en la Figura 6, dentro de AWS se tienen dos ambientes separados en distintas redes de Amazon VPC (Virtual Private Cloud). En el primer ambiente se encuentra el Back End y el segundo ambiente es para Front End, cuyo objetivo final es brindar la seguridad en ambos ambientes, aislándolos a cada uno en su propia red privada. Este tipo de red es similar a las tradicionales que se manejan en los propios data center, a diferencia de que se tiene una infraestructura escalable con AWS.

Dentro de la VPC de Back End existen cuatro servicios, que se explican a continuación:

1. Un servidor EC2, donde se seleccionó un sistema operativo para poder trabajar sobre este (Linux). En este ambiente, se utilizó la tecnología de Docker para automatizar el despliegue de las aplicaciones dentro de contenedores que, a su vez, pueden abstraer y automatizar la virtualización de aplicaciones en múltiples sistemas operativos. Por medio de esta tecnología se puede trasladar todo lo que se encuentra en el EC2 a otro ambiente, como son las configuraciones, conexiones, entre otros; de este modo, se evitan futuras reconfiguraciones. Además, estos contenedores son más ligeros que las máquinas virtuales.

No obstante, dentro de los contenedores se utilizaron microservicios que trabajan en conjunto y de forma autónoma con mucha rapidez. Con ello, los desarrolladores tienen la libertad de decidir qué tipo de aplicaciones necesitan, de acuerdo con la complejidad del entorno, pueden crear instancias en cuestión de segundos, haciendo el desarrollo más fácil y eficaz.

Los microservicios y contenedores han sido adoptados por varias organizaciones tanto en los ambientes de desarrollo como en producción, atendiendo diariamente

millones de transacciones (APIs, consumo directo).

En la Figura 7, se muestra la configuración que se realizó en el EC2, donde se tiene un microservicio llamado “api-person”; así mismo, se encuentra dentro de un contenedor, el cual da acceso a los datos del registro de persona, para luego darle roles y privilegios (cliente, secretaria, asistente, profesor, entre otros).

Al momento de crear un EC2 para los microservicios, se tiene que identificar qué servicios son transaccionales (lo más consultados) y estos se deben poner dentro de un EC2 para permitir su escalamiento. Los otros servicios que no son tan transaccionales, se pueden agrupar dentro de un EC2, donde pueden abastecer la poca demanda. Con esta estructura se asegura que solo los servicios que se vayan a utilizar puedan escalar libremente, más que todo para las aplicaciones que son muy transaccionales; por ejemplo, en un proceso de matrícula habrá un tráfico directamente con respecto a las materias, alumnos, pagos (área financiera), donde esos EC2 pueden escalar rápidamente para devolver respuestas a corto tiempo.

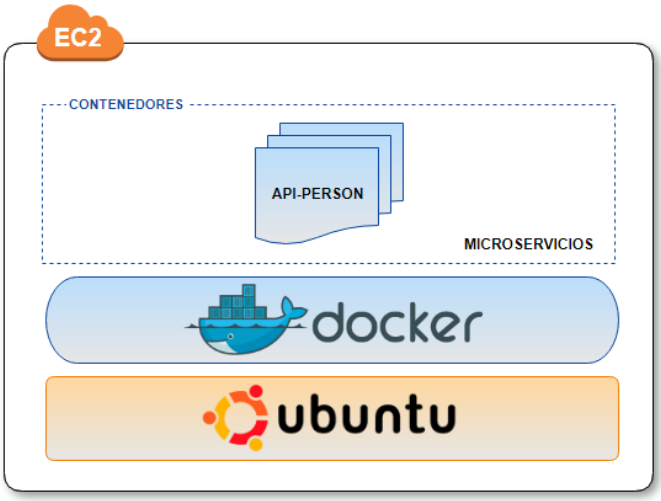


Figura 7. Configuración del EC2.

2. Se utilizó Amazon Machine Image (AMI) para el apoyo de la escalabilidad de los servicios EC2; realizando una imagen AMI (copia fiel) de toda la configuración que tenía EC2 para tenerlo listo al momento de escalar. Al momento de escalar se activa la AMI, generando nuevas copias de EC2 con la configuración que ya se tenía guardada con anterioridad en la imagen; de este modo, su creación es automática sin necesidad de ser realizado por el administrador.

3. Se creó la base de datos con el servicio Amazon RDS (Relational Database Service). Se trabajo con MySQL, que es la base de datos más popular en entornos de desarrollo web.

4. Se creó un ELB (Elastic Load Balancing), para que pueda balancear la carga de trabajo, equilibrando el tráfico de HTTP y HTTPS. Funciona por medio de solicitudes de direccionamiento de tráfico a destinos como instancias EC2, contenedores, almacenamiento y microservicios que se encuentran dentro del Amazon VPC, según el contenido de la solicitud. Mediante Elastic Load Balancing se tiene un óptimo control de las peticiones y/o solicitudes hacia algún servicio. Si se incrementa la demanda de cierto servicio o recurso, este rápidamente escala hasta atender todas las solicitudes y se desaparece de forma automática.

ELB mejora y simplifica la seguridad de las aplicaciones que se alojan. Así mismo, este servicio proporciona el direccionamiento de solicitudes destinadas a la entrega de arquitecturas de aplicaciones modernas, como aplicaciones basadas en contenedores y microservicios.

Por medio del Front End, se ingresan las peticiones para extraer o ingresar datos; su primer encuentro es con el ELB, que revisa qué servicios se solicitan para poder

direccionarlos para su atención; este es un escenario óptimo. Sin embargo, en caso de que un EC2 este saturándose, este escalará rápidamente, por medio de un AMI, para crear más EC2 del mismo tipo y el ELB pueda direccionar a los nuevos EC2 que se crearon para atender todas las peticiones.

5. Amazon CloudWatch es un servicio que monitorea y administra los recursos de las aplicaciones en AWS, donde los desarrolladores o la persona que monitorea el sitio puede tener el control de todo lo que ocurre en la nube por medio de los logs. Si hubiera alguna falla, revisa su ubicación y la atiende. Amazon CloudWatch recopila los datos de monitoreo, ofreciendo una vista del desempeño de todos los recursos y aplicaciones que se tienen en el proyecto, tales como instancias de Amazon EC2, la base de datos de Amazon RDS y Amazon ELB.

Amazon CloudWatch está relacionado con un administrador de consola (persona o equipo), que se encarga directamente de estar revisando este servicio y los recursos de la AWS. El administrador ingresa con sus accesos a AWS; mediante esa interfaz, revisa el servicio donde encuentra los logs que arroja el sistema Back End, como son logs de dibug, error, warning y otros.

Dentro del VCP Front End se encuentra toda la interfaz del software donde se tiene una relación directa con el cliente, ya que por este medio es la conexión entre el usuario y el Back End. El usuario emite una solicitud a través del Front End e internamente consulta al Back End por la respuesta de la solicitud que se refleja en el Front End, donde el usuario aprecia el resultado de su petición. En la Figura 6 se observa de color gris, ya que esta investigación no abarca esta sección. Amazon EC2 interactúa con Jenkins, que es el mediador entre el sitio de producción y el de desarrollo (ver

Figura 6). El equipo de desarrollo se refiere a las personas que están programando el proyecto; este ambiente puede ser on-premise o en la nube; eso dependerá del presupuesto que se tenga para él.

En otro ambiente de AWS se creó otro ambiente de EC2 para alojar al aplicativo Jenkins (ver Figura 8), que permite la relación entre la arquitectura propuesta de AWS con respecto al ordenador que se está utilizando para realizar las pruebas. Cabe mencionar que Jenkins puede estar albergado en otro servidor en la nube para separar el ambiente de desarrollo con el de producción, y tener un óptimo proceso.

Jenkins es un software de integración continua de código abierto y sirve de motor para la automatización de tareas. La etapa de las pruebas finales entre desarrollo y producción puede llevar mucho tiempo para el equipo de desarrollo, identificando los errores para su corrección. Jenkins posee diversos plugins que pueden integrar las etapas del proceso de desarrollo.

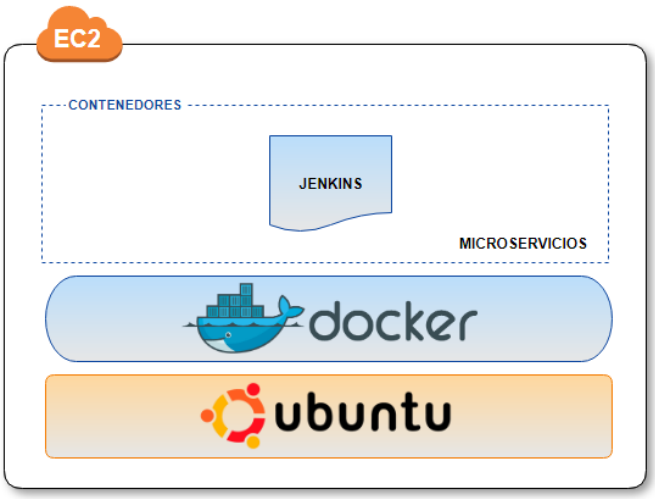


Figura 8. Configuración del EC2 para Jenkins.

Jenkins brinda una plataforma donde se configuran las conexiones del Amazon EC2 (producción) y de GitHub (código de desarrollo), que son los intermediarios al momento de trasladar el código a producción, realizando pruebas de integración continua, compilando el nuevo código, ejecutando las pruebas respectivas, comprobando la calidad al momento de desplegar a producción y, en caso de algún error, evitar subir el código y mantener la versión anterior para que pueda seguir en funcionamiento el sistema. Todo esto se realiza de forma automática e imperceptible para el usuario, evitando los errores en el ambiente de producción, que pueden ser un dolor de cabeza para el equipo encargado del proyecto.

La integración continua es una práctica muy utilizada en el mundo de la programación actual para los equipos de desarrollo que han preferido trabajar con metodologías agiles. Cuantas más integraciones automáticas tenga un proyecto, es mejor, ya que se está probando la calidad del software en todo momento, Jenkins ofrece esto descargando las fuentes necesarias, revisando el control de versiones de código, compilándolo, ejecutando pruebas y subiendo a producción para generar los informes de lo realizado.

Por último, se agregó la herramienta GitHub para apoyar al equipo de trabajo. GitHub es un servicio en la nube de alojamiento de código donde se puede almacenar y administrar. Se armó una base (esqueleto) que se puso en la master, donde todos los integrantes pueden verlo, y de esto parte para el desarrollo del código. Como GitHub tiene una estructura de árbol, se pueden crear tantas ramas a partir del master como se necesiten. Esta herramienta permite que el equipo de trabajo esté sincronizado y pueda acceder desde cualquier punto de la red. Diferentes personas pueden estar trabajando

en el mismo código sin tener que estar delante de la misma máquina; los cambios se reflejan por medio de colores.

Por medio del control de versiones que maneja GitHub se tiene un registro y administración de los cambios de código que se realizan en el proyecto. Se pueden tener muchas versiones para revisar si hubo algún error y poder reemplazarlo por otro y no afectar las funcionalidades nuevas que se van creando, sin tener que modificar el proyecto completamente.

A través de la bifurcación, el desarrollador trabaja de forma segura sin el temor de dañar el proyecto por completo, ya que duplica la parte del código fuente (repositorio) y puede hacer cambios, volver a versiones anteriores y continuar sin problemas, evitando afectar a todo el equipo de trabajo. Una vez que el desarrollador está seguro de que su código está listo y probado, lo puede hacer “oficial” y fusionarlo al código para que puedan partir futuros cambios o construcciones de códigos.

Con GitHub se pretende tener todo el código unificado, preciso y libre de errores, para ser utilizado en producción, donde todo está corriendo a tiempo real. Los cambios que se realizan son para reforzar el desarrollo actual o modificar algo que no se tenía contemplado al iniciar el proyecto y, posteriormente, se convirtió en un riesgo o cambio o una nueva versión del software.

CAPÍTULO IV

RESULTADOS

Introducción

A continuación, se muestran los resultados que se tuvieron de esta investigación. En varias pruebas realizadas se utilizaron algunas herramientas extras de apoyo y configuraciones de AWS, como se explica en la sección configuración.

Pruebas realizadas

Conexión entre desarrollo y producción

La intención de esta prueba es conocer lo rápido y sencillo que se puede realizar un cambio del ambiente de desarrollo a producción en pocos pasos; a su vez, cómo muestra AWS los cambios realizados. El ejemplo gráfico de esta prueba se puede observar en el Apéndice B.

Para esta prueba, se conectó el sitio de desarrollo (laptop personal) con el, repositorio de archivos, GitHub (nube) y, el integrador continuo, Jenkins (nube), para revisar que los cambios se estén efectuando con rapidez y, finalmente, reflejarse en el EC2 de AWS. Se editó el código de respuesta de la URL, donde se están mostrando los datos, como se puede ver en la Figura 9.

Se utilizó el framework Spring de Java y la herramienta Spring Tool Suite para desarrollar y conectarse directamente con GitHub.

En este caso, se modificó el archivo de la URL quitándole el “/32”. Según se vayan agregando o modificando archivos, la lista irá incrementando con los archivos que deben ser subidos al repositorio en la nube de GitHub; previamente se agrega un comentario, que es de campo obligatorio, para subirlos con “Commit and Push”. En el repositorio de GitHub muestra el archivo que se modificó con el mensaje que se agregó.

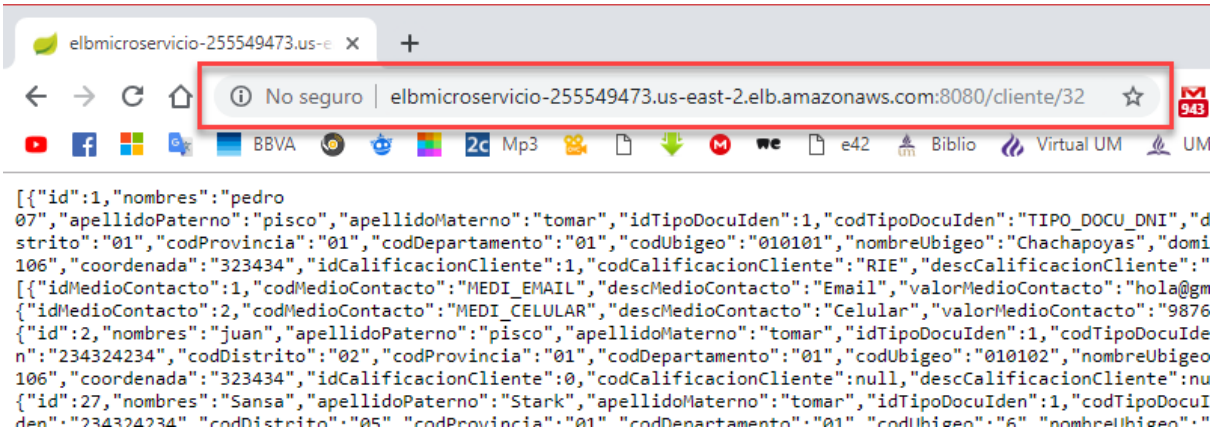


Figura 9. URL del microservicio EC2.

A continuación, Jekins detectó que hay un cambio y, como se puede observar en panel de Jenkins (ver Figura 10), se ejecutó la integración continua para verificar el cambio; por otro lado, si el cambio en el código va a dañar el ambiente de producción, Jenkins rechazará el cambio y continuará con los archivos anteriores, siendo esta la principal función de una herramienta de integración continua. Esta es la gran ventaja de Jenkins; de este modo se evita tener errores en el ambiente de producción, ya que es un sitio muy transaccional y siempre debe estar en funcionamiento. En esta prueba, no hubo ningún problema y se subió la modificación.

Dentro de “console output” de Jenkins se observó la salida de lo que se ejecuta en la consola del servidor del microservicio EC2. En el aplicativo de Jenkins hay muchas opciones para facilitar la integración continua entre el sitio de desarrollo y producción. No se abarcarán todas las secciones, ya que no es el propósito de este proyecto (ver Apéndice B).

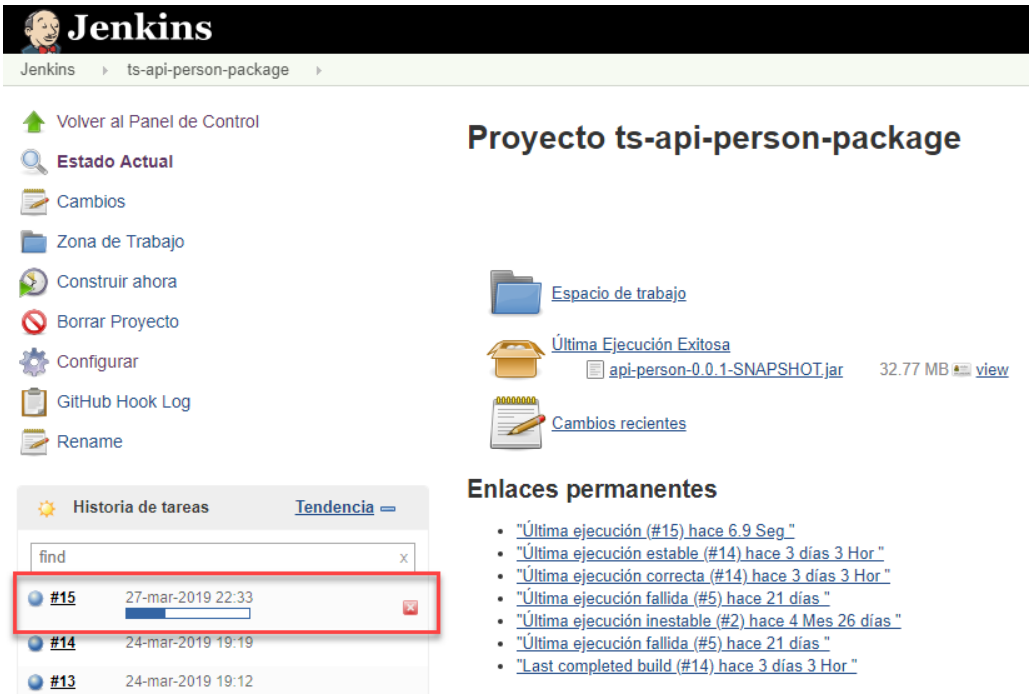


Figura 10. Verificación del cambio.

Como resultado de esta prueba, se puede observar en la Figura 11 que la URL se cambió, mostrando la salida de los datos. Esto refleja un cambio del ambiente de desarrollo a producción de manera óptima, sencilla y segura. Se utilizaron GitHub y Jenkins como herramientas de ejemplo; la selección de estas herramientas dependerá del equipo de desarrollo.

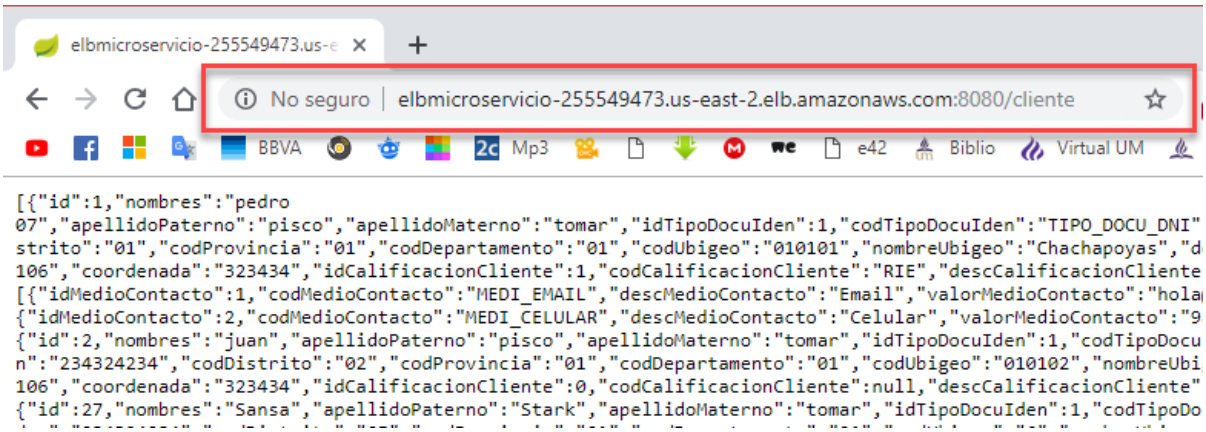
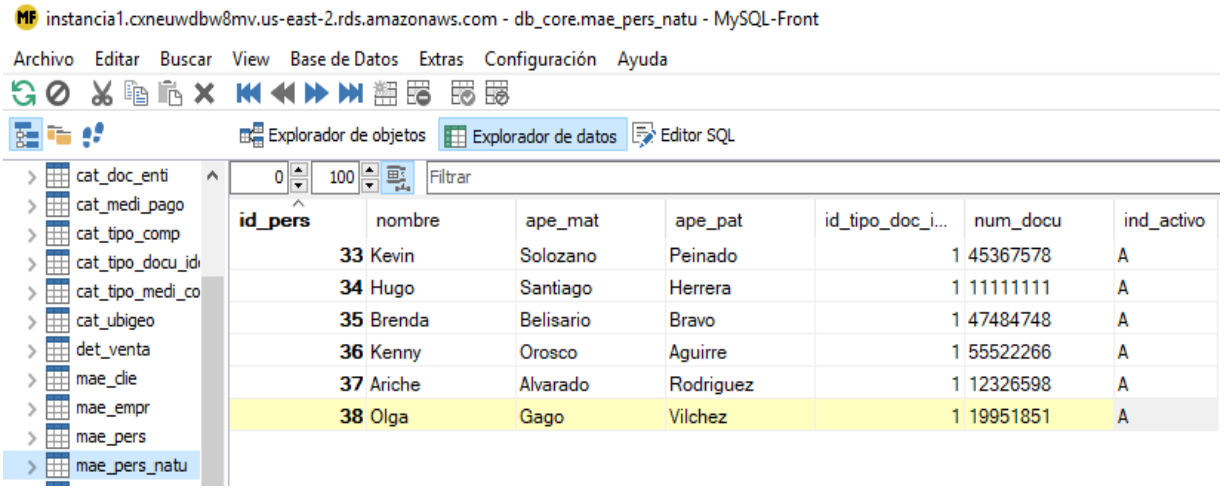


Figura 11. Cambio reflejado en la nueva URL.

Servicio REST

Se creó una base de datos (RDS) MySQL en AWS, donde se encuentran almacenados los datos de las personas que se ingresaron en forma de prueba. Por medio del programa MySQL-Front, se conectó con la instancia RDS para observar los registros de la tabla persona, donde el último registro es el número 38 (ver Figura 12). Todo ejemplo gráfico de esta prueba se puede observar en el Apéndice C.



id_pers	nombre	ape_mat	ape_pat	id_tipo_doc_i...	num_docu	ind_activo
33	Kevin	Solozano	Peinado	1	45367578	A
34	Hugo	Santiago	Herrera	1	11111111	A
35	Brenda	Belisario	Bravo	1	47484748	A
36	Kenny	Orosco	Aguirre	1	55522266	A
37	Ariche	Alvarado	Rodriguez	1	12326598	A
38	Olga	Gago	Vilchez	1	19951851	A

Figura 12. Registros de la tabla persona natural.

Por medio del servicio REST, se realizaron las pruebas para consultar la base de datos; para ello se utilizó la herramienta Postman para efectuar tareas de API REST, donde se pueden probar de manera rápida y sencilla las peticiones.

Se ingresó con el método GET un nuevo cliente por medio del Postman; después de su ejecución, muestra el status y el número de registro (39). Se ingresa en la URL, que se muestra en la sección “location” (ver Apéndice C), donde se encuentra el nuevo registro (ver Figura 13).

Figura 13. Lista del nuevo registro.

En la herramienta MySQL-Front, se revisó la tabla persona (ver Figura 14) donde se encontró el nuevo cliente que se registró (número 39). De esta misma manera se ejecutaron los otros servicios (POST, PUT y DEL); para eliminar a una persona, solo se cambia el status de esta, ya que se pretende tener un registro histórico de todas las personas (ver Apéndice C).

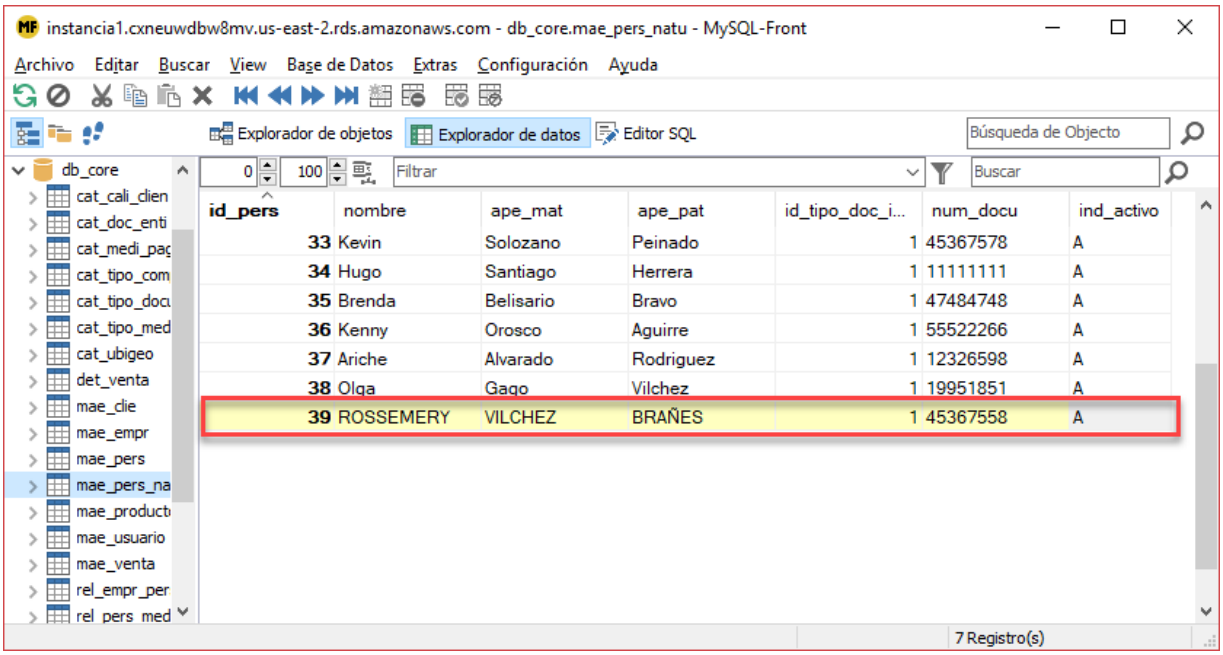


Figura 14. Nuevo registro en la base de datos.

Esta prueba se realizó para verificar si el servicio desarrollado en Java funcionaba correctamente con el servicio REST. Al trabajar con microservicios, en este caso “api-person” que se encuentra dentro de un EC2, y exponer sus datos, es más práctico realizarlo por el servicio REST; esto quiere decir que la relación entre microservicios es más rápida por medio del servicio REST donde puede utilizar GET, POST, PUT y DEL.

Al utilizar “Json”, el tamaño del request es más pequeño, optimizando la respuesta en comparación con un XML, que es más grande y, por ende, más lento. El código que se desarrolló para este servicio se encuentra en el Apéndice C, donde muestra sus cuatro métodos.

Escalabilidad de la arquitectura en AWS

Como se comentó anteriormente, los servicios de AWS pueden escalar al notar que un recurso tiene una alta demanda o al cumplir alguna regla que se configuró en el Auto Scaling. Para esta prueba se creó un servicio que, al momento de ser ejecutado, empieza a realizar un bucle pesado que consume los recursos del CPU (ver Apéndice D).

Se configuró una alarma, para que el EC2 escale. Cuando la utilización del CPU sea mayor a 50 bits por segundo, se creará una nueva instancia después de 30 segundos. En la consola del EC2, se encuentra el servicio llamado “Microservicio” que está detenido, ya que se creó una imagen (AMI) de este servicio para ser utilizada en la creación de instancias. Una de ella es la que lleva de nombre “EC2_TEST” (ver Figura 15), que se está ejecutando por la configuración que se realizó en grupo de Auto Scaling.

Se configuró el grupo Auto Scaling donde el mínimo de instancias iniciadas es de una instancia y el máximo es de cuatro instancias.

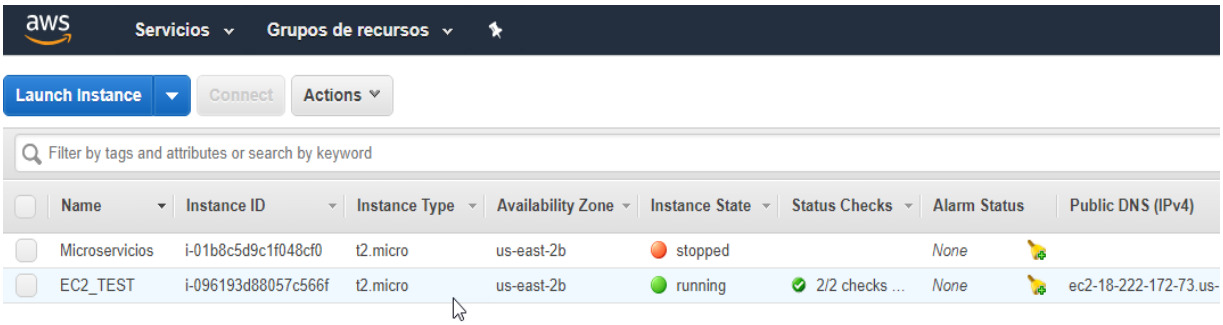


Figura 15. Interfaz del EC2.

Estas reglas son variables y ajustables: de acuerdo al tipo de tráfico que tenga el sitio web o el servicio, puede ser diariamente o semanalmente, entre otros. AWS muestra las estadísticas de entrada a los servicios para poder realizar una toma de decisiones si se desea cambiar la configuración.

Por medio de la herramienta PuTTy, que es un cliente Telnet y SSH, permite conectarse remotamente al servicio “EC2_TEST”, para revisar el uso del CPU. En un principio se dispuso de un índice del 0.3% (ver Apéndice D), porque no se encontraba en ejecución. Con la herramienta Postman, se ejecutó el servicio de nombre “servicio pesado” que empezó la prueba de escalabilidad de AWS (ver Figura 16).

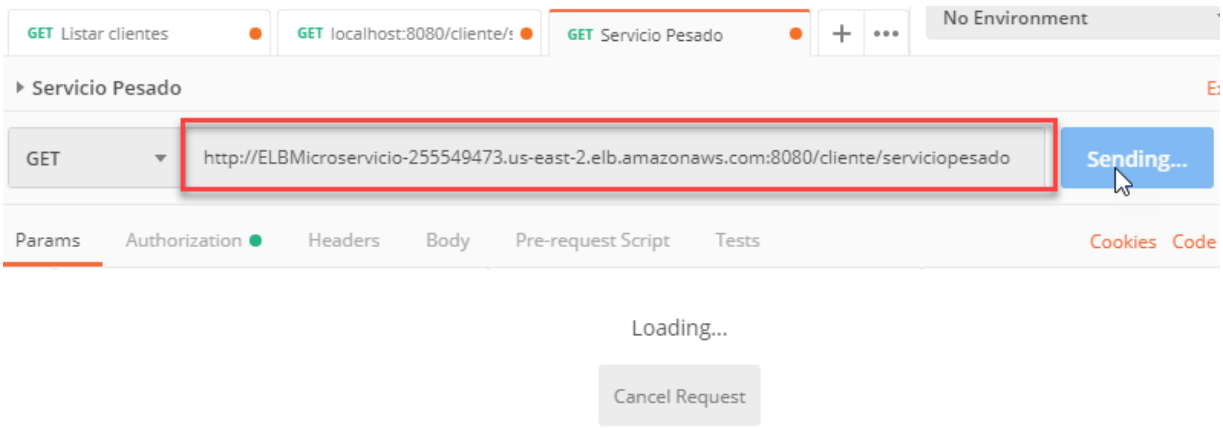


Figura 16. Ejecución del servicio pesado.

En la consola del servicio “EC2_TEST” (ver Figura 17), se mostró el incremento del porcentaje con respecto al uso del CPU. Empezó con un 0.3% y periódicamente ascendió a un 37% de uso. Esto quiere decir que se enestaba ejecutando el “servicio pesado”.

En el panel del AWS (ver Figura 18), se mostró la creación de las nuevas instancias para balancear la carga de trabajo que se estaba ejecutando.

Este servicio se detuvo cuando finalizó la carga de trabajo y otra vez volvió a un 0.3% de uso el CPU, ya que no hay funcionamiento de los servicios. Pero en un ejemplo real, en las horas laborables, un servicio siempre va a estar siendo utilizado por los clientes o usuarios propios de la empresa, así que su escalabilidad será más frecuente.

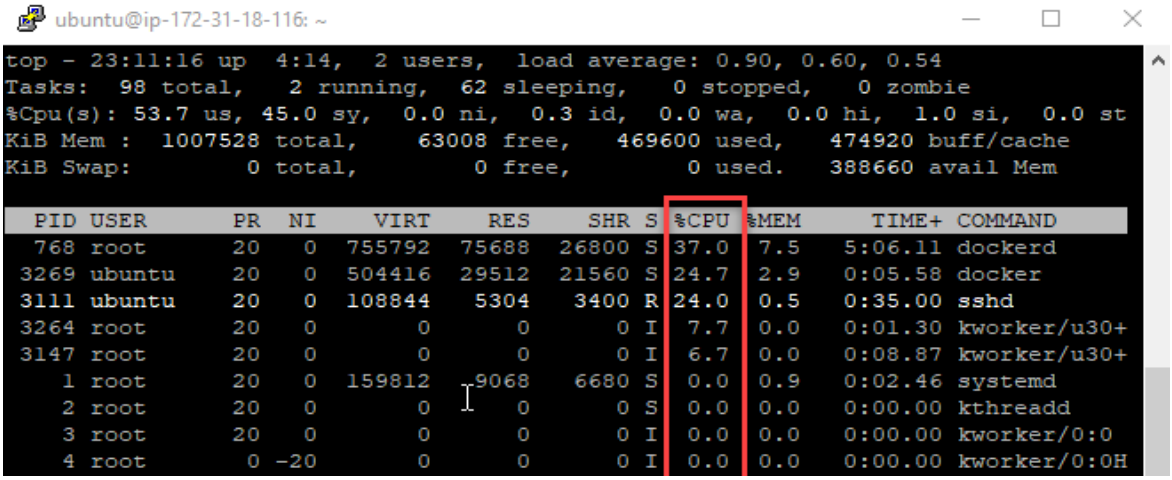


Figura 17. Consola del EC2, revisión del estado del CPU.

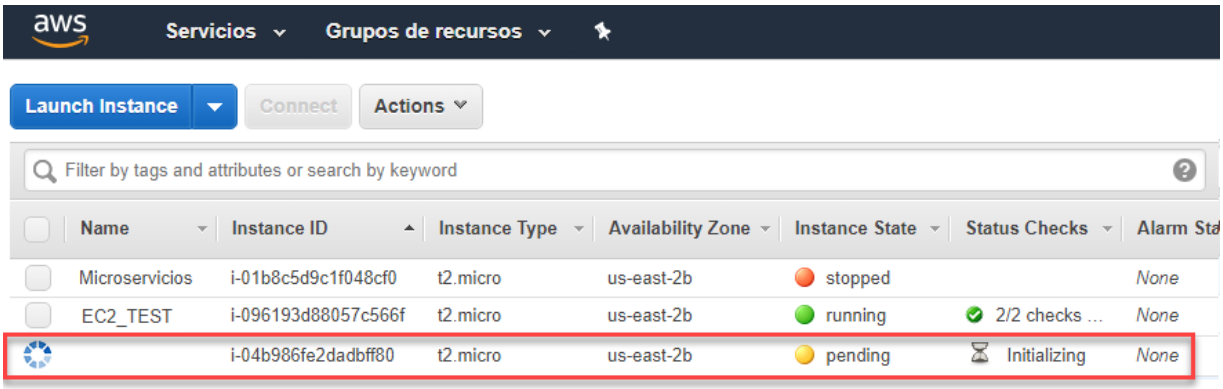


Figura 18. Creación de un nuevo EC2.

Por medio de Amazon CloudWatch se mostró gráficamente el trabajo que realizaron las instancias EC2, teniendo picos altos por los trabajos procesados y picos bajos donde no hay nada en ejecución. Esta herramienta es de gran apoyo, ya que por medio de los gráficos muestra los resultados y rápidamente se puede ver la fluctuación del tráfico: en qué horas son más recurrentes, días específicos o fines de semana, entre otros. También se pueden enviar las notificaciones al correo electrónico, en este caso, de la nueva instancia que se creó (ver Apéndice D).

En las Figuras 19 y 20 se muestran las gráficas de trabajo de los servicios EC2 con respecto a la ejecución del “servicio pesado”.

De este modo, se pueden configurar todos los servicios de AWS, para la base de datos y balanceo de carga, entre otros. De este modo se muestra que la principal ventaja de AWS es la escalabilidad en todos los servicios que ofrece.

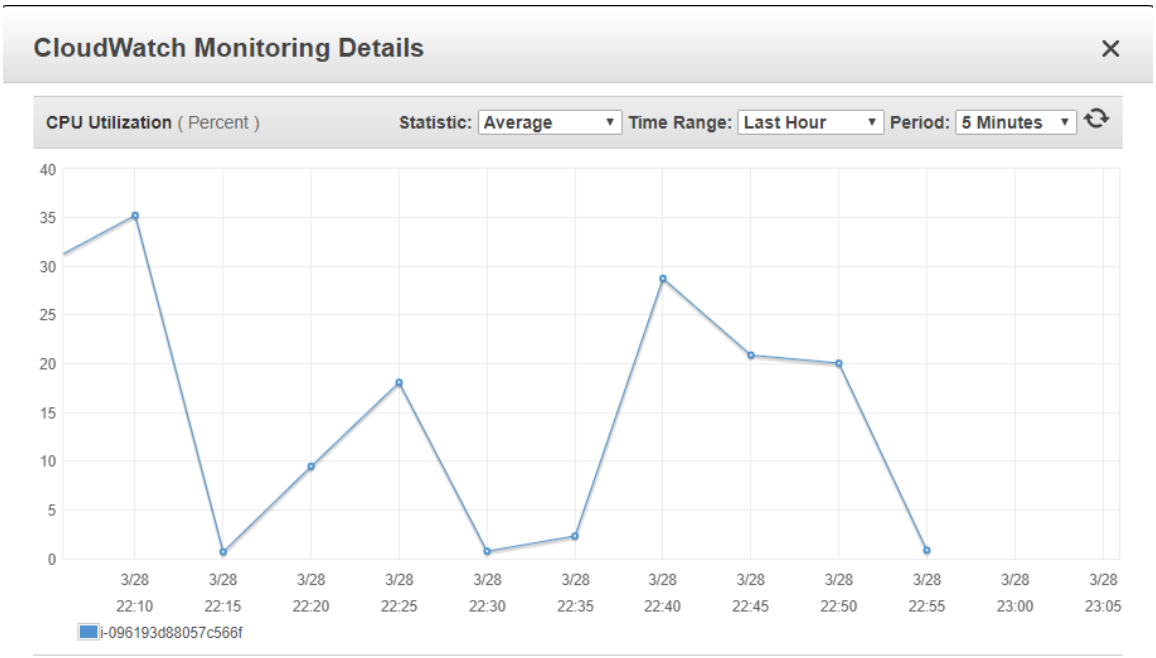


Figura 19. Historial de trabajo del CPU del “EC2_TEST”.

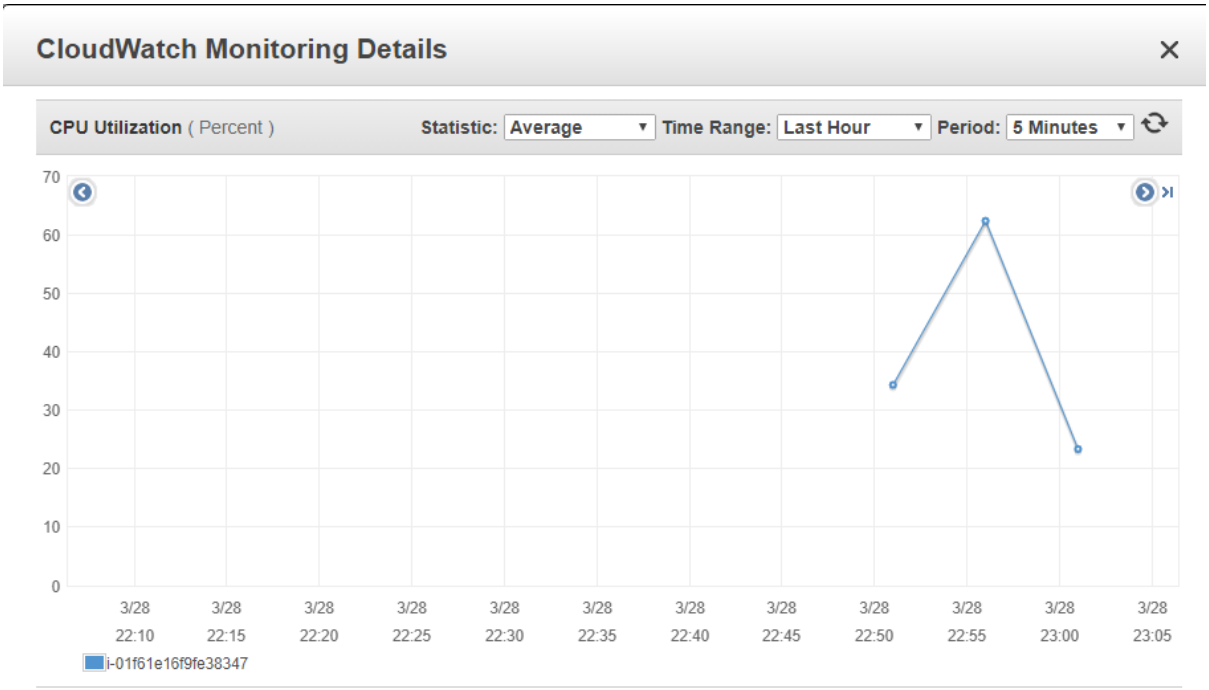


Figura 20. Historial de trabajo del CPU del nuevo EC2.

Por medio de esta prueba se consiguió comprobar la escalabilidad de las instancias de EC2.

Cuando se está solicitando un servicio y se genera una fila de peticiones, este servicio puede escalar rápidamente para dividir la carga de trabajo entre las instancias que se crean y cuando se dejan de utilizar, las instancias empezarán a eliminarse automáticamente al notar la baja demanda de trabajo, puesto que una sola instancia puede satisfacer la demanda.

Cabe mencionar que Amazon Web Services cobra por hora, permitiendo que se pueda pagar por lo que realmente se está utilizando; habrá tiempos más transitados (horas de trabajo) que otros, teniendo diferentes montos de facturación al año.

Seguridad de la arquitectura en AWS

Para realizar la prueba de seguridad, se creó una nueva alarma donde su configuración es el número de entradas al servicio, y que sea menor o igual a 500 byte; en caso contrario, se podrá escalar el EC2 en un minuto. El ejemplo gráfico de esta prueba se puede observar en el Apéndice E.

Se utilizó el aplicativo SoapUI, que es una herramienta de testing, para realizar métodos de repetición, captura y pruebas de carga, entre otros. Se configuró para enviar mil líneas de peticiones; es decir, que esta herramienta simula a mil usuarios que accedan a la instancia “EC2_TEST” (ver Figura 21).

En la consola del microservicio “EC2_TEST” (ver Figura 22), se mostró incremento con respecto a la carga de trabajo de 71.4% uso de CPU y 34.5% uso de memoria, activando las reglas de escalado para crear más servicios y atender las peticiones entrantes (ver Figura 23).

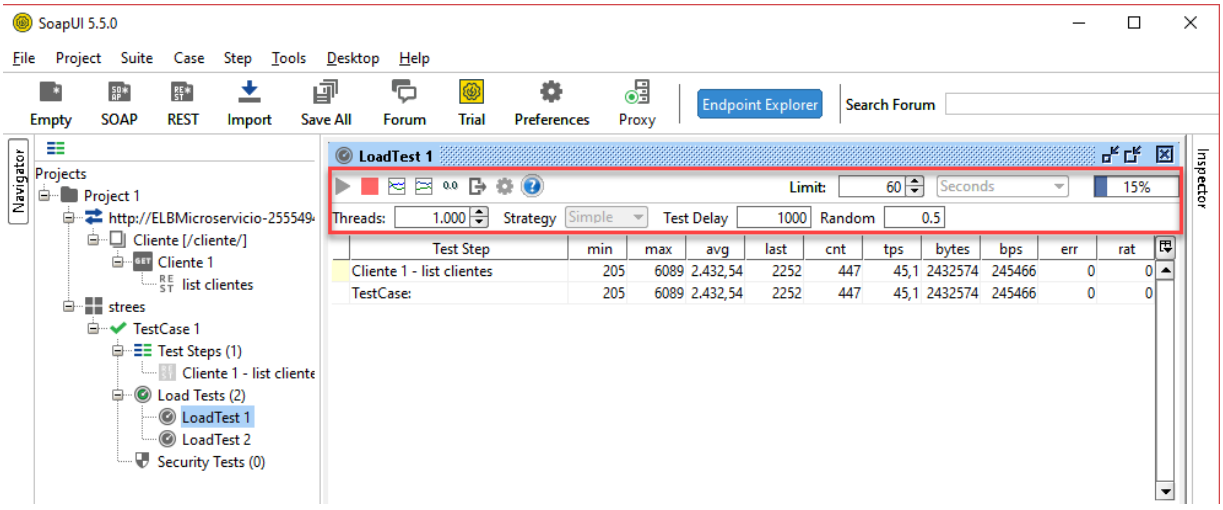


Figura 21. Envío de mil peticiones al EC2.

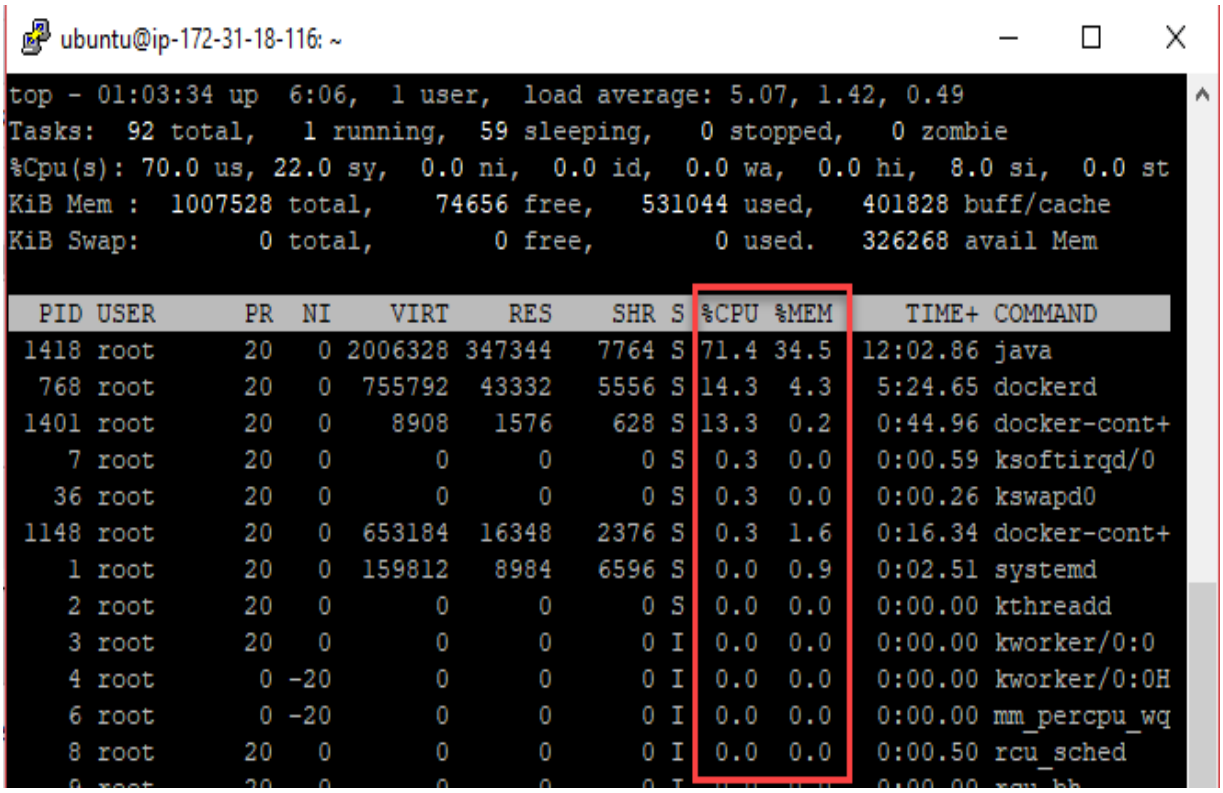


Figura 22. Revisión del uso de los recursos de memoria y CPU.

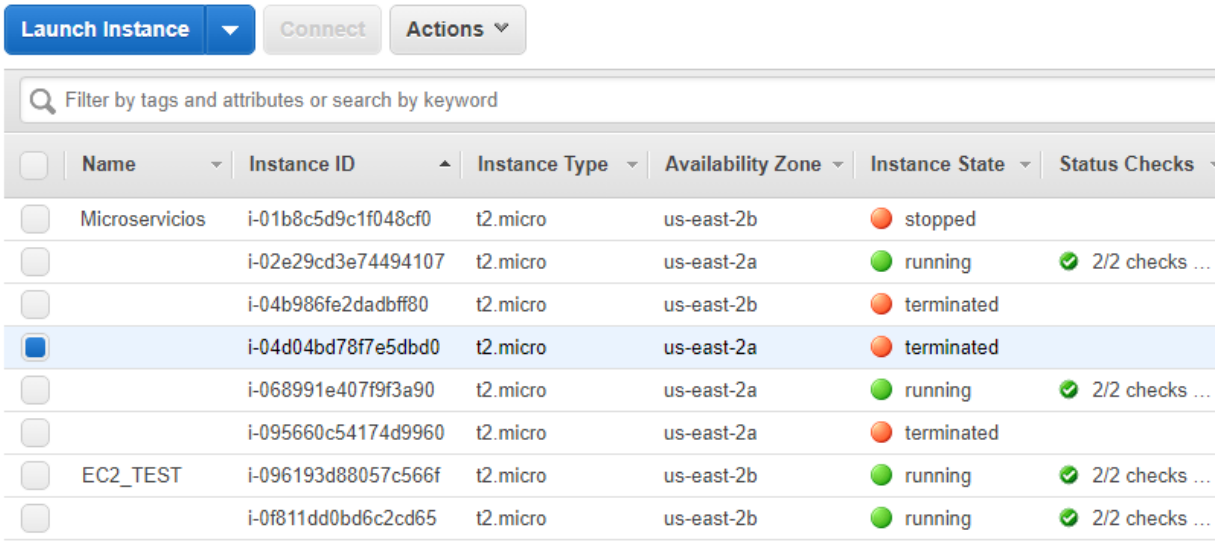


Figura 23. Ejecución de nuevas EC2.

Según como se fue repitiendo esta prueba, AWS observó un patrón sospechoso con respecto a los envíos masivos y lo consideró un ataque al aplicativo web. Entonces empezó a denegar el acceso a las múltiples peticiones que enviaba el aplicativo SoapUI; de esta manera, cerró la conexión y denegó el servicio, como se observa en la Figura 24.

Nuevamente se activó la alarma de seguridad que envió las acciones que se realizaron al administrador, vía correo electrónico, alertando de la actividad sospechosa. Por medio de CloudWatch se observó el historial de la alarma, con respecto a las pruebas realizadas del envío masivo de peticiones (ver Figura 25).

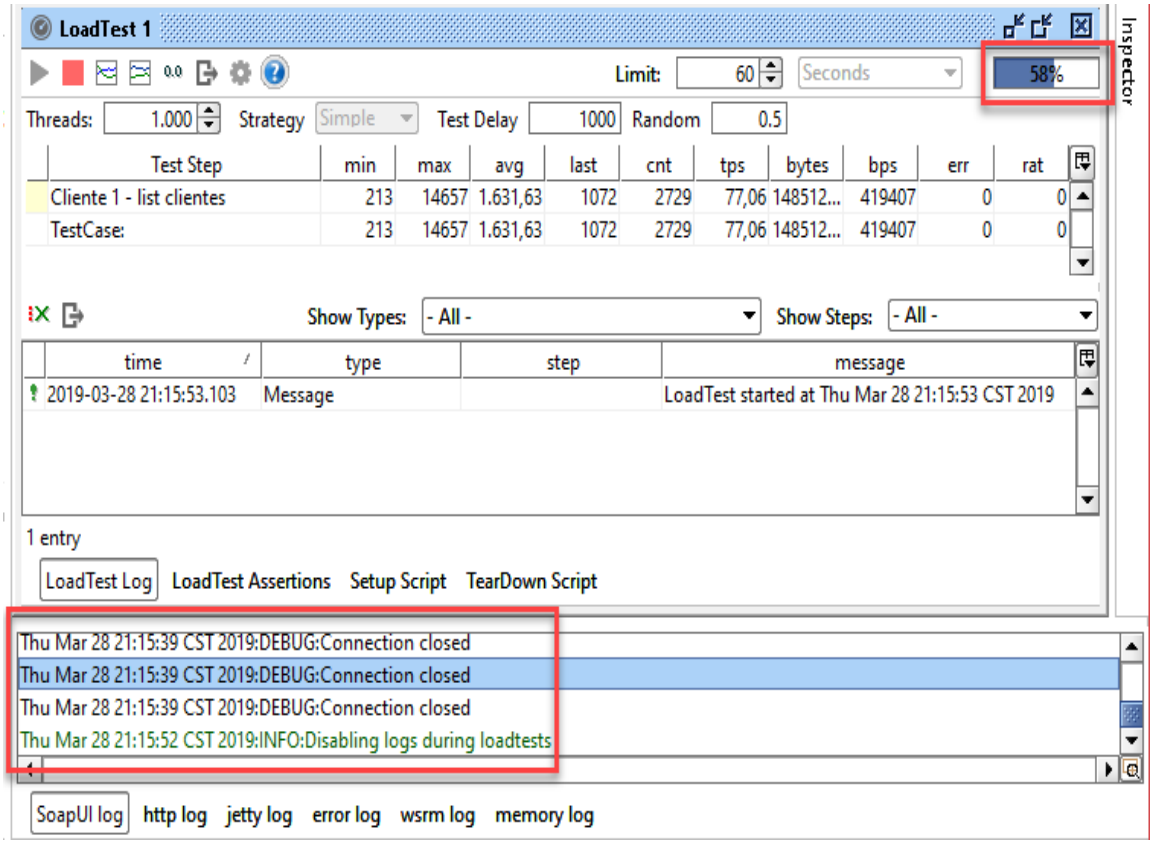


Figura 24. Errores de conectividad con EC2.

Se conoce la existencia de aplicaciones, virus que pueden ser enviados al ambiente de producción con la finalidad de dejar sin funcionamiento al aplicativo web, generando pérdida de información. Por medio de esta prueba, se verificó la seguridad que tiene el AWS con respecto a ataques maliciosos o ingresos de dudosa procedencia, ya que es importante proteger la información de la organización en todo momento.

AWS tiene toda una infraestructura dominante con respecto a la seguridad, activación de nuevos servicios y protección. Si algún servicio es altamente transaccional o necesita estar expuesto para transferir información se tiene una política para asegurarlo.

Figura 25. Detalles e historial de la alarma seguridad.

Configuración

En esta sección se encuentra la configuración que se realizó para la propuesta de la arquitectura Back End.

1. Como se explicó en la Figura 7, dentro de Amazon EC2 (ver Apéndice F) se creó una instancia donde está albergado el contenedor Docker, con el sistema operativo Linux, para trabajar con los microservicios (ver Figura 26).

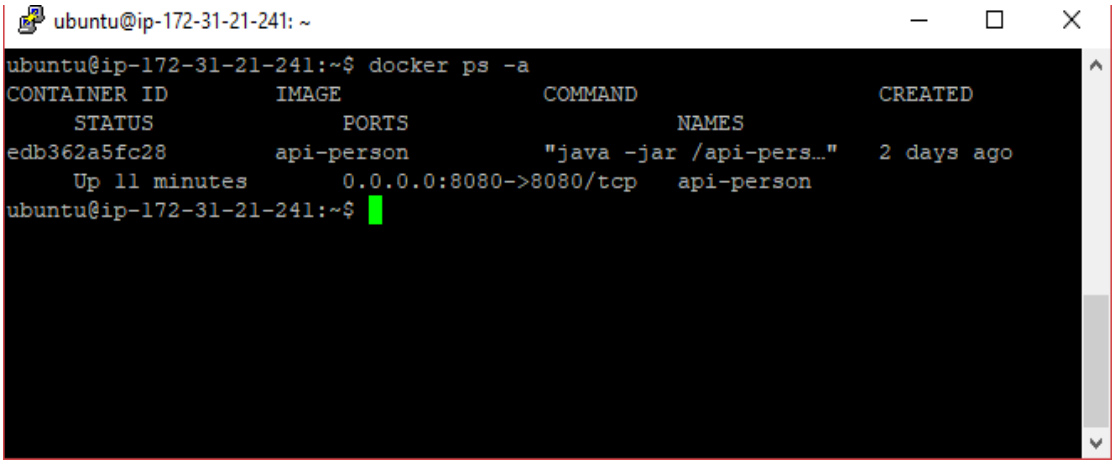


Figura 26. Consola de la instancia EC2 “microservicio”.

2. Se creó una imagen (AMI) del microservicio que contiene toda la configuración de este, facilitando el escalado de las instancias (ver Figura 27). Cabe mencionar que se puede hacer diversas copias de los EC2 que se configuré; de este modo, tenerlas listas para utilizarlas en el autoescalado.

3. Se creó una base de datos en MySQL y para ello se utilizó Amazon RDS. En la Figura 28 se muestra su panel de control.

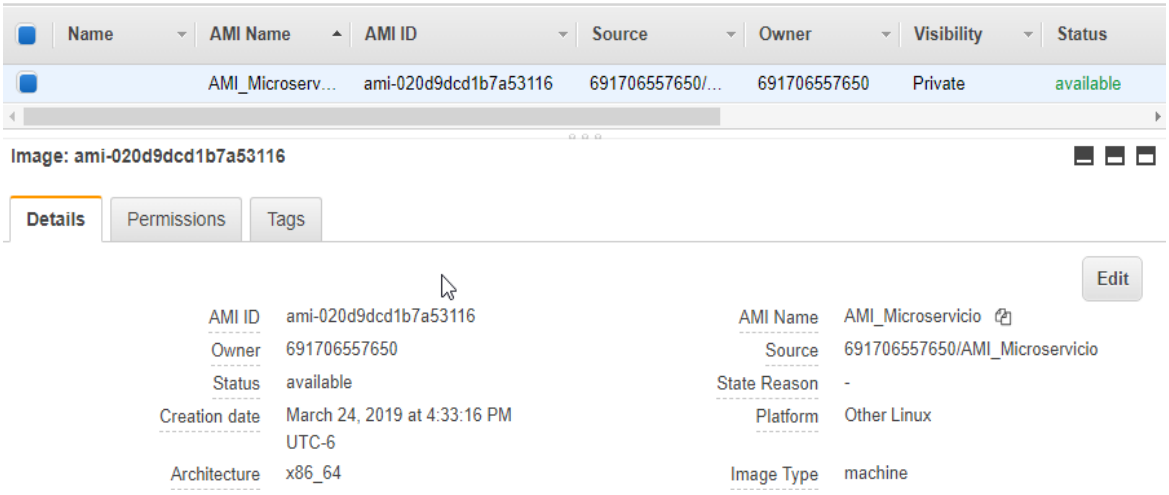


Figura 27. Detalles de la AMI creada.

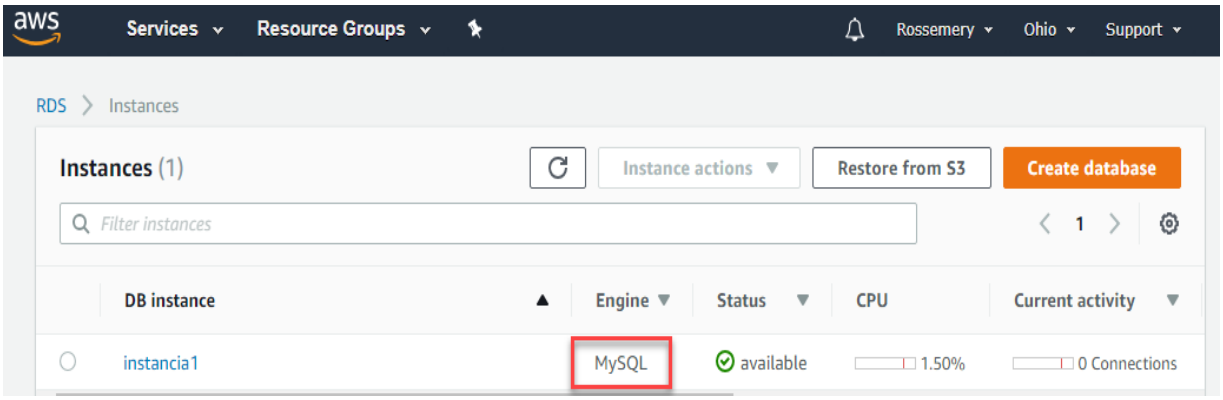


Figura 28. Instancia de la base de datos MySQL

4. Se creó un ELB (ver Figura 29) de tipo “balanceador de carga de aplicaciones” que son muy flexibles para las aplicaciones web de tráfico HTTP y HTTPS; con esto se pueden tener opciones de enrutamiento y visibilidad dirigidos para las arquitecturas de aplicaciones, como pueden ser microservicios y/o contenedores. El ELB apoyará en direccionar el tráfico de las peticiones entrantes a los EC2 que se tengan disponibles, agilizando así la carga de trabajo.

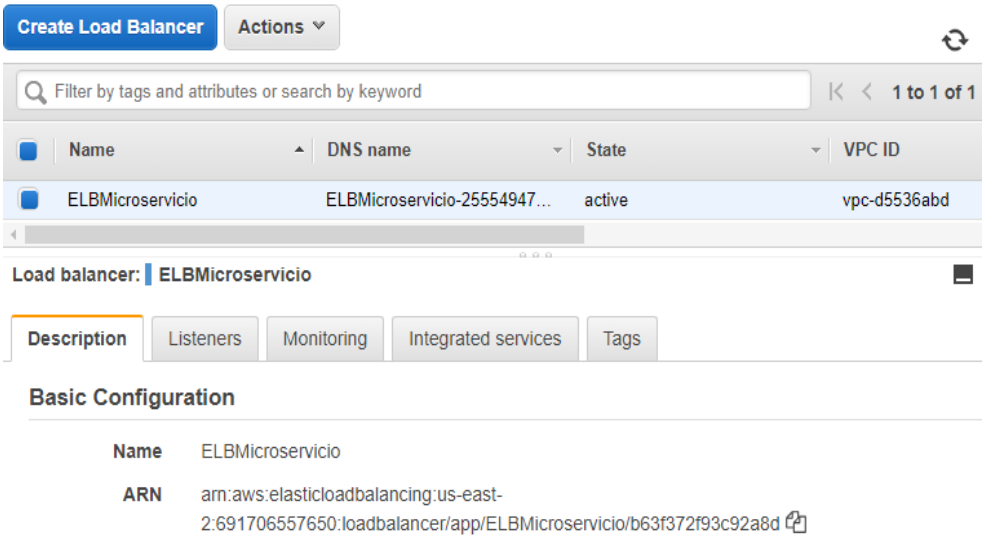


Figura 29. Creación del ELB.

5. Se configuró el Auto Scaling con la AMI del EC2 “Microservicio” (ver Apéndice F). Se creó un grupo de Auto Scaling (ver Figura 30) donde, por medio del AMI, que es una imagen del EC2 “Microservicio”, fue posible crear o reducir instancias según era necesario. Se configuró con un mínimo de uno y un máximo de cuatro instancias. Se recomienda tener un mínimo de dos instancias activas; sin embargo, por cuestión de costos, se optó por uno.

El Auto Scaling está muy ligado con el CloudWatch, ya que todas las configuraciones que activan el escalado son configurables por medio de alarmas que reportan al usuario que se está escalando porque se cumplió alguna regla. Por ejemplo las siguientes: si el rendimiento del CPU llega a 50bits, si hay muchas peticiones a un servicio, si un servicio está fallando, si hay un evento, promoción, día especial en una determinada fecha, que inciten a los usuarios a ingresar al sitio web y aumentar el tráfico de solicitudes a ser atendidas; se pueden configurar las reglas para determinadas fechas.

La respuesta será escalar el servicio que se configuró. Se creó la regla (ver Figura 31) para permitir la escalación de la instancia EC2 cuando la memoria es utilizada más de 50 o 60 bits por segundo; automáticamente se creará una nueva instancia para atender las peticiones entrantes o los servicios que se requieran.

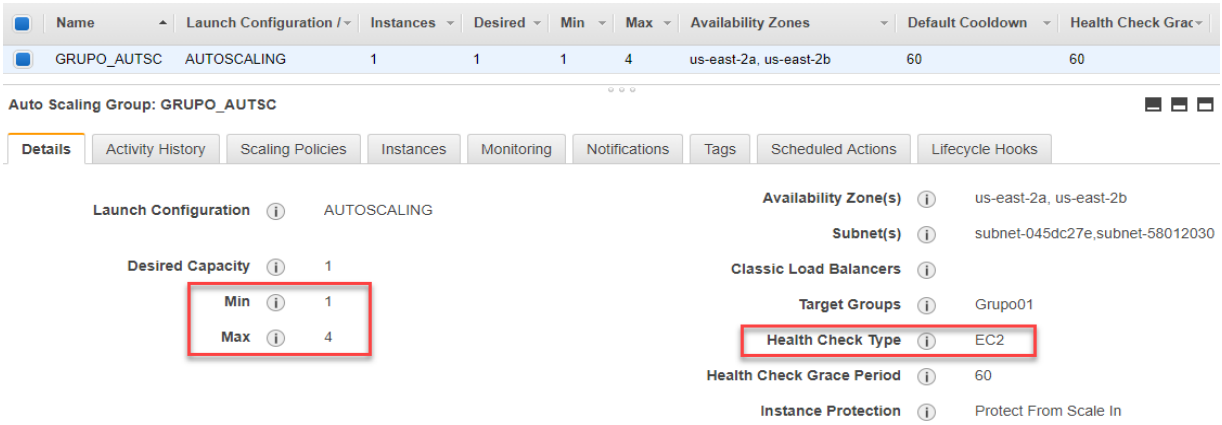


Figura 30. Detalles del grupo de Auto Scaling

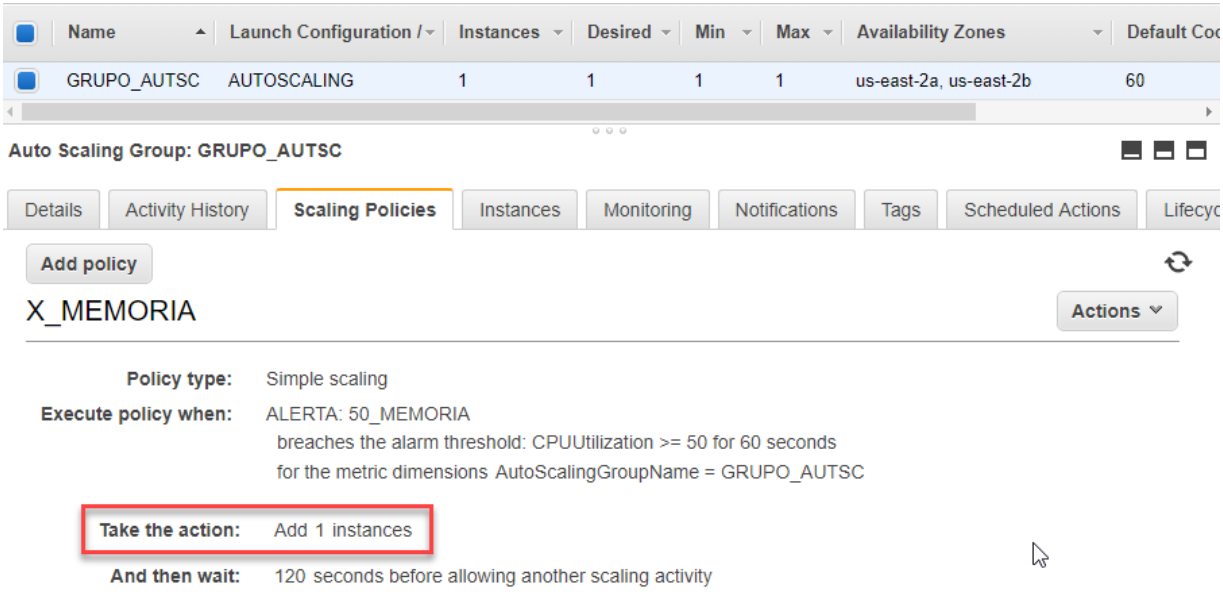


Figura 31. Creación de regla para el autoescalado.

6. Se creó otro ambiente de AWS para tener una instancia para Jenkins, con las mismas características de EC2 “Microservicios”, como se explicó anteriormente en la Figura 8; este microservicio interactúa con el ambiente de desarrollo y sube los cambios a producción en la nube de AWS (ver Figura 32).

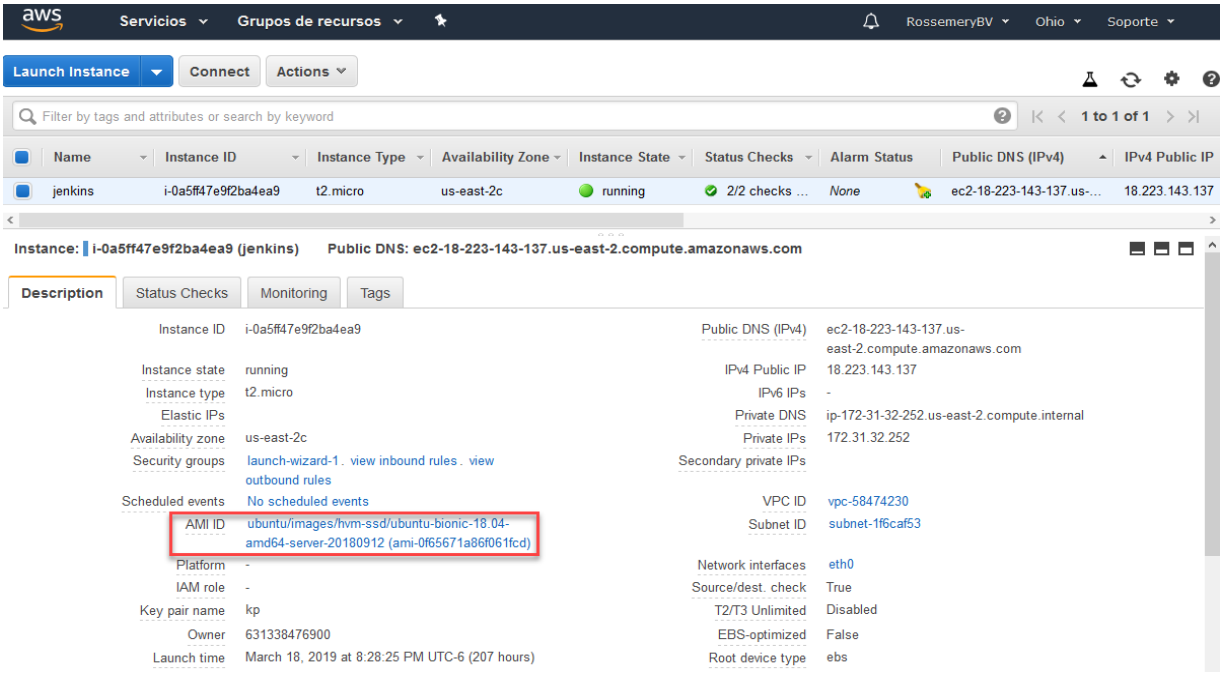


Figura 32. Creación de una instancia EC2 para Jenkins.

Dentro de EC2 “Jenkins”, se pueden crear más servicios y configuraciones de los antes mencionados, pero como este proyecto no está enfocando en la parte de desarrollo, no será necesario; solo se utilizaron para las pruebas.

7. Como repositorio de archivos, se utilizó la herramienta de GitHub, donde se encuentra todo el código de desarrollo que se sube a producción. Se creó un ambiente para realizar las pruebas con persona llamado “api-person” (ver Apéndice F).

8. Para el desarrollo, se utilizó el framework Spring de Java; así mismo, se utilizó Spring Tool Suite, por ser flexible y ágil para crear diversos tipos de arquitecturas dependiendo del proyecto, lo cual la hace una herramienta sencilla y eficaz. Por otro lado, se utilizó Spring boot para crear los microservicios (ver Apéndice F).

CAPÍTULO V

CONCLUSIONES Y TRABAJOS FUTUROS

Conclusiones

Como se expuso en los primeros capítulos de este documento, esta investigación tiene el propósito de proponer una arquitectura Back End que sirva de base para los sistemas escolares y continuar con el desarrollo en una etapa futura.

Como resultado de las pruebas realizadas a la arquitectura propuesta en la nube de Amazon Web Services, se puede concluir lo siguiente.

1. La arquitectura propuesta es flexible y escalable ante la necesidad de atender a varias solicitudes entrantes, y a la vez, minorizar la carga de trabajo ante un servicio que se está solicitando o es muy pesado (demanda más tiempo de respuesta), evitando prolongados tiempos de espera.

2. Sobre la seguridad de la arquitectura, AWS tiene un alto nivel de seguridad para proteger los datos de sus clientes ante situaciones sospechosas, alertando al usuario de lo sucedido. Esta característica puede ser configurable con múltiples reglas de seguridad, desencadenando eventos que ayudan a desviar e impedir exponer los datos.

3. La plataforma de AWS es muy flexible para crear un proyecto de cualquier dimensión desde pequeño hasta un gran tamaño. Como se expuso en el capítulo II, en AWS se tiene toda una infraestructura, contando con un abanico de opciones y configuraciones que se pueden seleccionar, desde el tipo de sistema operativo con el

que se va a trabajar hasta opciones de escalamiento si la organización crece.

Es por ello que se puede conectar con diversos programas que pueden ayudar en el proyecto, como conexiones y estructuras hibridas, según lo disponga el cliente.

4. Por último, se ve la robustez de la arquitectura propuesta. Gracias a los servicios de AWS, está diseñada y preparada para soportar transacciones de gran envergadura, para crear automáticamente nuevos EC2 cuando se requieran, facilitando el trabajo al equipo encargado de esta área. La clave de este proceso es que el arquitecto pueda crear y configurar todo el ambiente pensando en escenarios óptimos y desfavorables para la aplicación web. Una vez realizado esto, será más sencilla la monitorización de los servicios que se están ejecutando; sí se generara un problema o se identificara un patrón continuo, se pueden tomar medidas correctivas para evitar futuros inconvenientes.

Con las características antes mencionadas: seguridad, flexibilidad y escalabilidad, se comprueba la robustez de la arquitectura, ya que se necesita de estos elementos para reforzar todos los aspectos necesarios para tener una plataforma óptima.

A futuras investigaciones

En esta sección se exponen los trabajos posteriores que pueden surgir por medio de esta investigación, como son los siguientes:

1. Las tecnologías que se utilizaron en este proyecto para armar la arquitectura Back End son de las últimas tendencias de esta época, y poco a poco las empresas las han ido adoptando a sus plataformas, consolidando las ventajas que ofrecen. Es por ello que, según el avance la tecnología, esta arquitectura necesitará reafinar algunos cambios, con el apoyo de nuevos servicios, para mejorarla y potenciarla.

2. Esta arquitectura Back End es la plataforma para el proceso de desarrollo de software (programación). En otras palabras, se encuentra lista para que un equipo de desarrollo pueda empezar el proyecto de la construcción del software.

3. Con esta investigación, se pretende incentivar a futuros proyectos a incursionar en la computación en la nube, usar nuevos servicios o métodos para optimizar los procesos y mejorar la calidad, hacer que las maquinas trabajen más tiempo que las personas, evitando los errores humanos. En esta era digital, la tecnológica es un factor clave para la existencia de una empresa u organización, ya que los consumidores cada vez son más exigentes.

4. Con la ayuda de la inteligencia artificial, que últimamente está teniendo un fuerte impacto en la sociedad, como los carros autónomos. Todo se está optimizando, dando lugar a la programación autónoma; esta tecnología puede ayudar en gran manera a este proyecto, para que pueda cambiar su plataforma, integrando nuevos componentes, según se requiera, sin estar pensando si son compatibles unos con otros. Por ejemplo, puede ejecutar patrones de seguridad, si la arquitectura es atacada y se sobrepasan los patrones estándar de seguridad. Que antes de exponer la información, a los administradores les dé tiempo de actuar. La arquitectura autónomamente podrá defenderse al bloquearse o desconectarse para evitar el robo de información. Esto ya se está probando y ejecutando en proyectos pilotos que futuramente se usarán (Torres y Alférez, 2014).

APÉNDICE A

CARACTERÍSTICAS DE PROVEEDORES (SERVICIO EN LA NUBE)

CARACTERISTICA DE PROVEEDORES (SERVICIO EN LA NUBE

	amazon web services	Microsoft Azure	Google Cloud Platform
COMPUTACIÓN	▪ EC2 ofrece servicios informáticos para configurar múltiples máquinas virtuales con la ayuda de AMI predefinidas personalizadas. ▪ Servicios como: Contenedores EC2, AWS Auto Scaling y Lambda, así como Elastic Beanstalk para el despliegue de aplicaciones.	▪ Azure se basa en sus Máquinas virtuales que se adjuntan a otras herramientas como Resource Manager y Cloud Services para implementar aplicaciones en la plataforma en la nube.	▪ Utiliza un motor de cómputo para configurar las máquinas virtuales.
ALMACENAMIENTO	▪ AWS ofrece almacenamiento temporal que comienza a trabajar junto con la instancia. Una vez que se termina la instancia, el almacenamiento se destruye. ▪ También proporciona almacenamiento en bloque que es similar a los discos duros. El almacenamiento de bloque puede adjuntarse a cualquier instancia o mantenerse separado. ▪ Además, ofrece servicios de archivo con Glacier y almacenamiento de objetos con servicios S3.	▪ Microsoft Azure usa blobs de página y almacenamiento temporal para columnas basadas en máquinas virtuales. ▪ Servidor de archivos y block blobs para almacenamiento de objetos.	▪ La plataforma en la nube de Google tiene Google Cloud Storage, para el almacenamiento de objetos. ▪ GCP soporta DBs relacionales a través de Google Cloud SQL. Las tecnologías pioneras de Google, como Big Query, Big Table y Hadoop, son totalmente compatibles.
BASE DE DATOS	▪ AWS es totalmente compatible con las bases de datos relacionales y NoSQL, así como con Big Data. ▪ Amazon no ofrece ningún servicio de copia de seguridad, pero tiene el Glaciar para el almacenamiento de archivos a largo plazo a precios muy asequibles.	▪ Azure admite bases de datos NoSQL, bases de datos relacionales y Big Data a través de HDInsight y Windows Azure Table. ▪ Existe un Redis Cache que es particularmente necesario para el almacenamiento híbrido. ▪ Ofrece servicios de copia de seguridad y almacenamiento de archivos.	▪ Es particularmente popular para el almacenamiento unificado y también para un disco persistente. Tiene una base de datos relacional y la llave en la nube para administrar las cargas de trabajo críticas. ▪ Las dos opciones No SQL en GCP incluyen: Cloud Datastore y Cloud Bigtable. ▪ No hace ningún servicio de copia de seguridad o archivo.

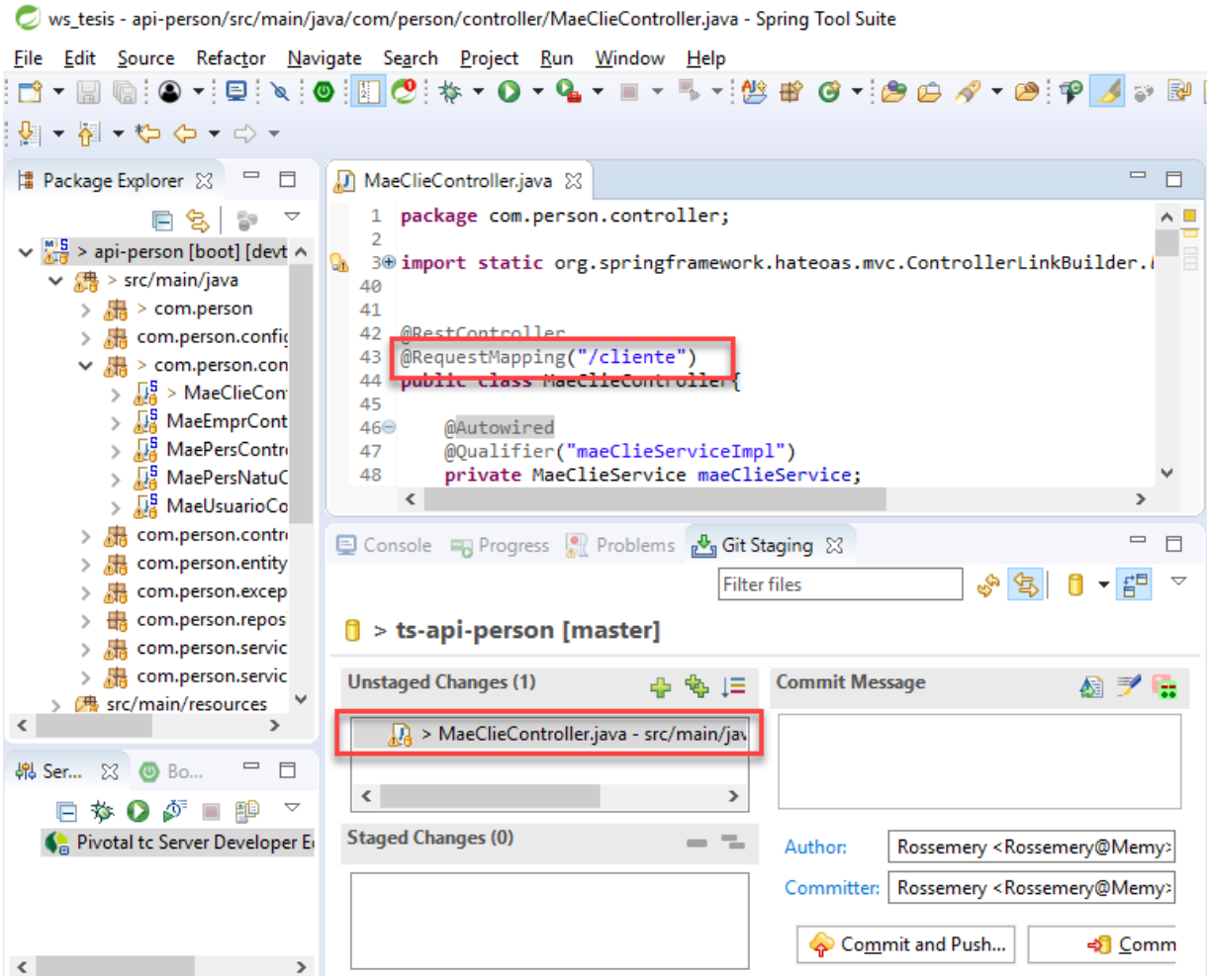
	amazon web services	Microsoft Azure	Google Cloud Platform
REDES	▪ En la nube privada virtual de Amazon (VPC), se puede configurar una red privada como pública, administrar usuarios, privilegios, activar direcciones IP, crear una topología de red y subredes. ▪ Esta plataforma en la nube ofrece soluciones avanzadas para ampliar su centro de datos local a la nube híbrida o pública. ▪ AWS es único en el suministro de Route 53, un servicio web de DNS.	▪ La red virtual de Azure (VNET) permite a los usuarios agrupar máquinas virtuales en redes aisladas en la nube. ▪ Los usuarios pueden identificar el rango de direcciones IP privadas, la topología de red, las puertas de enlace de red y crear subredes.	▪ Cada instancia de Google Compute Engine pertenece a una sola red, que define el rango de direcciones y la dirección de la puerta de enlace para todas las instancias conectadas a ella. ▪ Las reglas de firewall se pueden aplicar a una instancia y pueden recibir una dirección IP pública.
SEGURIDAD	▪ AWS ha establecido altos estándares de seguridad para ofrecer ventajas desde una arquitectura de red y un centro de datos a organizaciones preocupadas por la seguridad. ▪ Permite a los usuarios innovar y escalar la aplicación manteniendo un ecosistema seguro. ▪ Amazon garantiza a los usuarios tener mayor privacidad y más controles al menor costo.	▪ La seguridad de la nube de Azure se divide en cinco capas, como datos, aplicación, host, red y física. La infraestructura de Azure protege el ecosistema de Azure de todas las vulnerabilidades. ▪ Para la seguridad de los datos del usuario, Microsoft ofrece varios servicios, tales como: Controlar, gestionar el acceso y la identidad de los usuarios, redes de seguridad, proceso de cifrado de operación y comunicación, gestionando amenazas	▪ El firewall de próxima generación de FortiGate proporciona seguridad avanzada y firewall crítico para Google Cloud Platform (GCP).
PRECIO	▪ Por hora - redondeado	▪ Por minuto: compromisos redondeados (prepago o mensual)	▪ Por minuto - redondeado (mínimo 10 minutos)

APÉNDICE B

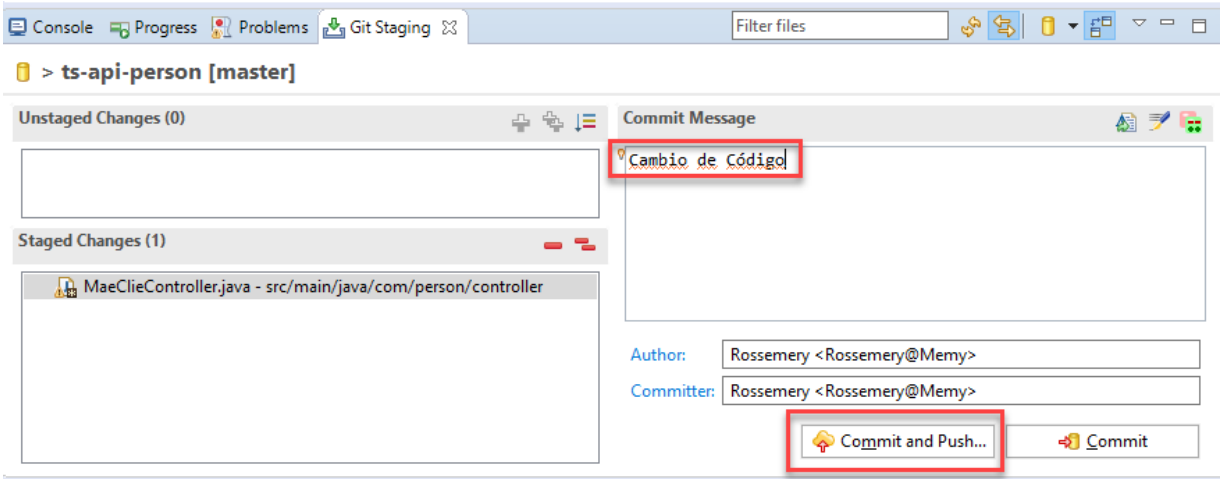
CONEXIÓN ENTRE DESARROLLO Y PRODUCCIÓN

REFLEJAR UN CAMBIO DEL AMBIENTE DE DESARROLLO A PRODUCCIÓN

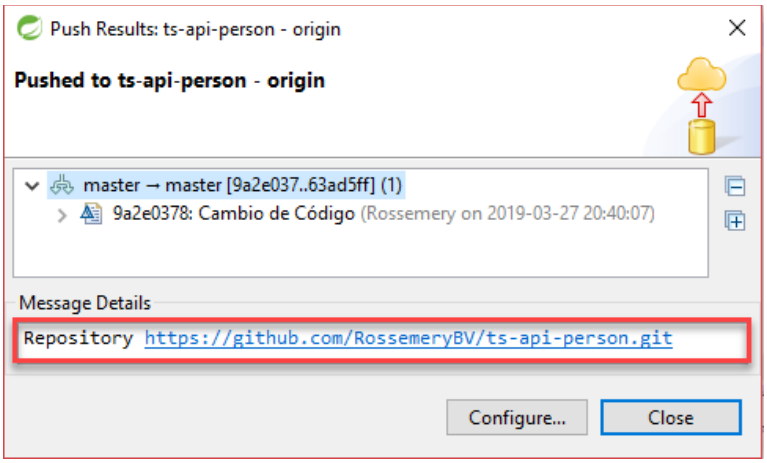
Archivo modificado en la herramienta Spring Tool Suite



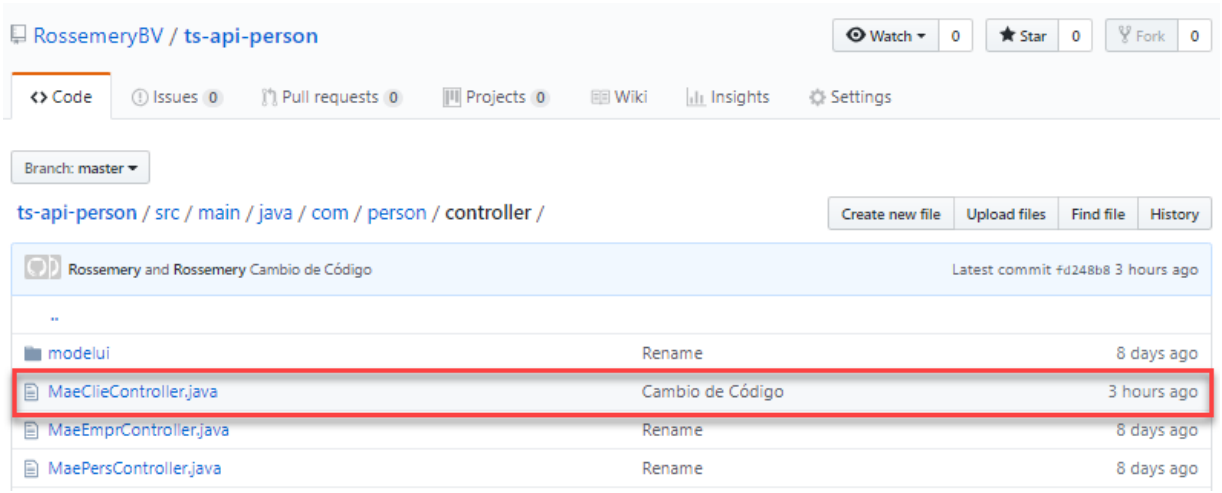
Archivo seleccionado para subir a GitHub



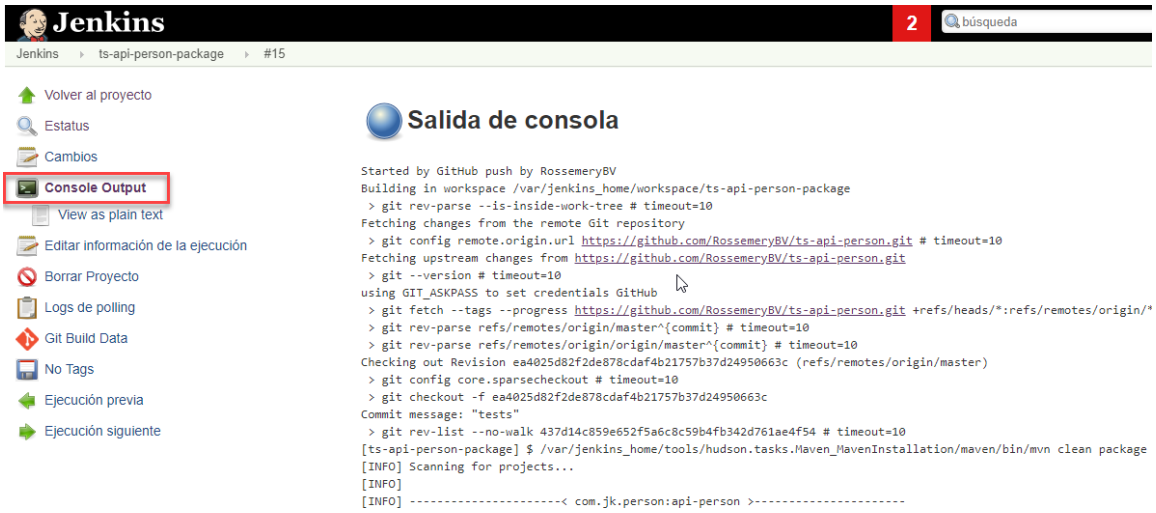
Dirección del repositorio que se subirán los cambios



Repositorio de GitHub.



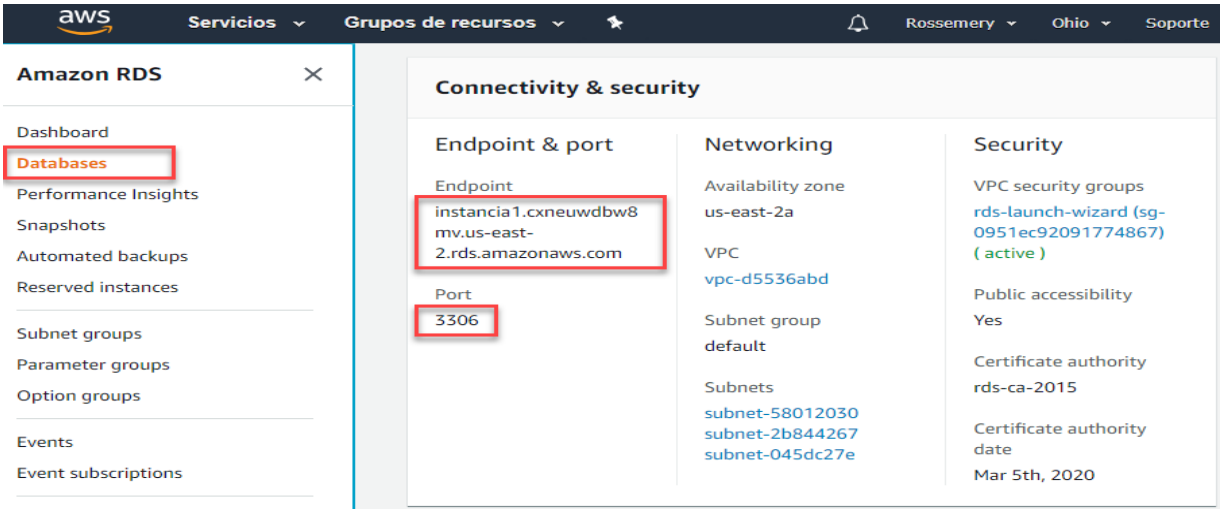
Salida de consola de Jenkins.



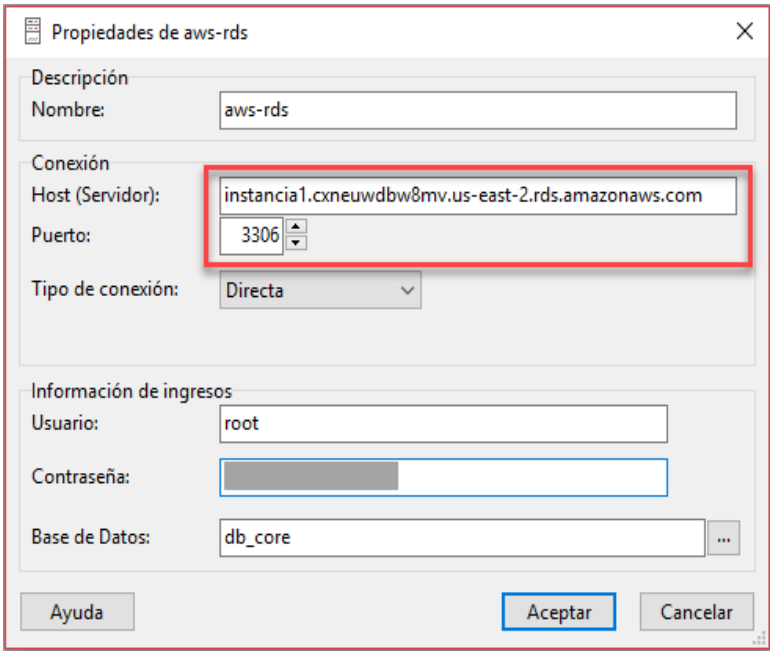
APÉNDICE C

SERVICIO REST

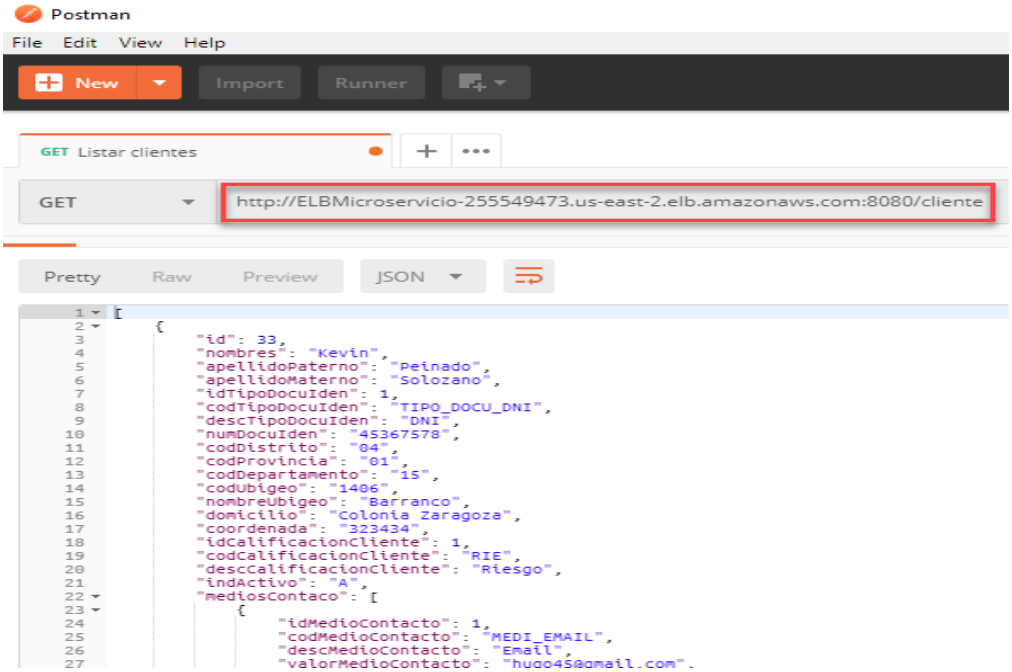
Base de Datos (RDS) en AWS.



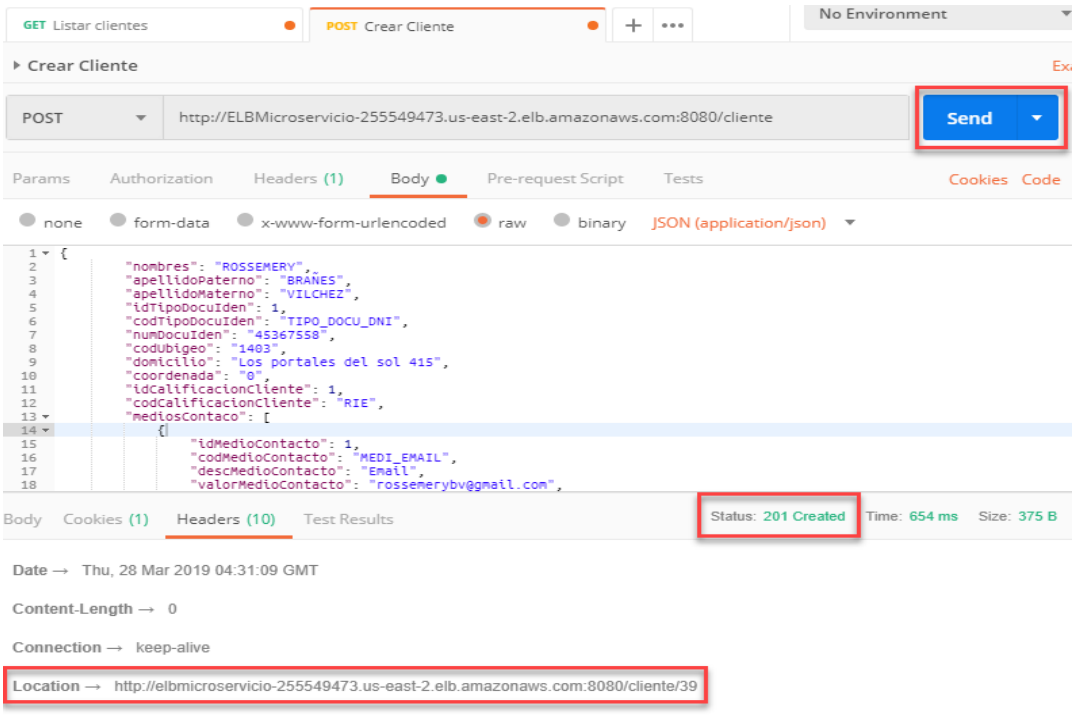
Conexión a la base de datos con la herramienta MySQL-Front



Lista de clientes con la herramienta Postman.



Creación de un nuevo cliente



Servicio PUT, actualizar un cliente

```
@PutMapping()
public ResponseEntity<?> update(@Valid @RequestBody Cliente cliente){
    List<MaeClie> listaClie = maeClieService.findOne(cliente.getId());

    if(listaClie == null) {
        throw new ClienteNotFoundException("id-"+cliente.getId());
    }else if(listaClie.size()<1) {
        throw new ClienteNotFoundException("id-"+cliente.getId());
    }

    MaeClie maeClie = maeClieService.update(cliente.toMaeClie());

    URI location = ServletUriComponentsBuilder
            .fromCurrentRequest()
            .path("/{id}")
            .buildAndExpand(maeClie.getIdClie()).toUri();

    return ResponseEntity.created(location).build();
}
```

Servicio GET, para listar todos los clientes o por ID.

```
@GetMapping
public List<Cliente> listAll(){

    List<MaeClie> maeClies = maeClieService.listAllActives();
    List<Cliente> clientes = new ArrayList<>();
    for(MaeClie mc: maeClies ) {
        Cliente cli = new Cliente();
        cli = cli.toCliente(mc);
        clientes.add(cli);
    }
    return clientes;
}

@GetMapping("/{id}")
public Resource<Cliente> buscarXId(@PathVariable int id){
    List<MaeClie> maeClie = maeClieService.findOne(id);

    if(maeClie == null) {
        throw new ClienteNotFoundException("id-"+id);
    }else if(maeClie.size()<1) {
        throw new ClienteNotFoundException("id-"+id);
    }

    //HATEOAS
    Cliente cliente = new Cliente().toCliente(maeClie.get(0));

    Resource<Cliente> resource = new Resource<Cliente>(cliente);
    ControllerLinkBuilder linkTo = linkTo(methodOn(this.getClass()).listAll());
    resource.add(linkTo.withRel("all-clients"));
    //
    return resource;
}
```

Servicio POST, creación de un cliente

```
@PostMapping()
public ResponseEntity<?> post(@Valid @RequestBody Cliente cliente){
    System.out.println("Guardando cliente");
    MaeClie maeClie = maeClieService.add(cliente.toMaeClie());

    URI location = ServletUriComponentsBuilder
            .fromCurrentRequest()
            .path("/{id}")
            .buildAndExpand(maeClie.getIdClie()).toUri();

    return ResponseEntity.created(location).build();
}
```

Servicio DEL, eliminar un cliente lógicamente

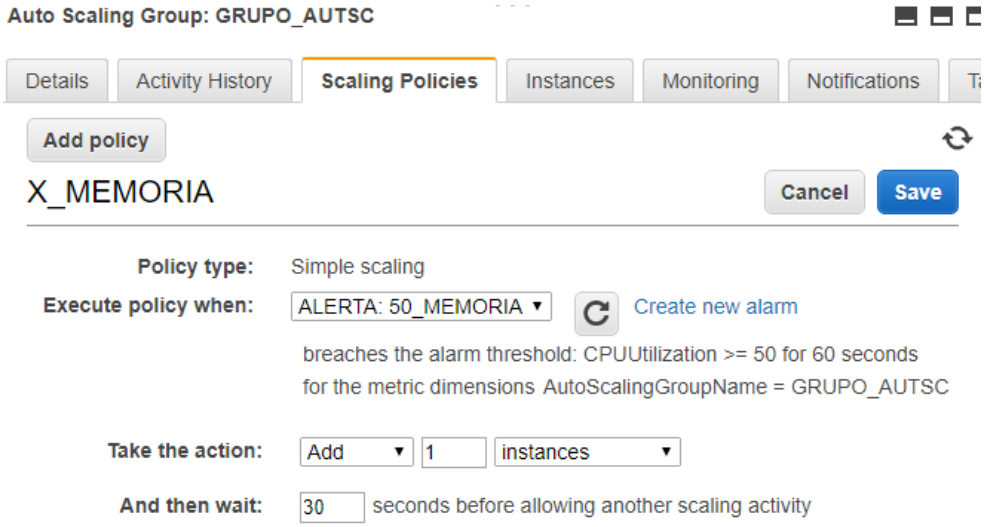
```
@DeleteMapping("/{id}")
public void deleteXId(@PathVariable int id){
    List<MaeClie> maeClie = maeClieService.findOne(id);

    if(maeClie == null) {
        throw new ClienteNotFoundException("id-"+id);
    }else if(maeClie.size()<1) {
        throw new ClienteNotFoundException("id-"+id);
    }
    maeClie.get(0).setIndActivo("I");
    maeClieService.update(maeClie.get(0));
}
```

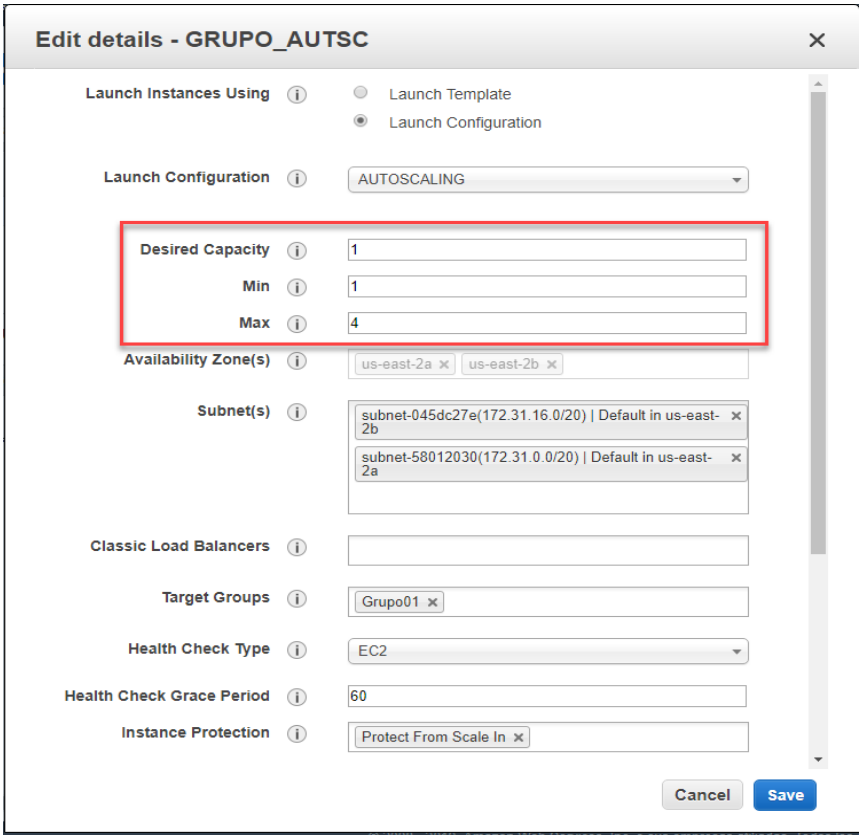
APÉNDICE D

ESCALABILIDAD DE LA ARQUITECTURA EN AWS

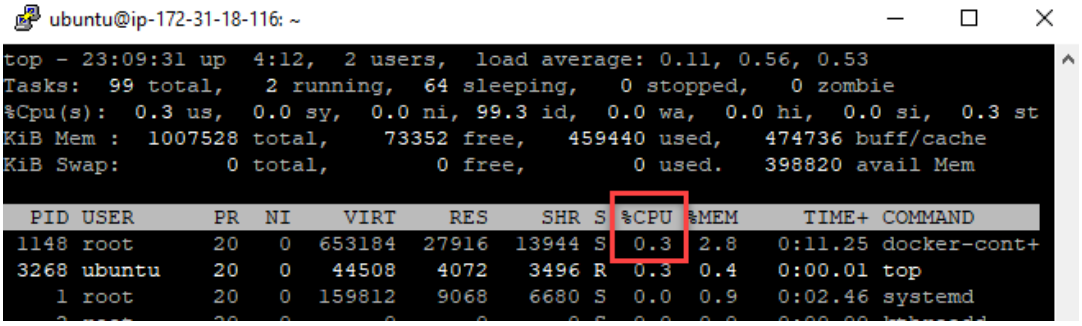
Creación de alarma por utilización del CPU.



Detalles del grupo Auto Scaling.



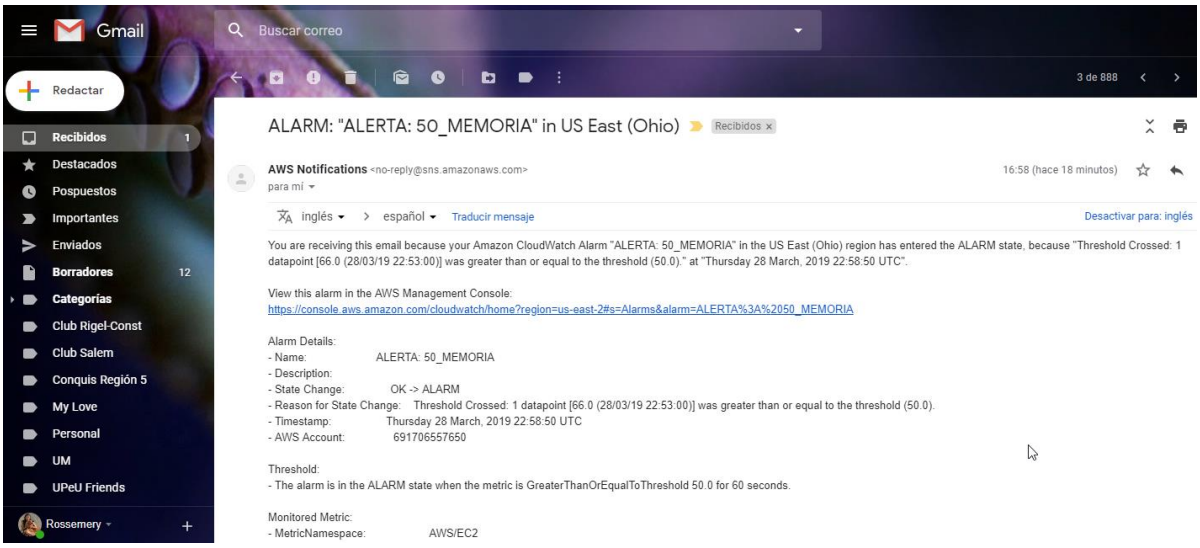
Consola del “EC2 _TEST”



Código del método “serviciopesado”

```
//Agregando Código
@GetMapping("/serviciopesado")
public String servicioPesado(){
    System.out.println("Ingreso al metodo");
    String var = "test 1";
    String var2 = "test 2";
    for(int i=0; i<100000; i++) {
        var = var+"test 1";
        var2 = var2+"test 2";
        System.out.println("Código Infinito "+i);
    }
    System.out.println("Se acabo el metodo");
    return "Se ejecuto correctamente";
}
// Fin: Agregando Código
```

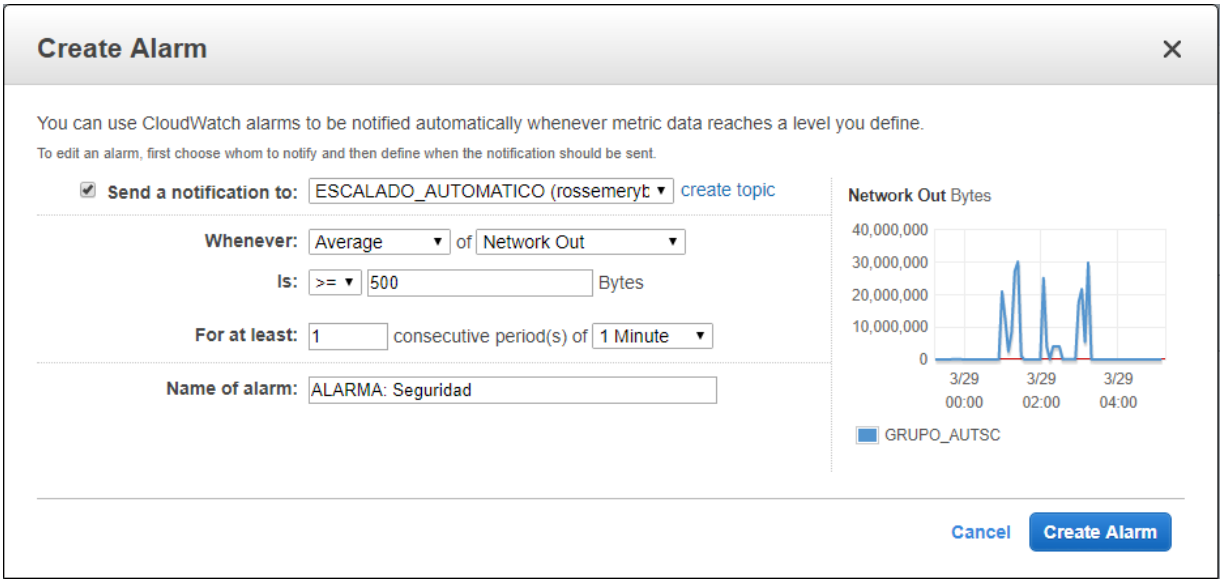
Alerta de AWS cuando se ocupa más de 50% del CPU.



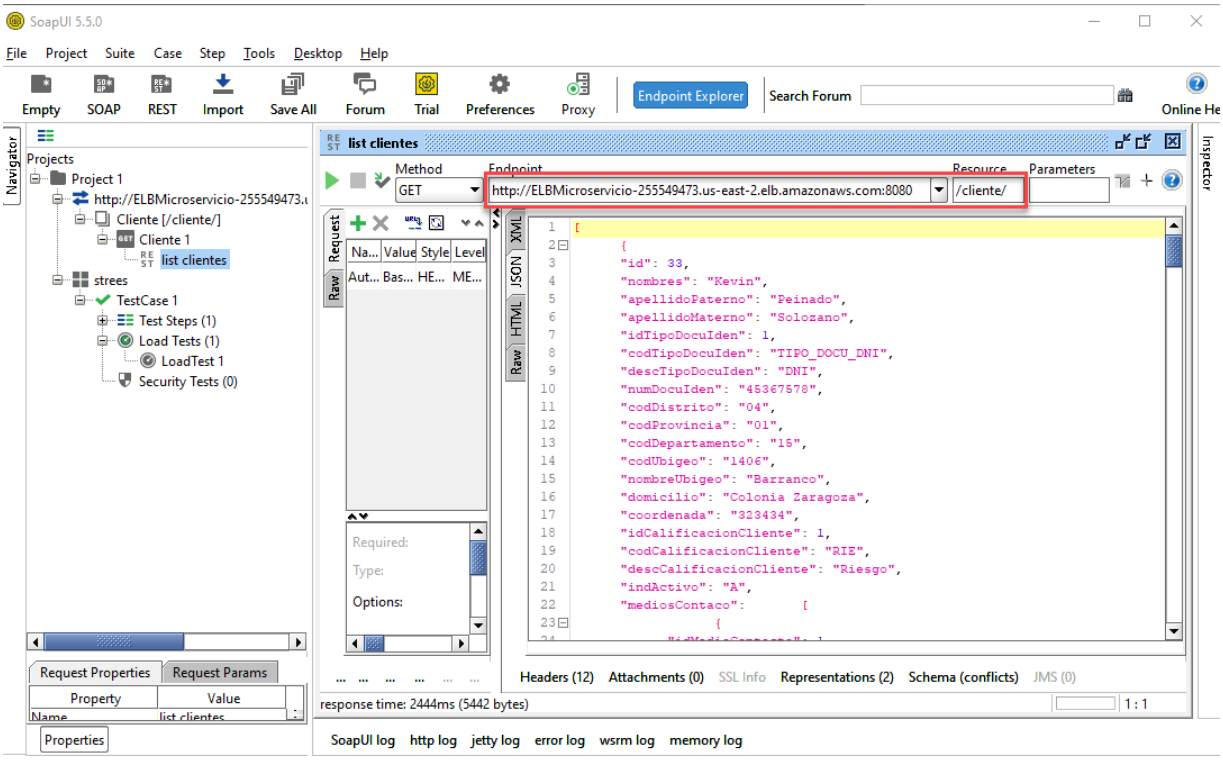
APÉNDICE E

SEGURIDAD DE LA ARQUITECTURA EN AWS

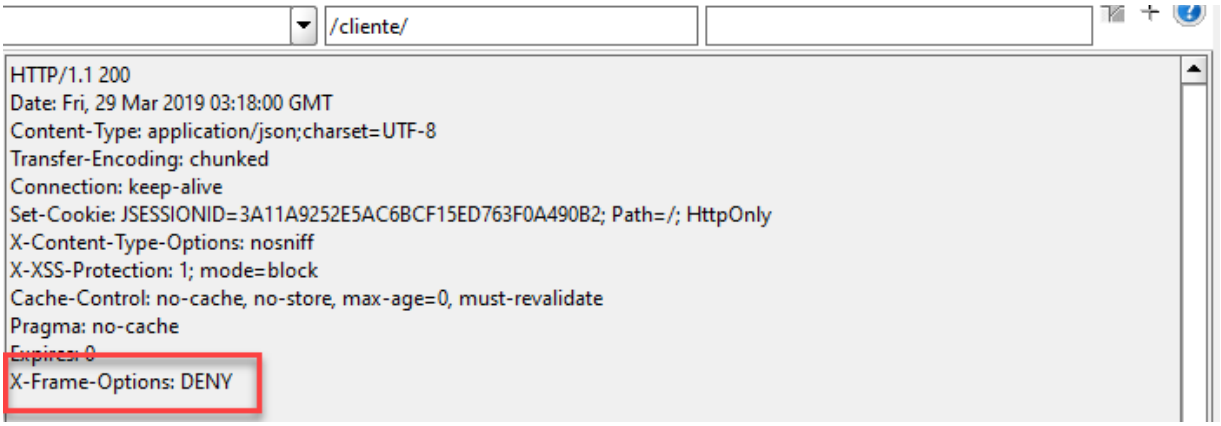
Creación de alarma para peticiones entrantes.



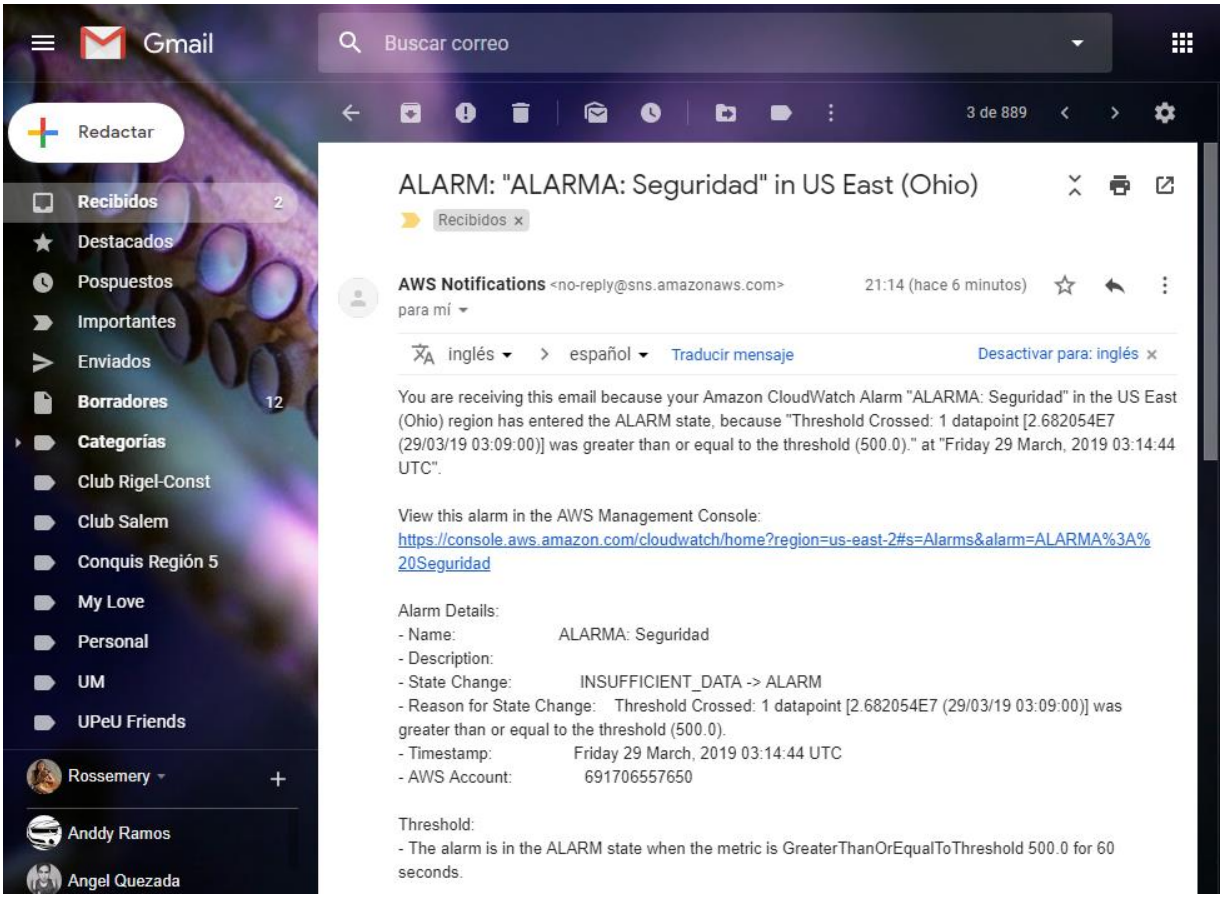
Interfaz del SoapUI, donde se llevó a cabo las pruebas.



Petición denegada



Aviso de alarma de seguridad

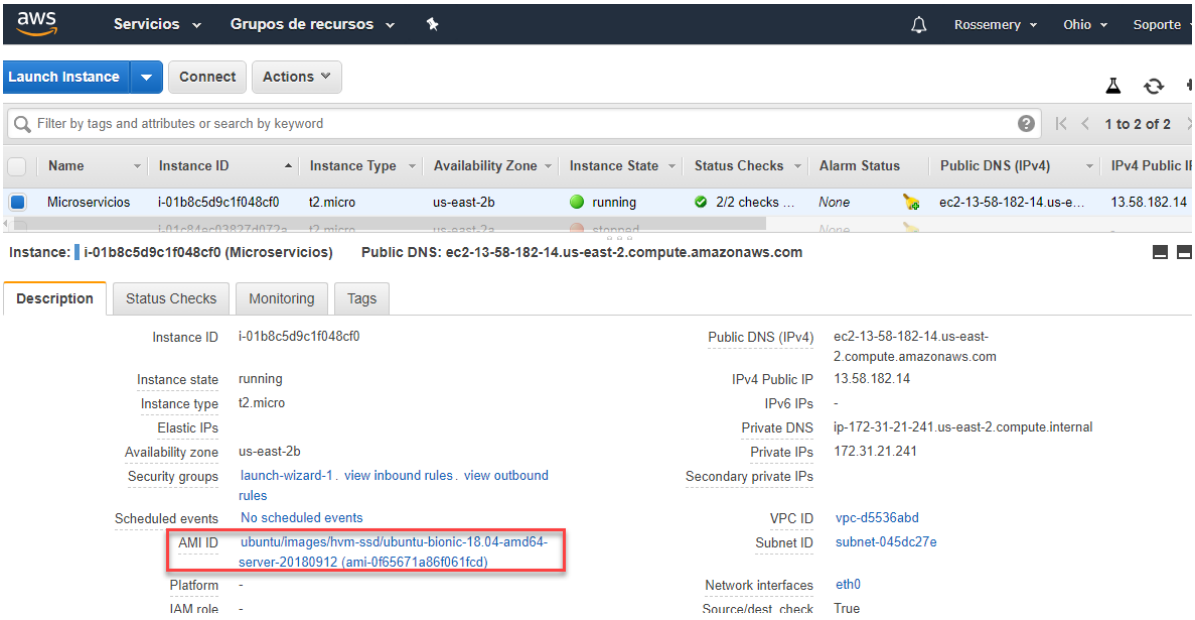


APÉNDICE F

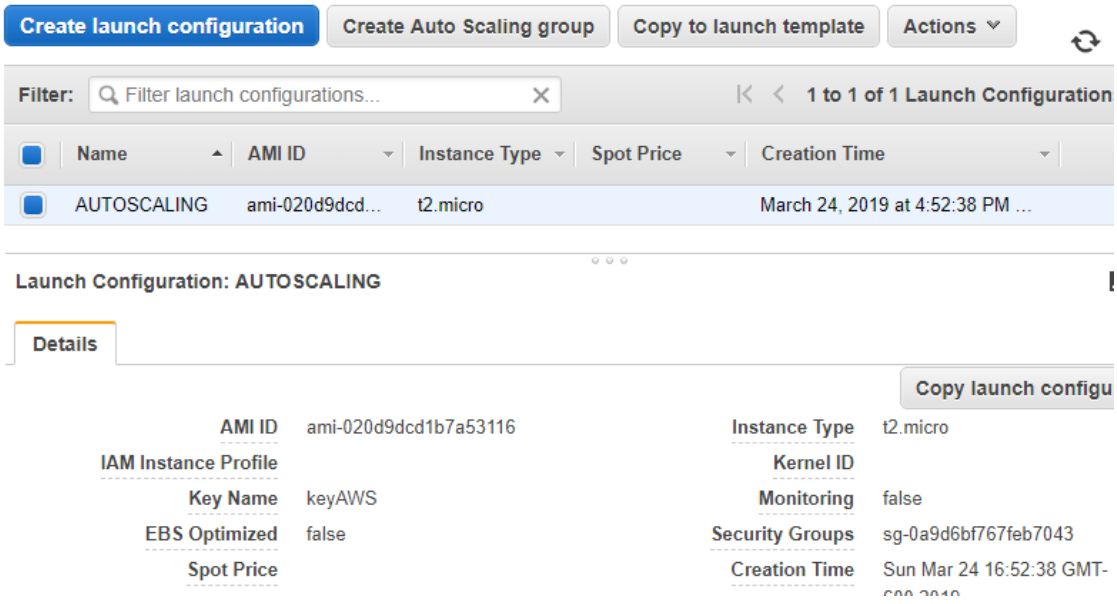
CONFIGURACIÓN

CONFIGURACIÓN

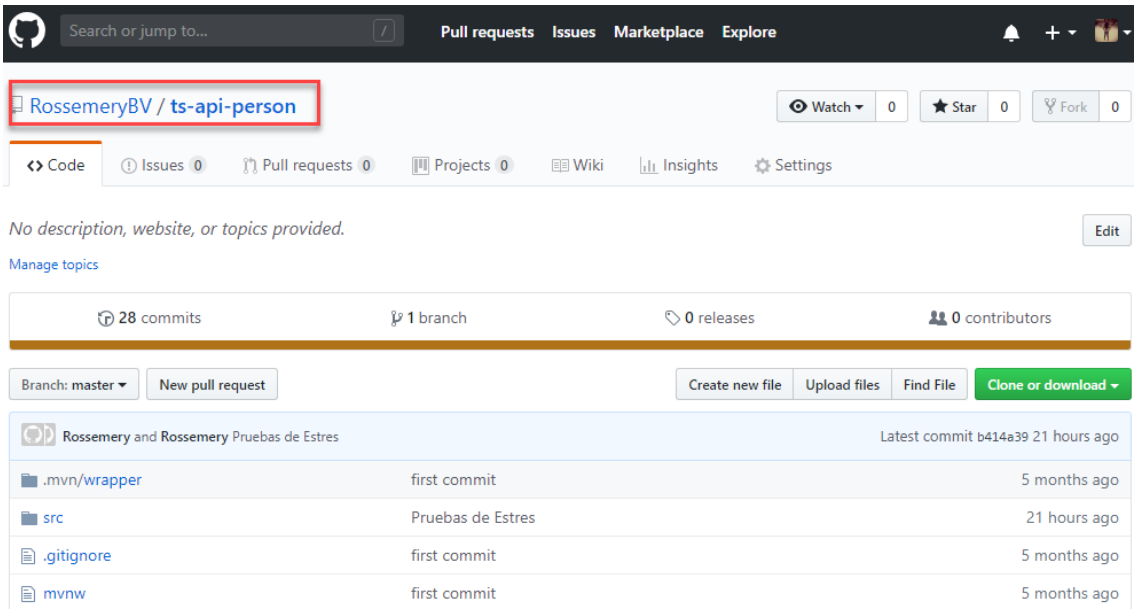
Creación de la instancia EC2 “microservicio”.



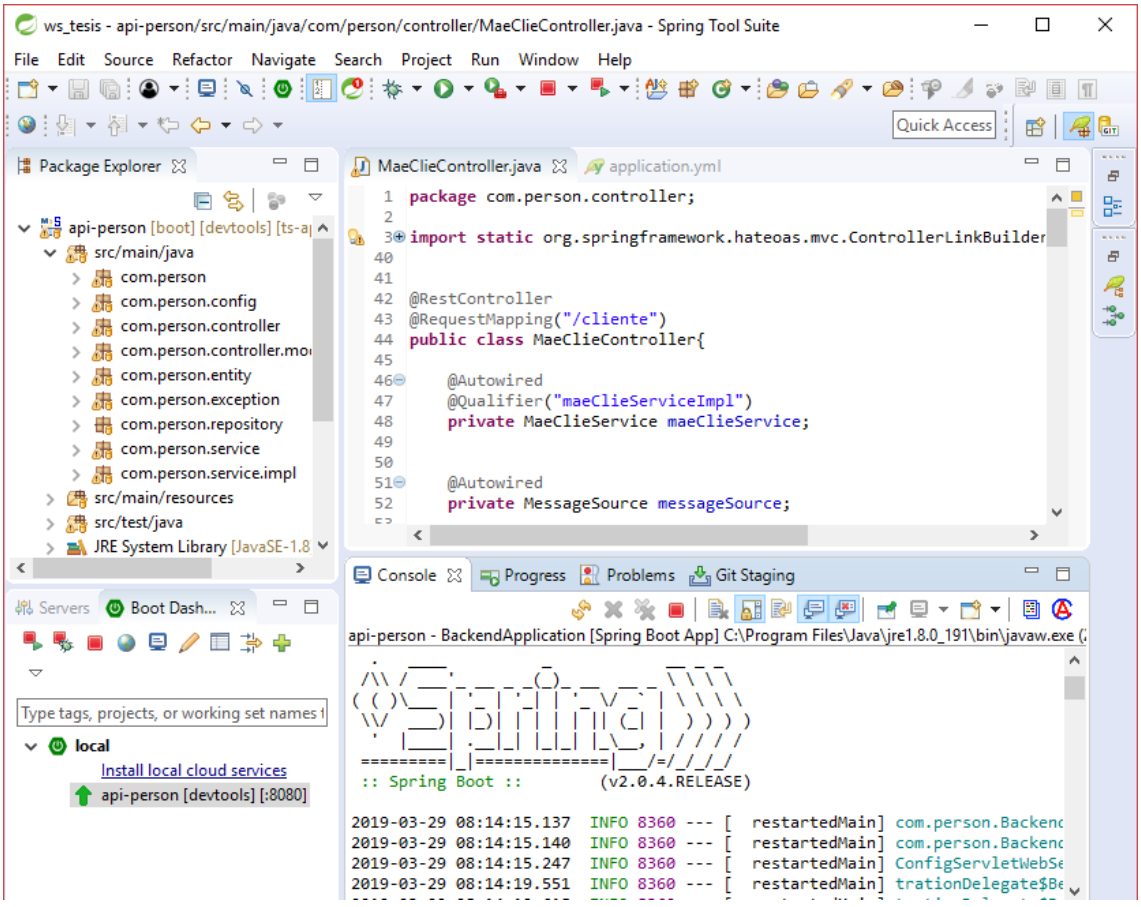
Detalle del Auto Scaling.



Repositorio de GitHub



Interfaz del Spring.



REFERENCIAS

Álvarez, P., Hernández, S., Fabra, J. y Ezpeleta, J. (2018). Cost-driven provisioning and execution of a computing-intensive service on the Amazon EC2. *The Computer Journal, 61*(9), 1407-1421. https://doi.org/10.1093/comjnl/bxy006

Agrawal, B. Wiktorski, T. y Rong, C. (2017). Adaptive real-time anomaly detection in cloud infrastructures. *Concurrency and Computation: Practice and Experience, 29*(24), 1-13. https://doi.org/10.1002/cpe.4193

Antunes, N. y Vieira, M. (2016). Designing vulnerability testing tools for web services: Approach, components, and tools. *Journal of Information Security,16*(4), 435-457. https://doi.org/10.1007/s10207-016-0334-0

Arias Orizondo, A. C. y Estrada Senti, V. E. (2013). Bases para crear un modelo de madurez para arquitecturas orientadas a servicios. *Scientific Electronic Library Online, 34*(3), 307-318. Recuperado de http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1815-59362013000300007 &lng=es&tlng=es.

Arias Preciado, C. A., Mariaca Lira, C. G. y Parodi Mendoza, J. A. (2011). *Cloud Computing, un aliado en la gestión de las TI en las empresas peruanas como parte de su estrategia* (Tesis de maestría). Universidad ESAN, Lima, Perú.

Axelrod, A. (2015). Box: Natural language processing research using Amazon Web Services. *The Prague Bulletin of Mathematical Linguistics, 104(*1), 27-38. https://doi.org/10.1515/pralin-2015-0011

Ayyapazham, R. y Velautham, K. (2017). Proficient decision making on virtual machine creation in laaS cloud environment. *The International Arab Journal of Information Technology, 14*(3), 314-323.

Bhardwaj, S., Jain, L. y Jain, S. (2010). Cloud computing: A study of infrastructure as a service (IaaS). *International Journal of Engineering and Information Technology, 2*(1), 60-63.

Braude, E. J. (2000). *Software engineering: An object-oriented perspective*. New York: Wiley.

Brogi, A. Neri, D. y Soldani, J. (2018). A microservice-based architecture for (customisable) analyses of docker images. *Softw Pract Exper, 48*(8), 1461-1474. doi: https://doi.org/10.1002/spe.2583

Burgos Coronel, A. E. (2015). *Evaluación de la arquitectura del sistema para el control de seguridad y auditoría: propuesta de calidad del software para gestión de lípidos en la Facultad de Ciencias Naturales de la Universidad de Guayaquil* (Tesis de licenciatura). Universidad de Guayaquil, Quito, Ecuador.

Caballer Fernández, M. (2014). *Gestión de infraestructuras virtuales configuradas dinámicamente* (Tesis doctoral). Universidad Politécnica de Valencia, Valencia, España.

Camacho, J. A., Chamorro, C. D., Sanabria, J. A., Caicedo, N. G. y García, J. I. (2017). Implementation of a service-oriented architecture for applications in physical rehabilitation. *Revista Facultad de Ingeniería, 26(*46), 113-121. https://doi.org/10.19053/01211129.v26.n46.2017.7323

Carvallo Vega, J. P. y Franch Gutiérrez, J. (2009, enero). *Descubriendo la arquitectura de sistemas de software híbridos: un enfoque basado en modelos i**. Conferencia presentada en Anais do WER09 - Workshop em Engenharia de requisitos, Valparaíso, Chile. Recuperado de: https://www.researchgate.net/publication/221235292_Descubriendo_la_Arquitectura_de_Sistemas_de_Software_Hibridos_Un_Enfoque_Basado_en_Modelos_i

Carillo Ordoñez, M. J. (2013). *Amazon EC2 en Guatemala: cambiando paradigmas análisis, uso e implementación* (Tesis de licenciatura). Universidad de San Carlos, Ciudad de Guatemala, Guatemala.

Castedo, L., Dávila, L., González, R., Hernando, M., López, S., Quesada, P., . . . Santos, C. (2008, julio). *Arquitectura cliente-servidor para un laboratorio remoto*. Conferencia presentada en Universidad Politécnica de Madrid, Madrid, España.

Chiriboga Mogollón, G. L. (2014). *Estudio de factibilidad para la migración de la infraestructura y servicios de los sitios web de grupo el comercio a un servicio en la nube* (Tesis de maestría). Universidad de las Américas, Quito, Ecuador.

Cliff, S. (2017). *Amazon web services outage shows vulnerability of cloud disaster recovery*. Recuperado de http://www.computerweekly.com/news/450414276/AWS-outage-shows-vulnerability-of-cloud-DR

Czarnul, P. (2013). An evaluation engine for dynamic ranking of cloud providers. *Informatica (Slovenia)*, *37*, 123-130. doi:10.31449/inf.v37i2.441

Dahan, F., El Hindi, K. y Ghoneim, A. (2017). Enhanced artificial bee colony algorithm for qos-aware web service selection problem. *Computing, 99*(5), 507-517. https://doi.org/10.1007/s00607-017-0547-8

Daza Corredor, A. P., Parra Peña, J. F. y Espinosa Rodríguez, L. M. (2015). Metodología de representación de software orientada al desarrollo ágil de aplicaciones: un enfoque arquitectural. *Redes de Ingeniería, 7*(1), 104-111. https://doi.org/10.14483/udistrital.jour.redes.2016.1.a3

De La Cruz Caicedo, A., Bolaños Bastidas, Y., Ordóñez Ante, L. y Corrales Muñoz, J. C. (2014). Semantic Annotation of SOAP Web Services based on Word Sense Disambiguation Techniques. *Ingeniería y Universidad, 18*(2), 369-392. https://dx.doi.org/10.11144/Javeriana.IYU18-2.sasw

Definición De. (2017). *Definición de términos*. Recuperado de https://definicion.de

Díaz Rodríguez, O. (2015). Metodología para diseñar bases de datos relacionales con base en el análisis de escenarios, sus políticas y las reglas del negocio. *3C TIC, 4*(3), 197-209. https://doi.org/10.17993/3ctic.2015.43.197-209

Digman, L. (2018). *Top cloud providers 2018: How AWS, Microsoft, Google, IBM, Oracle, Alibaba stack up*. ZDNet. Recuperado de https://www.zdnet .com/article/cloud-providers-ranking-2018-how-aws-microsoft-google-cloud-platform-ibm-cloud-oracle-alibaba-stack/

Duarte Vega, G. E. (2016). *Arquitectura propuesta para un servicio web completo: metodología de desarrollo e implementación* (Tesis de licenciatura). Universidad Distrital Francisco José de Caldas, Caldas, Colombia.

Engard, B. (2017). *The sides of software development: From Front End vs Back End to full stack. Software Guild*. Recuperado de https://www.thesoftwareguild.com/blog/front-end-vs-back-end/

Fusaro, V. A., Patil, P., Gafni, E., Wall D. P. y Tonellato, P. J. (2011). Biomedical cloud computing with Amazon Web Services. *PLOS: Computational Biology, 7*(8), 1-6. https://doi.org/10.1371/journal.pcbi.1002147

Gadge, S. y Kotwani, V. (2018). *Microservice architecture: API gateway considerations.* Recuperado de https://www.globallogic.com/pl/gl_news/microservice-architecture-api-gateway-considerations/

Galicia García, C., Ortega Ginés, H. B. y Curioca Varela, Y. (2015). Desarrollo de un Back-End adaptativo para portales web. *Revista de Sistemas Computacionales y TIC's, 1(*1), 52-60. Recuperado de https://ecorfan.org/spain/researchjournals/Sistemas_Computacionales_y_TICs/vol1num1/Sistemas%20Computacionales%20%20y%20TIC-52-60.pdf

Garfias Suárez, A. E. (2012). Front-end y back-end. *Revista Software Gurú (SG), 36*, 56-57. Recuperado de https://es.scribd.com/document/257838455/Front-End-y-Back-End

Gómez Fermín, L. V. y Moreno Poggio, T. R. (2014). Propuesta de modelo en cinco capas para aplicaciones web. *Revista Multidisciplinaria del Consejo de Investigación de la Universidad de Oriente, 26*(2), 168-173.

González García, C. (2017). *MIDGAR: Interoperabilidad de objetos en el marco de internet de las cosas mediante el uso de ingeniería dirigida por modelos* (Tesis doctoral). Universidad de Oviedo, Oviedo, España.

Guabtni, A., Ranjan, R. y Rabhi, F. A. (2013). A workload-driven approach to database query processing in the cloud. *Springer, 63*(3), 722-736. doi:10.1007/s11227-011-0717-y

Gunnulfsen, M. K. (2013). *Scalable and efficient web application architectures: Thin-clients and SQL vs. thick-clients and NoSQL* (Tesis de maestría). Universidad de Oslo, Oslo, Noruega.

Henríquez, C., Del Vecchio, J. F. y Peternina, F. J. (2015). La computación en la nube: un modelo para el desarrollo de las empresas. *Pospect, 13*(2), 81-87. http://dx.doi.org/10.15665/rp.v13i2.490

Huitrón Toledo, A., Zapata Nogales, E. D. y Zaragoza López, J. J. (2017). *Solución de infraestructura aplicativa en la nube para pequeñas y medianas empresas* (Tesis de licenciatura). Instituto Politécnico Nacional, Ciudad de México, México. Recuperado de http://tesis.ipn.mx/handle/123456789/21056

Jia bin, L., Shi Wei, H. y Wei Chuan, Y. (2018). The study of pallet pooling information platform based on cloud computing. *Scientific Programming, 5*, 1-5. https://doi.org/10.1155/2018/5106392

Jiménez Domingo, E. (2013). *Modelo de interoperabilidad para plataformas de cloud computing basado en tecnologías de conocimiento* (Tesis doctoral). Universidad Carlos III de Madrid, Madrid, España.

Jiménez Torres, V. H., Tello Borja, W. y Ríos Patiño, J. I. (2014). Lenguajes de patrones de arquitectura de software: una aproximación al estado del arte. *Scientia et Technica, 19*(4), 371-376. http://dx.doi.org/10.22517/23447214.8595

Khalid, A. y Shahbaz, M. (2017). Service architecture models for fog computing: A remedy for latency issues in data access from clouds. *KSII: Transactions on Internet and Information Systems, 11(5*), 2310-2345. https://doi.org/10.3837/tiis.2017.05.001

Kulkarni, G., Sutar, R. y Gambhir, J. (2012). Cloud computing infrastructure as service Amazon EC2. *International Journal of Engineering Research and Applications (IJERA), 2*(1), 117-125. Recuperado de https://www.researchgate.net/publication/234166267_CLOUD_COMPUTING-INFRASTRUCTURE_AS

López Hinojosa, J. D. (2017). *Arquitectura de software basada en microservicios para desarrollo de aplicaciones web de la Asamblea Nacional* (Tesis de maestría). Universidad Técnica del Norte, Ibarra, Ecuador.

Loten, A. (2018). Microsoft narrows gap con amazon en la nube. *The Wall Street Journal.* Recuperado de: https://blogs.wsj.com/cio/2018/08/02/microsoft-narrows-gap-with-amazon-in-cloud/

Luna Encalada, W. y Castillo Sequera, J. L. (2017). Model to implement virtual computing labs via cloud computing services. *Symmetry, 9*(7), 2-15. https://doi.org/10.3390/sym9070117

Matos Arias, Y. y Silega Martínez, N. (2013). Estilo arquitectónico para el sistema integrado de gestión Cedrux. *GECONTEC: Revista Internacional de Gestión del Conocimiento y la Tecnología, 1*(1), 24-36. Recuperado de https://www.upo.es/ revistas/index.php/gecontec/article/view/450/pdf_6

Meaurio, V. S. y Schmieder (2013). La arquitectura de software en el proceso de desarrollo: integrando MDA al ciclo de vida en espiral. *Revista Latinoamericana de Ingeniería de Software, 1*(4), 142-146. https://doi.org/10.18294/relais.2013.142-146

Meliá, S. (2007). *WebSA: un método de desarrollo dirigido por modelos de arquitectura para aplicaciones web* (Tesis doctoral). Universidad de Alicante, San Vicente del Raspeig, España.

Monroy, M. E., Arciniegas, J. L. y Rodríguez, J. C. (2016). Recuperación de arquitecturas de software: un mapeo sistemático de la literatura. *Información Tecnológica, 27*(5), 201-220. http://dx.doi.org/10.4067/S0718-07642016000500022

Mora Rodríguez, A. (2016). *Servicios en la nube con microsoft azure: desarrollo y operación de una aplicación android con DevOps.* Universidad Politécnica de Madrid, Madrid, España.

Olcina Valero, A. (2017). *Desarrollo de aplicaciones web con el API de Google Cloud* (Tesis de licenciatura). Universidad Politécnica de Valencia, Valencia, España.

Real Academia Española. (2017). *Diccionario de la lengua española*. Madrid: Autor.

RightScale (2018). *State of the Cloud report: Data to navigate your multi-cloud strategy.* Estados Unidos: RightScale. Recuperado de https://www.suse.com/media/report/rightscale_2018_state_of_the_cloud_report.pdf

Rojas Albarracín, G., Páramo Fonseca, J. y Hernández Merchán, C. (2017). Plataforma computacional sobre Amazon Web Services (AWS) de renderizado distribuido. *Revista Científica, 3*(30), 337-356. https://doi.org/10.14483/23448350.12362

Salazar Cárdenas, J. E. (2014). *Análisis comparativo de dos bases de datos SQL y dos bases de datos no SQL* (Tesis de licenciatura). Universidad Tecnológica de Pereira, Pereira, Colombia.

Salazar Hernández, W. E. (2017). *Implementación de arquitectura de microservicios utilizando virtualización por sistema operativo* (Tesis de licenciatura). Universidad de San Carlos de Guatemala, Ciudad de Guatemala, Guatemala.

Sangwan, R. S. (2014). *Software and systems architecture in action*. London: Auerbach.

Stankevicius, K. (2013). Rest architektürinio stiliaus palyginimas su soap, gyvendinant siuolaikeves interneto paslaugas, *Science Future of Lithuania, 5*(2), 92-95. https:// doi.org/10.3846/mla.2013.16

Suárez, J. M. y Gutiérrez, L. E. (2016). Tipificación de dominios de requerimientos para la aplicación de patrones arquitectónicos. *Información Tecnológica, 27*(4), 193-202. https://dx.doi.org/10.4067/S0718-07642016000400021

Suhartanto, H. Pasaribu, A. P., Siddiq, M. F., Fadhila, M. I. Hilman, M. H. y Yanuar. A. (2017). A preliminary study on shifting from virtual machine to docker container for insilico drug discovery in the cloud. *International Journal of Technology, 8*(4), 611-621. https://doi.org/10.14716/ijtech.v8i4.9478

Suryotrisongko, H., Puji Jayanto, D. y Tjahyanto, A. (2017). Design and development of backend application for public complaint systems using microservice spring boot. *Procedia Computer Science, 124,* 736-743. https://doi.org/10.1016/ j.procs.2017.12.212

Tahuiton Mora, J. (2011). *Arquitectura de software para aplicaciones Web* (Tesis de maestría), Instituto Politécnico Nacional, Ciudad de México, México.

The Institute of Electrical and Electronics Engineers. (2000). *1471-2000 - IEEE recommended practice for architectural description for software-intensive systems*. IEEE Unapproved Draft Std P42010/D6. doi:10.1109/IEEESTD.2000.91944

Tian, G., Wang, J., He, K., Sun, C. y Tian, Y. (2017). Integrating implicit feedbacks for time-aware web service recommendations, *Springer Link, 19*(1), 75-89. https:// doi.org/10.1007/s10796-015-9590-1

Tihomirovs, J. y Grabis, J. (2016). Comparison of SOAP and REST based web services using software evaluation metrics. *Information Technology and Management Science, 19*(1), 92-97. https://doi.org/10.1515/itms-2016-0017

Torres, M. y Alférez, G. H. (2014). *Software Architecture Evolution in the Open World through Genetic Algorithms*. Conferencia presentada en Nevada, Estados Unidos. Recuperado de http://worldcomp-proceedings.com/proc/p2014/SER3160.pdf

Unda, P. Coral, H. y Marcillo, D. (2013). *Sistema gestor fiducia fondos JEE mediante servicios de cloud computing, estudio de factibilidad e implementación* (Tesis de licenciatura). Universidad de las Fuerzas Armadas – ESPE, Sangolquí, Ecuador.

Valdivia Caballero, J. J. (2016). Modelo de procesos para el desarrollo del Front-End de aplicaciones web. *Interfases, 9*, 187-208. http://dx.doi.org/10.26439/interfases2016.n009.1245

Westcon-Comstor. (2018). *Distribuidor de soluciones en la nube Westcon-Comstor.* https://www.westconcomstor.com/au/en/vendors/wc-vendors/amazon-web-services.html

Zhang, N., Wang, J., Ma, Y., He, K., Li, Z. y Liu, X. (2018). Web service discovery based on goal-oriented query expansion. *The journal of Systems and Software, 142*, 73-91. https://doi.org/10.1016/j.jss.2018.04.046