UNIVERSIDAD CATÓLICA DE CUENCA
SEDE AZOGUES

UNIDAD ACADÉMICA DE TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

INGENIERÍA EN SISTEMAS

TEMA:

PROGRAMACIÓN EN LA NUBE COMO ALTERNATIVA A LA PROGRAMACIÓN DE APLICACIONES WEB TRADICIONALES

TRABAJO DE TITULACIÓN PRESENTADO EN CONFORMIDAD CON LOS REQUISITOS ESTABLECIDOS PARA LA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN SISTEMAS

AUTOR:

Cristian Paúl Guillén Parra

DIRECTOR:

Ing. Andrés Sebastián Quevedo Sacoto. MSc

AZOGUES – ECUADOR

2020

Aprobación del tutor

En calidad de tutor del trabajo de grado, presentado por el Sr. Cristian Paúl Guillén Parra para optar por el título de INGENIERO EN SISTEMAS, doy fe que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a presentación pública y evaluación por parte del jurado examinador que se designe.

En la ciudad de Azogues, a los 28 días del mes de febrero de 2020.

_____

ING. ANDRÉS SEBASTIÁN QUEVEDO SACOTO MSC.

C.I. 0301826434

Cesión de derechos de Autor

Yo, Cristian Paúl Guillén Parra, con documento de identidad 0302321880 manifiesto mi voluntad y cedo a la Universidad Católica de Cuenca la titularidad sobre los derechos patrimoniales en virtud de que soy el autor del trabajo de titulación: **"PROGRAMACIÓN EN LA NUBE COMO ALTERNATIVA A LA PROGRAMACIÓN DE APLICACIONES WEB TRADICIONALES"**, mismo que ha sido desarrollado para optar por el título de Ingeniero de Sistemas, en la Universidad Católica de Cuenca, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en mi condición de autor reservo los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Católica de Cuenca.

Azogues, marzo 2020

_____

Cristian Paúl Guillén Parra

C.I. 0302321880

CERTIFICADO DE AUTORÍA

El presente trabajo investigativo de proyecto profesional de grado previo a la obtención del título de Ingeniero en Sistemas, cuyo tema es *"Programación en la nube como alternativa a la programación de aplicaciones web tradicionales"*, corresponde al trabajo de investigación del autor, además certifico que he cumplido con todas las observaciones realizadas por el tribunal evaluador, por lo que las ideas, opiniones vertidas en el presente, son de exclusiva responsabilidad del autor.

Estudiante:

_____

Cristian Paúl Guillén Parra

C.I. 0302321880

Dedicatoria

Este trabajo está dedicado para todas las personas que aportaron con su granito de arena de una u otra forma a la realización de este. Su mano amiga en tiempos difíciles me ayudaron a salir de cada dificultad que se presentaba durante este largo trayecto profesional.

A mis padres Paúl y Fátima, que a pesar de la distancia siempre estuvieron junto a mi durante toda mi carrera estudiantil, aún más durante mi formación universitaria, por todo su apoyo, consejos, esto es para ustedes.

A Estuardo, Lolita, Piedad y Arsenio, quienes desempeñaron un rol como padres en mí, me brindaron sus consejos y amor cada vez que lo necesitaba, gracias por acompañarme en todo momento.

Para mi esposa y mi hijo, quienes con su presencia iluminan cada segundo de mi existencia y me dan las fuerzas de seguir adelante en cada momento difícil de la vida, cada segundo con ustedes es una eternidad de enseñanzas.

Agradecimiento

El amor recibido, dedicación y paciencia con la que mis padres se preocupaban por mí, por los proyectos cumplidos y las metas futuras, por todos sus consejos que poco a poco cimentaron en mi persona los valores para ser un buen profesional y más aún, un excelente ser humano, por todo esto, muchas gracias por haber confiado en mí.

A toda mi familia, que cada uno aportó con su granito de arena dentro de mi formación personal y profesional, gracias por toda su paciencia y sus sacrificios, a pesar de que el logro no era parte de su obligación.

A mi esposa Celena y a mi hijo Paquito, ustedes constituyen un pilar fundamental en mi vida sobre el cuál construiremos una familia llena de amor y prosperidad, llegaron a mi vida en un momento importante y lo único que hicieron desde entonces fue mejorar mi vida cada día, por todo esto muchas gracias, los amo un millón.

A la Universidad Católica de Cuenca y Facultad de Ingeniería en Sistemas de la sede Azogues, por el apoyo que me han ofrecido a lo largo de mi carrera como estudiante de tan prestigiosa institución. Mi más grande y sincero agradecimiento al Ing. Sebastián Quevedo, tutor de mi proyecto y modelo profesional, quien con sus enseñanzas, dirección y colaboración permitió el desarrollo de este trabajo.

A José y Raúl, futuros colegas, quienes me apoyaron con su conocimiento y me acompañaron durante esta etapa importante de la carrera, dejemos nuestra huella en el mundo.

Gracias a la vida, por permitirme seguir cumpliendo las metas propuestas, y a todas las personas que creyeron en mí en la realización de este trabajo, la vida les devuelva el doble.

Resumen

Este documento analiza cómo es posible la implementación de una nueva infraestructura de desarrollo web, la programación en la nube, mediante el desarrollo de un caso de estudio, el cual se encuentra optimizado para funcionar tanto en un entorno de servidor como en un entorno de desarrollo en la nube, para responder a una posible alternativa de desarrollo más económica a la tradicional. Para responder el tiempo de respuesta de ambos entornos de desarrollo se propone la metodología desarrollada por Papadopolous, optimizada para poder encapsular el entorno de pruebas y comparar el rendimiento en ambas tecnologías. En el capítulo 1 se describe la problemática a resolver durante el presente trabajo, así como bases teóricas que sustentan una posible solución, misma que se resume como objetivo principal del documento. En el capítulo 2 se sienta las bases teóricas que describen el contexto actual sobre el desarrollo web, así como la metodología a utilizar para proponer una solución al problema. En el capítulo 3 se aplica la metodología descrita en capitulo previo, dentro del cual se realiza el desarrollo de la aplicación. Luego, en el capítulo 4 y con toda la metodología aplicada, se analiza los resultados obtenidos durante la experimentación realizada en el capítulo 3. Finalmente, en el capítulo 5 se desglosan tanto las conclusiones como recomendaciones relacionadas al documento y el resultado obtenido.

PALABRAS CLAVE: PROGRAMACIÓN EN LA NUBE, AMBIENTES MONOLÍTICOS, DESARROLLO WEB, MICROSERVICIOS.

Abstract

This research work analyzes how the implementation of a new web development infrastructure, it means cloud programming, is possible through the development of a case study, which is optimized to work both in a server environment and in a development environment in the cloud, to respond to a possible economic alternative versus the traditional one. To response the time of both development environments, it is proposed the methodology conducted by Papadopolous, it was optimized to encapsulate the test environment and compare the performance in both technologies. Chapter 1 describes the problem to be solved during this work, as well as theoretical bases that support a possible solution, which is summarized as the main objective of the document. Chapter 2 lays the foundations of the theory that describe the current context on web development, as well as the methodology to be used to propose a solution to the problem. Chapter 3 applies the methodology described in the previous chapter, with this, it was carried out the development of the application. Then, in chapter 4 and with all the methodology applied, the results obtained during the experimentation in chapter 2 are analyzed. Finally, in chapter 5, it is presented the conclusions and recommendations and result obtained.

KEYWORDS: CLOUD PROGRAMMING, MONOLITHIC ENVIRONMENTS, WEB DEVELOPMENT, MICRO SERVICES.

Tabla de contenidos

Lista de tablas

Lista de figuras

Capítulo 1

Introducción

Desde el inicio del desarrollo de aplicaciones web, siempre su finalidad es trabajar como un intermediario entre la información y el cliente. El trabajo que los desarrolladores deben realizar es tremendo para lograr una fiabilidad en funcionamiento, así como ser lo más sencilla e intuitiva posible para el usuario, mientras menos se le tenga que enseñar al cliente, mejor diseñada y amigable será la aplicación. Existe una amplia gama de lenguajes de programación, así como *frameworks* construidos pensando en hacer el desarrollo web más sencillo, por lo que existen diferentes opciones para cada tipo de usuario. El desarrollo no es el principal problema, por experiencia personal, el principal problema al momento de poner en producción las aplicaciones construidas es la infraestructura informática destinada a almacenar y proveer acceso a la aplicación construida.

Existe una amplia gama de equipos informáticos, cada cual, con sus ventajas y desventajas, pero todo tiene su costo, y no barato precisamente. En el país, varias empresas se encargan de la importación de estos equipos, y se encuentran precios que van desde los $600 hasta equipos que superan fácilmente los $1000 (*Servidores - Productos*, n.d.). Recordemos que en este costo no se encuentra incluido la infraestructura de red, hablamos de equipos, cables, herramientas necesarias, ni tampoco mantenimiento y menos la aplicación web a usar. Los negocios emergentes analizan esta situación y ven casi imposible la inversión para contar con herramientas TI. En el presente trabajo, se realiza una comparación entre ambas tecnologías mediante la aplicación de un caso de

estudio, resultados que soportan la implementación o descarte de un nuevo paradigma de computación.

El Cloud Computing[1] es una tecnología que se presenta como una solución que abarata la implementación de sistemas de información, siendo su principal atractivo el desarrollo serverless[2] de aplicaciones, facturando únicamente el consumo que nuestro código servidor ocupe sin preocuparnos por la infraestructura subyacente que se necesita para dar el servicio, con la presencia de varios planes de consumo que abaratan considerablemente a la implementación de servidores locales (*Precios de los planes de AWS Support | A partir de 29 USD al mes | AWS Support*, n.d.). Al hablar de un desarrollo serverless se descarta el uso de hardware que se ha venido utilizando para "servir" las aplicaciones web, enviando todo el código que sólo podía ser ejecutado por el servidor, a la nube. Las empresas proveedoras de este servicio son grandes conocidas dentro del entorno informática, así como empresarial y un ejemplo para el mundo, y esta son sus ventajas, ya que al ser lo suficientemente importantes, su infraestructura de TI está lo necesariamente configurado para ofrecer la mejor experiencia. Básicamente, todo el código backend[3] se lo encargamos a estas para que nos la almacenen y ejecuten por nosotros; pero entonces, ¿cómo podemos hacer uso de las funciones backend desde nuestro frontend[4]?, la respuesta a esto es la implementación de la tecnología API REST

---

[1] Cloud computing. - es una tecnología que permite un acceso remoto a sistemas de información, bases de datos, funciones y procesos por medio de una conexión a Internet. Reemplaza la ejecución de software de manera localmente, abaratando costes y recursos.

[2] Serverless. – Ejecución de aplicaciones en la nube, mediante la ejecución de código almacenado en sus contenedores web.

[3] Backend. – Estructura de código que solo puede ser interpretado por el software del servidor.

[4] Frontend. – Estructura de código que se ejecuta en el lado del cliente al momento de ejecutar una aplicación o página web.

para la comunicación entre el frontend con el backend. Una API REST (*Representational State Transfer Application Program Interface*) es un tipo de arquitectura y comunicación comúnmente usada en el desarrollo de servicios web y que generalmente utiliza HTTP para el intercambio de información (*US10298656B2 - Extending representational state transfer application program interface (REST API) functionality - Google Patents*, n.d.). La facilidad de implementación ha hecho que las API REST sean las indicadas para poder comunicar las funciones Lambda, ofertadas por Amazon Web Services, con nuestro código.

1.1 Antecedentes

Desde que la computación hizo acto de presencia como uno de los grandes avances tecnológicos hace cuatro décadas, el término *cloud* ya había sido pronunciado, siendo utilizado para describir el transporte de información entre varios equipos, siendo hasta el inicio del nuevo milenio cuando finalmente esta tecnología pudo ser implementada (Rittinghouse & Ransome, 2017). Durante todo ese tiempo, el desarrollo de aplicaciones utilizaba un entorno de desarrollo de computación en *cuadrilla*[5], misma que fue implementada en varias áreas de ciencia informática, siendo esta un conjunto de herramientas que permitían un entorno de ejecución de aplicaciones de manera confiable(Iosup & Epema, 2011), similar a lo que en un futuro se vería en las tecnologías de desarrollo en la nube. No fue hasta que el avance de la programación en le nube hizo su aparición que emergieron nuevas tecnologías de apoyo al desarrollo de aplicaciones, como el uso de contenedores, máquinas virtuales, etc., las cuales disminuyeron

[5] Computación en cuadrilla. – Infraestructura de red diseñada para la investigación científica y de cooperación, utilizada para intercambio de información a gran escala y sistemas distribuidos.

considerablemente el tiempo de construcción e implementación de las aplicaciones (Van Eyk et al., 2019). Durante el transcurso de esa década, varias metodologías de desarrollo web aparecieron, mismas que utilizaron los recursos de HTML para poder centralizar la información, aligerando la carga de información al cliente (Molina Ríos et al., 2017), entre estas podemos el uso arquitectura de *Hypertext Design Model* (HDM), de los primeros modelos de interacción entre persona – computadora, mismo que ofrece al cliente varios servicios servidores, correspondientes a los niveles del modelo Dexte (*Sistemas de interacción persona-computador - José Bravo Rodríguez - Google Libros*, n.d.).

Con Amazon a la cabeza, quien en el año 2002 desarrollo una nueva infraestructura de desarrollo en la cual usuarios externos pueden acceder a ciertos servicios desde cualquier parte del mundo, denominándolo como "*as a service"* (Rittinghouse & Ransome, 2017). A pesar de todo esto, no se le dio el debido seguimiento a esta tecnología, siendo hasta el año 2006, cuando el CEO de Google Eric Schmidt usó el término para referirse a un modelo grande de negocios a través del alquiler de servicios por internet (Zhang et al., 2010).

1.2 Descripción del problema

Tradicionalmente y hasta la escritura de este trabajo investigativo, el desarrollo de aplicaciones web por lo general utilizan un entorno de servidor (Paz, 2015). Ante la necesidad de pequeñas empresas por cumplir con la resolución No. NAC-DGERCGC17-00000430 del SRI sobre facturación electrónica obligatoria(SRI, 2018), misma que se ha venido implementando paulatinamente en diferentes tipos de negocios desde el año 2018, la implementación de aplicaciones web en pequeñas y medianas empresas ha visto un crecimiento dentro de este aspecto. Esto conlleva a la adquisición de soluciones informáticas urgentes para encontrarse a nivel de otros negocios, organizaciones y empresas (Hernández, n.d.), mismas que conllevan una infraestructura de desarrollo compleja por debajo, un entramado de conexiones entre servicios, únicamente entendible por el desarrollador. Para ello, la programación en la nube descarta los inconvenientes propios de un entorno físico como problemas de entorno, conexiones, para únicamente preocuparse por devolver información al cliente que lo solicita, siendo fácilmente escalable, mantenible y accesible por los usuarios. Pero no se sabe que tan buenos, adecuados y eficientes son estos recursos.

1.3 Objetivos

1.3.1 Objetivo General

Comparar la programación en la nube frente a la programación tradicional a través del desarrollo de una aplicación web construida de manera tradicional y con programación en la nube, para medir el rendimiento entre ambas opciones de desarrollo web.

1.3.2 Objetivos Específicos

1. Desarrollar el marco teórico de la problemática entornos de desarrollo tradicionales contra desarrollo en la nube, explicando los conceptos informáticos presentes en cada una de las tecnologías a comparar, para demostrar lo que se quiere lograr y construir.
2. Construir la aplicación descrita en el caso de estudio mediante la elección de un *framework* de desarrollo web en un entorno servidor y de la nube aplicando los principios de la metodología escogida, para la aplicación de la metodología de calificación y obtener datos contables.
3. Analizar los datos obtenidos de las pruebas tanto en el servidor como en la nube, mediante los registros de AWS y la base de datos, para resolver la problemática encontrada.

1.4 Estado del arte

La computación en la nube es uno de los modelos de computación más prometedores en la actualidad, por no decir que ya es el presente debido al gran uso que se le está dando en diversas áreas no solo informáticas, por ejemplo, su implementación con el *Internet of Things (IoT)*, ha visto nacer un nuevo paradigma llamado el *CloudIoT* (Botta et al., 2016). Iot y la computación en la nube son grandes aliadas, debido a que la última ayuda a suplir las carencias de la primera tales como almacenamiento de información, procesamiento de datos, comunicación, suministrando de un acceso casi ilimitado a una gran cantidad de recursos online. Un claro ejemplo es dentro del área de salud en el año 2012, el proyecto *Virtual Cloud Carer* (VCC), mediante la computación

en la nube, tuvo como objetivo principal mejorar la socialización y objetivos sociales de personas mayores con enfermedades crónica, implementándose en un dispositivo inteligente alimentado con los datos de los sensores y procesado con los servicios en la nube, permitiendo entre otros casos de estudio, la rehabilitación desde casa (Gachet et al., 2012). Otro ejemplo claro de una correcta implementación de la computación en la nube, aunque resulte transparente para nosotros, es la implementación de las herramientas de *Google Apps for Education*, siendo este conjunto conformado por Gmail, herramientas ofimáticas de Google (Docs, Sheets, Slides), Calendar, Groups y Drive. Estas herramientas fueron aplicadas en un estudio realizado por (Gil-Mediavilla et al., 2015) en la Universidad Isabel I, el uso de estas herramientas permiten una mejor organización en cuanto al desempeño académico, definición de tareas, planificación de trabajos, comunicación entre compañeros de aula.

Una vez implementada y construida las aplicaciones, sobre los consumidores y proveedores de estos servicios siempre ha surgido la duda sobre la seguridad de su información en un entorno no controlado, cabe recordar que los usuarios no conocen la máquina física que se está alquilando al contratar servicios, siendo las mismas empresas proveedoras las encargadas de contratar diversos softwares de seguridad, así como la aplicación de las correctas medidas de seguridad de acceso remoto y físico. Para ello, en un análisis realizado por (Almorsy et al., 2016) conjuntamente con el soporte de la Universidad de Tecnología de Swinburne, los problemas de seguridad más frecuentes que podemos encontrar en este modelo de desarrollo son problemas vinculados al uso de tecnologías como la virtualización y el libre acceso por un amplio conjunto de personas

bajo el mismo alquiler de servicios; mismas que pueden ser evitadas mediante la aplicación de reglas más estrictas de seguridad, elasticidad y escalabilidad dentro del grupo de desarrollo.

A pesar de todo esto, el uso de la computación en la nube es una alternativa más que viable, por ejemplo, acorde a previsiones, el uso de *FaaS* se encuentra en un constante crecimiento, por lo que alrededor del año 2023, tendrá un crecimiento del 200% (Van Eyk et al., 2019).

1.5 Contribuciones

Mediante el desarrollo del presente trabajo de titulación, se da a conocer una nueva alternativa dentro del área de soluciones informáticas, dando la oportunidad a diferentes negocios que no cuentan con sistemas informáticos por diversos motivos, ya sean estos económicos, debido al alto costo de los equipos, motivos políticos o de alta dirección. Como se presenta en el documento, la principal ventaja del desarrollo en la nube es su completa independencia de servidores y equipos profesionales o complejos de manejar para los empleadores de equipos informáticos, desestimando gastos extras a la inversión inicial requerida para el emprendimiento del negocio.

1.6 Conclusiones

Gracias al constante crecimiento que existe dentro del ámbito de la informática, nuevas tecnologías han ido teniendo su presencia en diferentes puntos a lo largo de la historia del desarrollo, últimamente las grandes compañías han ampliado su enfoque de negocios para abarcar más el campo de la informática, por ejemplo, Facebook pasó de ser una red social a crear su propio *framework* de desarrollo, React JS; Google compró varios

servicios existentes como Youtube y se transformó en un monstruo dentro del ámbito de la informática, o Amazon, que construyó una infraestructura de desarrollo nueva, dando luz a la programación en la nube y el alquiler de servicios, facilitando el desarrollo web y la puesta en funcionamiento de los proyectos construidos.

Capítulo 2

Marco teórico

Dentro del desarrollo del estado del arte, se obtuvo una visión general sobre el uso y conclusiones de la programación en la nube, por lo que, en el presente trabajo y guiados con la teoría descrita, se sabe lo que se pretende construir y como lograrlo.

En un entorno empresarial, el flujo de la información, el almacenamiento de datos, el control de procesos dentro de la empresa y la veracidad de los datos conlleva a una demanda de nuevos servicios por parte de nuestros clientes. Actualmente, por ejemplo, toda empresa bancaria mínimo ofrece servicios de banca electrónica, facilitando a los usuarios el manejo de efectivo, el control de transacciones realizadas, impresión de certificados, etc. Para poder brindar estos servicios, las empresas deben contar con la capacidad suficiente de desarrollar soluciones a necesidades latentes, contando con el personal capacitado y además los equipos suficientes para su implementación. Todos estos servicios online son el resultado del correcto funcionamiento de aplicaciones web desarrolladas, las mismas que cuentan en su infraestructura de software con varias tecnologías funcionando en perfecta sincronía para proveer los resultados deseados, todo esto alojado dentro de equipos especializados para su ejecución.

2.1 Desarrollo de aplicaciones web

La web se tiende a confundir con internet, pero debemos tener en cuenta que no es lo mismo, resumiendo internet es una gran colección de tecnologías que trabajan entre sí para permitir la comunicación entre dispositivos en diferentes puntos del globo terrestre. Por tanto, web es solo uno de los muchos servicios que puede proporcionar internet. La

web fue creada por Tim Berners-Lee en el año 1989, la misma que consistía en una forma de conectar la información de internet con un medio físico mediante el protocolo HTTP (Ramos & Ramos, 2014) *Hypertext Transfer Protocol (HTTP)* es un protocolo a nivel de aplicación que permite la comunicación de ficheros de una manera simple y sencilla, simplificando la comunicación con el servidor (Warf, 2018). De esta manera mediante la fusión de internet con HTTP permite que todo el mundo pueda conectarse para compartir información usando internet, razón por la cual las páginas web comienzan con dicho protocolo, por ejemplo https://www.google.com.ec. Otro de las tecnologías implementadas por Tim Bernes-Lee fue el uso de URL (*Uniform Resource Locator*), que es un medio de localización de las distintas páginas web dentro de internet. Conforme a la evolución de internet en cuanto a los servicios que ofrece y la evolución y nuevas técnicas de ataques cibernéticos, se ve necesario una mejora en cuanto a seguridad hacia los clientes y sus transacciones, por lo que HTTP evoluciona y ahora también es HTTPS (*Hypertext Transfer Protocol Secure*), destinada a incrementar la seguridad en el intercambio de paquetes mediante el cifrado de datos basado en la seguridad de textos SSL/TLS. Aprovechando las ventajas de internet, se utiliza el servicio web como alternativa al desarrollo nativo de aplicaciones para alojar la solución, todo esto bajo las diferentes arquitecturas de desarrollo de servidor tradicional.

Una aplicación web funciona de manera similar a la arquitectura cliente – servidor, ya que implementa tres capas similares a esta, donde la se tiene la capa de presentación al cliente, interfaz accesible mediante cualquier navegador web actual; la capa intermedia que realiza el procesamiento de los datos, siendo el encargado de estos

procesos el servidor web; y una capa inferior, que la que provee los datos para el funcionamiento de la aplicación, siendo el encargado los diferentes motores de base de datos. Al hablar de una aplicación web, esta tiene un comportamiento casi igual a una página web, con todas las tecnologías implícitas dentro de la misma, las cuales son las siguientes:

- **HTML:** lenguaje de etiquetado, encargado de mostrar en el navegador web la información de la aplicación, nos permite interactuar con las diferentes funciones programadas a desarrollar.
- **CSS:** refiere al lenguaje de estilos en cascada, es el encargado de la representación visual del sitio web. Complemento perfecto para HTML. Tenemos otros ejemplos de lenguajes a usar como alternativa a CSS como por ejemplo SASS, SCSS; pero al compilar se convierte en un archivo CSS normal.
- **JavaScript:** lenguaje de programación destinado a la ejecución de acciones dentro de un entorno web. Es un lenguaje no compilado, funciona del lado cliente donde los navegadores web son los encargados de interpretar su funcionalidad. No es obligatorio su uso, ya que existen diferentes lenguajes compilados, pero en su mayoría compilan su código a JavaScript.

Nos quedamos un momento en la descripción de JavaScript. Como se define previamente, este es el encargado de agregarle acciones a los sitios web, pero ese no es solo su trabajo, ya que adicionalmente podemos crear sitios web únicamente con este lenguaje. Para esto, en la actualizar existen varios *frameworks* JavaScript de desarrollo web, mismos que incluyen todos los elementos necesarios para la construcción de sitios,

adicionalmente estos cuentan con su propia sintaxis de desarrollo, agilizando increíblemente el desarrollo web. Entre los *frameworks* más conocidos en la actualidad tenemos a ReactJS, Angular 2 y VueJS, cada uno con sus ventajas y desventajas, aplicables a diferentes requerimientos de desarrollo, resumidas en el siguiente cuadro comparativa.

Tabla 1. *Frameworks de desarrollo web*

	Angular 2	***React JS***	***Vue JS***
Desarrolladores	Google	Facebook	Comunidad
Sencillez de aprendizaje	Fácil con *Typescript*[6]	Medio	Si
Documentación	Si	Si	Si
Enlace de datos	Bidireccional	Uni-direccional	Bidireccional
Modelo de desarrollo	MVC	DOM virtual	DOM virtual
Código reusable	Si	Solo CSS	CSS y HTML
Casos de uso	Producción, aplicaciones empresariales con Material Design	Producción, aplicaciones personalizadas	Startups
Empresas	Netflix, Google, Paypal	Facebook, Yahoo, Atlassian	Alibaba, GitLab, Adobe

[6] Typescript. - Lenguaje de desarrollo libre y de código abierto, superconjunto de Javascript, desarrollado por Microsoft orientado al desarrollo web.

Acorde a la encuesta anual realizada por *StackOverflow* (StackOverflow, 2019), en el año 2019 dentro de la categoría de *frameworks* web, en los resultados de las encuestas aplicadas a los desarrolladores profesionales, dentro de los tres *frameworks* a comparar, se obtuvo a Angular como el más utilizado a nivel profesional con un 32,4%, seguido muy de cerca por React.js con un 32,3%, con un gran crecimiento respecto al del año 2018 (*Stack Overflow Developer Survey 2018*, n.d.) Esto resume la afinidad de los desarrolladores por el uso de Angular, especialmente útil por la reutilización completa de componentes, agilizando el desarrollo de una manera increíble, razón por la cuál será el *framework* elegido para el desarrollo del trabajo de titulación.

2.2 Angular 2

Se conoce como Angular 2+, a todas las versiones de Angular que prosiguen a la primera versión de Angular, conocido como Angular JS, actualmente se encuentra en la versión estable v8.2.8(Google, n.d.). Definiendo, Angular 2 es un marco de trabajo Javacript, desarrollado por la gigante Google, destinado a la construcción de aplicaciones web y móviles. Su uso apunta a un desarrollo basado en componentes, permitiendo la reutilización de los mismos varios lugares del sitio web, permitiendo que la aplicación se comporte como una *Single Page Application (SAP),* logrando un eficiencia y velocidad gigante, así como una escalabilidad en el desarrollo de una manera super sencilla (Google, n.d.). Para comenzar, no se necesita más conocimientos intermedios en cuanto al uso de las herramientas web comunes, como HTML, CSS y JavaScript.

Para el desarrollo de la aplicación final del trabajo, se escoge a Angular gracias a la compatibilidad con varias de las tecnologías existentes, así como las siguientes características que se complementan perfectamente con lo que se quiere desarrollar.

- Uso universal, con compatibilidad con las tecnologías de servidor más comunes como PHP, Node.js y .NET.
- Desarrollo basado en componentes, permitiendo cargar solo los componentes que queremos mostrar, ahorrando recursos, tiempos de carga y agilizando el desarrollo.
- Propio CLI, permitiendo generar proyectos con tan solo pocas líneas de comando.
- Constantes actualizaciones por parte del equipo de desarrollo.
- Documentación clara sobre el uso del marco de trabajo.
- La gran cantidad de librerías presentes para su implementación.

2.3 Arquitectura cliente – servidor

El modelo tradicional cliente – servidor es una arquitectura de software en la que existe únicamente un computador servidor central que maneja las peticiones de uno o varios clientes hacía el servidor, otorgando una respuesta a cada una de las solicitudes. El equipo servidor recibe todas las solicitudes HTTP por parte de los clientes, procesa los datos recibidos dentro de las funciones programadas de la aplicación y envía una respuesta de vuelta hacia el cliente, todo esto en una fracción de segundo y con varios clientes al mismo tiempo mientras la aplicación esté en ejecución.

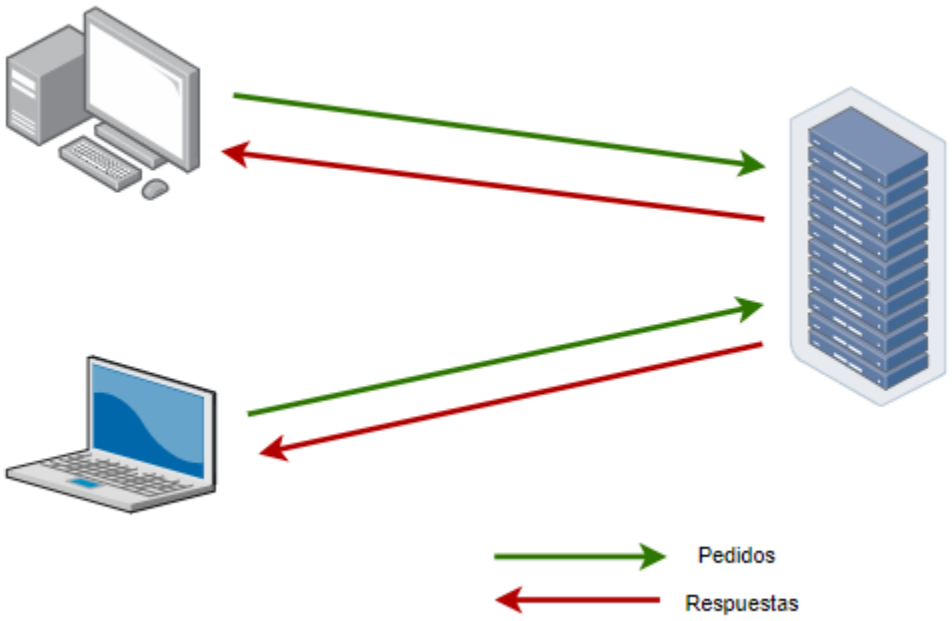

Ilustración 1. *Modelo cliente – servidor*

Fuente*: Autor*

Este modelo de arquitectura es el más usado y extendido dentro de la región, ya que permite centralizar la información en un solo equipo, evitando que los clientes ejecuten los procesos de servidor dentro de los propios dispositivos.(Marini, n.d.). Esta arquitectura nos permite distribuir las funciones acordes al servicio que va a ejecutar, pudiendo detallar cada función en una plataforma adecuada para una ejecución correcta. Adicionalmente, este tipo de arquitectura presentas ciertas ventajas:

- Segmentar procesos dentro de una red de servidores, dividiendo así la carga de los procesos que puede ralentizar la eficiencia del servidor.

- Escalabilidad de la aplicación, permitiendo una escalabilidad horizontal mediante el agregado de nuevos equipos clientes; y una escalabilidad vertical, pudiendo migrar a servidores más potentes y de mayor velocidad.
- Acceso a los datos, independientemente de la ubicación de los usuarios.
- El cliente, así como el servidor pueden existir dentro de una misma máquina.

2.4 Arquitectura en la nube

El *cloud computing* se presenta como una alternativa a la programación web tradicional, descartando todo lo que ya hemos conocido y ofreciendo nuevos servicios. Este término fue acuñado para referirse al uso y comercio de varios servicios informáticos a través de internet (Martin & E, 2012). Estos servicios que nos proporcionan son conocidos como “as a service” y nos permite trabajar de forma escalable en nuestro software(De Giusti et al., n.d.). Este modelo es presentado como una evolución del modelo tradicional de programación, que permite crear aplicaciones escalables en la nube, reduciendo considerablemente la configuración y el manejo que acarrea(Lynn et al., 2017). Las empresas más conocidas dentro del área tecnológica orientan sus servicios a la nube, como por ejemplo en la actualidad tenemos los servicios de Google Cloud (*Servicios de computación en la nube | Google Cloud | Google Cloud*, n.d.), Microsoft Azure (Microsoft, 2018) , y Amazon Web Services (Services, 2019) que ofrecen sus servicios a precios reducidos, pagando únicamente por el tiempo que consumimos.

La tecnología de programación en la nube en lugar de ofrecer la virtualización de servidores, el proveedor asigna los recursos que se necesita para la construcción de las

aplicaciones permitiendo el acceso a una red de ordenadores proveedores de diferentes servicios en tiempo real, con casi ningún tipo de configuración previa requerida (Kumanov et al., 2018). Actualmente, la programación en la nube está siendo aplicada en varios campos científicos, por ejemplo, dentro del área de la biomedicina, donde su masivo almacenamiento, su facilidad de escalabilidad, distribución de datos y el acceso inmediato a todos los recursos y aplicaciones (*Cloud Computing in Bioinformatics: current solutions and challenges*, 2016) lo convierten en la opción preferida para su implementación, dejamos de lado la configuración de servidores para que las aplicaciones puedan funcionar correctamente. Para probar la potencia de procesamiento de datos que ofrece la programación en la nube, realizando el procesamiento de comparación de 20,000 proteínas, mediante funciones implementadas en AWS (Amazon Web Services), mediante las funciones Lambda de este servicio. Al final, el resultado de procesamiento de la programación en la nube demostró la potencia de ejecución de las funciones necesarias dentro del campo de estudio y aun costo bajo, ya que, solo se paga por los recursos y funciones que se usan durante el procesamiento (Lloyd et al., 2018) .

Otro de los campos donde la programación en la nube ha hecho presencia, es en el estudio y procesamiento de la "Big Data"[7] (Zhang et al., 2010) . La programación en la nube actúa como soporte al cumplimiento de los desafíos tecnológicos que debe afrontar la Big Data dentro de los campos de industria, ciencias y gobierno, conocidos como las 5V (Mcafee & Brynjolfsson, 2012) . Dentro de cada uno de los desafíos de la Big Data, la

[7] Big Data. – Gran bloque de datos, que son destinado a su análisis mediante herramientas de TI, que revelan patrones, tendencias y asociaciones, relacionadas al comportamiento humano y sus interacciones con su medio.

programación en la nube soporta los procesos, dentro de los cuales tenemos: **almacenamiento de datos**, donde el almacenamiento de los servicios en la nube es prácticamente ilimitado, solucionando problemas de almacenamiento de información en servidores tradicionales, ofreciendo alta disponibilidad de datos, de manera fiable y rápida. La **transferencia de datos** de la Big Data dentro de la nube reduce considerablemente costos, aunque aún sigue siendo un desafío transferir datos desde un servidor físico a un servidor en la nube. El **manejo de los datos** dentro de la Big Data supone siempre un desafío a sistemas manejadores de base de datos, ya que normalmente, carecen de una escalabilidad frente al rápido crecimiento de los datos, así como el almacenamiento de datos no estructurados que maneja la Big Data. La programación en la nube oferta sistemas de almacenamiento de base de datos NoSQL[8] como Dynamo DB, propiedad de AWS (AWS, 2019) que junto a MongoDB se he hecho un espacio dentro de las empresas como su motor de base de datos predeterminado. A diferencia de un motor de base de SQL, en el que los datos deben tener una relación directa y no tener cabos sueltos, NoSQL nos permite tener una estructura JSON (*JavaScript Object Notation*) en sus datos, esto quiere decir que lo que almacenamos es un arreglo de información teniendo una "key" como identificador único, dejando de lado el manejo de llaves primarias y foráneas propias de SQL. Al manejar la información como un arreglo, podemos agregar, editar y eliminar la información alterando la estructura del array, pero sin alterar la integridad de este.

[8] NoSQL. - Sistemas propios para información con esquemas flexibles. Optimizados para aplicaciones que manejan grandes volúmenes de datos.

Hoy en día, gracias a la tecnología *serverless*, se ha conseguido convertir el uso de internet y sus servicios en una necesidad básica, en la que al igual que servicios eléctricos o de agua potable, se paga solo por su uso. La computación en la nube reúne un conjunto de tecnologías ya existentes en un solo lugar, dividiéndolo en servicios. Como principales servicios existentes en la nube tenemos:

2.4.1 Infraestructura como servicio (IaaS)

Trabaja en la capa más baja del desarrollo. Básicamente consiste en "alquilar" equipos tales como servidores, equipos de infraestructura de red, bases de datos, evitando la adquisición de los equipos y de toda la infraestructura relacionada al mantenimiento y conectividad. Permite instalar y configurar el sistema operativo deseado donde se puede desplegar y ejecutar nuestras aplicaciones. El pago por realizar es solo por el espacio en disco utilizado, espacio de disco reservado para base de datos, tiempo de uso de CPU, etc., dejando al proveedor como responsable de la mayor parte de los factores de gestión de las máquinas.

Uno de los principales IaaS es Amazon LightSail, el mismo qué es usado para el desarrollo de aplicaciones web simples gracias a *stacks*[9] de desarrollo preconfiguradas como Nginx, LAMP (Linux, Apache, MySQL, PHP), MEAN (MongoDB, ExpressJS, Angular 2+, Node.js) y Node.js; sitios web mediante creadores de contenido como WordPress, Magento, Plesk y Joomla; software para empresas como almacenamiento de archivos, copias de seguridad, funciones contables y finanzas; e iniciar entornos de

[9] Stack. – Conjunto de tecnologías que se complementan entre sí y trabajan conjuntamente para el desarrollo de un servicio.

prueba y desarrollo mediante la creación de entornos *sandbox*[10] (AWS, 2018) IaaS permite una escalabilidad vertical automática y semi automática, ya que posibilita aumentar las capacidades de IaaS mediante el contrato de recursos según lo que se requiera.

2.4.2 Software como servicio (SaaS)

Trabaja en la capa más alta de la red, consiste en ofrecer a una gran multitud de clientes el acceso a un software específico a través de la red mediante una distribución de uno a muchos, sin necesidad de que el usuario requiera una instalación previa en su equipo. La disponibilidad y funcionalidad del software depende de los administradores del servicio, retroalimentándose mediante los usuarios, ya que finalmente ellos son los beneficiados del software. Como ejemplos claros tenemos los gestores de correo electrónico como Gmail, los mismos que permiten el acceso a los usuarios mediante cualquier navegador de Internet. Dentro del mismo gigante de tecnología también tenemos sus herramientas ofimáticas de *Google Docs*, mismas que cuentan con varios servicios como hojas de cálculo, editores de texto, diseño de diapositivas, etc.

2.4.3 Plataforma como servicio (PaaS)

Siguiendo la estructura de capas web, *PaaS* se encuentra una capa bajo *SaaS*, donde su funcionalidad se centra en desplegar aplicaciones en la nube, las mismas que incluyen todo lo necesario para el desarrollo, pruebas y puesta en producción de aplicaciones y servicios web. El cliente se centra en la configuración del entorno de

[10] Sandbox. – Entorno aislado dentro de un sistema operativo dedicado a realizar pruebas sin afectar al funcionamiento del equipo.

desarrollo y en el control de la aplicación, dejando al proveedor de servicios encargarse de distribuir los recursos para que la aplicación se ejecute de manera eficiente.

Como ejemplo tenemos a Google App Engine, AWS Beanstalk; mismos que permiten desarrollar, compartir y alojas nuestras aplicaciones, así como de terceros dentro de su infraestructura de red. Prometen una escalabilidad en las aplicaciones desde cero hasta llegar a un alcance global. Su atractivo es la cantidad de lenguajes de desarrollo en los que se puede programar las aplicaciones, como, por ejemplo, dentro de Google App Engine podemos desarrollar en: Java, PHP, Node.js, Python, C#, .NET, Ruby y Go.(Google Developers, 2019).

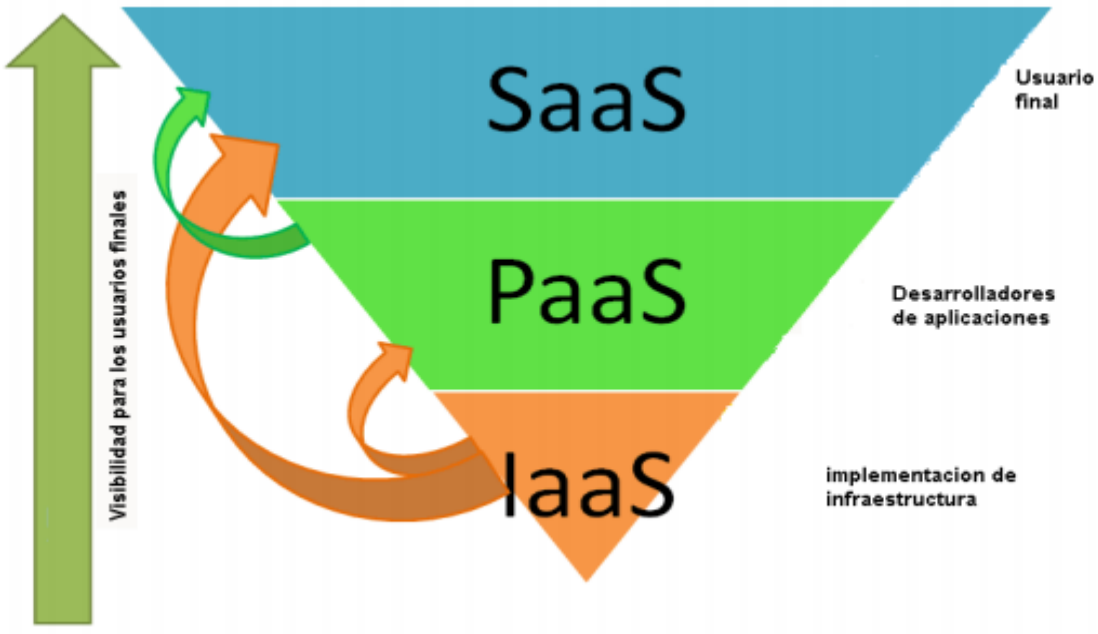

Ilustración 2. *Capas de la nube*

Fuente: *Obtenido de (Ávila Mejía, 2013)*

2.4.4 Funciones como servicio (FaaS)

Siguiendo con la estructura de capas, *FaaS* actúa como un pequeño *backend*, que permite una manera de ejecutar código servidor sin necesidad de una, todo esto mediante la ejecución de pequeñas líneas de código definidas ya por el desarrollador, estas se ejecutan como respuesta a la interacción del usuario con la interfaz, siendo activados mediante eventos dentro de las aplicaciones web. Como ventajas de implementación de este servicio tenemos que el desarrollador se concentra en el desarrollo de interfaz, lógica del programa, así como en la experiencia al usuario, descartando la tediosa tarea de configuración de servidores. Dentro de esta categoría tenemos varios ejemplos como AWS Lambda, Google Cloud Functions, Microsoft Azure Functions. AWS Lambda ofrece varios beneficios dentro de esta categoría, mismas que serán detalladas en el siguiente punto.

Si analizamos cada capa de la nube, en específico sus ejemplos, se observa que predominan tres proveedores de servicios en la nube, cada uno con más y menos catálogo de funciones. En (*AWS vs Azure vs Google vs IBM vs Oracle vs Alibaba | A detailed comparison and mapping between various cloud services*, n.d.), una página web de acceso libre, se observa el catálogo completo de servicios clasificados por proveedor, siendo AWS quién posee la mayoría de servicios y una documentación clara para su uso.

Tabla 2. *Proveedores de servicios en la nube*

Servicios	***Amazon Web Services***	***Microsoft Azure***	***Google Cloud Platform***

Servidores virtuales	Amazon EC2	Azure Virtual Machine	Compute Engine
Servidores virtuales privados	Amazon Lightsail	Azure App Service Enviroments	x
Microservicios de desarrollo	AWS Lambda	Azure Functions	Google Cloud Functions
Almacenamiento	Amazon Simple Storage Service	Azure Blob Storage	Cloud Storage
Migración de servidores	AWS Server Migration Service	Azure Websites Migration Assistant	X
Hosts virtuales dedicados	Amazon EC2 Dedicated Hosts	X	Sole Tenant Node (Beta)
Base de datos	Amazon DynamoDB	Azure CosmosDB	Cloud Datastore
Servidores virtuales	Amazon EC2	Azure Virtual Machine	Compute Engine

Para el desarrollo de la aplicación como caso de estudio, la tecnología escogida a usar es la infraestructura de *Amazon Web Services (AWS)*. La elección de esta fue apoyada gracias a la encuesta anual a profesionales de desarrollo realizada por

StackOverflow[11], para ello se revisa los resultados de la encuesta realizada en el año 2019 donde los resultados de tecnologías a utilizar fueron las siguientes:

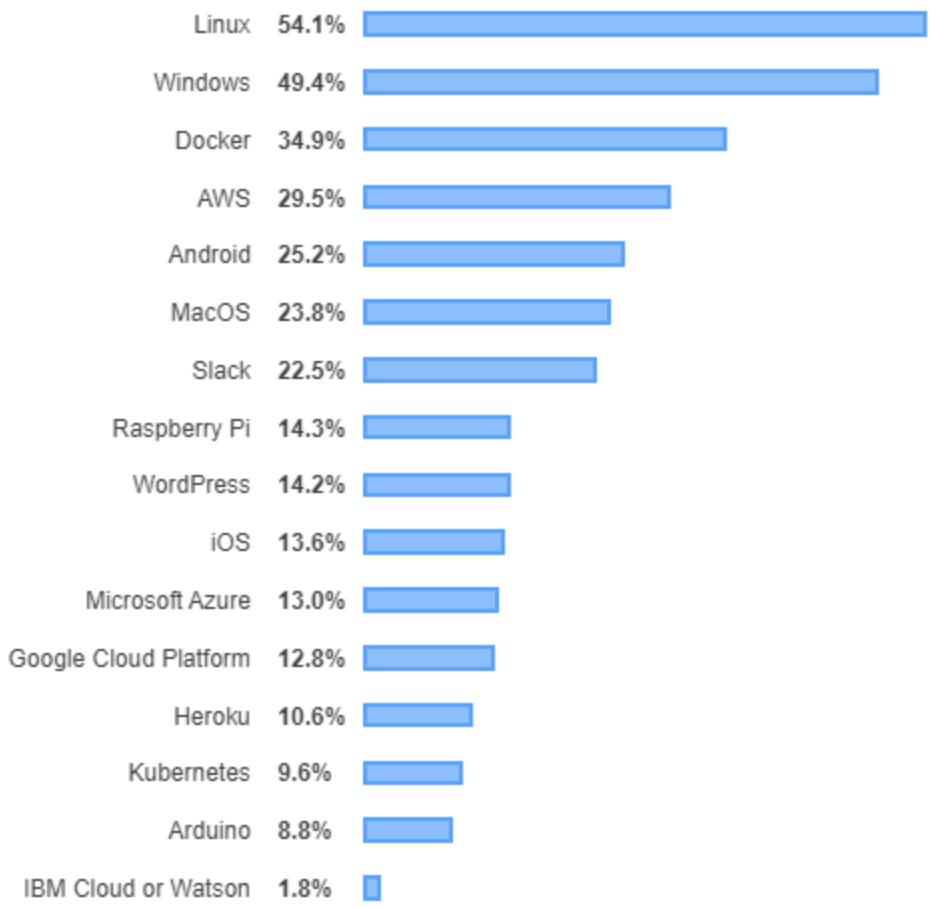

***Ilustración 3**. Tecnologías más populares del año 2019.*

***Fuente**: Encuesta 2019 de StackOverflow.*

En la ilustración 3, la encuesta fue aplicada a 67,243 profesionales mismos que dieron su opinión sobre las tecnologías que más utilizan, siendo Linux el sistema más

[11] StackOverflow. – Comunidad abierta orientada a desarrolladores profesionales y estudiantes, con la finalidad de compartir opiniones, soluciones a problemas, enseñanza privada y búsqueda de trabajo online.

utilizado para el desarrollo, pero lo que interesa al trabajo, la primera tecnología más utilizada para el desarrollo en la nube es AWS.

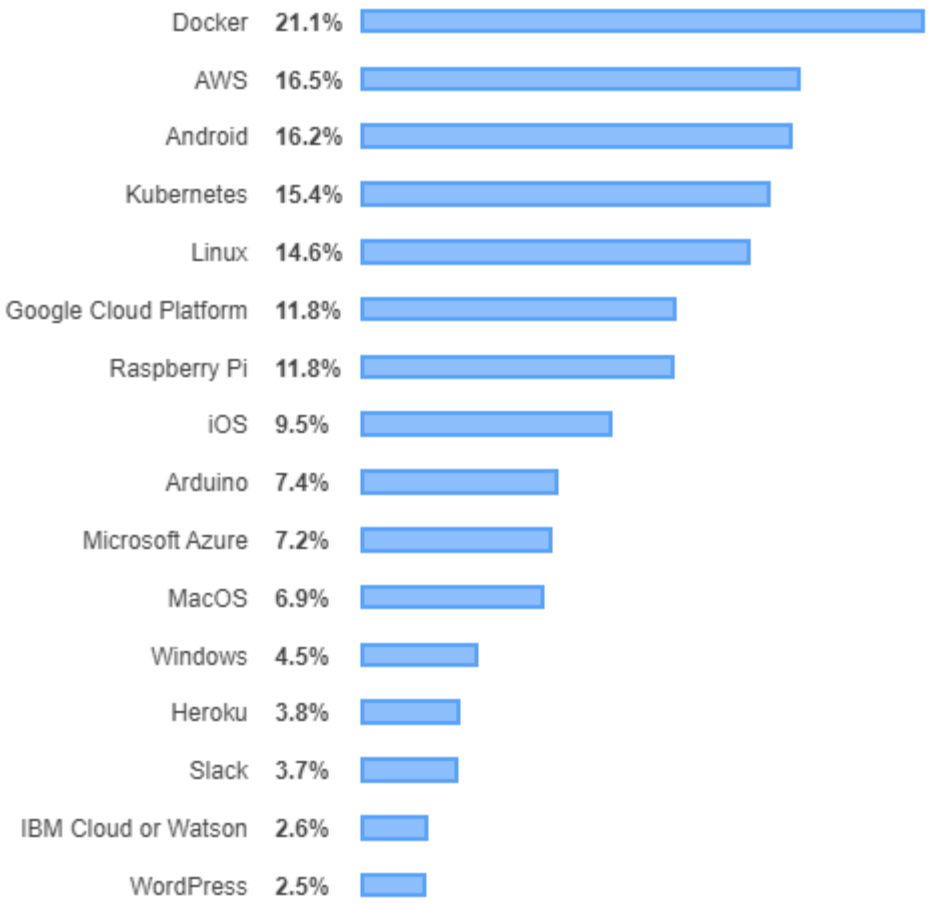

Ilustración 4*. Tecnologías más buscadas del año 2019.*

Fuente*: Encuesta 2019 de StackOverflow.*

En la ilustración 4, esta representa el porcentaje de desarrollares que presentan un interés por desarrollar con cierta tecnología, por lo que al ubicar a AWS en segundo lugar de la lista y primera en cuanto a tecnología en la nube se refiere, indica lo importante y deseado para el mundo informático es esta infraestructura.

Con todo lo anterior, se añade otro peso más a la elección y que marca una gran diferencia con el resto de los proveedores en la nube, el *Free-Tier*. Esta es una funcionalidad pensada para los que quieren ingresar al mundo de la nube, ofreciendo tres servicios gratis: (1) gratis para siempre, (2) gratis por 12 meses, (3) gratis para pruebas. Para el desarrollo del trabajo, se decanta por la primera opción ya que básicamente posee todos los servicios necesarios a coste cero, resumiéndose en: 25GB de almacenamiento en DynamoDB, un millón de solicitudes mensuales a AWS Lambda, y el uso de máquinas virtuales para el desarrollo del servidor en AWS EC2, entre otras, pero siendo estas tres las principales en uso para el trabajo.

2.4.5 Amazon Web Services Lambda (AWS Lambda)

Para el desarrollo del presente trabajo, dentro de toda la infraestructura de red de la nube, se escoge a *FaaS* como la opción más correcta para lograr el objetivo de la investigación. Dentro de todas las herramientas la elegida es Amazon Web Services, plataforma líder en la implementación de servicios en la nube, así como la más adoptada y completa en el mundo. La cantidad de herramientas ofertadas es grandiosa, con más de 165 servicios dentro de su catálogo (Services, 2019).

Una de estas herramientas es *AWS Lambda*, de la cuál podemos obtener todos los beneficios necesarios para el desarrollo de la aplicación. Dentro de estos tenemos:

- Descartar la administración de servidores, únicamente escribimos el código a ejecutar y lo ejecutamos en Lambda.
- Fácil escalabilidad, siendo cada función disparada de manera separada y pudiendo funcionar paralelamente.

- Respuesta rápida, consiguiendo tiempos de ejecución de las funciones en menos de un segundo, abaratando los costos de ejecución de las funciones.

Dentro de los diferentes casos de uso de Lambda podemos ejecutar código como respuesta a disparadores que pueden ser cambios en los datos, cambios aplicaciones al estado del sistema o acciones de los usuarios, creación de imágenes miniatura, codificar videos, procesos de creación, edición, borrado y lectura de datos de bases de datos, etc. Todos estos procesos los podemos ejecutar para administrar solicitudes web, móviles, Internet de las cosas y API de terceros (AWS | Lambda - Gestión de recursos informáticos, n.d.).

2.5 Metodología de calificación

El objetivo principal del presente trabajo es lograr calificar el rendimiento de las aplicaciones web desarrolladas en el servidor, contra la adopción del desarrollo en la nube. El principal desafío es poder calificar el rendimiento de las aplicaciones que adopten la nube, todo esto debido a que es un campo relativamente nuevo. El rendimiento de la nube centraliza el estudio en aspectos tecnológicos como los recursos y servicios adoptados en el desarrollo, costos de uso y métricas relacionadas con la elasticidad(Lakew et al., 2017).

Para el cumplimiento de la meta, se busca una metodología que nos permita comparar el rendimiento basados en varios puntos a calificables, para ello se diseñan pruebas que permitan reportar el rendimiento en la nube como en el servidor mediante métricas comunes, para ello se centraliza en análisis en la reproducibilidad de los resultados, característica propia de una metodología experimental (Feitelson, 2006).

Dentro del ámbito informático, una métrica se define como un valor resultante de varias medidas obtenidas a partir de un análisis experimental(Edwards, 1964).

Para ello, en (Papadopoulos et al., 2019), nos proporciona ocho principios metodológicos para la calificación de rendimiento de uso de la nube, aplicables también a un desarrollo local. Estos principios se desarrollaron pensando en la reproducibilidad de resultados de la experimentación, con los requerimientos de cumplimiento mínimos para su calificación, llegando a una conclusión medible de los resultados, estos son:

2.5.1 Repetición del experimento

Definir las variables a ser comprobadas mediante la experimentación de manera repetida, para lograr una cuantificación final que refleje de la manera más precisa el resultado final. Para ello, se debe reducir al máximo factores que puedan afectar el rendimiento en la nube, ya que existen factores como la concurrencia de solicitudes o la calidad de la señal de Internet que afectan de una manera u otra el rendimiento. El número de reproducibilidad del experimento va a ser definido acorde a la precisión requerida o un marco aceptable de errores.

2.5.2 Carga de trabajo y cobertura de la configuración

El desarrollo de la experimentación debe realizarse en diferentes escenarios de configuración para cubrir la mayoría de los aspectos que no puedan ser controlados de manera directa. La mayoría de los servicios en la nube poseen varias configuraciones que deben ser ejecutadas por el desarrollador, aunque eso no significa que la mayoría de las veces van a ser ejecutadas de la misma manera, así la configuración haya sido la correcta.

Para evitar una mala ejecución de la experimentación, la elección de los valores paramétricos debe basarse en configuraciones realistas, para ello, el encargado del experimento debe definir las variables del sistema que puedan afectar el resultado de los datos, para luego clasificarlos dentro de un conjunto de datos reales obtenidos.

2.5.3 Configuración del entorno de experimentación

Descripción detallada tanto de software como del hardware en el cual se va a experimentar, adicional a otros parámetros que puedan modificar el comportamiento de la experimentación. Dentro de las descripciones del entorno se debe incluir: el sistema n el cuál se va a realizar el experimento, el entorno en el que el sistema va a ser testeado, la carga de trabajo, el software o técnicas mediante las cuales se va a monitorizar los datos, y el objetivo de cada experimento, mismas técnicas que deben ser aplicadas tanto en el servidor como en la nube.

2.5.4 Libre acceso del artefacto

La comunidad debe tener acceso a un subconjunto representativo del software desarrollado, así como a los datos obtenidos, se incluye además una descripción detallada de la metadata propia del software. El artefacto entregable debe proveer un contenido educativo, facilitando a estudiantes datos interesantes para futuros estudios o trabajos.

2.5.5 Descripción probable del resultado a obtener

Desarrollo de un informe de los resultados que se esperan obtener mediante la experimentación, todo esto mediante aportaciones basadas en un conocimiento empírico. Se debe tener en cuenta los aportes, ya que no se deben desviar de la línea general del experimento. La información presentada debe estar bien elaborada mediante el uso de

diferentes diagramas de datos, acorde al volumen de datos y el resultado a obtener, tener en cuenta también una tendencia central.

2.5.6 Evaluación estadística

Proporcionar una evaluación estadística de la importancia de los datos obtenidos mediante la experimentación. Si bien la mayoría de los resultados pueden no ser reportados de una manera cuantitativa, al final estos aportan una conclusión sobre el problema estudiado, conclusión que puede ser tomada para una comparativa con diferentes instrumentos.

2.5.7 Unidades de medida

Todas las cantidades obtenidas durante el proceso deben ser reportadas con su unidad de medición correspondiente.

2.5.8 Costo

Durante la experimentación en la nube, se deben incluir: el modelo de costo asumido para la experimentación, los recursos contabilizados independientemente del modelo utilizado, y el costo total facturado acorde al modelo, ya que cada servicio en la nube tiene su propio cálculo de costo, así como los parámetros contables para el cálculo.

Capítulo 3

Experimento

3.1 Introducción

Como se describe en el capítulo previo, para poder realizar una calificación de rendimiento aplicables tanto a la nube como al ámbito local, se aplica ocho principios metodológicos en la aplicación web desarrollada. En este capítulo se desarrolla cinco de los ocho principios, los cuales refieren más a la parte teórica de estas, para ellos, el caso de estudio a aplicar para la realización del experimento que se expone a continuación.

3.2 Propuesta de diseño

Para el cumplimiento de los objetivos del presente trabajo, se propone el desarrollo de una pequeña aplicación web que va a contar con dos servicios, un S1 encargado de generar tablas de amortización para préstamos de dinero (máximo 180 meses) para luego almacenarlas dentro de la base de datos, y otro servicio S2 encargado de consultar los préstamos existentes, cuotas pagadas y pendientes de transacciones realizadas; servicios que van a ser desarrollados tanto dentro de una arquitectura de servidor como dentro de una arquitectura en la nube. Como lenguajes de desarrollo, dentro del lado del cliente se va a trabajar con el *framework* Angular 2, y como lenguaje de servidor se desarrolla con Node.js, ya que se acopla de manera perfecta tanto con la programación en la nube como con el desarrollo en servidor.

3.3 Desarrollo de la metodología

3.3.1 Repetición del experimento

Para el primer punto, debemos definir las variables que se van a medir durante el experimento, mismas que serán el resultado final del trabajo. Como se detalla a lo largo de todos los capítulos del documento, el principal objetivo es poder obtener resultados sobre duración de una solicitud y costo por solicitud, por lo tanto, estos tres parámetros van a ser las variables que se van a analizar en esta metodología.

Adicional, se debe definir la cantidad de iteraciones del experimento que nos permiten obtener datos realistas de las pruebas. Para ello se va a realizar pruebas mediante un continuo lote de solicitudes tanto a la nube como al servidor, lote que será de tamaño de unas 500 solicitudes continuas hacia los servicios.

3.3.2 Carga de trabajo y cobertura de configuración

Para este principio, el lote de solicitudes va a ser realizado en diferentes horas del día, lo que nos permite tener un gran número de datos en diferentes situaciones. Con estos datos, es capaz de realizar un cuadro estadístico más completo tanto del rendimiento en la nube como el rendimiento en el servidor, para posteriormente poder realizar una comparación entre los resultados.

La aplicación una vez construida, será desplegada en **Github Pages**, que no es más que un servicio de hosting para nuestras aplicaciones, proporcionando una dirección física a la que puede acceder desde cualquier parte. Adicional, como se detalló, los servicios en la nube se encuentran desplegados en AWS Lambda, accesibles mediante un api desarrollada en AWS Api Gateway. Para el servidor, este se encuentra desplegado en un servidor virtual mediante una instancia de AWS EC2, mismo que corre en un sistema operativo Ubuntu Server.

3.3.3 Configuración del entorno de experimentación

El desarrollo, configuración y tecnologías utilizadas para el experimento se encuentran detalladas en el capítulo 4, donde se realizó el desarrollo de aplicación, desarrollo del backend en el servidor y el desarrollo de las funciones en la nube que se van a utilizar.

3.3.4 Libre acceso al artefacto

El resultado de los datos auditados en la nube puede ser accesibles por el equipo de trabajo vinculado a la cuenta de desarrollo en AWS. En los repositorios de **Github** del autor se puede obtener acceso público al código fuente tanto del frontend como del backend desarrollado para el experimento. Adicional, el presente documento reposará en las bibliotecas de la institución para futuras investigaciones.

3.3.5 Descripción probable del resultado a obtener

Como es conocido, un entorno cliente – servidor es la conexión más directa entre dos ordenadores, evitando saltar de servidor en servidor, por lo que se obtiene una respuesta a las solicitudes mucho más rápida. Por lo tanto, el resultado que se espera obtener es corroborar que una conexión directa es más rápida, pero sin descartar la velocidad de respuesta de una tecnología sin servidor. Hay que tener muy en cuenta que el objetivo general no es sepultar la nube como alternativa de desarrollo, sino como una alternativa casi tan fiable y veloz como un entorno cliente - servidor.

Una vez que se han desarrollado los primeros cinco principios de la metodología, se continua con el proceso de desarrollo de la aplicación web que va a alojar los microservicios, así como configuraciones de los entornos de desarrollo, funciones a

implementar y descripción de uso de cada archivo de la aplicación. Para el desarrollo del experimento y basados en los *frameworks* de desarrollo previamente vistos en la *Tabla 1*, el desarrollo del lado del cliente se encuentra programado en Angular 2.

3.4 Desarrollo

La aplicación propuesta cuenta con dos microservicios alojados tanto en la nube como en el servidor Node.js mediante el *framework* Express, para poder realizar una calificación de rendimiento en entornos correctamente configurados y sin ventajas sobre el otro. Se escogen los servicios propuestos ya que conllevan un gran proceso de cálculo en cuanto a un servicio, y el otro por la cantidad de datos a leer desde la base de datos. En este capítulo se detalla el trabajo realizado del lado del cliente, así como la descripción de funcionamiento de cada uno de los servicios desarrollados. Como servicio de almacenamiento de datos se aprovecha la versatilidad de DynamoDB, el mismo que va a ser el encargado de alojar la información registrada por el usuario.

En la ilustración 5, se muestra la pantalla de interacción con el primer servicio, que consiste en un formulario para el ingreso de datos necesarios para generar una nueva tabla de amortización.

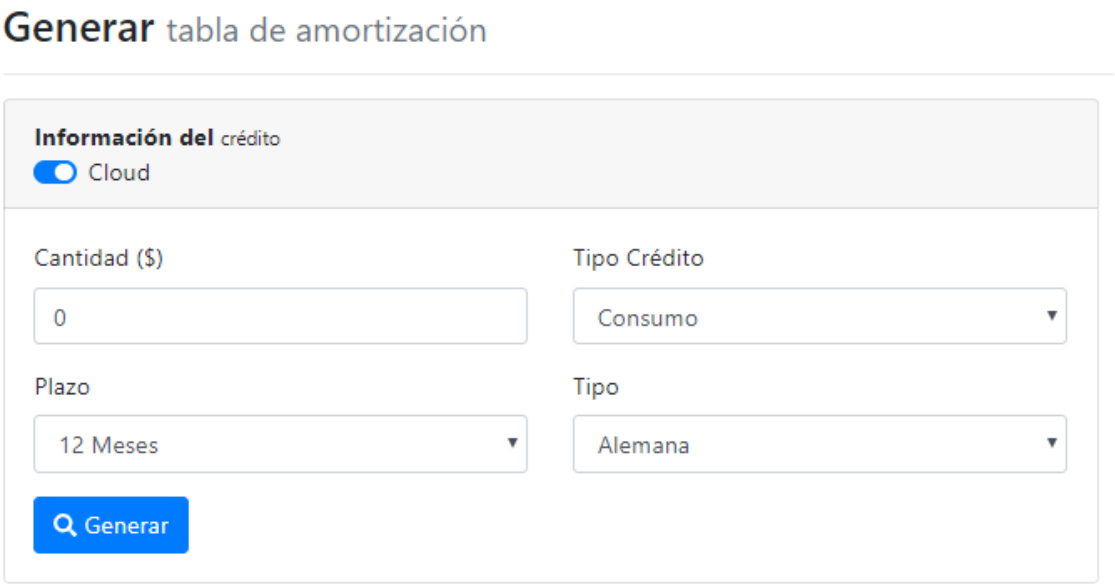

***Ilustración 5**. Formulario de consulta de nueva tabla de amortización.*

***Fuente**: Autor*

Cantidad: El monto de dinero en dólares solicitado, el cual sirve como base para el desarrollo de la consulta.

Tipo de crédito: El tipo de crédito a aplicar, cada uno con su valor de interés propio, interés que es utilizado en el cálculo de las cuotas del crédito. Para efecto del desarrollo del trabajo, los tipos de créditos aplicables son:

- **Consumo,** que cuenta con un interés anual del 15% sobre el monto.
- **Vivienda,** que cuenta con un interés anual del 10.44% sobre el monto.
- **Microempresarial,** que cuenta con un interés anual del 17% sobre el monto.
- **Comercial,** que cuenta con un interés anual del 10.99% sobre el monto.

Plazo: Cantidad en meses del número de cuotas para cubrir el monto del crédito, tanto dividendos de capital como interés generado mensualmente.

Tipo de amortización: Tipo de cálculo de dividendos para el cuál un cliente aplica. Para el ejercicio, los tipos aplicables son las amortizaciones de origen francés y de origen alemán.

Los datos del formulario son validados para evitar la existencia de datos inconsistentes o nulos. Una vez que el formulario es llenado de manera correcta, la información es enviada a la opción que escoja el usuario, ya sea que sea resuelta por el servidor o por las funciones en la nube. Para ello, bajo el título cuenta con *switch* que cambia el destino de la información.

En la ilustración 6 se observa los datos devueltos renderizados en el *DOM*, de manera que sean visibles para el usuario, siendo los cálculos totalmente abstractos tanto para el cliente como para el navegador web. Finalmente, el usuario decide si los datos son los correctos y se procede o no a grabar los registros en la base de datos.

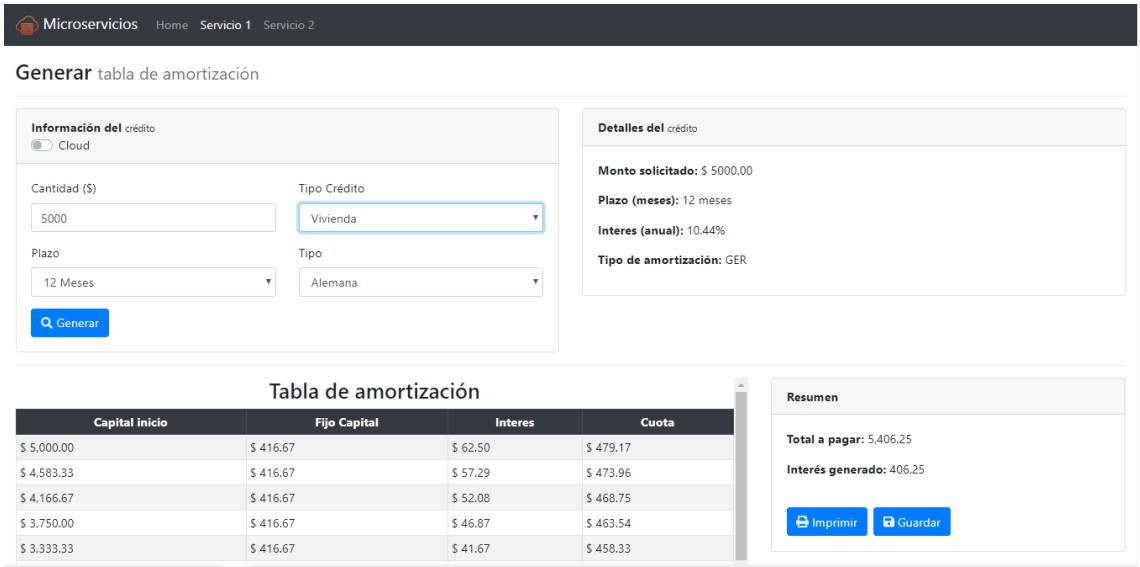

***Ilustración 6**. Pantalla principal de interacción con el servicio S1.*

***Fuente**: Autor*

El segundo servicio para implementar consiste en leer todos los datos ingresados dentro de la base de datos para poder realizar lo cobros de cada una de las cuotas correspondientes al crédito. Como restricciones contiene que no se puede cualquier cuota, sino que debe seguir un orden secuencial, por lo que, si no existe una cuota previa pagada, no se va a poder aceptar el cobro, adicional, una vez que todas las cuotas sean pagadas, el estado del préstamo pasa a pagado y ya no se puede hacer más cambios dentro del mismo. De igual manera que el servicio anterior, cuenta con un *switch* para permitir cambiar el origen de donde queremos obtener y transaccionar los datos.

En la ilustración 7 se observa cómo se encuentra ordenada la información de los créditos, mismos que fueron obtenidos desde la base de datos, permitiéndonos interactuar para obtener detalles de cada uno. En la ilustración 8 se observa los detalles de un crédito seleccionado, que incluye el número de cuotas, monto a pagar y estado de la cuota.

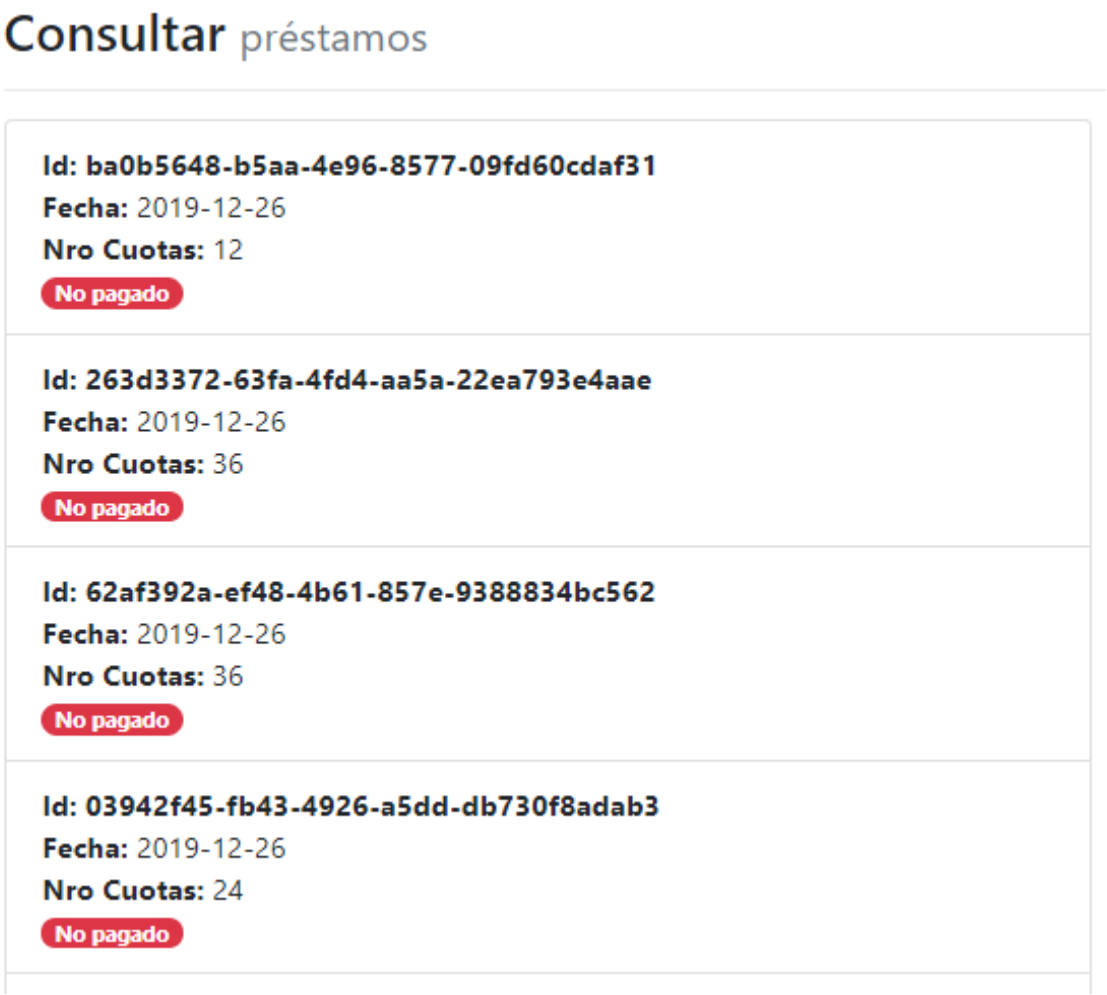

***Ilustración 7**. Carga de créditos desde DynamoDB*

***Fuente**: Autor*

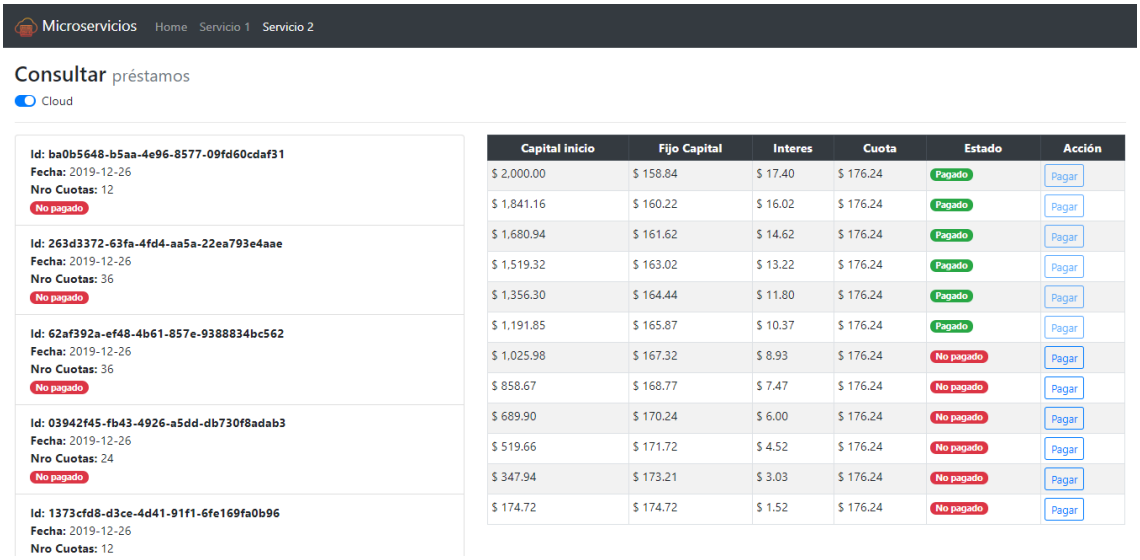

***Ilustración 8**. Detalle de crédito seleccionado.*

***Fuente**: Autor*

Antes de continuar con el desarrollo del backend, se definen las funciones necesarias y el flujo de información entre el *backend* y el *frontend*. Para el caso de estudio, los procesos necesarios son los siguientes:

1) Como primer proceso, se necesita una función que reciba los datos del formulario de solicitud de crédito, una vez calculada la tabla de amortización, la función retorna en formato JSON los datos de los pagos a realizar del crédito.
2) El siguiente proceso consiste en almacenar los datos generados en la base de datos para su posterior lectura. Este proceso es opcional para el cliente.
3) A continuación, se necesita un proceso que escanee todos los datos de DynamoDB y los retorne al cliente para poder trabajar con ellos.
4) Finalmente, es requerida una función que permita realizar el pago de cada de las cuotas, validando el pago a registrar.

En la ilustración 9 se grafica el flujo de procesos que realiza la aplicación. En un principio, el cliente realiza una petición desde el frontend, Javascript analiza el destino de la solicitud y resuelve si la petición va a la nube o al servidor, recibiendo la respuesta y proporcionando retroalimentación al usuario.

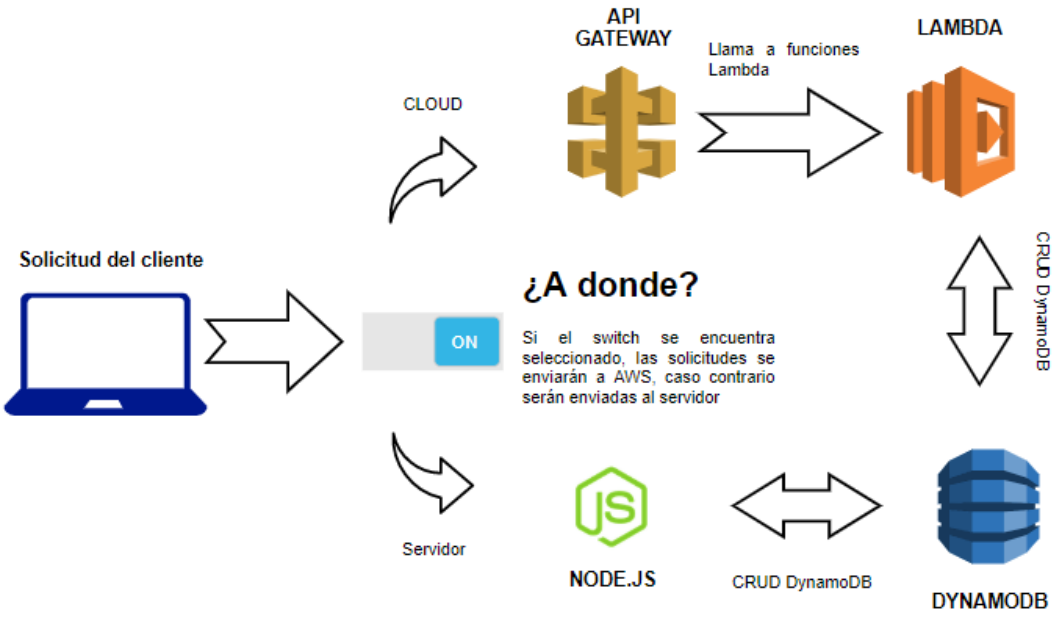

***Ilustración 9**. Flujo de trabajo de solicitudes.*

***Fuente**: Autor*

3.4.1 Desarrollo en una infraestructura en la nube

Tal como se detalló en capítulos anteriores, las funciones dentro de la infraestructura de la nube van a ser desarrolladas en JavaScript mediante el *framework* NodeJS en su versión 10, accesibles mediante API propias de AWS. Como primer paso, para que todo funcione correctamente debemos crear un rol de usuario con los permisos

necesarios para el funcionamiento correcto de las solicitudes; para ello, los roles se otorgan desde Amazon IAM(Amazon, n.d.), el cual no es más que un gestor de permisos, roles, y acceso a los recursos que provee Amazon. Para ello, desde la consola servicios de Amazon y accedemos a la opción de Amazon IAM, misma que se muestra en la ilustración 10 y 11.

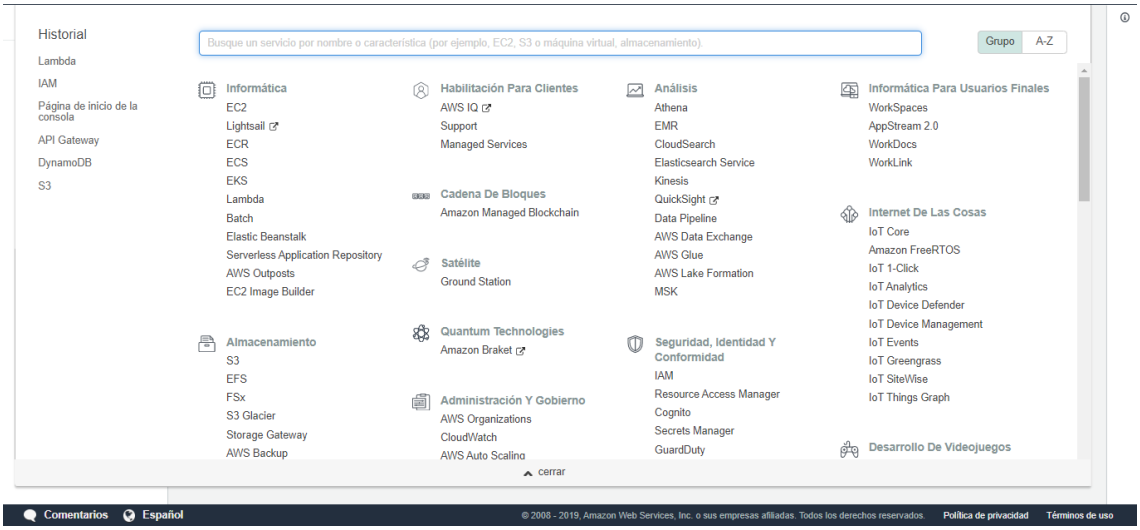

Ilustración 10. *Consola de servicios AWS.*

Fuente: *Autor*

Ilustración 11. *Pestaña de roles de usuario.*

***Fuente**: Autor*

Una vez dentro del servicio, se procede a la creación de un rol de desarrollo con los permisos a utilizar, estos son: Acceso completo a las funciones de Lambda, y opciones CRUD para la base de datos, permisos que podemos ver activos en la ilustración 12.

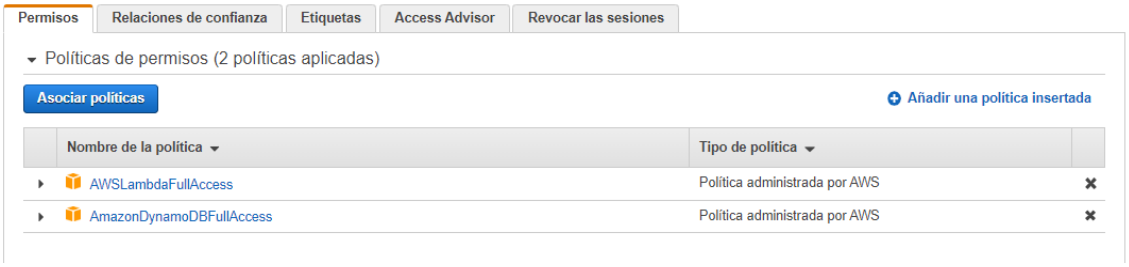

***Ilustración 12**. Políticas de permisos.*

***Fuente**: Autor*

Una vez definido el rol de acceso a servicios, procedemos a la creación de las funciones en el Amazon. Para ellos se sigue los procesos definidos previamente. Al final las funciones resultantes quedan establecidas como se observa en la ilustración 13:

- Dev-amortización
- Dev-crédito
- Dev-créditos
- Dev-pagar

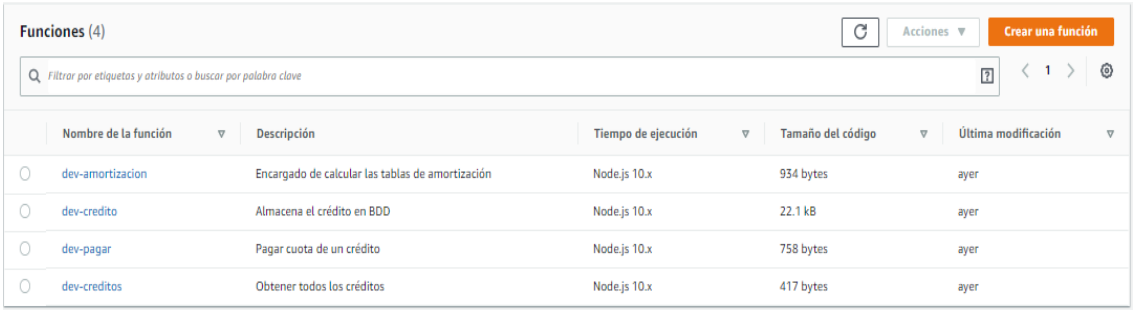

***Ilustración 13**. Funciones Lambda.*

***Fuente**: Autor*

Detallando más sobre las funciones, observamos información acerca del desarrollo de estas, como vemos, el lenguaje en el que está desarrollado es Node.js 10.X, además del tamaño de las líneas de código que abarca cada una de ellas, mismas que de una u otra forma afectarán al funcionamiento de cada función. A continuación, se muestra el código fuente de cada una de ellas:

Dev-amortización

```
'use-strict';

exports.handler = (event, context, callback) => {
    const tabla_amort = amortizacion(event.capital, event.plazo, event.amort,
event.interes);
    callback(null, tabla_amort);
};

function response(statusCode, message) {
    return {
      statusCode: statusCode,
      body: message
    };
}

function sortByDate(a, b) {
    if (a.createdAt > b.createdAt) {
      return -1;
    } else return 1;
}

function amortizacion(capitalR, plazoR, amortR, interesR){
    if( parseFloat(capitalR) < 1000) {
```

```
      return response(400, 'Monto mínimo para préstamo es $1000');
    }
    let capital = parseFloat(capitalR);
    let plazo = parseFloat(plazoR);
    let tipo = amortR;
    const tabla_amort = [];
    // Amortización alemana
    if ( tipo === 'GER') {
      let interes = (( parseFloat(interesR) / 12 ));
      const fijo_capital = ( capital / plazo );
      for( let i = 1; i <= plazo; i++) {
        let pago_interes = ((capital * interes) / 100);
        tabla_amort.push({
          capital: capital,
          cuota_interes: pago_interes,
          cuota_capital: fijo_capital,
          pago_final: (pago_interes + fijo_capital)
        });
        capital = (capital - fijo_capital);
      }
    }
    // Amortización francesa
    if( tipo === 'FR') {
      let int_fr = (parseFloat(interesR) / plazo) / 100;
      let interes = parseFloat(interesR) / plazo;
      const cuota_fija = ( capital * int_fr)/(1 - Math.pow( 1 + int_fr, -
plazo));
      for( let i = 1; i <= plazo; i++) {
        const cuota_interes = (capital * interes) / 100;
        tabla_amort.push({
          capital: capital,
          cuota_interes: cuota_interes,
          cuota_capital: cuota_fija - cuota_interes,
          pago_final: cuota_fija
        });
        capital = capital - (cuota_fija - cuota_interes);
      }
    }
    return response(201, tabla_amort);
}

exports.amortizacion  = amortizacion;
```

Dev-crédito

```
'use-strict';
const uuidv4 = require('uuid/v4');
const AWS = require('aws-sdk');
const DB = new AWS.DynamoDB.DocumentClient({region: 'sa-east-1'});

exports.handler = function(event, context, callback) {
  const cuotas = [];
  event.forEach( (det) => {
    cuotas.push({
```

```
      pagado: false,
      ...det
    });
  });

  const credito = {
    id: uuidv4(),
    fecha: new Date().toLocaleDateString(),
    cuotas: cuotas,
    pagado: false
  };

    const params = {
      TableName: 'Creditos',
      Item: credito
    };

    return DB
      .put(params)
      .promise()
      .then(() => {
        callback(null,  {
        statusCode: 201,
        body: JSON.stringify('Insertado Correctamente')});
      })
      .catch((err) =>  callback(null,  {
        statusCode: 400,
        body: JSON.stringify(err)}));
}
```

Dev-créditos

```
'use-strict';
const AWS = require('aws-sdk');
const DB = new AWS.DynamoDB.DocumentClient({region: 'sa-east-1'});

exports.handler = (event, context, callback) => {
    const params = {
      TableName: 'Creditos'
    };

    DB.scan(params, (err, data) => {
      if (err) {
        callback(null, response(400, `Error: ${err}`));
      } else {
        callback(null, response(201, data.Items ) );
      }
    });
};

function response(statusCode, message) {
    return {
      statusCode: statusCode,
      body: message
    };
```

```
}
```

Dev-pagar

```
'use-strict';
const AWS = require('aws-sdk');
const DB = new AWS.DynamoDB.DocumentClient({region: 'sa-east-1'});

exports.handler = (event, context, callback) => {
    const idTabla = event.amort.id;
    const cuotas = event.amort.cuotas;
    const cuotaIdx = event.cuotaIdx;

    if ( cuotaIdx === 0) {
      cuotas[0].pagado = true;
    } else if ( cuotaIdx > 0 && cuotaIdx < cuotas.length ) {
      if ( cuotas[cuotaIdx - 1].pagado ) {
        cuotas[cuotaIdx].pagado = true;
      } else {
        callback(null, response(400, `Error: Debe pagar la cuota anterior`));
        return;
      }
    }

    if (check_pagado_all( cuotas )) {
      event.amort.pagado = true;
    } else {
      event.amort.pagado = false;
    }

    const params = {
      TableName: 'Creditos',
      Key: {
        id: idTabla
      },
      UpdateExpression: "SET cuotas = :cuotas, pagado = :pagado",
      ExpressionAttributeValues: {
        ":cuotas": cuotas,
        ":pagado": event.amort.pagado
      }
    };

    DB.update( params, () => {
      callback( null, response(201, 'Guardado Correctamente') );
    });

};

function check_pagado_all( lista) {
    let cuotas_pagadas = 0;
    lista.forEach( (item) => {
      if (item.pagado) {
        cuotas_pagadas++;
      }
    });
```

```
    // reviso si las cuotas pagadas es igual a la cantidad de cuotas
    if ( cuotas_pagadas === lista.length) {
      return true;
    } else {
      return false;
    }
  }

function response(statusCode, message) {
    return {
      statusCode: statusCode,
      body: message
    };
}
```

Para poder consumir las funciones definidas, se debe hacer desde Amazon API Gateway, entonces continuando con el desarrollo llegó el momento de crear las peticiones necesarias para consumir cada una de estas y para ello desde la consola de AWS, ingresamos a la opción de API Gateway y creamos nuestra primera API. Una vez creada la API, se definen el tipo de peticiones que la misma va a realizar, para ello se establece lo siguiente:

- **POST:** esta petición nos va a permitir guardar los datos en la DynamoDB, para ello este recurso va a estar vinculado a la función **dev-crédito**, siendo el URL de petición.
- **GET:** mediante esta petición vamos a recibir una respuesta del servidor a una consulta mediante la cual pasamos ciertos parámetros. Este recurso está vinculado a las funciones **dev-amortización,** que va a recibir como parámetros el capital, el plazo, el interés y el tipo de amortización base para calcular el crédito; y **dev-créditos** que se encarga de escanear todos los registros de la tabla **Créditos** en DynamoDB.

- **PUT:** esta petición nos va a permitir actualizar registros que coincidan con los parámetros enviados. Este recurso está ligado a la función **dev-pagar**, encargada de actualizar el estado de una cuota y del crédito.

De esta manera, solamente queda realizar peticiones *Http* desde el frontend para poder consumir los recursos de las funciones definidas.

3.4.2 Desarrollo en una infraestructura de servidor

En este capítulo se explica la configuración del entorno de servidor requerido para el funcionamiento de la aplicación. Para ello se va a utilizar un servidor virtual de *Amazon EC2*, el mismo que viene preinstalado el sistema operativo UBUNTU Server 18.04, configurada con los puertos descritos en la ilustración 14 con su respectivo protocolo de uso. Además, como protocolo de envió de datos se utiliza la tecnología de Websockets, que va a permitir un intercambio de datos asíncrono con el cliente.

Type	Protocol	Port Range	Source		Description
HTTP	TCP	80	Custom	0.0.0.0/0	e.g. SSH for Admin Desktop
HTTP	TCP	80	Custom	::/0	e.g. SSH for Admin Desktop
All TCP	TCP	0 - 65535	Custom	0.0.0.0/0	e.g. SSH for Admin Desktop
All TCP	TCP	0 - 65535	Custom	::/0	e.g. SSH for Admin Desktop
Custom TCP I	TCP	45000	Custom	0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP I	TCP	45000	Custom	::/0	e.g. SSH for Admin Desktop
SSH	TCP	22	Custom	0.0.0.0/0	e.g. SSH for Admin Desktop
SSH	TCP	22	Custom	::/0	e.g. SSH for Admin Desktop
HTTPS	TCP	443	Custom	0.0.0.0/0	e.g. SSH for Admin Desktop
HTTPS	TCP	443	Custom	::/0	e.g. SSH for Admin Desktop
All ICMP - IPv	ICMP	0 - 65535	Custom	0.0.0.0/0	e.g. SSH for Admin Desktop
All ICMP - IPv	ICMP	0 - 65535	Custom	::/0	e.g. SSH for Admin Desktop

Ilustración 14. *Protocolos aceptados en el servidor.*

Fuente*: Autor*

Una vez configurados, Amazon nos entrega las llaves de acceso a nuestra máquina virtual, además de un IP pública y el nombre de nuestro servidor para poder acceder a este a través del protocolo SSH. Adicional, también tendremos acceso a nuestro servidor por medio del protocolo TCP, para poder subir los archivos necesarios al servidor y ponerle en funcionamiento. Una vez que tenga todo el acceso completo, procedemos a instalar y configurar los programas y servicios necesarios para que nuestro servidor esté completamente funcional, siendo aquí donde radica la principal diferencia con la programación en la nube.

Como primer paso, necesitamos instalar *Node.js* en la misma versión que usamos en AWS Lamba, o sea, Node.js 10.X y lo comprobamos desde la consola. En la ilustración 15 se observa la respuesta de Node, siendo esta la versión 10.17.0.

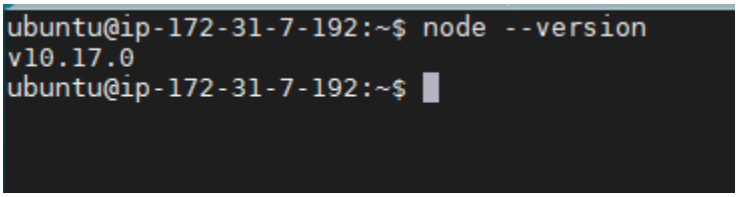

Ilustración 15. *NodeJS 10.X en servidor*.

Fuente: *Autor*

Asegurando que Node se encuentre instalado, podemos acceder a la instalación de paquetes, así como al motor V8 de Google que permite correr JavaScript como servidor, así como la librería Express, que facilita el desarrollo del servidor. Adicional, debemos instalar el compilador de Typescript, para poder transformar el código de servidor a JavaScript puro para que pueda correr en el servidor. Con todo esto ya instalado, se

procede a desarrollar el proyecto de Node encargado de complementarse con nuestra aplicación web. La estructura del proyecto se describe en la ilustración 16:

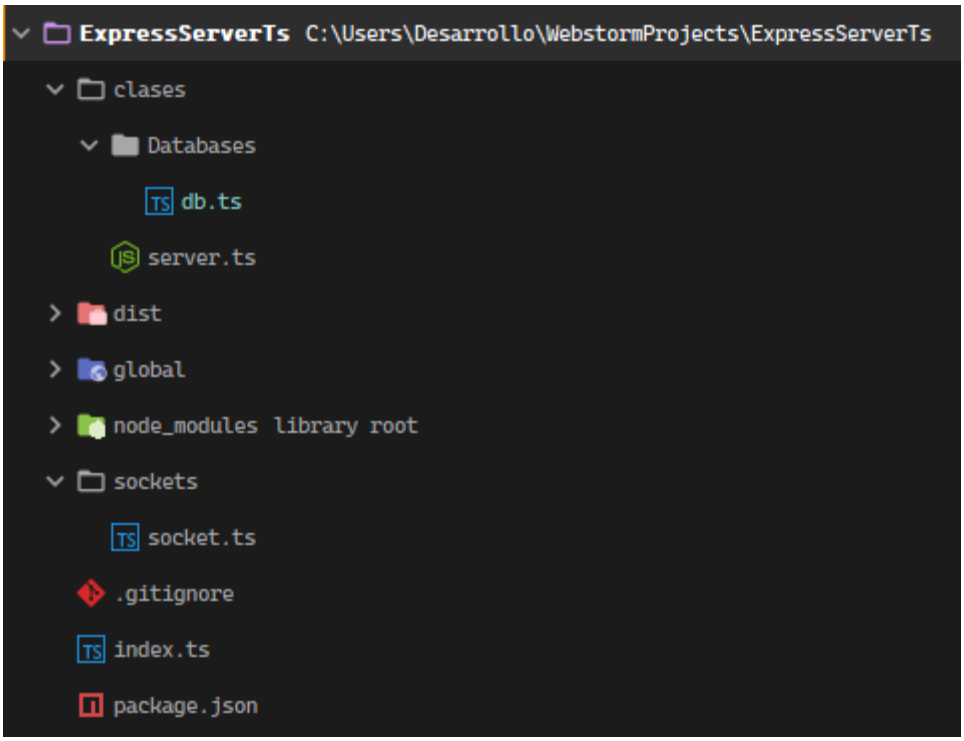

Ilustración 16*. Estructura del proyecto.*

Fuente*: Autor*

Clases, donde incluiremos las carpetas de conexión a la base de datos de DynamoDB, así como el archivo de inicialización de servidor.

Dist, carpeta donde se encuentran los archivos ya compilados a JavaScript, estos son los archivos que necesitaremos para poner en funcionamiento el servidor.

Global, donde podemos definir variables globales, por ejemplo, la configuración de conexión a los sockets.

Node_modules, no es más que las librerías instaladas necesarias para el correcto funcionamiento de la aplicación.

Sockets, donde se incluyen todos los eventos de sockets, tanto de envió como de respuesta a una petición.

Gitignote, no es más que un archivo del control de versiones, encargado de ignorar carpetas y archivos que no se van a subir al repositorio.

Index.ts, archivo principal, es el archivo que junta todos los archivos y pone en funcionamiento el servidor.

Package.json, archivo en formato JSON, donde se listan los paquetes utilizados en el proyecto, así como información de este.

A continuación, se incluye el código fuente del desarrollo del servidor, se incluirá solo el código ya compilado a JavaScript.

Clases/ Databases/ db.js

```
"use strict";
var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
Object.defineProperty(exports, "__esModule", { value: true });
const aws_sdk_1 = __importDefault(require("aws-sdk"));
const DynamoDB = require("aws-sdk/clients/dynamodb");
const uuidv4 = require('uuid/v4');
class Db {
    constructor() {
        aws_sdk_1.default.config.update({ region: 'sa-east-1' });
        this.DB = new DynamoDB.DocumentClient();
    }
    static get instance() {
        return this._instance || (this._instance = new this());
    }
    response(statusCode, message) {
        return {
            statusCode: statusCode,
            body: message
        };
    }
    sortByDate(a, b) {
        if (a.createdAt > b.createdAt) {
```

```
            return -1;
        }
        else
            return 1;
    }
    //=========================================================//
    // SIMULADOR CREDITO
    //=========================================================//
    // simulador_cred_server
    simulador_cred_server(payload, callback) {
        if (payload.monto < 1000) {
            callback(this.response(400, 'Monto mínimo para préstamo es $1000'));
        }
        let capital = payload.monto;
        let plazo = payload.plazo;
        let tipo = payload.amort;
        const tabla_amort = [];
        // Amortización alemana
        if (tipo === 'GER') {
            let interes = ((payload.interes / 12));
            const fijo_capital = (capital / plazo);
            for (let i = 1; i <= plazo; i++) {
                let pago_interes = ((capital * interes) / 100);
                tabla_amort.push({
                    capital: capital,
                    cuota_interes: pago_interes,
                    cuota_capital: fijo_capital,
                    pago_final: (pago_interes + fijo_capital)
                });
                capital = (capital - fijo_capital);
            }
        }
        // Amortización francesa
        if (tipo === 'FR') {
            let int_fr = (payload.interes / plazo) / 100;
            let interes = payload.interes / plazo;
            const cuota_fija = (capital * int_fr) / (1 - Math.pow(1 + int_fr, -
plazo));
            for (let i = 1; i <= plazo; i++) {
                const cuota_interes = (capital * interes) / 100;
                tabla_amort.push({
                    capital: capital,
                    cuota_interes: cuota_interes,
                    cuota_capital: cuota_fija - cuota_interes,
                    pago_final: cuota_fija
                });
                capital = capital - (cuota_fija - cuota_interes);
            }
        }
        callback(this.response(201, tabla_amort));
    }
    // save_credito_server
    save_credito_server(payload, callback) {
        const cuotas = [];
        payload.forEach((det) => {
            cuotas.push(Object.assign({ pagado: false }, det));
        });
        const credito = {
            id: uuidv4(),
            fecha: new Date().toLocaleDateString(),
```

```
            cuotas,
            pagado: false
        };
        const params = {
            TableName: 'Creditos',
            Item: credito
        };
        this.DB.put(params, (err, data) => {
            if (err) {
                callback(this.response(400, `Error: ${err}`));
            }
            else {
                callback(this.response(201, 'Insertado Correctamente'));
            }
        });
    }
    //=========================================================//
    // MANEJO DE PRÉSTAMOS
    //=========================================================//
    // get_prestamos_server
    get_prestamos_server(payload, callback) {
        const params = {
            TableName: 'Creditos'
        };
        this.DB.scan(params, (err, data) => {
            if (err) {
                callback(this.response(400, `Error: ${err}`));
            }
            else {
                callback(this.response(201, data.Items));
            }
        });
    }
    // pagar_cuota_server
    pagar_cuota_server(payload, callback) {
        const idTabla = payload.amort.id;
        const cuotas = payload.amort.cuotas;
        const cuotaIdx = payload.cuotaIdx;
        if (cuotaIdx === 0) {
            cuotas[0].pagado = true;
        }
        else if (cuotaIdx > 0 && cuotaIdx < cuotas.length) {
            if (cuotas[cuotaIdx - 1].pagado) {
                cuotas[cuotaIdx].pagado = true;
            }
            else {
                callback(this.response(400, `Error: Debe pagar la cuota anterior`));
                return;
            }
        }
        if (this.check_pagado_all(cuotas)) {
            payload.amort.pagado = true;
        }
        else {
            payload.amort.pagado = false;
        }
        const params = {
            TableName: 'Creditos',
            Key: {
                id: idTabla
```

```
            },
            UpdateExpression: "SET cuotas = :cuotas, pagado = :pagado",
            ExpressionAttributeValues: {
                ":cuotas": cuotas,
                ":pagado": payload.amort.pagado
            }
        };
        this.DB.update(params, () => {
            callback(this.response(201, 'Guardado Correctamente'));
        });
    }
    check_pagado_all(lista) {
        let cuotas_pagadas = 0;
        lista.forEach((item) => {
            if (item.pagado) {
                cuotas_pagadas++;
            }
        });
        // reviso si las cuotas pagadas es igual a la cantidad de cuotas
        if (cuotas_pagadas === lista.length) {
            return true;
        }
        else {
            return false;
        }
    }
}
exports.default = Db;
```

Clases/ server.js

```
"use strict";
var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
var __importStar = (this && this.__importStar) || function (mod) {
    if (mod && mod.__esModule) return mod;
    var result = {};
    if (mod != null) for (var k in mod) if (Object.hasOwnProperty.call(mod, k))
result[k] = mod[k];
    result["default"] = mod;
    return result;
};
Object.defineProperty(exports, "__esModule", { value: true });
const express_1 = __importDefault(require("express"));
const environment_1 = require("../global/environment");
const socket_io_1 = __importDefault(require("socket.io"));
const socket = __importStar(require("../sockets/socket"));
const http_1 = __importDefault(require("http"));
require('win-ca');
class Server {
    constructor() {
        this.app = express_1.default();
        this.port = environment_1.SERVER_PORT;
        this.httpsServer = http_1.default.createServer(this.app);
```

```
        this.io = socket_io_1.default(this.httpsServer);
        this.escucharSockets();
    }
    static get instance() {
        return this._intance || (this._intance = new this());
    }
    start(callback) {
        this.httpsServer.listen(this.port, callback);
        // this.StartService();
    }
    escucharSockets() {
        console.log('Escuchando conexiones - sockets');
        this.io.on('connection', cliente => {
            console.log('Cliente conectado');
            /*socket webstore*/
            socket.Queries(cliente);
        });
    }
}
exports.default = Server;
```

Global/ environment.js

```
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
exports.SERVER_PORT = Number(process.env.PORT) || 45000;
```

Sockets/ sockets.js

```
"use strict";
var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
Object.defineProperty(exports, "__esModule", { value: true });
const db_1 = __importDefault(require("../clases/Databases/db"));
exports.Queries = (cliente, io) => {
    const DDB = db_1.default.instance;
    //=========================================================//
    // SIMULADOR CREDITO
    //=========================================================//
    // simulador_cred_server
    cliente.on('simulador_cred_server', (payload, callback) => {
        DDB.simulador_cred_server(payload, callback);
    });
    // save_credito_server
    cliente.on('save_credito_server', (payload, callback) => {
        DDB.save_credito_server(payload, callback);
    });
```

```
    //=========================================================//
    // MANEJO DE PRÉSTAMOS
    //=========================================================//
    // get_prestamos_server
    cliente.on('get_prestamos_server', (payload, callback) => {
        DDB.get_prestamos_server(payload, callback);
    });
    // pagar_cuota_server
    cliente.on('pagar_cuota_server', (payload, callback) => {
        DDB.pagar_cuota_server(payload, callback);
    });
    //=========================================================//
    // MANEJAR DESCONEXION
    //=========================================================//
    cliente.on('disconnect', () => {
        console.log('Cliente Desconectado');
    });
};
```

Index.js

```
"use strict";
var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
var __importStar = (this && this.__importStar) || function (mod) {
    if (mod && mod.__esModule) return mod;
    var result = {};
    if (mod != null) for (var k in mod) if (Object.hasOwnProperty.call(mod, k))
result[k] = mod[k];
    result["default"] = mod;
    return result;
};
Object.defineProperty(exports, "__esModule", { value: true });
const server_1 = __importDefault(require("./clases/server"));
const bodyParser = __importStar(require("body-parser"));
const server = server_1.default.instance;
// BodyParser
server.app.use(bodyParser.urlencoded({ extended: true }));
server.app.use(bodyParser.json());
server.start(() => {
    console.log(`servidor inicializado, puerto: ${server.port}`);
});
```

Package.json

```
{
  "name": "ExpressServerTS",
  "version": "1.0.0",
  "description": "tesis UCACUE server",
```

```
  "main": "index.js",
  "scripts": {
    "nodemon": "nodemon dist/",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Cristian Guillén",
  "license": "ISC",
  "dependencies": {
    "@types/jsonwebtoken": "^8.3.2",
    "@types/mysql": "^2.15.7",
    "@types/node": "^12.7.8",
    "@types/request": "^2.48.3",
    "@types/sqlanywhere": "^1.0.2",
    "angular-font-awesome": "^3.1.2",
    "aws-sdk": "^2.466.0",
    "body-parser": "^1.18.3",
    "cors": "^2.8.4",
    "express": "^4.16.4",
    "jsonwebtoken": "^8.5.1",
    "mysql": "^2.17.1",
    "request": "^2.88.0",
    "socket.io": "^2.2.0",
    "sqlanywhere": "^1.0.24",
    "uuid": "^3.3.3",
    "win-ca": "^3.1.0"
  },
  "devDependencies": {
    "@types/cors": "^2.8.4",
    "@types/express": "^4.16.0",
    "@types/socket.io": "^2.1.0",
    "nodemon": "^1.19.4"
  }
}
```

Ya con todo listo, solo queda ejecutar el comando **node dist/** dentro de la carpeta del proyecto para que el servidor se encuentre en funcionamiento y escuchando las peticiones desde el frontend. En la ilustración 17 se observa como el servidor se encuentra levantando de manera correcta, a la espera de solicitudes desde el cliente.

```
ubuntu@ip-172-31-7-192:~$ cd ExpressServer/
ubuntu@ip-172-31-7-192:~/ExpressServer$ node dist/
Escuchando conexiones - sockets
servidor inicializado, puerto: 45000
```

***Ilustración 17**. Servidor Express en funcionamiento.*

***Fuente**: Autor*

Configurado y levantado el servidor, y a su vez realizadas las pruebas de conexión con el cliente, el siguiente paso es realizar las pruebas necesarias aplicando los tres últimos principios de la metodología descrita en el documento, misma que será resumida en el siguiente capítulo.

Capítulo 4

Análisis de resultados

Una vez que se ha aplicado los cinco primeros principios de la metodología y se ha desarrollado por completo las pruebas de la aplicación, se completa con los ocho principios metodológicos con la evaluación de rendimiento de la aplicación en los entornos ya descritos. A continuación, se desarrolla los últimos tres principios metodológicos.

4.1 Evaluación estadística

Una vez ya descritos todos los principios previos al actual, a continuación, obtenemos los datos en gráficos estadísticos acorde a los microservicios, que sirven como soporte de estudio al objetivo general del presente documento. A continuación, podemos observar las estadísticas de medida de tiempo consumido por cada uno de los microservicios en AWS.

4.1.1 Estadísticas de la nube

Microservicio 1

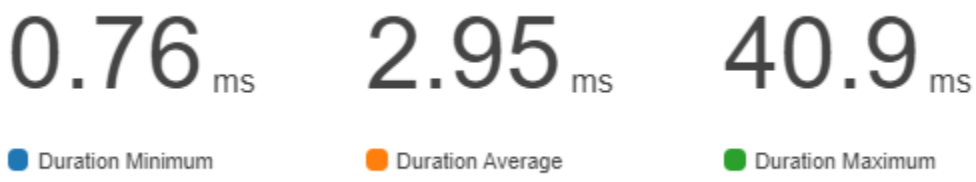

***Ilustración 18**. Estadísticas microservicio 1.*

***Fuente**: Autor*

En la ilustración 18, los detalles de monitoreo por parte de Amazon muestran un tiempo de ejecución mínimo de 0,76 ms, con un tiempo máximo de 40,9 ms, resumiéndose en un tiempo promedio de solicitud de 2,95 ms.

Microservicio 2

***Ilustración 19**. Estadísticas microservicio 2.*

***Fuente**: Autor*

En la ilustración 19, los detalles de monitoreo por parte de Amazon muestran un tiempo de ejecución mínimo de 32,6 ms, con un tiempo máximo de 1,36 s, resumiéndose en un tiempo promedio de solicitud de 269 s.

Microservicio 3

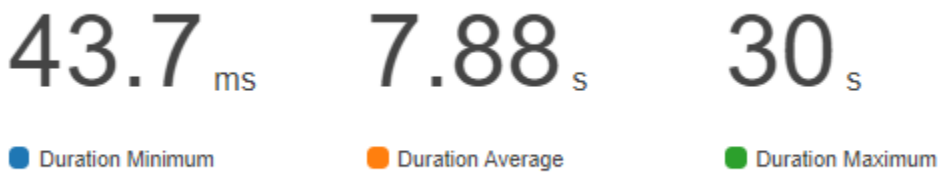

***Ilustración 20**. Estadísticas microservicio 3.*

***Fuente**: Autor*

En la ilustración 20, los detalles de monitoreo por parte de Amazon muestran un tiempo de ejecución mínimo de 43,7 ms, con un tiempo máximo de 30 s, resumiéndose en un tiempo promedio de solicitud de 7,88 s.

Microservicio 4

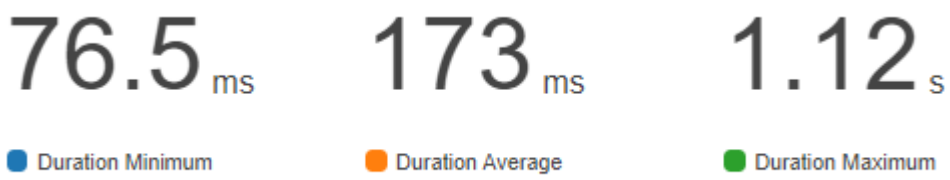

Ilustración 21. *Estadísticas microservicio 4.*

***Fuente**: Autor*

En la ilustración 21, los detalles de monitoreo por parte de Amazon muestran un tiempo de ejecución mínimo de 76,5 ms, con un tiempo máximo de 1,12 s, resumiéndose en un tiempo promedio de solicitud de 173 ms.

Para poder realizar la medición de rendimiento en el servidor, se procede a almacenar la diferencia de tiempo entre el inicio y fin de cada ejecución de función. Todos los datos se almacenaron en una base de datos para posteriormente poder graficarlos mediante Microsoft Excel. Los resultados obtenidos por lo tanto los incluimos con los resultados de AWS y lo condensamos en un solo gráfico por función.

4.1.2 Estadísticas del servidor

Microservicio 1

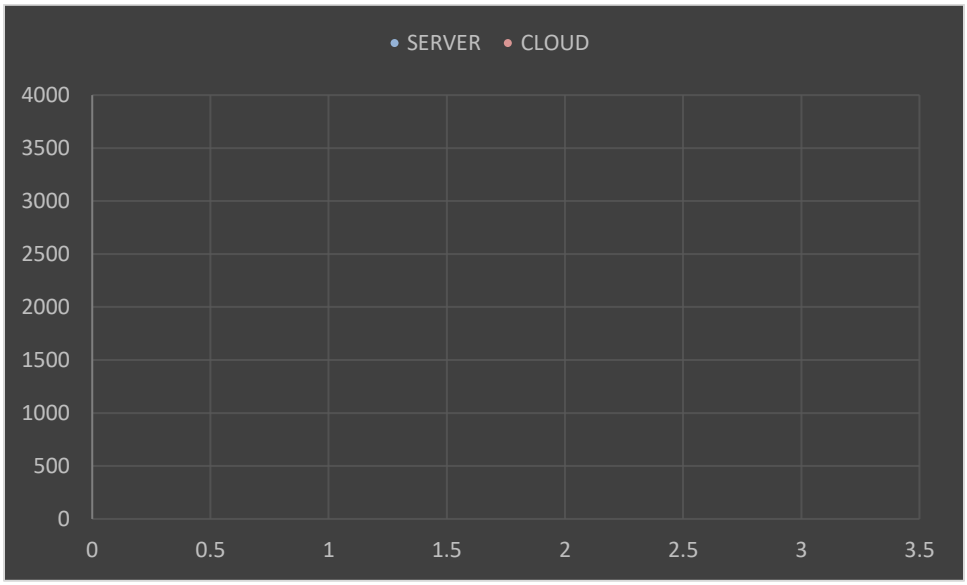

***Ilustración 22**. Resultados microservicio 1.*

***Fuente**: Autor*

En la ilustración 22, el gráfico muestra la diferencia entre el entorno servidor con el entorno en la nube, donde el servidor muestra una gran diferencia de respuesta a peticiones comparado con el tiempo en la nube, donde este segundo es mucho más rápido que el primero.

Microservicio 2

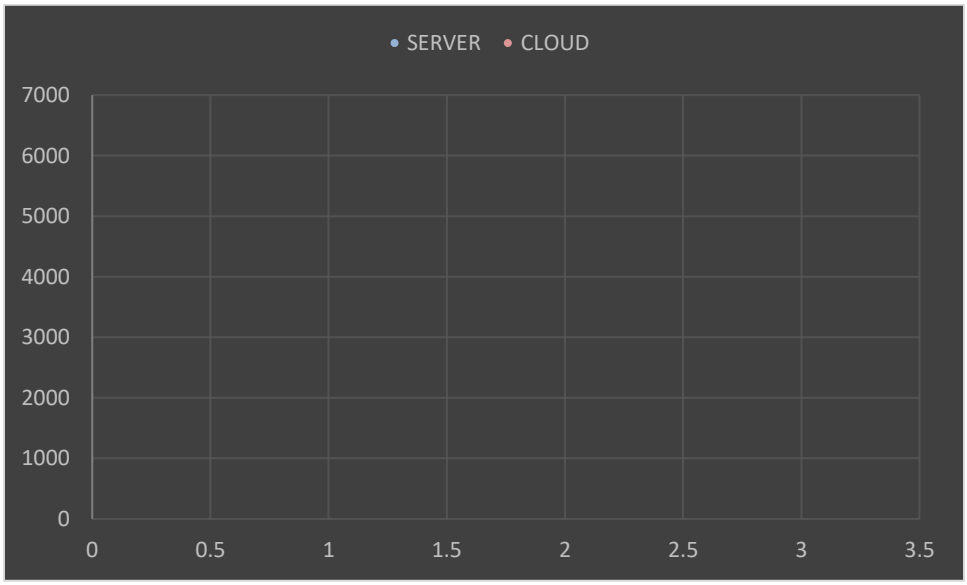

***Ilustración 23**. Resultados microservicio 2.*

***Fuente**: Autor*

En la ilustración 23, el gráfico muestra la diferencia entre el entorno servidor con el entorno en la nube, donde el servidor muestra una gran diferencia de respuesta a peticiones comparado con el tiempo en la nube, donde este segundo es mucho más rápido que el primero.

Microservicio 3

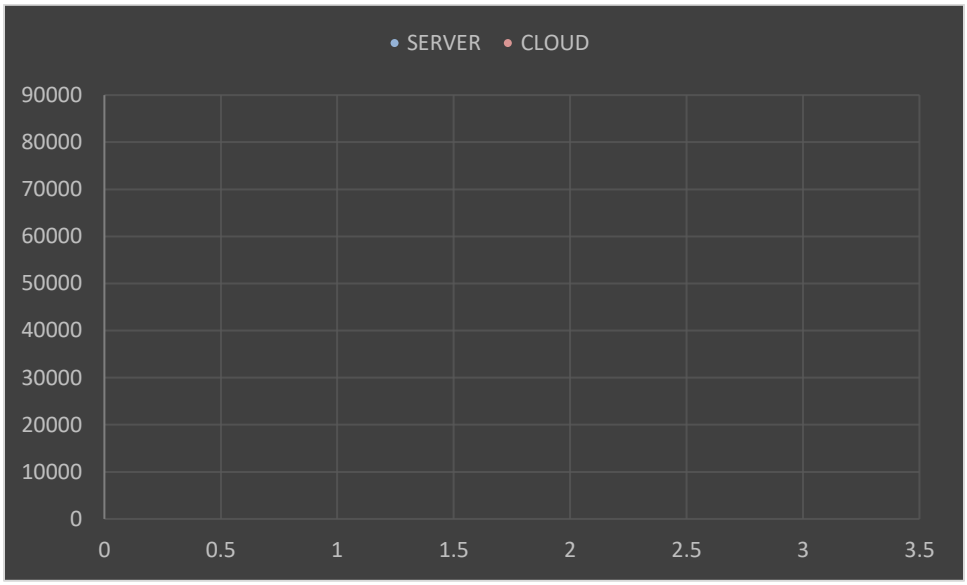

***Ilustración 24**. Resultados microservicio 3.*

***Fuente**: Autor*

En la ilustración 24, el gráfico muestra la diferencia entre el entorno servidor con el entorno en la nube, donde el servidor muestra una gran diferencia de respuesta a peticiones comparado con el tiempo en la nube, donde este segundo es mucho más rápido que el primero.

Microservicio 4

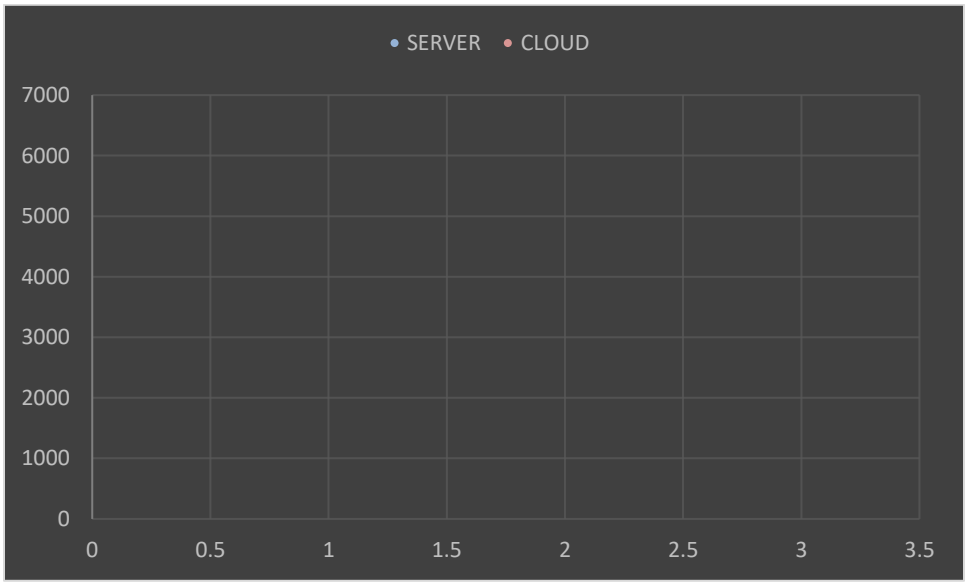

***Ilustración 25**. Resultados microservicio 4.*

***Fuente**: Autor*

En la ilustración 25, el gráfico muestra la diferencia entre el entorno servidor con el entorno en la nube, donde el servidor muestra una gran diferencia de respuesta a peticiones comparado con el tiempo en la nube, donde este segundo es mucho más rápido que el primero.

Como se observa en todos los gráficos, las funciones en la nube llevan una ventaja en lo que refiere a demora de ejecución de una función. En realidad, los resultados son por diferencias muy abultadas entre ellas, pero tenemos un factor que afecta a la ejecución que es el servidor en la nube. Explicando de una manera más detallada, los saltos que realiza el entorno servidor añaden tiempo en ejecución y resolución de la función ya que, primero, por medio de la comunicación con el socket llegan a un archivo de manejo de solicitudes del servidor, después a un archivo encargado

de resolver esas peticiones y a su vez desde ahí consumir el servicio de base de datos de AWS DynamoDB, siguiendo la misma ruta de regreso hasta el servidor; mientras que en el entorno de la nube, solo se realiza una petición HTTP por medio de una API y obtenemos una respuesta, por lo que la ruta que sigue la petición es más corta entre los dos puntos de comunicación.

4.2 Unidades de medida

Las unidades de medida bajo las cuales van a ser representados los datos son los siguientes: milisegundos (ms) para la duración de ejecución de las solicitudes y centavos de dólar (cents) para medir el costo de consumo de red para cada solicitud realizada por el software.

4.3 Costo

En cuanto al costo de implementación de los servicios en la nube, como se describe en el marco teórico de la infraestructura en la nube, el costo del servicio escogido fue la capa gratuita para siempre, por lo que el costo de implementación fue prácticamente de cero. Si lo comparamos con el desarrollo en el entorno servidor, solo con el costo de equipos de cómputo ya marca una diferencia en cuanto a la implementación de esta tecnología.

Capítulo 5

Conclusiones y recomendaciones

5.1 Conclusiones

Durante el desarrollo del marco teórico se da a conocer los conceptos básicos de cada una de las tecnologías a aplicar, apoyadas por la bibliografía referentes a cada una, por lo que una vez realizado el trabajo investigativo, se llega a la conclusión de utilizar Angular 2 para el desarrollo de interfaz de cliente de la aplicación web, así como el servidor desarrollado en NodeJS debido a la perfecta integración de lenguaje de desarrollo con la aplicación, permitiendo desarrollar tanto cliente como servidor en Javascript, mismo lenguaje que es también aplicado para el desarrollo de las funciones de AWS Lambda, acelerando considerablemente el desarrollo ya que no hay necesidad de utilizar un lenguaje diferente por tecnología, además de ser un lenguaje muy liviano y potente.

En cuanto a la aplicación de la metodología, gracias al aporte investigativo de Papadopolous y varios, podemos aplicar su teoría en el trabajo para poder llevarlo de una manera más organizada, correcta y que nos permite a su vez un mejor resultado al momento de realizar el caso de estudio.

Una vez aplicados los principios definidos en la metodología de investigación y con los resultados obtenidos, podemos concluir que la implementación de una tecnología *serverless* para pequeñas y medianas empresas es algo muy factible, los datos de respuesta respaldan a los tiempos de ejecución de solicitudes por lo que es una opción viable para implementación dentro de una empresa nueva o existente. Como punto

negativo se puede destacar que únicamente podemos desarrollar funciones dentro de los lenguajes de programación admitidos por AWS, siendo estos Java, Python, Go y Node.js; adicional, deben estar siempre actualizados ya que AWS progresivamente va descontinuando versiones antiguas de los lenguajes, pudiendo complicar lo desarrollado en un futuro.

Finalmente, acorde a los datos recolectados durante la experimentación, dentro de la capa gratuita de AWS, los costos prácticamente son nulos para la implementación de las funcionalidades al software, adicional, los tiempos de respuesta de ejecución reducen en más de un 50% la solución de las peticiones, variando estos valores acorde a la complejidad de ejecución de las funciones.

5.2 Recomendaciones

Para la implementación de una nueva tecnología, se debe tener en cuenta una correcta conexión a Internet para poder explotar al máximo las capacidades de la infraestructura en la nube y al final no tener problemas de retraso en las peticiones. Además, una correcta implementación de los lenguajes de programación, manteniéndolos siempre actualizados ya que por otra parte no se necesita realizar un mantenimiento de equipos.

Además, para iniciar con el desarrollo en la nube, se recomienda utilizar la capa gratuita por siempre de Amazon, que proporciona un costo significativamente menor al de la adquisición de equipos para un entorno de desarrollo de servidor por no decir completamente de cero en su implementación.

La aplicación de la computación en la nube abre nuevas oportunidades de desarrollo, como ya se describió en el marco teórico, su aplicación en varios campos muestra el potencial que tiene como una nueva infraestructura de desarrollo, no solo podemos optar por AWS, sino también probar con diferentes servicios en la nube y observar cuál se acopla mejor a los conocimientos de desarrollo y necesidades de los proyectos.

Lista de referencias

Almorsy, M., Grundy, J., & Müller, I. (2016). *An Analysis of the Cloud Computing Security Problem*. http://arxiv.org/abs/1609.01107

Amazon. (n.d.). *Amazon IAM*. Retrieved December 27, 2019, from https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/introduction.html

Ávila Mejía, Ó. (2013). Computaciòn en la nube. *La Propiedad Inmaterial*, *1*, 223–245.

AWS. (2018). *Amazon Lightsail*. https://aws.amazon.com/es/lightsail/

AWS. (2019). *Bases de datos no relacionales | Bases de datos de gráficos | AWS*. https://aws.amazon.com/es/nosql/

AWS | Lambda - Gestión de recursos informáticos. (n.d.). Retrieved February 2, 2020, from https://aws.amazon.com/es/lambda/

AWS vs Azure vs Google vs IBM vs Oracle vs Alibaba | A detailed comparison and mapping between various cloud services. (n.d.). Retrieved September 27, 2019, from http://comparecloud.in/

Botta, A., De Donato, W., Persico, V., & Pescapé, A. (2016). Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, *56*, 684–700. https://doi.org/10.1016/j.future.2015.09.021

Cloud Computing in Bioinformatics: current solutions and challenges. (2016). https://doi.org/10.7287/peerj.preprints.2261v1

De Giusti, L., Chichizola, F., Rodríguez, S., Sánchez, M., Paniego, J., & De Giusti, A. (n.d.). *Introduciendo conceptos de Cloud Computing utilizando el entorno CMRE*.

Edwards, E. (1964). On the theory of scales of measurement. In *Ergonomics* (Vol. 7,

Issue 4, pp. 504–505). https://doi.org/10.1080/00140136408956259

Feitelson, D. G. (2006). Experimental computer science: The need for a cultural change. *Internet Version: Http://Www. Cs. Huji. Ac. Il/~ Feit/Papers/Exp05. Pdf*, 1–37.

Gachet, D., De Buenaga, M., Aparicio, F., & Padrón, V. (2012). Integrating internet of things and cloud computing for health services provisioning: The virtual cloud carer project. *Proceedings - 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012*, 918–921. https://doi.org/10.1109/IMIS.2012.25

Gil-Mediavilla, M., Sánchez, A., Segura, A., & García de Vicuña, O. (2015). *Cloud computing en entornos educativos online. Análisis de experiencia en la asignatura 'Trabajo Fin de Grado' de la Universidad Isabel I.* https://www.redalyc.org/articulo.oa?id=31048902037

Google. (n.d.). *Angular*. https://angular.io/

Google Developers. (2019). *App Engine | Google Cloud*. https://cloud.google.com/appengine/

Hernández, R. (n.d.). *Ventajas y retos de la computación en la nube en empresas de Nuevo Laredo, Tamaulipas, México | Roque Hernández | Red Internacional de Investigadores en Competitividad*. Retrieved February 2, 2020, from https://riico.net/index.php/riico/article/view/1168

Iosup, A., & Epema, D. (2011). Grid computing workloads. *IEEE Internet Computing*, *15*(2), 19–26. https://doi.org/10.1109/MIC.2010.130

Kumanov, D., Hung, L.-H., Lloyd, W., & Yeung, K. Y. (2018). *Serverless computing*

provides on-demand high performance computing for biomedical research. http://arxiv.org/abs/1807.11659

Lakew, E. B., Papadopoulos, A. V., Maggio, M., Klein, C., & Elmroth, E. (2017). KPI-Agnostic Control for Fine-Grained Vertical Elasticity. *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, 589–598. https://doi.org/10.1109/CCGRID.2017.71

Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., & Pallickara, S. (2018). Serverless computing: An investigation of factors influencing microservice performance. *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*, 159–169. https://doi.org/10.1109/IC2E.2018.00039

Lynn, T., Rosati, P., Lejeune, A., & Emeakaroha, V. (2017). A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, *2017-December*, 162–169. https://doi.org/10.1109/CloudCom.2017.15

Marini, E. (n.d.). *El modelo cliente/servidor*. www.linuxito.com

Martin, D., & E, V. D. (2012). *CLOUD COMPUTING: DE LA VIRTUALIZACIÓN DE APLICACIONES Y DE ESCRITORIO, A LA VIRTUALIZACIÓN DE SERVIDORES. Ing. Edgar Gutiérrez*. 1–5. http://www.itcelaya.edu.mx/ojs/index.php/pistas/article/view/289

Mcafee, A., & Brynjolfsson, E. (2012). *HBR.ORG Spotlight on Big Data Big Data: The Management Revolution*.

Microsoft. (2018). *Qué es Azure: Servicios en la nube de Microsoft | Microsoft Azure*. Microsoft. https://azure.microsoft.com/es-es/overview/what-is-azure/

Molina Ríos, J. R., Zea Ordóñez, M. P., Contento Segarra, M. J., & García Zerda, F. G. (2017). ESTADO DEL ARTE: METODOLOGÍAS DE DESARROLLO EN APLICACIONES WEB. *3C Tecnología_Glosas de Innovación Aplicadas a La Pyme*, *6*(3), 54–71. https://doi.org/10.17993/3ctecno.2017.v6n3e23.54-71

Papadopoulos, A. V., Versluis, L., Bauer, A., Herbst, N., Von Kistowski, J., Ali-eldin, A., Abad, C., Amaral, J. N., Tuma, P., & Iosup, A. (2019). Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering*, 1–1. https://doi.org/10.1109/TSE.2019.2927908

Paz, R. L. G. La. (2015). *Desarrollo de aplicaciones web en el entorno servidor. IFCD0210*. https://books.google.es/books?hl=es&lr=&id=OO91CQAAQBAJ&oi=fnd&pg=PT5&dq=Desarrollo+de+aplicaciones+web+en+el+entorno+servidor&ots=vOkiRJQJr2&sig=qiYhEebFGYWSMdonVvJqpmLb-ZY#v=onepage&q=Desarrollo de aplicaciones web en el entorno servidor&f=false

Precios de los planes de AWS Support | A partir de 29 USD al mes | AWS Support. (n.d.). Retrieved February 2, 2020, from https://aws.amazon.com/es/premiumsupport/pricing/

Ramos, J., & Ramos, A. (2014). *Aplicaciones Web - Alicia Ramos Martín, Maria Jesus Ramos Martín - Google Libros*. https://books.google.es/books?hl=es&lr=&id=43G6AwAAQBAJ&oi=fnd&pg=PA1

&dq=Aplicaciones+Web+ramos+martin&ots=Dgb6q3y3IJ&sig=0fY3a5Sm1o_KGktNdXogLBkaVG0#v=onepage&q=Aplicaciones Web ramos martin&f=false

Rittinghouse, J. W., & Ransome, J. F. (2017). *CLOUD COMPUTING: Implementation, Management, and Security*. http://www.webcitation.org/5ntFJvrGR

Services, A. W. (2019). *¿Qué es AWS?* https://aws.amazon.com/es/what-is-aws/?nc1=f_cc

Servicios de computación en la nube | Google Cloud | Google Cloud. (n.d.). Retrieved February 2, 2020, from https://cloud.google.com/

Servidores - Productos. (n.d.). Retrieved February 2, 2020, from http://www.tecnosmart.com.ec/v2/productos/servidores

Sistemas de interacción persona-computador - José Bravo Rodríguez - Google Libros. (n.d.). Retrieved February 10, 2020, from https://books.google.es/books?hl=es&lr=&id=V6a0l-JbRX8C&oi=fnd&pg=PA15&dq=Sistemas+de+interacción+persona-computador+ortega+cantero&ots=zYUTV7E9J6&sig=IQ0WPCBaFELFUiL_EvS1mBCQhQk#v=onepage&q=Sistemas de interacción persona-computador ortega cantero&f=false

SRI. (2018). *Contribuyentes obligados a emitir comprobantes electrónicos - Servicio de Rentas Internas del Ecuador*. https://www.sri.gob.ec/web/guest/contribuyentes-obligados-a-emitir-comprobantes-electronicos

Stack Overflow Developer Survey 2018. (n.d.). Retrieved September 26, 2019, from https://insights.stackoverflow.com/survey/2018

StackOverflow. (2019). *Stack Overflow Developer Survey 2019*. Stack Overflow Insights. https://insights.stackoverflow.com/survey/2019

US10298656B2 - Extending representational state transfer application program interface (REST API) functionality - Google Patents. (n.d.). Retrieved February 2, 2020, from https://patents.google.com/patent/US10298656B2/en

Van Eyk, E., Iosup, A., Grohmann, J., Eismann, S., Bauer, A., Versluis, L., Toader, L., Schmitt, N., Herbst, N., & Abad, C. (2019). The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms. *IEEE Internet Computing*. https://doi.org/10.1109/MIC.2019.2952061

Warf, B. (2018). Hypertext Transfer Protocol. In *The SAGE Encyclopedia of the Internet*. https://doi.org/10.4135/9781473960367.n128

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, *1*(1), 7–18. https://doi.org/10.1007/s13174-010-0007-6

ANEXOS

TESIS CRISTIAN V1

INFORME DE ORIGINALIDAD

2%	2%	0%	1%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	aws.amazon.com Fuente de Internet	<1%
2	www.ssbg.com.cn Fuente de Internet	<1%
3	patents.google.com Fuente de Internet	<1%
4	trid.trb.org Fuente de Internet	<1%
5	www.freecodecamp.org Fuente de Internet	<1%
6	www.credencys.com Fuente de Internet	<1%
7	www.academiacarceller.net Fuente de Internet	<1%
8	www.vtc.com Fuente de Internet	<1%
9	en.wikipedia.org Fuente de Internet	<1%

10	Bijeta Seth, Surjeet Dalal, Raman Kumar. "Chapter 8 Securing Bioinformatics Cloud for Big Data: Budding Buzzword or a Glance of the Future", Springer Science and Business Media LLC, 2019 Publicación	<1%
11	Submitted to Excelsior College Trabajo del estudiante	<1%
12	www.bartleby.com Fuente de Internet	<1%
13	www.turismoaventura.com Fuente de Internet	<1%
14	www.idemsoft.com Fuente de Internet	<1%
15	psicopedagogia.com Fuente de Internet	<1%
16	Submitted to Universidad Internacional de la Rioja Trabajo del estudiante	<1%
17	treaties.un.org Fuente de Internet	<1%
18	www.fegor.com Fuente de Internet	<1%
19	www.tecnomarkets.com	

	Fuente de Internet	<1%
20	www.geocities.com Fuente de Internet	<1%
21	Submitted to Universidad Estatal de Milagro Trabajo del estudiante	<1%
22	Submitted to Infile Trabajo del estudiante	<1%
23	Submitted to Uniagustiniana Trabajo del estudiante	<1%
24	Submitted to Mississippi State University Trabajo del estudiante	<1%
25	www.prefabricadosaljema.com Fuente de Internet	<1%
26	Submitted to Universidad Catolica De Cuenca Trabajo del estudiante	<1%
27	ciberaula.com Fuente de Internet	<1%

Excluir citas	Apagado	Excluir coincidencias	Apagado
Excluir bibliografía	Activo		

PERMISO DEL AUTOR DE TESIS PARA SUBIR AL REPOSITORIO INSTITUCIONAL

Yo, CRISTIAN PAUL GUILLEN PARRA, portador/a de la cédula de ciudadanía Nro.,0302321880 En calidad de autor/a y titular de los derechos patrimoniales del trabajo de titulación **"PROGRAMACIÓN EN LA NUBE COMO ALTERNATIVA A LA PROGRAMACIÓN DE APLICACIONES WEB TRADICIONALES"** de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de Los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos, Así mismo; autorizo a la Universidad para que realice la publicación de éste trabajo de titulación en Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Azogues, 05 de marzo de 2020

F: .....

CRISTIAN PAUL GUILLEN PARRA

0302321880

El Bibliotecario de la Sede Azogues

CERTIFICA:

Que: **GUILLEN PARRA CRISTIAN PAUL,** con cédula de ciudadanía Nro. **0302321880,** de la Carrera de: **INGENIERIA EN SISTEMAS**

No adeuda libros, a esta fecha: **4 de marzo del 2020**.

Byron Alonso Torres Romo
Bibliotecario

Biblioteca Universitaria
MONS. "FROILAN POZO QUEVEDO"

CENTRO DE IDIOMAS

ABSTRACT

Author: Cristian Guillén.

This research work analyzes how the implementation of a new web development infrastructure, it means cloud programming, is possible through the development of a case study, which is optimized to work both in a server environment and in a development environment in the cloud, to respond to a possible economic alternative versus the traditional one. To response the time of both development environments, it is proposed the methodology conducted by Papadopolous, it was optimized to encapsulate the test environment and compare the performance in both technologies. Chapter 1 describes the problem to be solved during this work, as well as theoretical bases that support a possible solution, which is summarized as the main objective of the document. Chapter 2 lays the foundations of the theory that describe the current context on web development, as well as the methodology to be used to propose a solution to the problem. Chapter 3 applies the methodology described in the previous chapter, with this, it was carried out the development of the application. Then, in chapter 4 and with all the methodology applied, the results obtained during the experimentation in chapter 3 are analyzed. Finally, in chapter 5, it is presented the conclusions and recommendations and result obtained.

KEYWORDS: CLOUD PROGRAMMING, MONOLITHIC ENVIRONMENTS, WEB DEVELOPMENT, MICRO SERVICES.

Azogues, 10 de marzo del 2020

EL CENTRO DE IDIOMAS DE LA UNIVERSIDAD CATÓLICA DE CUENCA, CERTIFICA QUE EL DOCUMENTO QUE ANTECEDE FUE TRADUCIDO POR PERSONAL DEL CENTRO PARA LO CUAL DOY FE Y SUSCRIBO

Abg. Liliana Urgilés Amoroso, Esp.

COORDINADORA CENTRO DE IDIOMAS AZOGUES