Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

CÓMO APLICAR LA METODOLOGÍA DE DOCE FACTORES EN LA CREACIÓN DE UNA APLICACIÓN MODERNA PARA LA WEB UTILIZANDO ANGULAR, FLASK Y MARIADB

José Carlos Véliz Castro

Asesorado por el Ing. Ricardo Enrique Ibarra Cabrera

Guatemala, octubre de 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

CÓMO APLICAR LA METODOLOGÍA DE DOCE FACTORES EN LA CREACIÓN DE UNA APLICACIÓN MODERNA PARA LA WEB UTILIZANDO ANGULAR, FLASK Y MARIADB

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JOSÉ CARLOS VÉLIZ CASTRO
ASESORADO POR EL ING. RICARDO ENRIQUE IBARRA CABRERA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martinez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Kevin Vladimir Armando Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADORA	Inga. Virginia Victoria Tala Ayerdi
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Ing. Luis Fernando Espino Barrios
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

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

CÓMO APLICAR LA METODOLOGÍA DE DOCE FACTORES EN LA CREACIÓN DE UNA APLICACIÓN MODERNA PARA LA WEB UTILIZANDO ANGULAR, FLASK Y MARIADB

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería de Ciencias y Sistemas, con fecha agosto de 2021.

José Carlos Véliz Castro

Guatemala, 01 de agosto de 2022

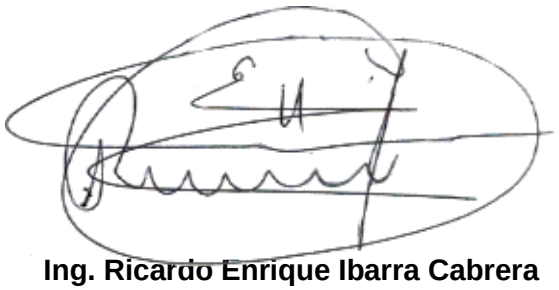
Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Trabajos de Tesis
Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería - USAC

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **José Carlos Véliz Castro** con carné **201700343** y CUI **3054 41272 0207** titulado **"Cómo aplicar la metodología de doce factores en la creación de una aplicación moderna para la web utilizando Angular, Flask y MariaDB",** lo he revisado y luego de corroborar que el mismo se encuentra concluido y que cumple con los objetivos propuestos en el respectivo protocolo, procedo a la aprobación respectiva.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Ing. Ricardo Enrique Ibarra Cabrera
Colegiado No. 13655

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 11 de agosto de 2022

Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **JOSÉ CARLOS VÉLIZ CASTRO** con carné **201700343** y CUI **3054 41272 0207** titulado **"CÓMO APLICAR LA METODOLOGÍA DE DOCE FACTORES EN LA CREACIÓN DE UNA APLICACIÓN MODERNA PARA LA WEB UTILIZANDO ANGULAR, FLASK Y MARIADB"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

LNG.DIRECTOR.196.EICCSS.2022

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **CÓMO APLICAR LA METODOLOGÍA DE DOCE FACTORES EN LA CREACIÓN DE UNA APLICACIÓN MODERNA PARA LA WEB UTILIZANDO ANGULAR, FLASK Y MARIADB**, presentado por: **José Carlos Véliz Castro**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

"ID Y ENSEÑAD A TODOS"

Msc. Ing. Carlos Gustavo Alonzo
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, septiembre de 2022

Decanato
Facultad de Ingeniería
24189101- 24189102
secretariadecanato@ingenieria.usac.edu.gt

LNG.DECANATO.OI.664.2022

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **CÓMO APLICAR LA METODOLOGÍA DE DOCE FACTORES EN LA CREACIÓN DE UNA APLICACIÓN MODERNA PARA LA WEB UTILIZANDO ANGULAR, FLASK Y MARIADB,** presentado por: **José Carlos Véliz Castro,** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANA
FACULTAD DE INGENIERÍA

Inga. Aurelia Anabela Cordova Estrada
Decana

Guatemala, octubre de 2022

AACE/gaoc

ACTO QUE DEDICO A:

Evolución	Por los millones de años de evolución humana necesarios para brindarme el conocimiento suficiente para cumplir esta maravillosa meta.
Mi madre	Hermelinda Castro Alvarado, por su amor incondicional y su sacrificio incomparable, por estar siempre por mí y ayudarme a conseguir esta meta.
Mis hermanos	Laura y José Véliz, por ser un fuerte apoyo y motivación personal, ya que gracias a ellos no abandone mi meta y logre cumplirla.
Mi familia	Familia Castro Alvarado, por su apoyo incondicional cuando lo necesite.

AGRADECIMIENTOS A:

Mi madre	Hermelinda Castro Alvarado, por su apoyo incondicional, ya que sin ella no hubiese logrado alcanzar mis metas.
Mis hermanos	Laura y José Véliz, por su cariño y motivación para conseguir lo que me había planteado.
Mi asesor	El ingeniero Ricardo Ibarra por su paciencia y apoyo para completar esta etapa en mi vida.
Mis amigos	Victor Aguilar, Javier Golon López, Javier Monterroso Lopez, Andhy Solís, Marco Lopez y Heidy Miranda por acompañarme durante esta etapa de mi vida y apoyarme cuando lo necesite.
Universidad de San Carlos de Guatemala	Por ser la casa de estudios que me brindo la oportunidad de convertirme en un profesional y en una persona más consciente.
Facultad de Ingeniería	Por brindarme la oportunidad de convertirme en un ingeniero y regalarme maravillosos momentos con las enseñanzas aprendidas.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
yml	Extensión de un archivo en formato YAML.

GLOSARIO

Angular	*Framework* de código abierto para la creación y mantenimiento de aplicaciones web desarrollado en TypeScript.
AWS	Plataforma de servicios en la nube, que ofrece la posibilidad de adquirir desde potencia de cómputo hasta el almacenamiento de información de diversas maneras.
Back-end	Componente del desarrollo web encargado del manejo de la lógica de la aplicación.
Build	Paquete ejecutable creado a partir de la transformación del código fuente.
CACIF	Comité coordinador de asociaciones agrícolas, comerciales, industriales y financieras del país de Guatemala.
CI/CD	Método para la distribución continua de aplicaciones mediante el uso de la automatización en las etapas del desarrollo.

Cloud native	Enfoque en el desarrollo de software que utiliza la computación en la nube para construir aplicaciones escalables en entornos modernos.
Commit	Copia instantánea de un proyecto almacenada en un sistema gestor de versiones.
Crash-only	Diseño de software enfocado en tratar los fallos de manera segura y la recuperación de este de manera rápida. La única manera de detener un programa bajo este diseño es por un fallo y la única manera de iniciarlo es por una recuperación.
Daemons	Procesos que se ejecutan en segundo plano en lugar de ser controlado directamente por el usuario.
Docker	Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software.
Docker Hub	Repositorio público en la nube que permite la extracción, almacenamiento y envío de imágenes de aplicaciones contenerizadas.
Docker-compose	Herramienta para la definición y ejecución de aplicaciones Docker de múltiples contenedores.
DORA	Programa de investigación perteneciente a Google, que busca comprender las practicas, procesos y

	capacidades que permiten a los equipos de trabajo conseguir un alto desempeño del software.
Extreme programming	Marco de desarrollo de software ágil que tiene como objetivo producir software de mayor calidad para mejorar la eficiencia del equipo de desarrollo.
Flask	*Framework* minimalista escrito en Python que permite crear aplicaciones web rápidamente y utilizando la menor cantidad de líneas de código.
Framework	Esquema o marco de trabajo que ofrece una estructura base para la elaboración de un proyecto con objetivos específicos.
Front-end	Componente del desarrollo web encargado de la interfaz gráfica en la cual un usuario puede ver o interactuar con la información.
Git	Software de control de versiones enfocado en la eficiencia, confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones.
GitHub	Plataforma de desarrollo colaborativo en la nube para alojar proyectos, utilizando el sistema de control de versiones Git.

GitHub Pages	Servicio de alojamiento de sitios web estáticos que utiliza los archivos web directamente desde un repositorio en GitHub.
GitLab	Plataforma Git y DevOps basada en la nube que permite supervisar, probar y desplegar software.
Gitflow	Flujo de trabajo basado en Git que brinda un mayor control y organización sobre un repositorio.
Google Cloud	Conjunto de recursos físicos, como computadoras y unidades de disco duro, y recursos virtuales, como máquinas virtuales, alojadas en los centros de datos de Google.
Grafana	Software libre que permite la visualización y el formato de datos métricos de sistemas computacionales.
Heroku	Plataforma como servicio que permite la construcción, ejecución y operación de aplicaciones enteramente en la nube.
HTML	Protocolo de transferencia de hipertexto.
Java	Lenguaje de programación orientado a objetos que permite la construcción de aplicaciones y sistemas.

JavaScript	Lenguaje de programación utilizado para el desarrollo web, funcionando en el lado del servidor y a su vez al lado de la interfaz gráfica de la aplicación.
Jenkins	Herramienta de código abierto para la automatización de procesos, permitiendo conseguir la integración y entrega continua de software.
Kanban	Sistema visual para el manejo del flujo de trabajo a nivel individual, por grupos u organizacional, permitiendo un desarrollo ágil de un proyecto.
Kubernetes	Sistema de código libre para la automatización del despliegue y manejo de aplicaciones en contenedores, diseñado por Google.
Linux	Sistema operativo de código abierto basado en Unix, desarrollado por Linus Torvalds.
Log	Grabación secuencial en un archivo o en una base de datos de todos los acontecimientos que afectan a un proceso particular.
MariaDB	Sistema de gestión de bases de datos de código abierto, desarrollado por Michael Widenius.
Mercurial	Sistema de control de versiones multiplataforma, elaborado utilizando el lenguaje de programación Python.

Microsoft TFS	Conjunto de herramientas de desarrollo de software colaborativo, alojada localmente. Actualmente se le conoce como Azure DevOps Server.
Nginx	Servidor web ligero de alto rendimiento, que además es software libre y código abierto.
NoSQL	Es el conjunto de tecnologías de bases de datos no relacionales.
NPM	Sistema de gestión de paquetes utilizado por uno de los entornos de ejecución de JavaScript.
Petabyte	Unidad de almacenamiento de información cuyo símbolo es PB y equivales a 1,000,000,000,000,000 bytes.
Pip	Sistema de gestión de paquetes, utilizado para la instalación y administración de paquetes de software escritos en Python.
Prometheus	Software de código abierto utilizado para la supervisión y alerta de eventos.
Python	Lenguaje de programación de alto nivel interpretado, cuya filosofía se basa en la legibilidad del código, utilizado para la creación de múltiples tipos de aplicaciones.

Runtime	Intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.
SCRUM	Metodología de desarrollo ágil, que consiste en un conjunto de buenas prácticas para el trabajo colaborativo buscando conseguir el mejor resultado dentro de los proyectos en donde se implementa.
Share-nothing	Arquitectura distribuida en el que cada nodo es independiente y autosuficiente.
SQL	Lenguaje de consulta estructurada utilizado para el manejo de conjunto de datos y las relaciones existentes entre ellos.
Stack	Combinación de herramientas aplicaciones y servicios utilizadas para construir una aplicación.
Subversion	Herramienta de control de versiones de código abierto basada en un repositorio y cuyo funcionamiento se asemeja a un sistema de ficheros.
The twelve-factor app	Metodología de desarrollo de software enfocada en aplicaciones de software como servicio que busca la creación de programas de alto rendimiento y resilientes antes los problemas.
TortoiseSVN	Cliente de Subversion gratuito, implementado como una extensión de la consola de Windows.

TypeScript	Lenguaje de programación libre y de código abierto desarrollado por Microsoft, creado a partir de Javascript con un tipado fuertemente establecido.
URL	Dirección única y especifica que se asigna a cada uno de los recursos disponibles en internet.
Virtualenv	Herramienta que se utiliza para crear entornos Python de manera aislada.

RESUMEN

Se tiene como propósito crear una aplicación sencilla pero que cumpla con la metodología de doce factores, dicha metodología garantiza que las aplicaciones sean altamente resilientes y también permite que un equipo de desarrollo se mueva rápidamente. Se planea explicar la manera en que se puede conseguir que se cumplan los requisitos que los doce factores proponen, utilizando tecnologías y herramientas actualmente vigentes.

Como resultado de la investigación se tiene como propósito crear un conjunto de videotutoriales que explican los doce factores y la forma en que se implementan en la aplicación que se estará trabajando, utilizando un *stack* tecnológico conformado por Angular, Flask (Python) y MariaDB.

OBJETIVOS

General

Demostrar a través de un videotutorial cómo se puede construir una aplicación web que cumpla con la metodología The twelve-factor app usando el *stack* Angular, Flask y MariaDB.

Específicos

1. Explorar el desarrollo de aplicaciones siguiendo la metodología The twelve-factor app.

2. Entender los beneficios y riesgos que involucra utilizar ciertas tecnologías siguiendo la metodología The twelve-factor app.

3. Generar contenido audiovisual que permita a las personas ejemplificar cómo usar las tecnologías para construir aplicaciones en esta forma.

4. Exponer públicamente el contenido audiovisual creado.

INTRODUCCIÓN

La metodología The twelve-factor app está escrita por distintos colaboradores, con vasta experiencia, que describen las prácticas ideales para el desarrollo de aplicaciones, tomando en cuenta el crecimiento a largo plazo, la dinámica de colaboración y la complejidad que puede llegar a tener el mantenimiento del software.

Los 12 factores se traducen en requerimientos no funcionales que debe cumplir la aplicación para dotarla con propiedades como la escalabilidad, portabilidad, resiliencia y agilidad.

Dado que cumplir con cada uno de estos requerimientos está estrechamente ligado a la tecnología y lenguajes que se utilicen para construir la aplicación, se debe de analizar la correcta manera de implementarlo, es por ello por lo que a continuación, se estará exponiendo cómo aplicar la metodología The twelve-factor app utilizando el *stack* tecnológico de Angular, Flask y MariaDB, que en conjunto a prácticas DevOps dará como resultado una aplicación moderna.

También parte de la investigación será desarrollar una serie de videotutoriales, los cuales describirán paso a paso la implementación de cada factor y además explicar cómo las tecnologías que se emplearán lograrán que se cumplan los factores mencionados.

Los videotutoriales serán publicados en una plataforma gratuita para que se encuentren al alcance del que esté interesado en construir una aplicación de este tipo.

1. FUNDAMENTOS DE LA INVESTIGACIÓN

La investigación posee una naturaleza cualitativa dado que se darán a conocer las características de las herramientas y tecnologías a estudiar, luego se utilizará la de mayor conveniencia según la solución que se plantee.

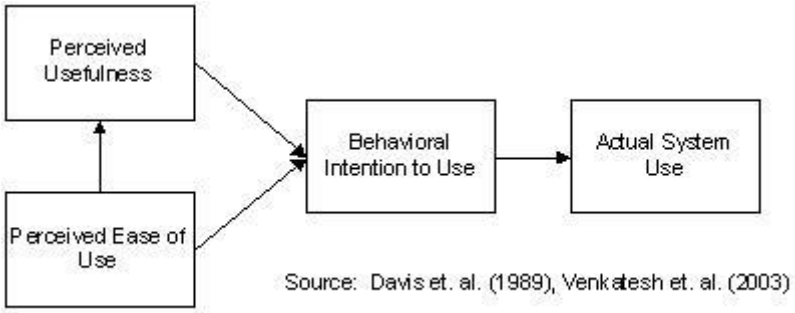
1.1. Teoría de la investigación

Para el presente trabajo de investigación se definió que se utilizará el Modelo de Aceptación de la Tecnología, el cual se identifica por las siglas TAM *(Technology Acceptance Model).*

1.1.1. TAM

El modelo de aceptación de la tecnología, postula que la utilidad y la facilidad de uso percibida determinan la intención de un individuo de utilizar un sistema, sirviendo la intención de uso como mediador del uso real del sistema.

Figura 1. **Modelo de aceptación de tecnología**



Fuente: Theorizeit (2020). *Technology acceptance model*. Consultado el 20 de agosto del 2021. Recuperado de https://is.theorizeit.org/w/images/9/90/Tam.JPG.

El TAM proporciona una base teórica para comprender y evaluar la| aceptación de los usuarios hacia las nuevas tecnologías, permitiendo desarrollar e implementar mejores sistemas.

1.1.2. Relación con la investigación

Uno de los objetivos de la presente investigación es introducir a estudiantes al estudio de la tendencia de construcción de aplicaciones *cloud native*, por lo que se pretende presentar las tecnologías disponibles en las que se pueden desarrollar este tipo de aplicaciones.

Por medio del contenido audiovisual generado, se tiene el objetivo de mostrar la manera en que se utilizan las herramientas y tecnologías definidas, en este caso, para implementar una aplicación utilizando Angular, Flask y MariaDB. Los doce factores sirven de senda para presentar la utilidad de las diferentes tecnologías disponibles, la adopción de una tecnología u otra dependerá de la utilidad y facilidad de uso que tengan los desarrolladores,

tomando en cuenta los requisitos del proyecto a desarrollar, un punto medio entre ambos factores permitirá la construcción de una aplicación efectiva y en un tiempo razonable.

1.2. Situación actual en la escuela de Ciencias y Sistemas de la Facultad de Ingeniería en la Universidad de San Carlos de Guatemala

De acuerdo con AWS (2020), el software ya no se limita a respaldar un negocio, sino que se convierte en componente integral de cada aspecto de este. Las compañías interactúan con sus clientes a través de software proporcionado como aplicaciones o servicios en línea en todo tipo de dispositivos. También utilizan software para incrementar la eficacia operativa al transformar cada sección de la cadena de valor, como la logística, la comunicación y las operaciones. Guatemala no es la excepción, acorde a la primera Encuesta Empresarial 2021 presentada por el CACIF reveló que el 69 % de las compañías consultadas han acelerado la digitalización de procesos para la automatización, por lo que adoptar una filosofía de DevOps dentro de la industria puede ser un factor diferencial en el desempeño de la empresa.

Durante el transcurso de la carrera de ciencias y sistemas se utilizaron algunas herramientas y practicas relacionadas con DevOps, sin embargo, en ningún momento se imparte de forma integrada cómo debe organizarse el ecosistema en el que se desarrolla un proyecto, aplicando los principios que DevOps propone. A continuación, se presentan conceptos que se han utilizado a lo largo de la carrera de ciencias y sistemas con la red de estudios vigente, así como también los cursos en los que se han impartido dichos temas y las tecnologías en que se han implementado:

Tabla I. **Conceptos asociados a DevOps**

Concepto	**Cursos**	**Tecnologías**
Control de Versiones	Estructuras de Datos, Compiladores, Análisis y Diseño de Sistemas, Sistemas Operativos	GitHub, GitLab, Microsoft TFS, TortoiseSVN
Metodologías Ágiles	Análisis y diseño de sistemas I y II	SCRUM, Extreme Programming, Kanban
Integración Continua / Entrega Continua	Análisis y diseño de sistemas I y II, Compiladores II	Jenkins, Heroku, GitHub Pages
Contenedores	Seminario de Sistemas	Docker
Orquestación	Sistemas Operativos I y II	Google Kubernetes Engine
Monitoreo	Sistemas Operativos I y II	Prometheus, Grafana

Fuente: elaboración propia, realizado con Microsoft Excel.

2. APLICACIONES MODERNAS PARA LA WEB

Acorde a un artículo publicado por Scanlan (2019), llamado *Defining The Modern Application*, una aplicación moderna se puede definir como un servicio de software resiliente y compatible con varias nubes, es decir, que pueda ser ejecutado sobre distintos proveedores de nube, y que está conformado por despliegues organizados de recursos como máquinas virtuales, contenedores o funciones sin servidor. Anteriormente las arquitecturas como la monolítica era muy utilizada y funcionaba correctamente en la mayoría de los casos de desarrollo web, sin embargo, de acuerdo con los datos de Digital 2021: Global Overview Report, únicamente en el 2021, la cantidad de usuarios de internet ha alcanzado los 4,660 millones, lo que representa un crecimiento del 7.3 % en comparación al año anterior que fue de 3,880 millones. El número de usuarios en 2021 equivale al 59.5 % de la población mundial, todo este crecimiento requiere un escalamiento que lo soporte, por lo que existe la necesidad de crear y desplegar servicios web rápidamente. En 2011, Adam Wiggins, junto con la experiencia y observaciones de otros colaboradores definieron doce factores que se recomienda cumplir para que una aplicación sea altamente resiliente y que les permita a los equipos de desarrollo desplegar rápidamente nuevos servicios o actualizaciones de estos.

2.1. Defectos en aplicaciones web actuales

Las aplicaciones web actuales presentan diferentes características que no benefician el crecimiento o la rapidez con la que se modifican, los doce factores plantean requerimientos no funcionales que agregan un valor extra a los proyectos que los implementan.

Según Khan (2017), existen problemas en los equipos de desarrollo, pero a su vez pueden llegar a ser resueltos con ayuda de la práctica constante.

El equipo de DORA (DevOps Research and Assessment) de Google Cloud ha llevado a cabo un programa de investigación y determinó que existen cuatro métricas que miden de forma efectiva el desempeño operativo y la entrega de software en una organización, las empresas más desarrolladas consistentemente han demostrado tener una mejor valuación en dichas métricas.

- Frecuencia de despliegue: se refiere a la frecuencia con la que se despliega el código a producción.
- Tiempo de espera para los cambios: define el tiempo que se tarda en pasar de la confirmación del código a su ejecución en producción.
- Tiempo medio de recuperación: cuánto tiempo se tarda en recuperarse de los fallos y las interrupciones no planificadas.
- Tasa de fallos en los cambios: se refiere a qué porcentaje de los cambios requiere una corrección posterior.

Dichas métricas se ven afectadas en gran proporción por los problemas que se exponen a continuación.

2.1.1. Largos tiempos de ciclo

Regularmente los equipos de desarrollo no despliegan una nueva versión de la aplicación hasta que todas las modificación y nuevas funcionalidades están totalmente terminadas, también ocurre que el equipo de desarrollo y el de operaciones no se encuentran alineados correctamente, lo que causa que código ya desarrollado se despliegue días o hasta semanas después que fue

terminado. El proceso de despliegue manual regularmente es secuencial, lento, con muchos pasos y complicado, además que puede suponer un gran riesgo ya que cada vez que se ejecuta, si uno de los pasos falla, puede fallar toda la aplicación, provocando una suspensión de los servicios durante el periodo de tiempo que tome recuperarse.

2.1.2. Despliegues poco confiables

Para realizar un despliegue tradicionalmente es necesario realizar una serie de pruebas manuales, las cuales son repetitivas y que, en un buen escenario, existe la documentación adecuada, pero se debe revisar cada vez que se realiza una modificación, esto causa mucho estrés en el equipo de control de calidad, como resultado pueden pasar por alto problemas críticos, y en consecuencia una mayor tasa de fracaso de los cambios. Dicho problema, sumado a que muchas veces se incentiva a que los programadores envíen el código rápidamente sin enfocarse en la calidad de este, da lugar a envíos poco confiables y el incremento de los fallos en el servicio.

2.1.3. Pérdida de productividad

El tiempo perdido en actividades que no producen resultados puede causar problemas, los equipos de desarrollo frecuentemente deben esperar a que nuevos ambientes se modifiquen o se creen, se necesita una aprobación de las peticiones de cambio, el equipo de control de calidad debe esperar a que el equipo de desarrolle termine su trabajo para poder realizar las pruebas, toda esta espera se traduce en despliegues poco frecuentes y largos periodos de espera para realizar los cambios.

2.1.4. Problema de la transferencia del conocimiento

Muchos proyectos se crean sin la documentación adecuada, las personas que desarrollaron el sistema probablemente ya no se encuentren al alcance, por lo que no es fácil identificar a las personas que conocen bien el sistema y puede convertirse en un problema. La persona encargada de monitorear el sistema, no necesariamente lo conoce a profundidad, lo que resulta en que sea muy complicado para esta persona realizar un cambio importante, ya que el conocimiento fundamental de la aplicación no puede ser consultado o no existe, todo esto provoca que en caso de una falla critica, los tiempos de recuperación sean muy elevados. A este problema se le puede sumar la ausencia de estándares definidos para el desarrollo, el cual dificulta la compresión y el mantenimiento del código, creando una dependencia hacia las personas que construyeron la aplicación desde un inicio.

2.1.5. Problemas para crear los ambientes correctos

La configuración de los ambientes de producción no siempre es documentada y se maneja únicamente entre el equipo de operaciones. Cualquier cambio en la configuración es controlado manualmente en los servidores, por lo que frecuentemente hay largos periodos de espera para que un ambiente sea preparado. Servidores de una misma aplicación pueden requerir configuraciones diferentes o librerías diferentes, por lo que todo este trabajo manual resulta en despliegues poco frecuentes y periodos de entrega más largos.

2.1.6. Falta de extensibilidad y escalamiento

Una aplicación puede ser lanzada con un numero de funcionalidades y contenido reducido, pero que luego de un periodo de tiempo se planee una experiencia más detallada y de abundante contenido. La extensibilidad es la capacidad que la aplicación tiene para incorporar nuevas capacidades y funcionalidades conforme pasa el tiempo. Si una aplicación no se construye desde el inicio con este atributo, seguramente necesitará un rediseño y desarrollo en el futuro.

Si una aplicación tiene despliegues lentos o largos periodos de espera para la creación de ambientes de producción, el tema de escalamiento se torna virtualmente imposible, la escalabilidad permite manejar diferentes tipos de usuario, manejar el aumento del tráfico y a tener una mejor disponibilidad de los servicios. Una falta de escalabilidad tiene consecuencias como la mala experiencia del usuario, un incremento en la tasa de deserción de clientes y un tiempo de vida más corto para la aplicación. Para que una aplicación se construya bien hoy, se necesita saber lo máximo posible de lo que se necesita que haga en el futuro.

2.2. Cloud Native

De acuerdo con un artículo publicado por Oracle (2020), *cloud native* hace referencia al concepto de construir, desplegar y mantener aplicaciones tomando ventaja del modelo de entrega que presenta la computación en la nube, de tal manera que permite implementar sus principios esenciales tales como escalabilidad, elasticidad, agilidad, velocidad, entre otros.

Sobre la idea presentada es posible determinar que realmente *cloud native* empodera a las organizaciones para construir y desplegar sus aplicaciones de manera escalable, utilizando cualquiera de los diversos tipos de nubes existentes.

Otro aspecto que se puede relacionar al término *cloud native* es el uso de tecnologías desarrollo modernas, ya que como tal estas tecnologías potencian la efectividad del uso de una solución en la nube, como por ejemplo tenemos dentro de estas tecnologías el uso de contenedores, arquitecturas de microservicios, DevOps, entre otras.

2.3. DevOps

Según una publicación realizada por Microsoft (2020) el término DevOps procede de la fusión de dos palabras de origen inglés, las cuales son Development (Desarrollo) y Operations (Operaciones), lo cual nos puede dar una noción sobre este tema, pero en la realidad representa un conjunto de ideas o prácticas que van más allá de las dos palabras que lo conforman.

DevOps es un cambio cultural que busca reducir o eliminar la brecha existente entre el equipo de desarrollo y operaciones, los cuales históricamente han trabajado de manera separada. Esta división ha generado grandes problemas de comunicación entre los equipos ya mencionados, creando una barrera imaginaria entre ambos que impide un correcto traslado de información de un equipo a otro.

Sobre la base anterior surge DevOps como una cultura que trata de no solamente eliminar esa brecha o barrera imaginaria, sino además generar una automatización de los procesos que existen dentro del ciclo de vida del

desarrollo de software, que abarca desde la creación del software hasta la implementación y mantenimiento de este.

Algo importante que debe ser mencionado es que la implementación de las prácticas que DevOps contiene puede generar que una empresa pueda construir, probar, y liberar software de manera más rápida y confiable.

2.4. Metodología The twelve-factor app

Según Adam Wiggins (2017), creador de la metodología The twelve-factor app, en la actualidad el software tiende a distribuirse ya como un servicio, lo que lo podemos denominar comúnmente como Web Apps o Software as a Service (SaaS), y en conjunto con este tipo de distribución han surgido diversas metodologías para poder crearlo, siendo una de las más importantes la metodología The twelve-factor app.

La metodología The twelve-factor app permite construir aplicaciones de tipo SaaS que:

- Tienden a utilizar formatos declarativos para la automatización de la configuración.
- Tienen un contrato claro con el sistema operativo que trabajan,
- Son adecuadas para desplegarse en plataformas modernas en la nube,
- Reducen las diferencias existentes entre los entornos de desarrollo y producción.
- Puede lograr escalar sin la necesidad de la realización de grandes cambios en sus herramientas, arquitectura o prácticas elegidas inicialmente.

Lo anterior nos permite evidenciar que este tipo de metodología puede ser empleada en múltiples aplicaciones sin importar el lenguaje de programación que se elija, y permitiendo la construcción de aplicaciones en cualquier combinación de recursos consumibles a través de la red, como una base de datos.

A continuación, se iniciará con la descripción de cada uno de los factores que conforman esta metodología, tratando de abarcar y presentar cada uno de ellos.

2.4.1. Código base (Codebase)

El código base nos referencia al código que se tiene dentro de un control de versiones que permite realizar múltiples despliegues.

Una aplicación desarrollada bajo la metodología The twelve-factor app, debe ser gestionada siempre con un sistema de control de versiones, dentro de ellos podemos encontrar por ejemplo Git, Mercurial o Subversion. Dentro de los sistemas de control de versiones se maneja un término importante que básicamente es una copia de la base de datos de seguimiento de versiones y es conocido comúnmente como repositorio de código.

El código base es cualquier repositorio (en un sistema de control de versiones centralizado como por ejemplo Subversion) o cualquier conjunto de repositorios que comparten un *commit* raíz (en un sistema de control de versiones descentralizado como Git).

Algo importante que se debe saber es que solo debe existir un código base esto para mantener centralizado el código, pero pueden existir múltiples

despliegues de la aplicación como se requiera. Un despliegue como tal es una instancia de la aplicación que está en ejecución.

Por último, se debe saber que, aunque el código base debe ser único puede existir una manera de tener múltiples códigos base, y esa forma es cuando se construye un sistema distribuido, ya que como bien se sabe cada parte del sistema distribuido es una aplicación.

2.4.2. Dependencias

Este factor nos menciona como tal que las dependencias al ser declaradas deben ser aisladas explícitamente, esto nos dice que una aplicación bajo la metodología The twelve-factor app no debe depender de la declaración explícita de paquetes instalados en un sistema.

Lo primero a realizar es la declaración de las dependencias, esto se tiene que hacer completa y explícitamente, mediante un manifiesto de declaración de dependencias. Además, se debe utilizar una herramienta o herramientas que permitan aislar las dependencias durante el tiempo de ejecución, esto permitirá que las dependencias, implícitamente, no afecten al resto del sistema.

Para conseguir lo antes mencionado muchos lenguajes cuentan con las herramientas necesarias para declarar y aislar las dependencias del sistema, y en nuestro caso en específico Python cuenta con dos herramientas para lograrlo, en primer lugar, PIP se utiliza para la declaración de las dependencias y Virtualenv se utiliza para el aislamiento, con ello se logra cumplir con el requerimiento que la metodología actual solicita.

2.4.3. Configuraciones

La configuración de una aplicación es de los únicos aspectos que puede variar entre los diversos despliegues que existen, dentro de ellas pueden incluirse los recursos que maneja una base de datos, credenciales para servicios externos, valores de despliegue, entre otros.

Las aplicaciones creadas bajo la metodología The twelve-factor app, almacenan sus configuraciones en variables de entorno, esto debido a que las variables de entorno se pueden modificar fácilmente entre los diversos despliegues sin la necesidad de realizar un cambio dentro del código de una aplicación. Además, las variables de entorno son un estándar que es independiente del lenguaje donde se trabaje y del sistema operativo que se utilice.

En una aplicación twelve-factor, las variables de entorno son controles granulares, en donde cada una de ellas tiende a ser ortogonal a las otras. Nunca se deben agrupar estas variables como “entornos”, en cambio deben de gestionarse de manera separada en cada despliegue, esto permite que sea escalable fácilmente.

2.4.4. Backing services

Un *backing service* se refiere a cualquier recurso que una aplicación puede llegar a consumir a través de la red como parte del funcionamiento natural. Un ejemplo de lo anterior sería una base de datos, sistemas de mensajería y de colas, servicios SMTP de email, entre otros.

El código de una aplicación twelve-factor, no debe realizar distinción al utilizar ya sea un servicio local o un servicio proporcionado por terceros, esto se debe a que la aplicación sólo debería visualizarlos como recursos conectados que pueden ser accedidos mediante una URL u otro localizador.

Un aspecto que debe tomarse muy en cuenta es que al momento de realizar un despliegue de una aplicación utilizando la metodología The twelve-factor app, el cambio de una base de datos local por una gestionada por terceros debe poder realizarse sin cambiar nada del código de la aplicación, esto es porque lo único que debería de cambiarse es la configuración.

2.4.5. Construir, distribuir y ejecutar

El código base se transforma en un despliegue al completar las siguiente tres fases o etapas:

- Etapa de construcción: es una transformación en donde básicamente se convierte un repositorio de código en un ejecutable llamado *build*. En esta etapa se traen todas las dependencias y todo el código utilizando para ello una versión concreta del mismo, esto debe ser correspondiente a *commit* ya determinado para este proceso.
- Etapa de distribución: se utiliza la construcción ya desarrollada en la fase anterior y se fusiona con la configuración del despliegue donde se realizará, Por lo cual en esta fase ya se tiene tanto la construcción como la configuración necesaria para la próxima ejecución.
- Etapa de ejecución: es conocida también como *runtime*, en esta etapa se realiza la ejecución en el entorno elegido para ello.

Las aplicaciones twelve-factor deben de tener una separación total de las tres etapas o fases que se describieron anteriormente, esto quiere decir por ejemplo que no debe ser posible la modificación del código en la etapa de distribución o ejecución.

2.4.6. Procesos

Una aplicación debe de ejecutarse como uno o más procesos en el entorno de ejecución, para ejemplificar esto podemos verlo de la siguiente manera, imaginemos que el código es un script independiente, el entorno de ejecución es un portátil de un desarrollador, el compilador o intérprete correspondiente ya se encuentra instalado, y el proceso se lanza mediante línea de comandos, en este caso podemos ver como puede ser que una aplicación puede convertirse en un proceso aunque solamente sea un ejemplo sencillo.

Los procesos twelve-factor no deben de tener un estado como tal y deben ser del tipo share-nothing. Ahora bien, si lo que se necesita es que la información sea persistente se debe almacenar en un *backing service*, por ejemplo, en una base de datos

2.4.7. Asignación de puertos

Las aplicaciones web se ejecutan normalmente mediante contenedores web, por ejemplo, las aplicaciones JAVA se suelen ejecutar como módulos en Tomcat, pero en el caso de las aplicaciones twelve-factor, es distinto, ya que son completamente autocontenidas, esto nos quiere decir que realmente no dependen de un servidor web en ejecución para poder crear un servicio web público.

Una aplicación web comúnmente utiliza HTTP como un servicio al que se le puede asignar un puerto, y escucha las peticiones que puede recibir por dicho puerto, aunque HTTP no es el único servicio que permite la asignación de puertos, esto se debe a que cualquier servicio puede ejecutarse por medio un proceso al cual es posible asignarle un puerto y esperar las peticiones que pueden llegar.

Algo importante que se debe saber es que la asignación de puertos puede permitir que una aplicación sea un *backing service* para otra aplicación, esto es posible si se utiliza la URL de la aplicación con la que se desea conectar como un recurso declarado en la configuración de la aplicación que realiza el consumo.

2.4.8. Concurrencia

Las aplicaciones twelve-factor tratan a los procesos como ciudadanos de primera clase. Los procesos utilizan como fuente de inspiración el modelo de procesos desarrollados por Unix para ejecutar *daemons*. Este modelo le permite a un desarrollador distribuir la ejecución de su aplicación para poder gestionar diversas cargas de trabajo, asignando cada tipo de trabajo a un tipo de proceso. Un ejemplo claro de lo anterior puede ser que las peticiones de tipo HTTP puedan ser realizadas con un proceso y las tareas que contienen una carga bastante grande puede procesarse con hilos.

El potencial del uso del modelo antes mencionado se puede visualizar al momento de tratar de escalar, ya que al utilizar la naturaleza del share-nothing, que divide los procesos de manera horizontal, permite el aumente de la concurrencia de una forma simple y fiable.

2.4.9. Desechabilidad

Los procesos dentro de la metodología The twelve-factor app son desechables, esto significa que se pueden iniciar o finalizar en el momento que se desee. Esto conlleva que se pueda escalar de manera rápida y flexible, así también permite un despliegue rápido del código y configuraciones, y despliegues más robustos.

Los procesos como tal deberían reducir su tiempo de arranque, esto proporciona al proceso mayor agilidad en la distribución y el escalado, y lo hace más robusto, esto anterior es debido a que el gestor de procesos puede mover los procesos de forma segura entre máquinas físicas más fácilmente.

Una aplicación twelve-factor es una arquitectura que trata con las finalizaciones inesperadas y peligrosas, como tal se utiliza el diseño *crash-only* para llevar este concepto a su conclusión lógica.

2.4.10. Igualdad entre desarrollo y producción

Históricamente ha existido dos entornos bastante diferenciados en la construcción de un software, en primer lugar, tenemos el entorno de desarrollo y en segundo lugar encontramos el entorno de producción, estas diferencias entre dichos entornos se explican a continuación:

- Diferencias de tiempo: un desarrollador puede estar trabajando en un código durante días, semanas o incluso meses antes de que llegue a producción.
- Diferencias de personal: los desarrolladores escriben el código y los ingenieros de operaciones lo despliegan.

- Diferencias de herramientas: los desarrolladores pueden utilizar un conjunto de herramientas distinto a los utilizados en producción.

Las aplicaciones twelve-factor se diseñan para tratar de realizar despliegues de forma continua que tratan de disminuir las diferencias entre los entornos de desarrollo y producción, y para lograrlo trabajan de la siguiente manera en cada una de las diferencias mencionadas anteriormente:

- Reducción de diferencias de tiempo: un desarrollador puede escribir código y tenerlo desplegado en cuestión de horas, o incluso, minutos.
- Reducción de las diferencias de personal: los desarrolladores que escriben el código están muy involucrados en el despliegue y pueden conocer su comportamiento en producción.
- Reducción en diferencias de herramientas: se trata de manejar el mismo conjunto de herramientas hasta donde sea posible, tanto en desarrollo como en producción.

Lo que podemos obtener de lo mencionado anteriormente es que lo que se debe buscar es el mantener los entornos de desarrollo, preproducción y producción tan parecidos como sea posible.

2.4.11. Historiales (*logs*)

Los historiales permiten observar el comportamiento de una aplicación en tiempo de ejecución. En los entornos en los cuales se basan en servidores es muy común observar que se almacenen en un fichero en el disco, denominado fichero histórico, pero como tal solamente es un tipo de los formatos de salida.

Una aplicación *twelve-factor* nunca se preocupa del direccionamiento o almacenamiento de sus diversas transmisiones de salida, esto se debe a que no se debería escribir o gestionar ficheros de salida. En su lugar, cada proceso en ejecución escribe sus eventos a la salida estándar (*stdout*), de esta manera, los desarrolladores verán el flujo en su terminal para observar el comportamiento de la aplicación.

Ahora bien, en despliegues de preproducción y producción, cada transmisión debe ser capturada por el entorno de ejecución y redirigidas a uno o más destinos finales para ser revisadas o archivadas. Es importante aclarar que estos destinos donde se archivan no son visibles o configurables por la aplicación, se gestionan totalmente por el entorno de ejecución.

2.4.12. Administración de procesos

El juego de procesos es como tal el conjunto de procesos que se utiliza para realizar las tareas habituales de una aplicación. Por otro lado, suele suceder que en algunas ocasiones los desarrolladores necesitan ejecutar procesos de administración o mantenimiento en una única ocasión, un ejemplo de esto podría ser ejecutar tareas de migración de bases de datos, también ejecutar una consola, o ejecutar scripts incluidos.

Los procesos mencionados anteriormente deben de ejecutarse en un entorno idéntico al que se utiliza normalmente en un proceso habitual dentro de la aplicación. Estos procesos se deben ejecutar en una distribución concreta, utilizan el mismo código base y la misma configuración que cualquier otro proceso que ejecuta esta esa distribución. Además, el código de administración se debería enviar con el código de la aplicación para evitar problemas de sincronización.

Por último, se debe aclarar que estos procesos de administración deberían utilizar las mismas técnicas de aislamiento de dependencias que los demás procesos, ya que todos deberían utilizar la misma técnica de aislamiento de dependencias.

2.5. Arquitectura

La arquitectura de un programa o un sistema de software se representa como tal por la organización o estructura de cada uno de sus componentes, especificando sus atributos, sus relaciones y la forma en la cual realizan sus interacciones.

Algo importante que se debe conocer es que cuando hablamos de arquitectura debemos saber que se logra obtener por medio de ella un nivel de detalle bastante amplio, ya que es posible conocer desde el nivel más alto del sistema hasta los componentes más pequeños que lo integran.

2.5.1. Patrones comunes

Los patrones de arquitectura permiten expresar como tal un esquema organizacional de la aplicación, logrando establecer los componentes, relaciones y cómo pueden llegar a ser utilizados los componentes del sistema.

En este caso estos patrones que se mencionan establecen ciertas cualidades comunes de una solución necesitada, por ejemplo, puede definirse por medio de las restricciones del sistema o topología dada.

Los patrones que a continuación se definirán, son los patrones más comunes utilizados en la construcción de soluciones a diversos problemas de

software presentados, y como tal permiten definir un tipo de arquitectura funcional.

2.5.1.1. Arquitectura orientada a servicios (SOA)

La arquitectura orientada a servicios es un estilo de diseño de software se basa en crear aplicaciones utilizando servicios interoperables que pueden llegar a ser integrados con varias aplicaciones. La idea detrás de este uso es el brindar las ventajas de integración, reutilización de los servicios, encapsulamiento de las aplicaciones y sobre todo seguridad.

Un aspecto importante que se debe respetar es que los servicios que se construyen deben ser de estándares abiertos, dicho de otra manera, debe ser independiente del sistema operativo en el que se ejecuta o el lenguaje de programación que se utiliza.

Estos servicios utilizados en este tipo de arquitectura se le conoce comúnmente como *Web Services*, y el utilizarlos como tal nos otorga la facilidad de escalamiento, además permite la reutilización y reducir el acoplamiento. La imagen que a continuación se presenta nos muestra cómo se maneja y distribuye la arquitectura orientada a servicios, por lo cual nos permite evaluar de mejor manera como trabaja dicha arquitectura.

Figura 2. **Arquitectura orientada a servicios**

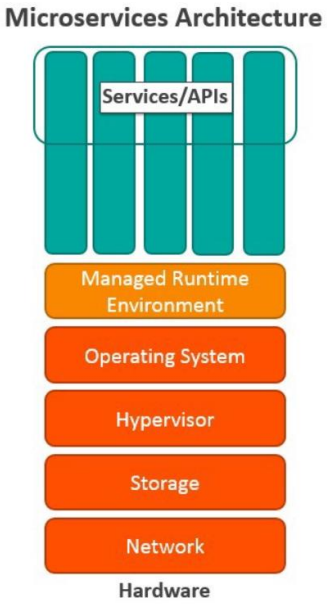
Fuente: Software development community (2019). *What is a service-oriented architecture?* Consultado el 27 de agosto de 2021. Recuperado de https://miro.medium.com/max/840/1*qAFyYAQSE3e-flZSqprHlg.jpeg.

2.5.1.2. Arquitectura de microservicios

Este tipo de arquitectura se basa en dividir su funcionalidad en pequeños componentes autosuficientes e independientes del resto de componentes. Los microservicios que pertenecen a esta arquitectura deben ser servicios autónomos ya que deben funcionar de manera separada cada uno de ellos, además cada microservicio debe implementar una sola funcionalidad del negocio al cual pertenece el software.

Los beneficios detectables de implementar este tipo de arquitectura son la alta escalabilidad que provee, la facilidad para realización de pruebas y la interoperabilidad utilizando estándares abiertos. La siguiente imagen nos mostrará cómo funciona en realidad una arquitectura microservicios, por lo cual es importante analizar las partes que el mismo contiene.

Figura 3. **Arquitectura microservicios**



Fuente: Software development community (2019). *What is a service-oriented architecture?* Consultado el 27 de agosto de 2021. Recuperado de https://miro.medium.com/max/840/1*qAFyYAQSE3e-flZSqprHlg.jpeg.

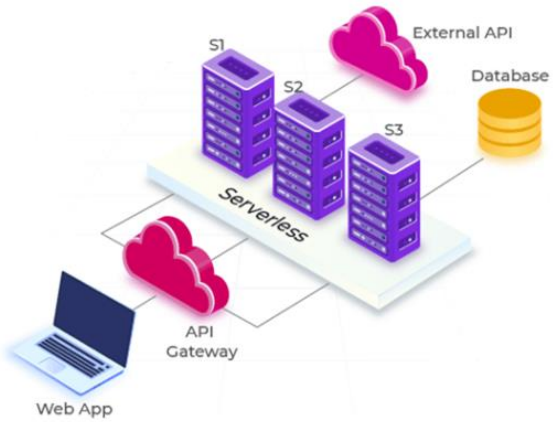
2.5.1.3. Arquitectura Serverless

La arquitectura Serverless pueda llegar a generar una noción un poco errónea en su nombre, ya que da la idea de que no existen servidores dentro de su implementación, pero en la realidad esto tiende a ser muy diferente, debido a que no existen servidores físicos o en la nube administrados por la empresa que crea el software, sino en cambio esta tarea se delega a terceros.

Este tipo de arquitectura se caracteriza por delegar los temas de despliegue, infraestructura, escalabilidad y disponibilidad a una o varias empresas, logrando así solamente enfocarse a desarrollar la lógica de la aplicación que se desea realizar.

La forma de implementación que tiene este tipo de arquitectura permite un gran ahorro de tiempo, en la administración de los servidores, además permite una mayor facilidad para desarrollar debido a que al dejar de un lado la infraestructura solamente se centra en los servicios a implementar.

Figura 4. **Arquitectura Serverless**



Fuente: Opcito (2019). *Serverless isn't exactly serverless!* Consultado el 27 de agosto de 2021. Recuperado de https://www.opcito.com/hubfs/thumbnail%20%288%29-1.png.

2.5.2. Ejemplos de casos de éxito

A continuación, se mostrarán un conjunto de ejemplos de cómo las arquitecturas anteriormente presentadas lograron mejorar y resolver los

problemas que presentaron algunas de las empresas más conocidas en la actualidad.

2.5.2.1. Arquitectura orientada a servicios (SOA)

Un caso de éxito que se puede observar en la implementación de la arquitectura SOA, lo podemos tener con la empresa eBay que implemento esta arquitectura como un nivel de integración basado en software.

McKendrick (2006) señala en su artículo que el uso de esta arquitectura ya mencionada le permitió a eBay manejar más de dos *petabytes* de datos, lo que nos demuestra que realmente la implementación de este tipo de arquitectura realmente puede ser muy eficiente en el manejo de la información.

2.5.2.2. Microservicios

Un caso de éxito que podemos encontrar claramente en esta arquitectura es el caso de Netflix, ya que como tal es una de las primeras compañías en pasar de una arquitectura monolítica a una arquitectura de microservicios basada en la nube.

Netflix decidió trasladarse a este tipo de arquitectura debido a que sus datos e información estaban creciendo rápidamente, y necesitaban una arquitectura que se pudiese acoplar a ello, y encontraron la solución en la implementación de los microservicios.

En la actualidad Netflix tiene más de 1000 microservicios desarrollados y cada uno de ellos maneja una parte separada del sitio completo.

2.5.2.3. Serverless

Un caso de éxito encontrado en esta arquitectura es el caso de Shamrock, esta empresa especializada en los servicios de transporte, financiamiento y tecnología inicialmente decidió utilizar Docker para poder crear una aplicación de facturación, pero esta aplicación se volvió difícil de manejar, por lo cual Shamrock decidió migrar a la arquitectura Serverless en 2018.

Esta transición ayudó grandemente a la empresa ya que permitió un aumento en los usuarios que utilizaban su aplicación y una reducción en costos mensuales, que en comparación con la implementación con Docker era 10 veces menor el costo, además de lo anterior le permitió a la compañía poder desechar aplicaciones heredadas, crear nuevas herramientas de automatización y optimizar el flujo del equipo.

2.6. Tendencias

En base a una investigación realizada por Mariia Lozhko (2021) las tendencias detectadas en las aplicaciones web modernas son bastante importantes ya que nos permiten prepararnos para los temas que maneja en la actualidad el mercado, es por ello por lo que debemos conocer los principales exponentes que dominan las nuevas tecnologías presentadas, y a continuación se realizará un recuento de ellos.

2.6.1. Principales exponentes

Los principales exponentes en torno a las aplicaciones web se obtuvieron de la investigación previamente mencionada, y se detallarán en este apartado, tratando de mostrar no solamente el tipo de tendencia que se está

desarrollando, sino también agregar una pequeña descripción sobre lo que trata cada tema que se lista a continuación.

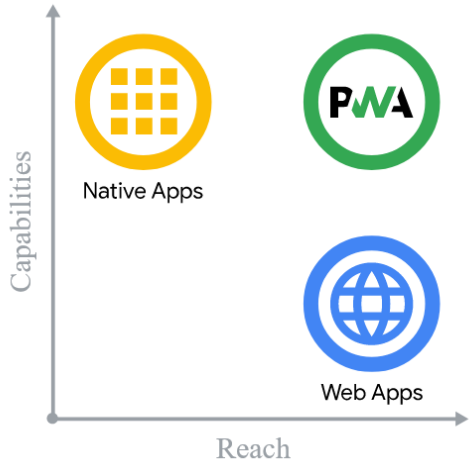
2.6.1.1. Progressive web app (PWA)

Son aplicaciones web que fueron diseñadas para ser capaces, confiables e instalables. Como tal se puede evidenciar que dentro de los tres aspectos ya mencionados hay uno que sobresale y es que este tipo de aplicaciones pueden ser instaladas en el sistema operativo.

Las PWA utilizan *service workers*, manifiestos y otras características de plataformas web en combinación con la mejora progresiva, permitiendo así que el usuario pueda sentir que se encuentra en una aplicación nativa.

La siguiente imagen muestra cómo se encuentran las aplicaciones web progresivas, y como se puede evidenciar se encuentran en medio de las web apps y las native apps.

Figura 5. **Comparación del diseño de PWA**



Fuente: Mahendra (2021). *Progressive web app.* Consultado el 28 de agosto de 2021. Recuperado de https://webdev.imgix.net/image/tcFciHGuF3MxnTr1y5ue01OGLBn2/1DKtUFjXLJbiiruKA9P1.svg.

2.6.1.2. Acelerated mobile pages (AMP)

Esta es una de las tendencias de desarrollo web que busca la aceleración del rendimiento de la página y tratar de disminuir el riesgo de que un usuario deje la página demasiado pronto.

Este tipo de tecnología, aunque puede ser parecida a la tendencia anterior, realmente no lo es, ya que ambas son construidas de diferente manera, por ejemplo, las AMP presentan su aceleración por medio del complemento de open-source desarrollado por Twitter y Google.

Por último, solo hay que mencionar que las AMP son páginas optimizadas que tienen a operar de manera rápida y simplificada, contando solo con un

diseño conveniente con características básicas ya que su fuerte es la velocidad a comparación de un sitio web de gran alcance.

2.6.1.3. Simple page application (SPA)

Esta tendencia nos hace referencia a la construcción de una aplicación web en donde la interacción con el usuario se da en una única página, la cual se reescribe dinámicamente con los nuevos datos proveídos por el servidor, evitando así crear nuevas páginas.

Lo mejor de este tipo de aplicación web es el menor consumo que se tiene al acceder a ella, y que además puede atraer mayormente la atención del usuario debido a que no debe estar interactuando con muchas páginas a la vez.

Según las tendencias actuales puede ser muy probable que exista en el futuro una cantidad mucho mayor de personas que construyan sus aplicaciones web de esta forma ya que el uso va en crecimiento.

2.6.2. Retos para su construcción

Según un artículo publicado por Microsoft (2018) actualmente existen muchos retos para poder desarrollar o construir una aplicación web moderna, ya que, aunque existan múltiples opciones para poder realizarlo, aún no cubren todos los aspectos necesarios para resolver todos los problemas o retos encontrados, por ello a continuación se tratará de mostrar un conjunto de ejemplos de este tipo.

2.6.2.1. Data managment

El manejo de los datos es unos de los principales retos que se tienen en la construcción de aplicaciones web y esto como tal tiende a influir en aspectos de calidad del producto a entregar. Los datos como tal tienden a provenir de diversas fuentes o diversas ubicaciones y esto puede presentar una gran variedad de desafíos para lograr manejarlo correctamente.

El desafío anterior nos da un primer vistazo de los retos que se tienen, pero para entender mejor a lo que se refiere este reto se puede utilizar de ejemplo que durante el manejo de los datos se tiene que mantener una coherencia de estos, y por lo general, los datos deben de lograr sincronizarse en diferentes ubicaciones.

2.6.2.2. Design and implementation

Un buen diseño tiende a abarcar factores como por ejemplo la consistencia y la coherencia dentro del diseño, así también la implementación de los componentes, la capacidad de mantenimiento y la reutilización de los componentes en otros escenarios o aplicaciones, por lo cual es evidenciable que representa un reto en la construcción de una aplicación.

Las decisiones que se tomen durante la fase de diseño e implementación tienden a generar un gran impacto en la calidad y costo resultante de las aplicaciones que se desarrollan, por ello que a este desafío se debe abarcar inteligentemente, porque un mal diseño o implementación puede traer consigo grandes problemas.

2.6.2.3. Messaging

Las aplicaciones modernas tienen a ser generadas en arquitecturas como por ejemplo de microservicios o SOA, en la cual se requiere de una infraestructura de mensajería o de comunicación que pueda conectar cada uno de los servicios que conforman la aplicación, siendo lo más idea que se implemente de manera que permita la escalabilidad.

Un tipo de mensajería que puede ser bastante funcional para este aspecto puede ser la mensajería asincrónica que contiene muchos beneficios, aunque se debe estudiar la mejor opción que se acople al tipo de problema que se desea solucionar.

2.6.3. Oferta académica

La oferta académica para la construcción de aplicaciones de tipo *cloud native* ha crecido en estos últimos años, esto se debe al gran desarrollo que se ha dado en la tecnología en la nube, además de la agregación de múltiples empresas en este ámbito.

2.6.3.1. Cloud native computing foundation

Esta plataforma representa un gran beneficio para poder crecer a nivel académico, esto se debe a que ofrece una gran variedad de posibles certificaciones o aprendizajes sobre temas bastante interesantes en la actualidad, un ejemplo de ello son las certificaciones relacionadas a Kubernetes, por lo cual poder unirse a esta fundación permitirá obtener grandes beneficios a nivel académico.

2.6.3.2. Certificaciones relacionadas a la construcción de aplicaciones *cloud native*

Las certificaciones a nivel cloud han adquirido gran valor en los últimos años, esto se debe al creciente traslado y adopción de múltiples softwares dentro de la nube, es por ello por lo que a continuación mostraré tres certificaciones de las compañías más grandes que se dedican a entregar servicios en la nube.

- AWS Foundational**:** esta certificación requiere de por lo menos 6 meses de conocimientos sobre la nube de AWS además del conocimiento general del sector, por lo cual para poder obtener como tal esta certificación se debe haber practicado o conocido los servicios que ofrece esta nube, y que permite certificar a la persona como un profesional de la nube.
- Azure Fundamentals**:** esta certificación perteneciente a Microsoft, requiere que una persona conozca los servicios en la nube y como son proveídos por Microsoft Azure. Esta certificación además es para personas que están iniciando a trabajar con la nube o está comenzando a conocer estos temas.
- Google Foundational**:** esta certificación perteneciente a Google es para personas que aún no conozcan el trabajo con los servicios en la nube y deseen comenzar a aprender sobre ello.
- Google Associate: este tipo de certificación es para personas que ya tienen conocimiento sobre los servicios que trabaja Google, y además ya cuentan con más de 6 meses de utilizar esos servicios.

3. TECNOLOGÍAS PARA APLICACIONES MODERNAS

Existen una diversidad muy amplia de tecnologías para desplegar una aplicación moderna, la nube es un término que se utiliza para describir una red de servidores que están conectados para funcionar como un único ecosistema.

3.1. Tipos de nube

La nube se puede clasificar en pública, privada, híbrida y multicloud, anteriormente se distinguían según su ubicación y propiedad, pero con el pasar de los años, la distinción entre cada clasificación requiere tomar en cuenta muchos aspectos.

3.1.1. Nube pública

Una nube pública es un entorno de nube que se crea a partir de una infraestructura de TI ajena al usuario final. Algunos de los principales proveedores son Amazon Web Services (AWS), Google Cloud, IBM Cloud y Microsoft Azure.

Antes las nubes públicas se ejecutaban de forma externa a las empresas, sin embargo, los proveedores actuales han comenzado a ofrecer estos servicios dentro de los centros de datos de los clientes. Debido a esto la ubicación y la propiedad no es un factor determinante para distinguir entre una nube pública o privada. Una nube se convierte en pública cuando los entornos se dividen y distribuyen entre varios usuarios, usualmente siguiendo una estructura tarifaria.

3.1.2. Nube privada

Una nube privada es un entorno de nube que se destina exclusivamente a un usuario o grupo finales, usualmente se ejecuta detrás de su propio cortafuegos. Regularmente este tipo de nubes se ejecutaba dentro de la infraestructura de TI de la empresa, sin embargo, se está optando por alquilar centros de datos de terceros. Una nube se vuelve privada cuando la infraestructura de TI subyacente se destina a un solo cliente con acceso completamente aislado.

3.1.3. Nube híbrida

Una nube híbrida es un entorno de TI aparentemente único creado a partir de múltiples entornos que se conectan mediante redes de área local (LAN), redes de área amplia (WAN), redes privadas virtuales (VPN) o API. Las características de las nubes híbridas son complejas y los requisitos pueden variar, pero casi siempre se incluye una mezcla entre nube privada y pública comunicándose entre sí.

Una nube se convierte en híbrida cuando las aplicaciones pueden trasladarse a varios entornos distintos, pero que aún están conectados, y fuera de ellos. Al menos algunos de estos entornos deben provenir de recursos de TI consolidados que puedan ampliarse, según se requiera. Asimismo, todos estos entornos deben gestionarse como un solo entorno con una plataforma integrada de organización y gestión.

3.1.4. Multicloud

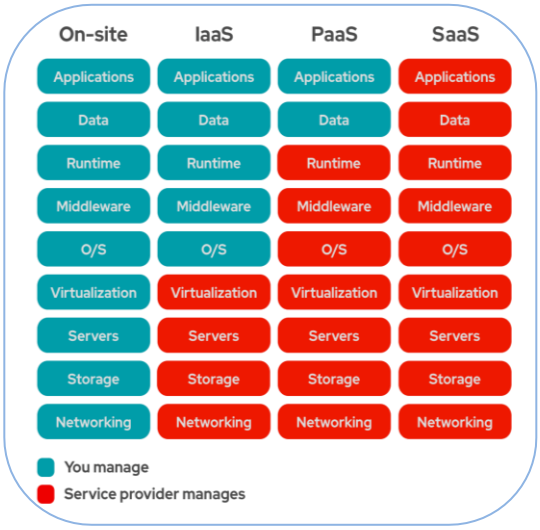
Multicloud se refiere a un enfoque de nube compuesto por al menos dos servicios de nube, que proporcionan por lo menos dos proveedores de nube pública o privada. Todas las nubes híbridas son multiclouds, pero no todas las multiclouds son híbridas. Las multiclouds se vuelven híbridas cuando se conectan varias nubes con algún tipo de integración u organización.

Un entorno multicloud puede crearse para controlar mejor los datos confidenciales o como un espacio de almacenamiento redundante para una mejor recuperación ante desastres, de cualquier forma, cada vez es más común que las empresas con una amplia gama de servicios busquen mejorar la seguridad y el rendimiento utilizando varias nubes.

3.2. Tipos de servicios en la nube

Los servicios de nube son infraestructuras, plataformas o sistemas de software que alojan los proveedores externos y que se ponen a disposición de los usuarios a través de Internet. Los servicios se pueden clasificar según el nivel de control que se posee sobre la infraestructura que sostiene una aplicación.

Figura 6. **Tipos de servicios en la nube**



Fuente: Red Hat (2020). *Diferencias entre IaaS, Paas y SaaS.* Consultado el 27 de agosto de 2021. Recuperado de https://www.redhat.com/cms/managed-files/iaas-paas-saas-diagram5.1-1638x1046.png.

3.2.1. On-premise

On-premise se refiere a que el software y la tecnología se aloja en centros de datos administrados por la organización, pueden ser propios o rentados, en todo caso el personal de TI tiene acceso físico a los datos y puede controlar directamente la configuración, la gestión y la seguridad de la infraestructura informática y los datos. Normalmente su utilización requiere la adquisición de licencias de software para cada servidor o para el usuario final que vaya a trabajar con el software.

3.2.2. IaaS

IaaS significa Infrastructure as a Service, en este tipo de servicio el proveedor gestiona la infraestructura, es decir que gestiona los servidores, la

red, la virtualización y el almacenamiento de datos reales a través de una conexión a Internet. Se realiza la renta de la infraestructura y el usuario accede a ella con una API o un panel.

El usuario gestiona el sistema operativo, las aplicaciones y el middleware, mientras que los proveedores se encargan de los sistemas de hardware, las redes, los discos duros, el almacenamiento de datos y los servidores. Además, son los responsables de prevenir las interrupciones, hacer reparaciones y solucionar los problemas de hardware. Este es el modelo de implementación tradicional de los proveedores de almacenamiento en la nube.

3.2.3. PaaS

PaaS significa Platform as a Service, en este tipo de servicio el proveedor proporciona y gestiona el hardware y una plataforma de software de aplicaciones. El usuario es el que maneja las aplicaciones que se ejecutan en la plataforma y los datos en los que se basa la aplicación. La utilización de PaaS ofrece una plataforma en la nube compartida para desarrollar y gestionar aplicaciones sin tener que diseñar ni mantener la infraestructura generalmente asociada con el proceso, lo cual resulta especialmente útil para los desarrolladores y los programadores.

3.2.4. SaaS

Saas significa Software as a Service, en este servicio se ofrece a los usuarios una aplicación de software que es gestionada por el proveedor de servicios de nube. Las aplicaciones SaaS son aplicaciones web o aplicaciones móviles a las que los usuarios pueden acceder a través de un explorador web.

Las actualizaciones de software, las correcciones de fallos y otros mantenimientos generales del software están a cargo del usuario, y se conectan a las aplicaciones de la nube a través de un panel o una API. El SaaS también elimina la necesidad de instalar localmente una aplicación en la computadora de cada usuario, lo cual da lugar a mejores métodos de acceso grupal o en equipo al sistema de software.

3.2.5. Otros servicios

El número de servicios disponibles en la nube crece continuamente, algunos proporcionan funciones muy concretas a los usuarios, por lo que prácticamente cualquier servicio de software disponible tiene su opción equivalente en la nube, algunos que se pueden mencionar son:

- Analytics as a Service
- Backup as a Service
- Business as a Service
- Communications as a Service
- Content as a Service
- Logging as a Service
- Monitoring as a Service
- Search as a Service
- Security as a Service
- Storage as a Service

Estos son algunos ejemplos de otros servicios que se pueden encontrar en la nube, sin embargo, se pueden contener en alguna de las clasificaciones anteriores. Existen otros servicios de los que vale la pena hacer énfasis debido a que recientemente se consideran como nuevas categorías.

3.2.5.1. FaaS

FaaS significa Functions as a Server, es un modelo de ejecución informática que se basa en eventos y se ejecuta en contenedores sin estado. Esas funciones son las que gestionan la lógica y el estado de los servidores a través del uso de servicios. FaaS permite a los desarrolladores diseñar, ejecutar y gestionar esos paquetes de aplicaciones como funciones, sin tener que mantener su propia infraestructura.

FaaS es una forma de implementar la llamada computación sin servidor, donde los desarrolladores escriben la lógica empresarial que luego se ejecuta en contenedores de Linux totalmente gestionados por una plataforma. Algunos ejemplos de FaaS conocidos:

- AWS Lambda
- Google Cloud Functions
- Microsoft Azure Functions (open source)
- OpenFaaS (open source)

3.2.5.2. CaaS

CaaS significa Containers as a Service, consiste en un modelo informático de servicios de nube que permite a los usuarios implementar y gestionar aplicaciones, a través de la abstracción basada en contenedores. El proveedor ofrece el marco, o la plataforma de organización, en el que se implementan y gestionan los contenedores; y gracias a esta organización se automatizan las funciones de TI más importantes.

Los contenedores crean entornos uniformes que agilizan el desarrollo de aplicaciones en la nube, las cuales pueden ejecutarse en cualquier parte, y también aceleran su distribución. Entre los proveedores más populares están:

- Kubernetes (K8S)
- Docker Swarm
- Google Container Engine (GKE)
- Azure Container Service (AKS)
- Amazon ECS

3.3. Proveedores más populares

Existe una gran variedad de proveedores de servicios de nube y cada día aparecen nuevos, sin embargo, es necesario exponer aquellos que tienen mayor relevancia en el mercado actual. De acuerdo con el cuadrante mágico de Gartner AWS lidera como proveedor de infraestructura, seguido de Microsoft y Google.

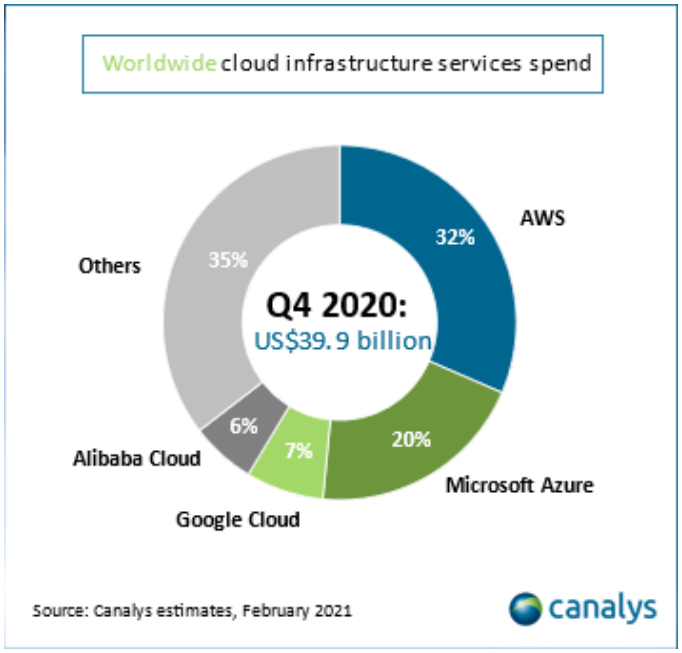
Figura 7. **Cuadrante mágico de IaaS en la nube del 2019**

Fuente: Chand (2019). *Top 10 cloud service providers in 2021.* Consultado el 29 de agosto de 2021. Recuperado de https://csharpcorner-mindcrackerinc.netdna-ssl.com/article/top-10-cloud-service-providers/Images/CloudIaaS.png.

Según el estudio realizado por Canalys (2020), los cuatro principales proveedores representaron el 65 % del gasto total en la nube en el cuarto trimestre de 2020. La demanda de servicios en la nube se mantuvo fuerte en todos los segmentos de clientes empresariales, incluidos los sectores más afectados por la pandemia, como el comercio minorista y la fabricación. "El ritmo de la digitalización, liderado por la nube, se está acelerando. Las empresas están ahora más seguras de liberar presupuestos para la

transformación del negocio", dijo el analista de investigación de Canalys, Blake Murray. "Los grandes proyectos que se pospusieron a principios de año se están volviendo a priorizar, liderados por la modernización de aplicaciones, las migraciones de SAP y la transformación del lugar de trabajo. La sanidad, los servicios financieros y el sector farmacéutico se encuentran entre los sectores que lideran el camino, pero incluso los que se encuentran bajo mayor presión están desviando las inversiones a la nube, abriendo nuevas vías de ingresos y diversificando los modelos de negocio." Al mismo tiempo, las pequeñas y medianas empresas siguen recurriendo a los servicios en la nube para ayudarles a mantener sus operaciones y controlar los costes.

Figura 8. **Gastos en los servicios de infraestructura en la nube**



Fuente: Canalys (2020). *Global cloud infrastructure market Q4 2020.* Consultado el 29 de agosto de 2021. Recuperado de https://www.canalys.com/newsroom/global-cloud-market-q4-2020.

3.4. Categorización

El presente apartado permitirá conocer una categorización de todos los proveedores de tecnologías más populares por cada uno de los servicios ya mencionados anteriormente, de tal manera que permitirá conocer quiénes son los mayores exponentes de cada categoría.

3.4.1. Más populares por categoría

La categorización que se hará a continuación pertenece a los servicios ya mencionados con anterioridad por lo cual se establecerá quiénes son los líderes de cada categoría presentada a continuación.

- IaaS: esta categoría perteneciente a infraestructura como servicio es liderada por la empresa Amazon Web Services, seguida a su vez de Microsoft y en el tercer lugar tenemos a Google, como se evidenciará en la imagen que se adjunta abajo, esas tres empresas lideran esta categoría.
- PaaS: esta categoría perteneciente a plataforma como servicio es liderada por la empresa Amazon Web Services, seguida a su vez de Microsoft y en el tercer lugar tenemos a Google, y aunque suceda lo mismo que la categoría anterior es porque como tal los servicios que se ofrecen lideran siempre en el mismo lugar, como se evidenciará en la imagen que se adjunta abajo, esas tres empresas lideran esta categoría.
- SaaS: esta categoría perteneciente a software como servicio está liderada por empresas que ofrecen software a los usuarios para su uso, aquí podemos encontrar por ejemplo a empresas como Microsoft o Google que lideran esta categoría.

- On-premise: esta categoría pertenece a las soluciones creadas fuera de la nube, esto quiere decir que directamente se realiza la instalación física de la infraestructura, y se encuentra liderada por IBM, Software AG y Oracle, los cuales ocupan los primeros lugares de esta categoría.

Figura 9. **Cuadrante mágico de servicios en la nube del 2021**

Fuente: AWS (2021). *Magic Quadrant de 2021 para servicios de infraestructura y plataforma en la nube*. Consultado el 29 de agosto de 2021. Recuperado de https://aws.amazon.com/es/resources/analyst-reports/gartner-mq-cips-2021/.

3.5. Tendencias observadas

Este apartado tratará de mostrar las tendencias encontradas sobre las tecnologías que se encuentran disponibles para la construcción de aplicaciones

modernas, permitiendo así prepararse ante las nuevas tecnologías que existen o existirán pronto.

3.5.1. Frameworks

La primera tendencia detectada es la aparición de *frameworks* para la construcción de aplicaciones web, dichos *frameworks* facilitan la creación de nuevas aplicaciones además que permiten la inclusión de nuevas características a un software nuevo que se desee crear

Según un estudio realizado por el ingeniero Diego Pacheco (2021), se determina que algunos de los *frameworks* más solicitados son los siguientes que se describen.

- React: lenguaje base JavaScript
- Vue.js: lenguaje base JavaScript
- Angular: lenguaje base JavaScript
- Spring: lenguaje base Java
- Django: lenguaje base Python
- Flask: lenguaje base Python
- Laravel: lenguaje base PHP
- Ruby on Rails: lenguaje base Ruby

3.5.2. Manejo de logs

El manejo de los registros de lo que sucede con nuestras aplicaciones tiende a ser un factor de suma importancia al desarrollar un nuevo software ya que se hace necesario determinar qué es lo que realmente sucede con nuestro

software, por ello a continuación se lista las herramientas para este tipo de manejo de registros.

- Graylog
- Logstash
- Fluentd
- Rsyslog
- LOGalyze

3.5.3. Almacenamiento

Las tendencias en el almacenamiento de datos han presentado importantes cambios para las personas, pero estos cambios son para mejorar como tal el servicio, ya que estas nuevas tendencias lograran poder almacenar de mejor manera las grandes cantidades de datos que van a ir creciendo conforme el tiempo pase, por lo cual a continuación se realiza una recopilación sobres estas nuevas tendencias en almacenamiento.

- Bases de datos que unen SQL/NoSQL, esta tendencia se basa en tratar de obtener lo mejor de ambos mundos de las bases de datos, de tal manera que un usuario podrá obtener mejores capacidades al poder acceder a ambos tipos de datos.
- Bases de datos en la nube (PaaS), esta tendencia nos refiere a la gran variedad de posibles bases de datos que podemos encontrar en la nube. Estas bases de datos se trabajan como plataforma como servicio debido al tipo de implementación que se realiza.
- Gestión automatizada, es otra tendencia importante a tomar en cuenta y trata sobre el conjunto de técnicas y herramientas para la automatización

del manejo de la base de datos, tratando de simplificar el mantenimiento, aprovisionamiento, actualizaciones, entre otros.

- Big Data, esta tendencia que ya ha crecido con el paso del tiempo, hace referencia como tal a las técnicas o herramientas necesarias para un correcto manejo de grandes volúmenes de datos, tratando como tal de reducir los tiempos de procesamiento de estos y obtener resultados más rápida y fácilmente.

4. CONSTRUCCIÓN DE UNA APLICACIÓN MODERNA

El presente capítulo tratará lo relacionado con la creación de una aplicación moderna para la web, tratando de detallar los pasos necesarios para lograrlo, mostrando no solamente las tecnologías utilizadas sino también los requerimientos e instrucciones para conseguir una aplicación moderna.

4.1. Descripción de la aplicación

La aplicación que se desarrolla es una plataforma que muestra información relacionada al clima, permitiendo a los usuarios registrados obtener información climática acerca de los países que sean de su preferencia.

Las funcionalidades principales que se encontrará el usuario al entrar a la aplicación es que podrá registrar su información personal para poder acceder a la plataforma, además podrá administrar sobre qué países desea recibir información, permitiendo visualizar solamente los países en los cuales se ha interesado.

Por último, se debe aclarar que la construcción de esta aplicación es solamente con fines educativos por lo cual solo cuenta con funciones básicas necesarias para mostrar cómo se implementa la metodología 12 factor app.

4.2. Arquitectura elegida

La arquitectura tecnológica seleccionada para la construcción de nuestra aplicación moderna está conformada por tres elementos importantes que

cubren tres áreas necesarias las cuales son la visualización, el procesamiento de datos y almacenamiento de estos.

El área de visualización, que es la encargada de interactuar con el usuario, se encuentra desarrollada con Angular, que permite construir aplicaciones web.

El área de procesamiento de datos, que se encarga de recibir los datos que proporciona el área anterior y transformarlos para su posterior almacenamiento, se encuentra construida con Python, utilizando el *framework* denominado Flask.

Por último, nos encontramos con el área de almacenamiento de datos, la cual nos proporciona un lugar adecuado donde guardar nuestra información y que persista por el tiempo que se encuentre funcional o se desee. Esta área se encuentra realizada en una base de datos de tipo relacional denominada MariaDB.

4.3. Implementación del factor código base

El código base que se utiliza consta de dos aspectos importantes a ser mencionados, en primer lugar, es el software o sistema encargado de controlar las versiones, que en este caso específico es Git, en segundo lugar, se debe mencionar la plataforma elegida para utilizar dicho control de versiones, y para ello se eligió a GitHub.

Luego de ya tener seleccionadas las dos herramientas anteriores se procedió a la creación de dos repositorios dentro de la plataforma, este paso es sumamente importante ya que permite tener una estructura inicial

imprescindible para una aplicación construida bajo la metodología de los 12 factores.

Los repositorios creados corresponden a cada una de las funcionalidades que se tienen, en primer lugar, se estableció un repositorio para el desarrollo de la interfaz de usuario construida con Angular y el segundo repositorio corresponde al apartado del procesamiento de datos, el cual se desarrolló en Python.

4.3.1. Comandos de Git utilizados

Lo comentado en el apartado anterior se realizó utilizando una serie de comandos que permitieron manejar el comportamiento del repositorio, es por ello por lo que a continuación se describen los comandos más importantes que se utilizaron.

4.3.1.1. Git clone

Este comando permitió como tal realizar una copia del repositorio existente en el entorno remoto en un entorno local, esto brinda la posibilidad de poder trabajar o desarrollar la aplicación desde cualquier entorno local si se tienen los accesos requeridos.

4.3.1.2. Git branch

Este comando permitió la creación de nuevas ramas dentro de los repositorios realizados, esto se hace necesario debido al tipo de flujo de trabajo utilizado, además nos permitió consultar las ramas que ya se tenían dentro del entorno de trabajo.

4.3.1.3. Git checkout

Este comando permitió cambiar de una rama a otra, esto se hace necesario al estar trabajando en diversas ramas, por lo cual este comando fue de suma importancia.

4.3.1.4. Git add

Este comando se utilizó para poder agregar todos los cambios realizados dentro del repositorio local, esto es importante ya que si no es utilizado no se guardarán los cambios en el siguiente commit realizado.

4.3.1.5. Git commit

Este comando permite establecer los puntos de control dentro del repositorio y brindando así la posibilidad de regresar si se presenta la necesidad de retornar a este punto.

4.3.1.6. Git push

Este comando se utilizó para poder guardar los *commits* realizados y enviarlos al repositorio remoto, de tal manera que la información se almacenaba dentro de GitHub.

4.3.1.7. Git pull

Este comando permitió recibir todas las actualizaciones que se realizaban dentro del repositorio remoto, manteniendo actualizado el repositorio local.

4.3.1.8. Git merge

Este comando permitió fusionar las ramas, este comando se utilizó al momento de terminar una nueva funcionalidad, de tal manera que al momento de finalizar la construcción de una funcionalidad nueva se utilizaba este comando para poder fusionarla con el código principal.

4.3.2. Git flow

Git flow es una estrategia o flujo de trabajo el cual se utilizó dentro de nuestra implementación, esto fue debido a que nos permitió organizar de mejor manera nuestro desarrollo del software.

La implementación de esta estrategia se realizó siguiente el estándar previamente establecido, para ello se utilizó de base una publicación realizada por Atlassian.

Ahora bien, para poder comprender de mejor manera la forma en la cual se implementó dentro de nuestra investigación, se detalla a continuación el flujo de trabajo que se siguió.

4.3.2.1. Flujo de trabajo

El flujo de trabajo que presenta Gitflow nos permito organizar la codificación realizada, logrando establecer un estándar de cómo construir cada una de las funcionalidades de la aplicación.

Ahora bien, para comprender este flujo de trabajo hay que comenzar describiendo cada una de las ramas que utiliza para mantener todo más

organizado, por lo cual a continuación se describirán las ramas utilizadas dentro de nuestra implementación.

La rama *master* fue utilizada para almacenar las versiones finales de nuestro software que van a producción, de tal manera que dentro de esta rama almacenamos solamente versiones funcionales de la aplicación.

La rama *develop* nos permitió mantener nuestro código actualizado, esta rama se utilizó para poder mantener el código funcional previo a ser enviado a la sección de pruebas para crear una nueva versión de la aplicación.

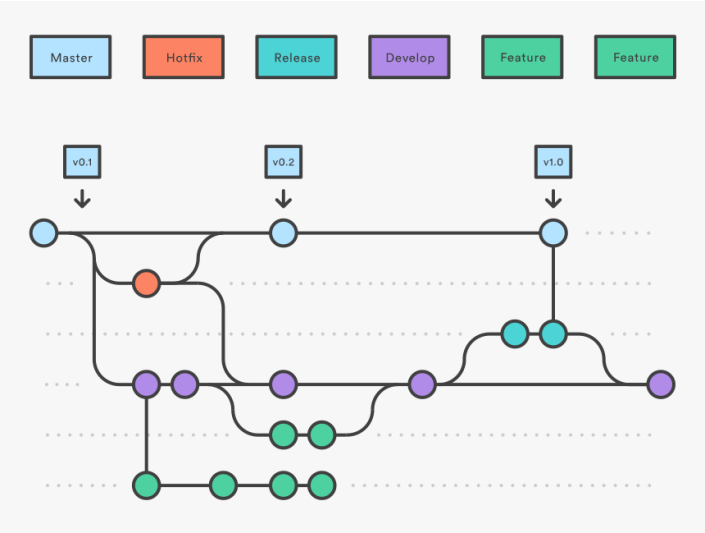
Las ramas *feature* nos permitieron agregar nuevas funcionales a nuestro software, por lo cual al momento de crear una nueva funcionalidad para la aplicación se creaba una rama de este tipo.

La rama *release* permitía realizar las pruebas para validar que una versión del código fuese funcional, de tal manera que nos permitió poder trasladar un código funcional hacia una nueva versión en la rama máster.

Por último, se tenían las ramas *hotfix* y *bugfix* estas ramas permitían resolver los errores encontrados, la rama *hotfix* resolvía errores encontrados en la rama master o en el código que iría a producción y la rama *bugfix* nos permitió resolver errores que se encontraron dentro de la rama *develop*.

Lo descrito anteriormente permite comprender de mejor manera como se implementó realmente el flujo de trabajo seleccionado, y para finalizar esta explicación se agrega una imagen para visualizar el flujo de trabajo de forma gráfica.

Figura 10. **Flujo de trabajo Gitflow**



Fuente: Atlassian (2020) *Flujo de trabajo de Gitflow.* Consultado el 06 de octubre de 2021. Recuperado de https://www.pngitem.com/pimgs/m/356-3561396_gitflow-workflow-git-flow-hd-png-download.png.

4.4. Implementación del factor dependencias

La aplicación propuesta utiliza para el *front-end* el *framework* Angular desarrollado en TypeScript, un superconjunto de JavaScript, y el *back-end* se implementa con el *framework* Flask desarrollado en Python.

4.4.1. Dependencias utilizando Angular

El *framework* de Angular y los componentes que utiliza están empaquetados como paquetes de npm, por lo que otras dependencias también se instalan utilizando este gestor.

4.4.1.1. Npm

Npm es el directorio de software más grande del mundo, en el cual desarrolladores de código abierto pueden compartir y descargar paquetes. Npm contiene tres componentes que son el sitio web, la interfaz de línea de comandos (CLI) y el directorio. Para utilizar npm lo primero que se debe hacer es instalar Node.js, debido a que npm está desarrollado en este lenguaje, luego dentro de la carpeta donde se desea gestionar los paquetes se debe ejecutar el comando npm init para inicializar el proyecto. Dicho comando genera un archivo llamado “package.json” el cual contiene la información necesaria para ejecutar el proyecto.

En el archivo “package.json” se encuentra el elemento “dependencies” que contiene una lista con los nombres de las dependencias necesarias para ejecutar el proyecto, así como también la versión instalada. También existe un apartado llamado “devDependencies” el cual se refiere a dependencias que son necesarias únicamente durante el desarrollo.

Figura 11. **Ejemplo de archivo package.json**

Fuente: elaboración propia, empleando Visual Studio Code 1.51.1.

4.4.2. Dependencias utilizando Flask

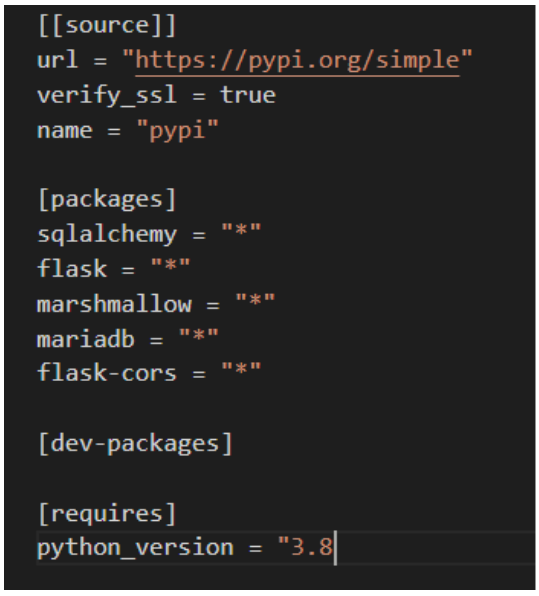
Flask es un *framework* desarrollado en Python, PIP es el administrador de paquetes por defecto de Python, sin embargo, a diferencia de npm este no ofrece la función de administrar las dependencias, por lo que en este lenguaje existen diferentes métodos para tener el control de estas. Anteriormente la práctica estándar era escribir en un archivo de texto requirements.txt una lista con las dependencias necesarias, que luego con el comando pip install -r requirements.txt serían instaladas dentro de un entorno virtual que también se creaba manualmente con el comando virtualenv. La administración de dicho archivo se vuelve poco práctica, especialmente cuando crece la cantidad de

dependencias o se deben hacer cambios en las versiones estipuladas, por esta razón se crea Pipenv, el cual permite administrar las dependencias de forma óptima.

4.4.2.1. Pipenv

Pipenv es una herramienta que automáticamente crea y maneja un entorno virtual para el proyecto en donde se utiliza, también permite instalar y desinstalar paquetes en el mismo. Pipenv administra las dependencias por medio de un archivo que genera llamado “Pipfile”, acompañado de otro archivo llamado “Pipfile.lock” que es usado para producir un *build* determinado.

Figura 12. **Ejemplo de archivo pipfile**



Fuente: elaboración propia, empleando Visual Studio Code 1.51.1.

La figura 12 muestra la forma en que está organizado un archivo Pipfile, las dependencias se manejan en la sección de packages y las dependencias de

desarrollo en la sección dev-packages, el asterisco significa que utiliza la versión más reciente del paquete, este puede ser sustituido por un numero de versión en caso sea requerido.

Pipfile.lock guarda la versión exacta de cada paquete instalado con sus hashes, los cuales se aseguran de instalar la misma versión cuando el proyecto pase a producción, también agrega todas las dependencias de los paquetes que se encuentran en el archivo Pipfile.

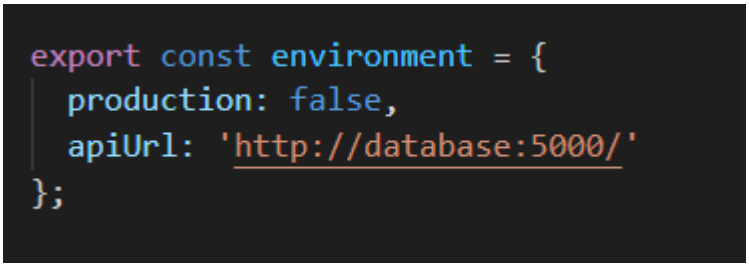
4.5. Implementación del factor configuraciones

El factor configuración permite parametrizar la aplicación según el ambiente en el que se esté ejecutando, cada lenguaje y *framework* se adapta a este factor con diferentes métodos y para este caso en particular se describirá como se puede cumplir utilizando Angular y Flask.

4.5.1. Configuración con Angular

Angular permite definir diferentes configuraciones para cada ambiente en el que se ejecute el proyecto, como *develop*, *stage* o *production*, esto lo hace por medio de los archivos *environment* los cuales se encuentran en el directorio del proyecto *./src/environments/*.

Figura 13. **Ejemplo de archivo environment.ts Angular**



Fuente: elaboración propia, empleando Visual Studio Code 1.51.1.

El archivo environment.ts es el que se utiliza por defecto para desplegar la aplicación, sin embargo, Angular permite remplazar dicho archivo según el ambiente que se configure. Para sustituir el archivo de configuración primero se debe crear con el nombre *environment*.<nombre del ambiente>.ts, por ejemplo *environment*.prod.ts, *environment.stage*.ts o *environment.testing*.ts según sea lo requerido. Luego se debe configurar el remplazo en el archivo angular.json en la raíz del proyecto, dentro del elemento *configurations* se debe agregar el ambiente con su respectivo *fileReplacement* que le indique el archivo correcto que debe utilizar para el despliegue.

Figura 14. **Ejemplo de archivo angular.json**

Fuente: elaboración propia, empleando Visual Studio Code 1.51.1.

Por último, para compilar la aplicación se utiliza el comando *ng build* y se le indica el ambiente en el que se ejecutará, por ejemplo, *ng build -- configuration = staging*.

4.5.2. Configuración con Flask

Flask no ofrece una forma nativa de implementar este factor, sin embargo, al ser Python existen algunos métodos alternativos, para este caso se ha escogido utilizar un archivo de Python que almacena las credenciales necesarias para su funcionamiento.

Figura 15. **Ejemplo de archivo angular.json**

Fuente: elaboración propia, empleando Visual Studio Code 1.51.1.

Se debe tomar en cuenta que las credenciales de la base de datos se envían por medio de variables de entorno, esto es conveniente ya que el código se ejecuta en contenedores y es sencillo configurar variables de entorno dentro de ellos. Para acceder a los valores de la configuración únicamente se debe importar el archivo de configuración y utilizar la variable que se necesite.

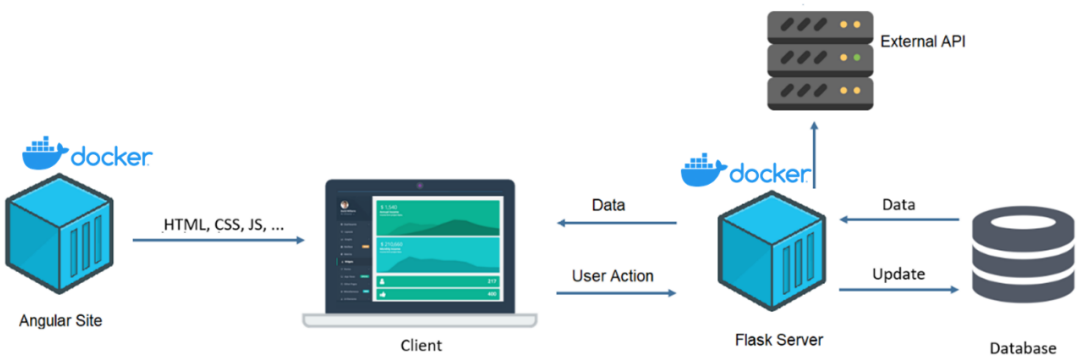
Figura 16. **Ejemplo acceso a la configuración en Python**

Fuente: elaboración propia, empleando Visual Studio Code 1.51.1.

4.6. Implementación del factor *backing services*

Este factor se cumple a consecuencia de utilizar contenedores y a que se cumple el factor de configuración, ya que los recursos se conectan a través de la red, el *front-end* y el *back-end* se comunican por medio de peticiones “http” y a su vez, el *back-end* se comunica con el API de OpenWeather y la base de datos utilizando las credenciales que se encuentran en la configuración.

Figura 17. **Flujo de la aplicación Weather App**



Fuente: elaboración propia, empleando Microsoft Paint.

4.7. Implementación del factor construir, desplegar y ejecutar

Luego de contener todo el código fuente dentro de los respectivos repositorios, cumpliendo así con el primer factor, se debe realizar la ejecución del software en un ambiente de producción.

Para ello se debe cumplir como tal con tres fases importantes, las cuales deben estar separadas una de otra para su correcto funcionamiento, es por ese motivo que a continuación se detalla cómo se logró cumplir con cada una de ellas.

4.7.1. Fase de construcción

La manera en la cual se realizó esta fase fue utilizando dos herramientas importantes, las cuales permitían automatizar y empaquetar el código fuente, siendo una de ellas Jenkins y la segunda Docker.

Jenkins es un servidor de automatización, el cual permite tomar el código fuente, que se encuentra en el repositorio, y transformarlo, mediante el uso de Docker, en contenedores.

Los contenedores que se generan se trasladan a un repositorio de contenedores en la nube, para esta ocasión se utilizó Docker Hub, guardando así cada uno de ellos para su próximo despliegue.

El proceso que se explicó anteriormente se detalla dentro de un archivo denominado jenkinsfile, en donde se encuentran un conjunto de comandos que permiten la automatización, para generar una mayor comprensión, a continuación, se explicará dicho archivo.

4.7.1.1. Jenkinsfile

La construcción del archivo jenkinsfile consta del seguimiento de una serie de reglas que generan una automatización sobre el proceso deseado, por ello se muestra en la siguiente imagen la estructura de uno de los archivos utilizados.

Figura 18. **Archivo jenkinsfile para la interfaz de usuario**

```
pipeline {
    agent none
    environment {
        DH_USERNAME = credentials('dockerhub_username')
        DH_PASSWORD = credentials('dockerhub_password')
    }
    stages {
        stage('Build') {
            agent {
                label 'master'
            }
            steps {
                echo "Realizando build previo a la subida"
                sh '''
                    docker build -t weatherapp_frontend .
                '''
            }
        }
        stage('TagAndPush') {
            agent {
                label 'master'
            }
            steps {
                echo "Tag and Push"
                sh '''
                    docker login --username ${DH_USERNAME} --password ${DH_PASSWORD}
                    docker tag weatherapp_frontend jacevel97/weatherapp_frontend:latest
                    docker push jacevel97/weatherapp_frontend:latest
                '''
            }
        }
    }
}
```

Fuente: elaboración propia, utilizando Visual Studio Code 1.61.2.

Inicialmente podemos visualizar que uno de los primeros parámetros es el agente, el cual permite definir el nodo encargado de realizar la automatización, el cual se deja en *none* para poder definirlo en cada una de las etapas explicadas posteriormente.

El siguiente parámetro encontrado es el *environment*, en donde podemos extraer las variables de entorno definidas dentro de Jenkins.

Por último, encontramos el parámetro de las etapas (*stages*), en el cual definimos cada una de las etapas que necesitamos dentro de nuestra automatización, para esta ocasión solamente se tienen dos, la primera que realiza la construcción del contenedor, y la segunda que conecta con Docker Hub y sube el contenedor a dicho repositorio.

Lo explicado anteriormente permite obtener una mejor perspectiva acerca de cómo se realiza la automatización de un proceso utilizando un servidor dedicado a ello.

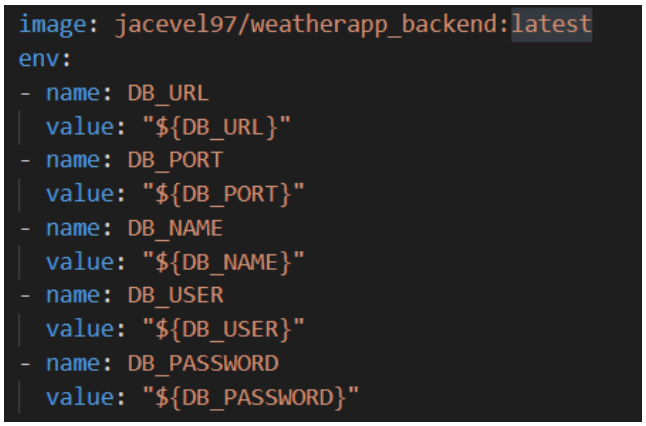
4.7.2. Fase de distribución

La presente fase, denominada distribución, se realizó utilizando los contenedores resultantes de la fase anterior, y combinándolos con la configuración que se deseaba, utilizando para ello las variables previamente establecidas.

El uso de contenedores facilita de nuevo la implementación de esta fase, esto es debido a que los contenedores permiten la implementación de una configuración personalizada, siendo así que se le puede establecer por cada despliegue que se realice.

El cumplimiento de esta fase se realiza al obtener los contenedores previamente generados, y aplicarles la configuración que se desea, tal como se muestra a continuación.

Figura 19. **Implementación de la etapa de distribución**



Fuente: elaboración propia, utilizando Visual Studio Code 1.61.2.

Es evidenciable en la ilustración previa, como se obtiene la imagen del contenedor, y se le aplica la configuración que se desea, logrando así una correcta fase de distribución.

4.7.3. Fase de ejecución

La última fase explicada dentro de este factor es la fase de ejecución, en donde se ejecuta la aplicación en un entorno de ejecución, para esto se utiliza como tal Kubernetes.

Luego de obtener los contenedores y aplicarles la configuración que se desea, el último paso es ejecutar la aplicación, para ello se utiliza un archivo de extensión yml previamente creado en donde se define el comportamiento que tendrá la ejecución que se desea.

Un aspecto importante que se determino es que, al momento de construir el contenedor, ya no se permite la modificación de este, logrando de esta manera que exista una separación completa de las diversas fases explicadas.

4.8. Implementación del factor procesos

El cumplimiento de este factor requiere de varias decisiones al momento de diseñar la aplicación:

- En primer lugar, se encuentra la arquitectura, la cual al separar los componentes en *front-end*, *back-end* y base de datos, permite que cada uno se ejecute de forma independiente.
- Como segundo punto, al decidir utilizar contenedores para ejecutar los componentes, cada proceso se mantiene aislado del otro y se comunican únicamente a través de la red.
- En tercer lugar, los desarrolladores conocen la metodología que se está utilizando, por lo que evitan almacenar información que deba ser persistente dentro de cada proceso, ya que estos deben ser sin estado.

4.9. Implementación del factor asignación de puertos

El presente factor se implementó al momento de construir la aplicación, esto se debe a que realmente los *frameworks* utilizados para la construcción lo permitían.

En primer lugar, para el caso de la interfaz de usuario (*front-end*), en donde se utilizó Angular, se publicó un servicio mediante la asignación de un puerto, en el cual quedo levantada la aplicación dentro del puerto 4200, no dependiendo así de un servidor web externo para poder funcionar.

Ahora bien, del lado del servidor para el procesamiento de información, desarrollado en Flask, se construyó la aplicación dentro del puerto 5000, siendo así que cumple con el presente factor.

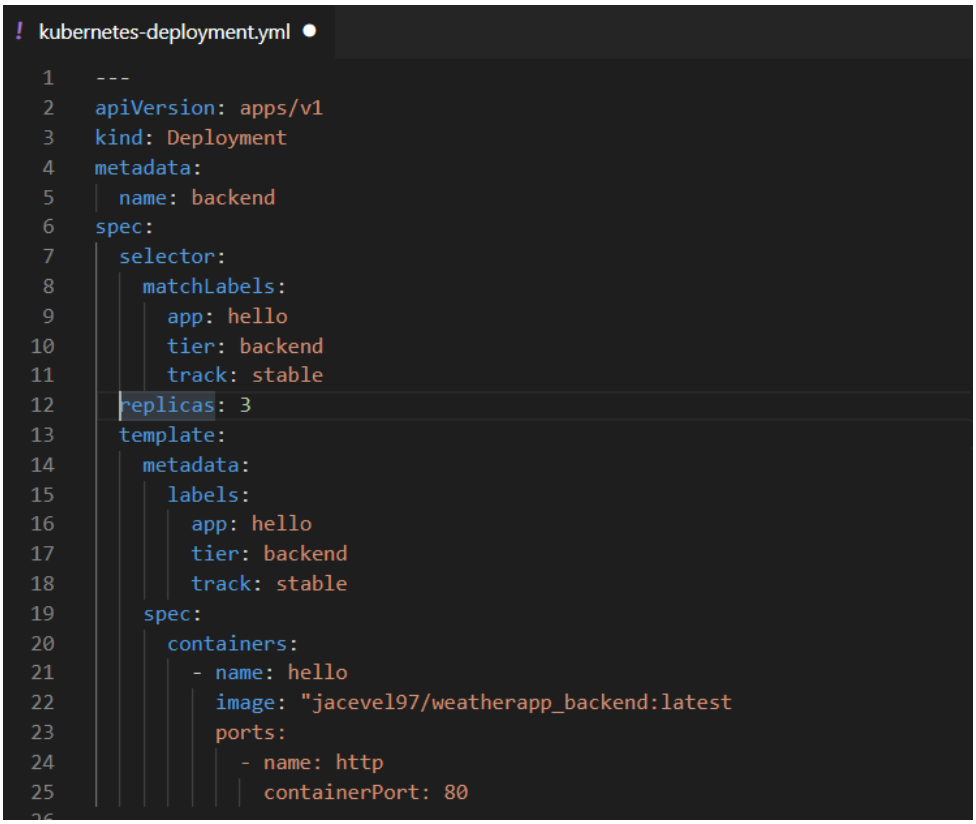
Un aspecto importante que fue tomado en cuenta al momento de la implementación de este factor fue establecer cada uno de los puertos como variables de entorno, esto debido al tipo de implementación que se realizaría, ya que toda la construcción se ejecutaría por medio de contenedores y la configuración se establecería dentro de un archivo *docker-compose*, permitiendo de esta manera poder crear un archivo específico para el tipo de construcción que se deseaba realizar, tanto para producción como para desarrollo.

Por todo lo anterior se puede determinar que la aplicación que se ha desarrollado es una aplicación autocontenida la cual no necesita un servidor web externo para poder funcionar de la manera correcta, además permite la construcción en diversos entornos según sea necesario, permitiendo el cambiar los puertos en el lugar donde se desee utilizar.

4.10. Implementación del factor concurrencia

El factor concurrencia permite que la aplicación escale según la cantidad de procesamiento que se requiera, este factor se cumple fácilmente gracias a la flexibilidad que Kubernetes ofrece, el archivo kubernetes-deployment.yml contiene los objetos de kubernetes que se desean crear, el objeto de tipo *deployment* posee la propiedad “replicas” el cual le indica al clúster de kubernetes cuantos contenedores debe desplegar.

Figura 20. **Ejemplo de configuración de replicas en Kubernetes**



Fuente: elaboración propia, empleando Visual Studio Code 1.51.1.

Para crear o aplicar cambios en esta configuración se debe ejecutar el comando kubectl apply -f kubernetes-deployment.yml, el cual se conecta al clúster de kubernetes y realiza los cambios según lo indique el archivo.

4.11. Implementación del factor desechabilidad

La implementación de este factor se realizó utilizando contenedores dentro de la construcción de nuestra aplicación, el uso de los contenedores proveía una mayor robustez al momento de crearlo.

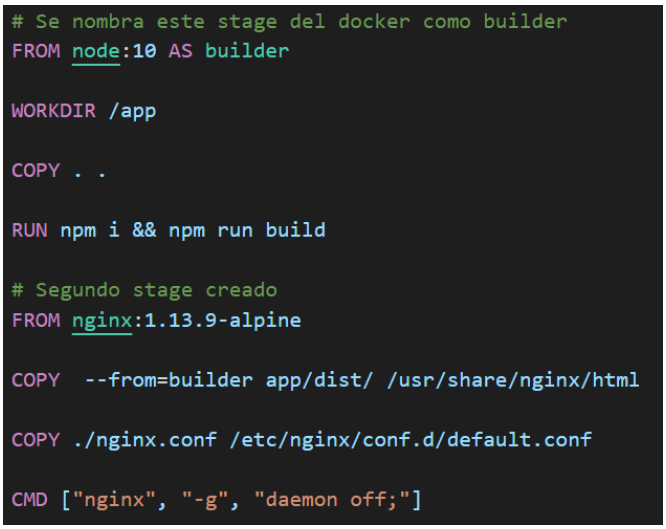
Ahora bien, para agilizar tanto el despliegue como la detención de los servicios, se utilizó Docker-Compose que permite el levantamiento de varios contenedores a la vez.

El procedimiento para poder implementar el uso de contenedores dentro de nuestra aplicación se explicará a continuación, mostrando los dos archivos más importantes que nos permitieron levantar nuestro servicio de una manera más rápida y robusta.

4.11.1. Contenedor de la interfaz de usuario

Para la creación del contenedor que presenta la interfaz de usuario, se utilizó como tal una serie de comandos dentro de un archivo denominado dockerfile, estos comandos realizaban el proceso necesario para que la interfaz del usuario funcionara, y se levantase en el momento que el contenedor iniciará su ejecución. Para entender este funcionamiento se adjunta a continuación una imagen del archivo ya mencionado.

Figura 21. **Archivo dockerfile para *front-end***



```
# Se nombra este stage del docker como builder
FROM node:10 AS builder

WORKDIR /app

COPY . .

RUN npm i && npm run build

# Segundo stage creado
FROM nginx:1.13.9-alpine

COPY  --from=builder app/dist/ /usr/share/nginx/html

COPY ./nginx.conf /etc/nginx/conf.d/default.conf

CMD ["nginx", "-g", "daemon off;"]
```

Fuente: elaboración propia, utilizando Visual Studio Code 1.61.2.

Los comandos presentados en la imagen anterior permiten como tal crear un nuevo contenedor para la interfaz de usuario, ahora bien, para entender de mejor manera, se explicará cada una de las dos áreas más importantes de la imagen anterior.

La primera área, que abarca desde la primera sección de comentarios hasta la segunda sección, permite instalar las dependencias necesarias, y además construir la solución, luego la segunda área permite copiar la construcción realizada dentro del servidor web nginx, de tal manera que se tiene un servidor web bastante óptimo.

La creación del contenedor para la interfaz de usuario se realiza tal y como se visualizó en el apartado anterior, y de esta manera ya podría crearse un contenedor para una interfaz visual desarrollada en angular.

4.11.2. Contenedor para el procesamiento de datos

El servidor encargado del procesamiento de datos también debe estar dentro de un contenedor, para ello se mostrará a continuación como está constituido el archivo dockerfile que permite la construcción del contenedor correspondiente.

Figura 22. **Archivo dockerfile para *back-end***

```
FROM python:3.8

WORKDIR /usr/src/

RUN pip install pipenv

COPY . .

RUN pipenv lock --keep-outdated --requirements > requirements.txt

RUN pip install -r ./requirements.txt

EXPOSE 5000

CMD ["./bootstrap.sh" ]
```

Fuente: elaboración propia, utilizando Visual Studio Code 1.61.2.

Los comandos que se pueden visualizar en la imagen previa permiten como tal la construcción de un contenedor para el servidor de procesamiento de datos, ahora bien, para entender de mejor manera, se explicará de manera generalizada cómo funciona el archivo previamente mostrado.

En primer lugar, se tiene dentro de las primeras líneas el establecimiento de la versión y tipo de imagen a utilizar para crear el contenedor, luego se realiza una instalación necesaria para el establecimiento de las dependencias,

para luego copiar todo el contenido dentro del contenedor, por último, se realiza la instalación de las dependencias necesarias para el funcionamiento del servidor y se expone el puerto a utilizar.

Tal como se mostró en el texto anterior funciona el dockerfile para la creación del presente servidor, logrando así mantener en contenedores tanto la parte desarrollada en Angular como lo desarrollado en Python.

4.12. Implementación del factor paridad en desarrollo y producción

La idea principal dentro del cumplimiento del presente factor nos indica que se debe reducir la diferencia existente dentro del ambiente de desarrollo y el ambiente de producción, tratando de abarcar tres áreas principales.

Debido a lo anterior, se muestra el detalle de cómo se lograron reducir las diferencias notables dentro de los ambientes ya mencionados, estas diferencias se obtuvieron de la publicación realizada por Adam Wiggins (2017) en su estudio de la metodología The twelve-factor app.

4.12.1. Tiempo

La primera diferencia detectada es el tiempo, el cual tiende a ser un factor importante en el desarrollo de software, logrando reducirlo por medio de la implementación de Jenkins, el cual automatiza la construcción de los contenedores.

Jenkins, no solo permite que el proceso de generación de nuevos contenedores sea más sencillo, sino además reduce de gran manera el tiempo

de despliegue, ya que, al momento de eliminar las tareas de construcción y distribución de contenedores, genera un gran ahorro de tiempo.

Debido a lo anterior, desde el momento de la construcción de una nueva funcionalidad, el tiempo que se tarda en llegar a un despliegue puede contarse en minutos.

4.12.2. Personal

Esta diferencia se puede resolver de una manera no tan compleja, esto se consigue por medio del involucramiento de los desarrolladores dentro del despliegue, tratando de observar cómo se comporta en producción.

En este caso específico nos encontrábamos involucrados en ambos ambientes, de tal manera que podíamos entender tanto el ambiente de desarrollo como el de producción, generando un mejor panorama del software desarrollado.

Algo importante a tomar en cuenta es que, aunque no siempre suceda lo descrito previamente dentro proyectos de una magnitud mayor, es importante que se permita a los desarrolladores involucrarse dentro del despliegue para entender de mejor manera el panorama completo del software a desarrollar.

4.12.3. Herramientas

La última diferencia encontrada nos habla acerca de las herramientas que se trabajan, debido a que en muchas ocasiones las herramientas utilizadas tienden a variar dependiendo del entorno donde se utilizan.

Lo descrito anteriormente se resolvió al implementar herramientas similares en ambos entornos, por ejemplo, se utilizó el mismo sistema operativo (Linux), el mismo tipo de base de datos y herramientas dentro de del despliegue en ambos.

Por último, queda mencionar que la resolución de estas tres principales diferencias permite que tanto el ambiente de desarrollo como el de producción sean lo más parecido posibles, logrando resolver diferencias de tiempo, personal y herramientas.

4.13. Implementación del factor logs

Este proyecto posee dos ambientes de ejecución, siendo estos desarrollo y producción. En ambos ambientes se debe tener la posibilidad de ver el comportamiento de la aplicación, tal visibilidad está determinada por la tecnología que se utilice para desplegar la aplicación, en este caso para el ambiente de desarrollo se utilizó docker-compose y para producción kubernetes.

4.13.1. Logs con docker-compose

Docker-compose permite visualizar fácilmente los logs de cada container, se puede tener una vista en tiempo real de la aplicación utilizando el comando docker-compose up, este comando despliega los contenedores y muestra en la línea de comandos los historiales que se generan. Por otra parte, se puede ejecutar docker-compose up -d, este comando no muestra los logs en tiempo real, pero se puede acceder a ellos con el comando docker-compose logs o si se requiere ver los logs de un contenedor en específico se puede utilizar el comando docker-compose logs <nombre del contenedor>.

4.14. Implementación del factor administración de procesos

El uso de contenedores permite crear ambientes idénticos para el despliegue de componentes de la aplicación, por lo tanto, ejecutar una tarea de gestión o administración como un proceso de una sola vez requiere únicamente el desplegar un contenedor y ejecutar la tarea dentro del mismo. De acuerdo con el factor de desechabilidad, los contenedores están diseñados para ser descartados en cualquier momento y no afectar el desempeño de la aplicación, por lo que una vez terminada dicha tarea, el contenedor puede destruirse.

En el ambiente de desarrollo, se puede ingresar a la terminal del contenedor con el comando docker-compose exec backend sh, en este ejemplo se inicia una terminal del contenedor *back-end* definido en el archivo docker-compose.yml. Al estar dentro del contenedor se tiene acceso a un entorno de ejecución idéntico en donde se encuentra el código y la configuración necesaria para realizar cualquier tarea de administración.

El ambiente de producción también ofrece una herramienta similar al de desarrollo, conectado al clúster de kubernetes se puede ejecutar el comando kubectl exec --stdin --tty --*namespace* my-app-*namespace* <*deploy-name*> -- /*bin*/*bash*, este comando inicia un terminal en donde se pueden realizar ejecutar la tarea de gestión, la configuración y el código que contendrá el contenedor corresponderá a la configuración escrita para producción.

5. CURSO SOBRE LA METODOLOGÍA THE TWELVE FACTOR APP

El estudio realizado sobre la metodología The Twelve-Factor App a lo largo de este documento permite evidenciar una forma de aplicarla sobre un problema en concreto, pero aun así quedan ciertas áreas prácticas que no se cubren del todo, es por ello por lo que se desarrolló un curso que cubría el uso de esta metodología desde un punto de vista más ilustrativo, logrando de esta manera completar algunas áreas que no se visualizaron dentro del presente documento.

El curso desarrollado busca enseñar el cómo implementar la metodología The Twelve-Factor App de una manera teórica-practica de forma que cualquier persona que complete este curso podrá comprender e implementar esta metodología.

5.1. Características

El curso previamente mencionado proporcionará a la persona que lo reciba un mayor conocimiento sobre la metodología que se ha estudiado, es por ello por lo que a continuación se describen las características principales del curso realizado.

- Enfoque teórico – práctico: el curso que se ha desarrollado cuenta con un enfoque que cubre ambas áreas, tanto el conocimiento teórico como la implementación práctica logrando de esta manera explicar a mayor

detalle la conformación de los factores correspondientes a la metodología.

- Implementación real: el desarrollo del curso que se presenta cuenta como tal con una implementación de una aplicación web, mostrando así el correcto funcionamiento de los temas que se estudian dentro de cada video.
- Contenido didáctico y amigable: la presentación del contenido busca que las personas que visualicen el curso puedan comprenderlo y absorber la mayor cantidad de información posible, es por ello por lo que la construcción de este se enfocó en presentar toda la información de una forma entendible y amigable al estudiante.
- Información estructurada: toda la información presentada dentro del curso se encuentra estructurada, facilitando así la recepción de la información, y evitando que exista confusión al momento de aprender sobre estos temas.

5.2. Camino por seguir

La recepción de este curso no debe ser el final del camino sobre el aprendizaje sobre la metodología The Twelve-Factor App, esto es debido al continuo avance de la tecnología que tiende a presentar nuevas herramientas e información que permiten optimizar de mejor manera la construcción de aplicaciones modernas, es por ello por lo que esto se debe visualizar este curso como una base inicial del camino que se puede seguir.

Debido a lo anterior a continuación se detallan una serie de recomendaciones orientadas a ilustrar algunas opciones para continuar con el aprendizaje inicial que se obtiene de este estudio.

- Certificaciones: según una publicación de la universidad Columbia Southern el obtener una certificación permite visualizar el compromiso de una persona con la profesión que maneja, siendo así que el obtener una certificación en las áreas que cubre la metodología estudiada permitirá reconocer el conocimiento que se ha adquirido, un ejemplo de una certificación importante por realizar es una relacionada a DevOps.
- Actualización continua: la investigación continua sobre las nuevas tendencias que surgen es un apartado importante, es por ello que se recomienda el siempre estar al pendiente sobre las tecnologías que aparecen, buscando siempre estar al día con los nuevos conocimientos relacionados.
- Búsqueda de proyectos relacionados: es recomendable el poder trabajar en proyectos en los cuales se pueda aplicar la metodología previamente estudiada, esto permite asegurar el conocimiento que se posee utilizándolo en el ámbito practico.

Las recomendaciones previas muestran un posible camino a continuar, pero las posibilidades que se tienen para aumentar el conocimiento sobre esta metodología son varias, y se recomienda al lector el poder establecer una ruta continua de aprendizaje que refuerce su conocimiento actual y le permita mejorar día tras día.

5.3. Publicación del material de aprendizaje

La publicación del material desarrollado busca el poder compartir la información que el curso contiene, de tal manera que cualquier persona que se encuentre interesada en aprender pueda acceder a él sin mayor inconveniente.

Lo anterior genera la necesidad de encontrar un sitio adecuado para la publicación de la información, y esto es solucionado con el uso de plataformas web de enseñanza, por ello a continuación se describe las plataformas elegidas y la razón por la cual se ha elegido una plataforma e-learning para la publicación del material desarrollado.

5.3.1. Plataforma e-learning para publicación de un curso

De acuerdo con la ITU (2021), para el 2021 se observó un crecimiento de 800 millones de personas con acceso a internet, esto a su vez reflejo un aumento del uso de plataformas web para el aprendizaje, esto se observa en una investigación realizada por la GMI.

Debido a lo anterior se observa que las plataformas e-learning ofrecen un sitio adecuado para la publicación del curso desarrollado ya que brinda la posibilidad de acceso desde múltiples lugares y por múltiples personas, es por ello por lo que a continuación se describen las plataformas elegidas, ilustrando que beneficios ofrece y como poder acceder a ella.

5.3.1.1. Plataformas elegidas

Se han seleccionado varias plataformas *e-learning* debido a que de esta manera se ofrece la oportunidad de acceso desde múltiples lugares, brindando al usuario la elección de la plataforma con la cual se sienta más a gusto.

Es por lo anterior que a continuación se listan las plataformas seleccionadas para la publicación del curso, mostrando en total cinco plataformas.

- Canvas Network
- Khan Academy
- Udemy

La publicación del curso en dichas plataformas se conformará en dos fases, iniciando con la publicación inicial del material de aprendizaje en Udemy, para luego terminar con la publicación en las dos últimas plataformas mostradas previamente.

El acceso a este contenido se detalla a continuación en el siguiente apartado, ilustrando la forma en la cual se podrá obtener el curso.

5.3.1.2. Acceso

El curso desarrollado que se encuentra en las plataformas *e-learning* podrá ser accedido por cualquier persona en cualquier ubicación, para ello solamente se debe realizar una búsqueda por medio del título e inscribirse en el mismo dentro de cada portal educativo.

El título escogido para el curso desarrollado es el siguiente: metodología *Twelve-Factor App*, este título será el mismo para cada una de las plataformas mencionadas anteriormente.

Por último, se presentará un listado de pasos a seguir para acceder al curso creado, ilustrando la manera correcta de realizarlo.

- El primer paso por realizar es el ingreso en la plataforma elegida para la recepción del curso, para ello se puede utilizar cualquier navegador y situarse dentro de la plataforma.

- El segundo paso por realizar es crear un usuario dentro de la plataforma elegida, esto permitirá acceder al portal *e-learning* y realizar la búsqueda correspondiente del curso.
- El tercer paso por realizar es la búsqueda del curso dentro del portal, para ello se debe colocar el título del curso dentro de la sección de búsqueda del portal *e-learning*, logrando así ubicarlo.
- El cuarto paso es la suscripción al mismo, esto permitirá recibir todos los videos correspondientes del curso desarrollado.
- Por último, solo queda la visualización de cada uno de los tutoriales dentro de la plataforma escogida, tratando de aprender lo más posible de ello.

El listado previo muestra un camino a seguir para recibir el curso sobre la metodología The Twelve-Factor App*,* las personas que deseen recibirlo pueden seguir los pasos mostrados y con ello poder obtener el conocimiento que se desea entregar.

Gracias a todo lo anterior, podemos comprender de mejor manera la metodología The twelve-factor app, logrando entender por igual el apartado teórico y práctico de esta metodología, permitiendo así conseguir una implementación resiliente en la creación de aplicaciones web modernas.

CONCLUSIONES

1. Se construyó una aplicación que permitía poder visualizar la información del clima a los usuarios registrados, siguiendo la metodología The twelve-factor app.

2. Se determinó que al utilizar la metodología The twelve-factor app en la construcción de una aplicación moderna se puede obtener una mayor resiliencia, soportando múltiples fallos y recuperándose rápidamente de ellos.

3. Se estableció que no existen mayores riesgos al utilizar la metodología The twelve-factor app sobre la construcción de una aplicación moderna. Únicamente los riesgos que pueden surgir se deben a una incorrecta aplicación de la metodología.

4. Se creó un curso al estilo MOOC, el cual contiene todo el proceso que se debe llevar a cabo para poder implementar la metodología The twelve-factor app sobre la construcción de una aplicación moderna.

5. Se publicó el curso creado en una plataforma gratuita para que cualquier persona pueda acceder a ellos en el momento que lo deseen.

RECOMENDACIONES

1. Implementar la metodología de los 12 factores con diversos *stack* tecnológicos para visualizar la adaptación que la metodología posee, debido a que se puede adaptar a cualquier *stack* actual.

2. Utilizar otro servicio web para el control de versiones como por ejemplo GitLab, esto se debe a que ofrece diversos servicios ya incluidos dentro de su plataforma como la implementación de CI/CD sin necesidad de utilizar otro software externo.

3. Desarrollar aplicaciones web de mayor tamaño o alcance, esto permitirá conocer experimentar de mejor manera todos los beneficios que la metodología The Twelve-Factor App ofrece.

4. Continuar el estudio sobre la metodología The Twelve-Factor App, logrando descubrir y documentar nuevos usos y funcionalidades que ofrece esta metodología.

REFERENCIAS

1. Amazon (2020) *AWS Certification.* AWS. Recuperado el 11 de agosto de 2021 de https://aws.amazon.com/es/certification/.

2. Canalys (2020) *Global cloud infrastructure market Q4 2020.* Canalys. Recuperado el 29 de agosto de 2021 de https://www.canalys.com/newsroom/global-cloud-market-q4-2020.

3. Casasola Girón, N. E. (2019) *Diseño de un modelo de integración continúa utilizando Jenkins para proyectos de desarrollo en una empresa de telecomunicaciones para el módulo de contabilidad* (Tesis de licenciatura). Universidad de San Carlos de Guatemala, Guatemala. Recuperado de http://www.repositorio.usac.edu.gt/13836/1/Norman%20Eduardo%20Casasola%20Gir%C3%B3n.pdf.

4. Cloud native computing foundation (2021) *Training.* Cloud native computing foundation. Recuperado el 29 de septiembre de 2021 de https://www.cncf.io/certification/training/.

5. Díaz González, J.R. (2018) *Diseño e implementación de guía de migración DevOps desde una cultura de desarrollo de software tradicional.* (Tesis de maestría). Universidad Mariano Gálvez, Guatemala. Recuperado de https://glifos.umg.edu.gt/digital/97878.pdf.

6. Galdámez, G. (14 de junio de 2021) *Cómo ser un Super Desarrollador: introducción a git-flow.* Medium. https://medium.com/snappler/c%C3%B3mo-ser-un-super-desarrollador-introducci%C3%B3n-a-git-flow-parte-1-7a3f7027d3fd.

7. Gamboa Cruz, G. A. (2021) *Análisis y diseño de infraestructura para sistema de visualización de información utilizando prácticas de automatización de procesos mediante DevOps* (Tesis de licenciatura). Universidad de San Carlos de Guatemala, Guatemala. Recuperado de http://www.repositorio.usac.edu.gt/15766/1/Gustavo%20Adolfo%20Gamboa%20Cruz.pdf.

8. Google (2018) *Google Cloud certification*. Google Cloud. Recuperado el 11 de agosto de 2021 de https://cloud.google.com/certification.

9. Khan, A. (2017) *Why DevOps: Problems the Practice Helps Solve in your Business.* SPR. Recuperado el 10 de Agosto de 2021 de https://spr.com/why-devops-and-why-you-need-it-now/.

10. Kim G., Behr K. y Spafford G. (2013) *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*.

11. Kim G., Debois P., Willis J., Humble J. y Allspaw J. (2016) *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*.

12. Lozhko, M. (2021) *15 Top web development trends in 2021*. Lanars. https://lanars.com/blog/top-web-development-trends.

13. Microsoft (03 de enero de 2021) *Cloud Design Patterns.* Microsoft. Recuperado el 10 de agosto de 2021 de https://docs.microsoft.com/en-us/azure/architecture/patterns/.

14. Microsoft (2020) *¿Qué es DevOps?* Azure Microsoft. Recuperado el 09 de agosto de 2021 de https://azure.microsoft.com/es-es/overview/what-is-devops/#devops-overview.

15. Microsoft (2020) *Microsoft Certified: Azure Fundamentals*. Microsoft. Recuperado el 11 de agosto de 2021 de https://docs.microsoft.com/en-us/learn/certifications/azure-fundamentals/.

16. Oracle (2020) *What is Cloud Native?* Oracle Cloud Infraestructure. Recuperado el 10 de agosto de 2021 de https://www.oracle.com/cloud/cloud-native/what-is-cloud-native/.

17. Pacheco, D. (2021) Los 10 frameworks de desarrollo web más solicitados en 2021. *Diegoo*. https://diegooo.com/frameworks-de-desarrollo-web-mas-solicitados-en-2021/.

18. Scanlan, T. (2019) *Defining the modern application*. VMWare. Recuperado el 07 de agosto de 2021 dehttps://octo.vmware.com/defining-modern-application/.

19. Wiggins, A. (2017) *The twelve-factor app*. the twelve-factor app. Recuperado el 07 de agosto de 2021 de https://12factor.net.