# Universidad Nacional de San Juan

## FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES


## Departamento de Informática

Licenciatura en Ciencias de la Computación

Trabajo final:

**“Comparación de la eficiencia de Tecnologías Tradicionales de Front End versus Funciones Serverless”**

Autor: Villa Rivera, Darciel Owen Francis
Asesor: Lic. Rodríguez Nelson

San Juan, Argentina
2023

# DATOS PERSONALES DEL AUTOR

**APELLIDO Y NOMBRE:** Villa Rivera, Darciel Owen Francis

**FECHA DE NACIMIENTO**: 11 de Abril de 1991

**LUGAR DE NACIMIENTO:** San Juan, Argentina

**DOMICILIO:** Pablo Rojas MK 8 S/N PB B B° Enoe Bravo, Santa Lucia

**TELÉFONO:** +54 9 264 585 6250

**E-MAIL:** darylvilla10@gmail.com

**ESTUDIOS SECUNDARIOS:**
Colegio Central Universitario "Mariano Moreno"

**ESTUDIOS UNIVERSITARIOS:**
Lic. en Ciencias de la Computación

*A mi esposa, hija, mi vieja y hermanos: por la paciencia, comprensión y contención a lo largo de toda mi carrera…*
*A mis amigos y compañeros que siempre han estado apoyando…*
*A los profesores que siempre me apoyaron y estimularon, permitiendo hacer de esta carrera mi profesión y parte de mi vida…*
*A quienes no menciono y olvido, que siempre han estado presente…*

***Muchas Gracias…***

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS Y TABLAS

# Capítulo I

# Introducción

# 1.1. Introducción General

Inicialmente las empresas construían su infraestructura informática con el modelo on-premise, que significa que son propietarios de sus propios servidores y de toda la instalación correspondiente (Red, impresoras, dispositivos de conectividad, etc.). Posteriormente surgió Cloud Computing. Este último modelo de computación se ha consolidado como una arquitectura que permite la provisión de recursos bajo demanda y de forma elástica; es ampliamente utilizada y ha demostrado su efectividad.

Luego de varios años de que las diferentes proveedoras de servicios Cloud, ofrecieran sus servicios y la tecnología evolucionara, surgió Serverless Computing, como un nuevo paradigma de desarrollo de software en el Cloud. Surge como una evolución tecnológica impulsada por los microservicios.

En sus inicios Cloud Computing se ofrecía en una arquitectura monolítica, luego se implementan máquinas virtuales, posteriormente contenedores y por último Serverless (en algunos casos se conoce como función como servicios). La misma está enfocada en proveer una arquitectura que permita la ejecución de funciones arbitrarias con mínima sobrecarga en la administración del servidor y soportada bajo la programación orientada a eventos.

Por otro lado, hace unos años se dividió la arquitectura de desarrollo, una parte interactúa con el usuario (Front End), se conoce como “lado del cliente”, y Back End, es la parte que se conecta con la base de datos y el servidor que utiliza dicho sitio web, por eso se dice que Back End corre del lado del servidor. Tanto Front End como Serverless han transitado caminos diferentes, sin embargo, para determinadas aplicaciones son comparables, con la diferencia de que Serverless maneja al Back End como servicio.

Se han realizado muchos trabajos sobre comparar Serverless con el Cloud tradicional, sin embargo, no se ha analizado convenientemente cuales son las ventajas y desventajas comparado con una infraestructura on-premise.

El objetivo del presente trabajo final, es comparar la eficiencia y las ventajas económicas del uso de las tecnologías tradicionales de Front End con Serverless Computing.

# 1.2. Antecedentes

Cloud Computing es un modelo de arquitectura y también de negocios, dado que permite brindar recursos bajo demanda tanto de infraestructura, como de plataforma y de software, además de nuevos servicios que han surgido recientemente.

Cloud Computing evolucionó desde un modelo monolítico, luego pasó al uso de micro servicios, para aparecer recientemente el término Serverless Computing. Como puede apreciarse en la figura 1.

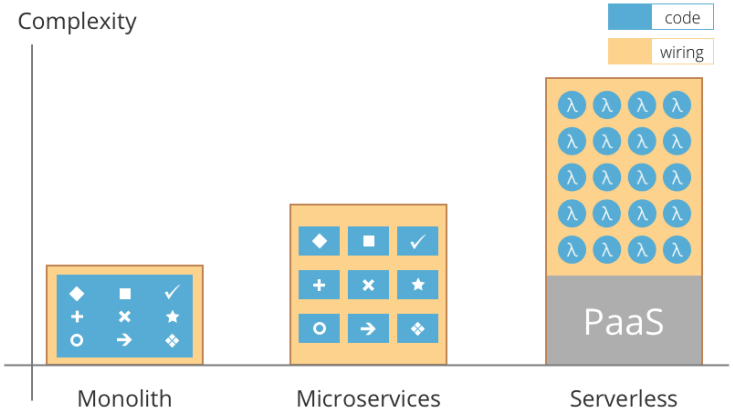



**Figura 1:** Evolución de Cloud Computing [1]

El término puede dar lugar a equívocos, dando la impresión de que se habla de computación sin necesidad de disponer de servidor. Esto no es exactamente así, en Serverless Computing existen por supuesto servidores, pero en vez de ser administrados por una persona, están gestionados de forma automatizada por un proveedor cloud.

Si bien la idea de este modelo bajo demanda o de “pago por uso” similar a lo que es actualmente Serverless se remonta al año 2.006 con una plataforma llamada Zimki, uno de los primeros usos del término Serverless nace de Iron, en 2012 con su producto llamado IronWorker, una plataforma de trabajo bajo demanda basada en contenedores.

Desde ese entonces hay nuevas implementaciones de Serverless tanto en la nube privada como en la pública. Primero hubo ofertas de BaaS (Backend as a Service), como Parse en 2011 y Firebase en 2012 (adquirido por Facebook y Google, respectivamente). En noviembre de 2014 se lanzó AWS Lambda, y a principios de 2016 se vieron anuncios para IBM OpenWhisk en Bluemix (ahora IBM Cloud Functions, con el proyecto principal de código abierto Apache OpenWhisk), Google Cloud Functions y Microsoft Azure Cloud Functions. Huawei Function Stage se lanzó en 2017. También hay numerosos frameworks de código abierto. Cada uno

de estos, tanto públicos como privados, tiene conjuntos únicos de tiempos de ejecución de lenguaje y servicios para manejar eventos y procesar datos. [1]

Por otro lado, debido a que es una tecnología reciente, no existen gran cantidad de trabajos de investigación. La mayoría de los mismos analizan aspectos de la arquitectura o de las ventajas en el costo del desarrollo de aplicaciones en ambientes específicos, pero pocas publicaciones se enfocan en la programación de aplicaciones en Plataformas Serverless.

Un mapeo sistemático realizado en 2018 [2], indica qué 2016 es cuando surge realmente Serverless. Durante ese año solo se publicaron 7 trabajos de investigación, y además del total de trabajos solo 4 tienen que ver con FaaS (Functions as a Service).

Debido a que la migración de aplicaciones a las arquitecturas Serverless no es una tarea sencilla, algunos autores proponen el modelo de actor, es un método computacional con un patrón muy popular para crear aplicaciones concurrentes. El modelo simplifica el trabajo de componer en paralelo y ejecuciones distribuidas mediante el uso de una unidad básica de cálculo: el actor [3].

Otro trabajo similar, publicado en 2019, desarrolla un análisis sistemático sobre plataformas y herramientas para FaaS (Functions as a Service) [4]. En dicha publicación se afirma que teniendo en cuenta el ítem modelos de programación solo existen 3 trabajos. Por otro lado, también afirma que la mayoría de las publicaciones (36%) se basan en AWS Lambda para realizar los desarrollos y pruebas. De las tres publicaciones citadas ninguna trata el tema de programación específicamente. El trabajo de Abhinav Jangda et al [5], presenta una formalización de la computación Serverless, la publicación de Meissner et al, [6] presenta una plataforma para aplicaciones Serverless con soporte de computación reactiva, y por último los investigadores  Piotr Moczurad y Maciej Malawski [7] desarrollan un frameworks para computación Serverless.

## 1.3. Justificación

En los últimos años se puede notar una clara tendencia hacia el desarrollo de aplicaciones web debido a las ventajas que estas ofrecen y a la utilización de servicios cloud para alojarlas. Cada vez crece más la demanda de estas aplicaciones, por lo que las empresas suelen querer reducir cada vez más los tiempos de desarrollo. Con la aparición de la Computación Serverless se han logrado reducir las operaciones de mantenimiento y configuración de los servidores. Los programadores, por lo tanto, pueden concentrarse en el código y no pierden tiempo con temas correspondientes a la configuración de los servidores. Otra ventaja de utilizar los Servicios Serverless tiene que ver con la reducción de costos que una infraestructura de este estilo puede ofrecer. No se necesita tener un servidor que esté activo permanentemente y generando costos incluso en los tiempos donde no se está ejecutando código.

Por otro lado, los desarrolladores Web hace un tiempo se han dividido en dos especialidades Front End y Back End. Estos términos se refieren a la separación de preocupaciones entre la capa de presentación (Front End) y la capa de acceso a datos (Back End) de una pieza de software , o la infraestructura física o hardware. En el modelo cliente-servidor , el cliente generalmente se considera el Front End y el servidor generalmente se considera el Back End, incluso cuando parte del trabajo de presentación se realiza en el mismo servidor

Tanto Serverless Computing como Front End han evolucionado desde lugares diferentes, pero ambas tienen que ver con el desarrollo Web. Sin embargo, Serverless tiene algunas ventajas notables en determinadas aplicaciones, pero no es adecuado si no se puede llevar todo a funciones independientes que cooperan e invocar al Back End como servicio. Analizar la conveniencia de Serverless comparado con el desarrollo clásico de Front-End es una temática que no ha sido abordada ni por la industria ni por la academia y es precisamente la propuesta central de este trabajo final.

# 1.4. Objetivos del Trabajo

## 1.4.1. Objetivo General

- Comparar la eficiencia en el uso de recursos, performance y costos, entre el desarrollo Front End convencional y las Funciones Serverless.

## 1.4.2. Objetivos Específicos

- Investigar las plataformas Serverless más populares del mercado
- Analizar en profundidad la plataforma para desarrolladores Front End Vercel, sus características, herramientas adicionales y desarrollo de aplicaciones.
- Evaluar los parámetros de eficiencia del sistema utilizando Front End.
- Desarrollar el sistema sobre la plataforma Serverless y sobre la plataforma on-premise.
- Determinar los test necesarios para comparar la eficiencia y otros parámetros de evaluación.
- Análisis de los resultados y comparación de los mismos en ambas plataformas.

# Capítulo II

# Marco Teórico

# 2.1. Introducción

Con el objetivo de comprender de mejor manera las Funciones Serverless y las Tecnologías Tradicionales, dentro del ámbito del Front End es que se procede durante toda esta sección a aclarar conceptos relacionados y tecnologías asociadas al área temática. La distribución del presente marco teórico se ha diagramado de manera jerárquica, empezando desde lo más general, básico y necesario, hasta finalizar con la explicación de los costos del uso de diversas Plataformas Serverless y la eficiencia de un sitio web. Los conceptos explicados en las subsecciones siguientes se tornan fundamentales para llevar a cabo la comparación de la eficiencia de las Funciones Serverless con las Tecnologías Tradicionales, en relación al Front End.

# 2.2. Historia del Front End

La historia del desarrollo Front End viene de la mano de la Web, y surge a partir del desarrollo de páginas HTML. La estructura de toda página web se divide en dos partes igualmente importantes, del lado del cliente el Front End y del lado del servidor el Back End. El Front End es la sección que interactúa con los usuarios que visitan dicha página web, es decir, es la parte visible. Muestra el diseño, enlaces, botones, efectos, entre otros contenidos permitiendo a los visitantes navegar por donde lo deseen.

Es la encargada de la experiencia del usuario, englobando y mostrando todo el trabajo de diseño web. Por lo general, reúne en su interior hasta 3 lenguajes diferentes: HTML, CSS y JavaScript. Cada uno orientado a determinados fines, se suman para conseguir el resultado final que aparece por la pantalla de cada usuario que ingresa en una web, por lo que la interfaz debe ser sencilla de usar, atractiva y funcional.

Se hace mención a la Historia de Front End comenzando HTML, el mismo se origina en 1980 cuando el físico Tim Berners-Lee (trabajador de CERN, *Centro Europeo para la Investigación Nuclear*) propuso un nuevo sistema de "*hipertexto*" con la finalidad de compartir documentos. Tim Berners-Lee se encontró con la problemática de poder facilitar el acceso a la información desde cualquier computadora de CERN o de otras instituciones. Nacen el protocolo HTTP y las páginas HTML, ya que se buscaba una forma sencilla y estándar de acceder a dicha información permitiendo que las páginas estuvieran unidas entre sí, enlazadas (dando origen al concepto de *Hipertexto*).

Para la definición del estándar HTML, Tim se basó en el lenguaje de marcado SGML (*Standard General Markup Language*), el mismo define las reglas de etiquetado y estructura general. HTML originariamente surge como un Sistema de

Hipertexto Interno, luego Tim junto con el Ingeniero de Sistemas Robert Cialliau presentan World Wide Web (*W3*) un Sistema de Hipertexto en Internet. Posteriormente W3 se convertiría en el organismo encargado de crear todos los estándares relacionados con la web, llamado W3C (*World Wide Web Consortium*). [8]

Desde la creación de SGML se observó la necesidad de definir un mecanismo que permitiera aplicar diferentes estilos a los "*documentos electrónicos*". Surgen las Hojas de Estilo, el lenguaje de las mismas se populariza con el crecimiento de HTML, pero era muy dificultoso crear documentos con la misma apariencia en diferentes navegadores, debido a la falta de un estándar para la definición de los estilos.

Por ese motivo W3C propuso la creación de un lenguaje de hojas de estilos específico para HTML. Las dos propuestas más relevantes fueron CHSS (*Cascading HTML Style Sheets*) realizada por Harton Wium Lie y SSP (*Stream-based Style Sheet Proposal*) llevada a cabo por Bert Bos. Finalmente Lie y Bos se unieron para definir un nuevo lenguaje, tomando lo mejor de cada propuesta, surge CSS (*Cascading Style Sheets*).

En 1995 W3C decide apostar por el desarrollo y estandarización de CSS, siendo incorporado a su grupo de trabajo de HTML. La adopción de CSS por los navegadores llevó un largo periodo de tiempo, el mismo año que se publicó CSS 1, Microsoft lanza su navegador Internet Explorer 3.0, disponiendo de un soporte bastante reducido de CSS. El primer navegador con soporte completo de CSS 1 fue la versión para Mac de Internet Explorer 5, que se publicó en el año 2000. [9]

Actualmente la versión usada por los navegadores es CSS3, con dicha versión se ha obtenido una actualización importante en el estándar de definición de estilos para documentos HTML, incluye características potentes tanto para aplicar aspecto avanzado en elementos de una página como para ayudar a realizar una maquetación más precisa.

El objetivo de CSS3 es que los programadores sean capaces de aplicar estilos a los documentos HTML de forma versátil, sin necesidad de hack y facilitar la separación de código entre contenido y presentación. [10]

También cabe mencionar que en 1995 se introdujo JavaScript como una forma de agregar programas a las páginas web. Fue una idea muy novedosa debido a que en los comienzos de World Wide Web, HTML era bastante simple y los usuarios consultaban mayormente contenido basado en texto.

A principios de los años 90, la mayoría de los usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. Esa velocidad era más que suficiente para la época salvo que se quisiera descargar imágenes de cierto tamaño. La web en aquel entonces no ofrecía gran cosa más que servir como una gran biblioteca donde los usuarios consultaban mayormente contenido basado en texto.

Luego comenzaron a desarrollarse las primeras aplicaciones web, donde los formularios en ese entonces eran muy complejos y la velocidad de navegación era muy lenta. Surge JavaScript como la necesidad de dotar a la web de capacidades interactivas, permitiendo crear una interfaz de usuario activa, lo que ofrece retroalimentación a los visitantes según navegan por sus páginas. Por ejemplo, es común usar JavaScript en la validación de formularios para asegurar que la información introducida es válida, sin necesidad de enviar ninguna información al servidor, el programa realiza los cálculos necesarios ahorrando tiempo y recursos del lado del servidor.

Con JavaScript se puede crear sobre la marcha páginas HTML personalizadas, dependiendo de las acciones ejecutadas por el usuario. Por ejemplo, si se necesita crear una web de seguros, con JavaScript se puede realizar consultas en el servidor sin necesidad de recargar la página, mostrar opciones personalizadas, lanzar eventos en función del día y hora en donde la persona que ingrese se encuentre, etc.

Las ventajas más destacadas de JavaScript son:

- Es un lenguaje muy sencillo.
- Es rápido, tiende a ejecutar las funciones inmediatamente.
- Cuenta con múltiples opciones de efectos visuales.
- Es soportado por los navegadores más populares y es compatible con los dispositivos más modernos, incluyendo el navegador nativo para iPhone y Android.
- Es muy versátil, puesto que es muy útil para desarrollar páginas dinámicas y aplicaciones web.
- Es una buena solución para poner en práctica la validación de datos en un formulario.
- Es multiplataforma, puede ser ejecutado de manera híbrida en cualquier sistema operativo móvil.
- Es el único lenguaje que permite trabajar modo FullStack en cualquier tipo de desarrollo de programación.

En la actualidad los navegadores no son los únicos en los que se utiliza JavaScript. También es posible usarlo en un entorno servidor. Por ejemplo, las bases de datos, como MongoDB y CouchDB, usan JavaScript como su lenguaje de scripting y consulta. Además, varias plataformas para la programación de escritorio y servidor, en particular el proyecto Node.js proporcionan un entorno para la programación de JavaScript fuera del navegador.

La evolución de JavaScript sigue creciendo mediante nuevas implementaciones, frameworks y librerías que se utilizan para diferentes usos. Todo esto hace imposible para los desarrolladores poder conocer todo de este lenguaje, es aquí donde los frameworks se presentan para facilitar su trabajo.

Estos son los más populares:

- **Angular.js**: Es desarrollado por Google, se ha encargado de traer orden a las aplicaciones JavaScript y potenciar las Arquitecturas SPA.

- **React.js**: Librería de Facebook orientada a la gestión de Interfaces de usuario. Muy extendido a la hora de desarrollar aplicaciones móviles.

- **Meteor.js**: Pensado para desarrollar aplicaciones JavaScript que puedan ejecutar su código en entornos cliente y servidor.

- **jQuery.js**: uno de los clásicos, han pasado muchos años desde que apareció en el mercado y se convirtió en el standard de facto a la hora me manipular el árbol DOM. [11]

# 2.3. Cloud Computing

La Computación en la Nube es la entrega de servicios informáticos, incluidos servidores, almacenamiento, bases de datos, redes, software, análisis e inteligencia, a través de Internet (la nube) para ofrecer una innovación más rápida, recursos flexibles y economías de escala.

Un concepto fundamental detrás de la computación en la nube es que la ubicación del servicio y muchos de los detalles, como el hardware o el sistema operativo en el que se ejecuta, son en gran medida irrelevantes para el usuario. La metáfora de la nube se tomó prestada de los esquemas de redes de telecomunicaciones antiguas, en las que la red telefónica pública (y posteriormente Internet) a menudo se representaba como una nube para indicar que la ubicación no importaba, solo era una nube de cosas.

Por lo tanto, en lugar de que la empresa u organización posea su propia infraestructura informática o centros de datos, pueden alquilar el acceso por ejemplo para aplicaciones, almacenamiento, a un proveedor de servicios en la nube.

Uno de los beneficios de usar servicios de computación en la nube es poder evitar el costo inicial, además de la complejidad de poseer y mantener una propia infraestructura de TI, simplemente se debe pagar por lo que usa, cuando lo usa.

A su vez, los proveedores de servicios de computación en la nube pueden beneficiarse de importantes economías de escala al brindar los mismos servicios a una amplia gama de clientes.

Los servicios de computación en la nube cubren una gran cantidad de opciones, desde los conceptos básicos de almacenamiento, redes, hasta el procesamiento del lenguaje natural y la inteligencia artificial, así como las aplicaciones de oficina estándar. Casi cualquier servicio que no requiera que se esté físicamente cerca del hardware de la computadora que se está utilizando se puede entregar a través de la nube, incluso la computación cuántica.

Este modelo se está convirtiendo en la opción predeterminada para muchas aplicaciones, los proveedores de software ofrecen cada vez más sus aplicaciones como servicios a través de Internet en lugar de productos independientes a medida que intentan cambiar a un modelo de suscripción.

El analista tecnológico Gartner predice que, hasta la mitad del gasto en los mercados de software de aplicaciones, software de infraestructura, servicios de procesos comerciales e infraestructura de sistemas se habrá trasladado a la nube para 2025, frente al 41 % en 2022. Se estima que casi dos tercios del gasto en el software de la aplicación será a través de la computación en la nube, frente al 57,7% en 2022. [12]

## 2.4. Serverless

En la actualidad todo tipo de empresa, compañía o entidad necesita la tecnología informática para su funcionamiento, tan importante es que puede representar el éxito o el fracaso de las mismas. Debido a eso, es necesario de una infraestructura que dé soporte a todo su sistema informático.

Cada empresa en lo que a TICs se refiere, cuenta con una arquitectura que se ha ido actualizando con el paso de los años, gracias a los avances tecnológicos. Por arquitectura se define a toda plataforma tecnológica que utiliza el sistema para brindar servicios al usuario final, convirtiéndose en la base del sistema informático de cualquier organización.

Dicha plataforma como el resto de la tecnología ha evolucionado de manera radical. Uno de los elementos de la misma siempre estuvo acompañando a las empresas en su evolución tecnológica, el servidor. Hace varios años la mayoría de las compañías eran completamente responsables de las operaciones y aplicaciones del servidor. Dando lugar, a la misma a mantener un departamento dedicado a la configuración y mantenimiento de los conmutadores de red, servicios de base de datos, entre otras. Posteriormente llegó la computación en la nube, donde los datos y aplicaciones se reparten en nubes de computadoras, es decir, son alojados en cientos de miles de servidores pertenecientes a los gigantes de Internet, Google, Microsoft, IBM, Oracle, Amazon, etc. Permitiendo a las empresas la reducción de costes y de personal, incrementando la productividad y aumentando la disponibilidad del propio producto. Las arquitecturas comenzaron a ser más complejas, pero más flexibles aumentando la potencia de cómputo y de almacenamiento. Los años siguientes permitieron a la nube evolucionar y

asentarse dentro de las compañías, generando nuevos conceptos como IaaS, PaaS y Saas. Conceptos que aportaron mayor rapidez, facilidad y flexibilidad al desarrollo de las arquitecturas, dando lugar a lo que hoy en día se conoce como Cloud Computing.

En los comienzos del Cloud Computing, las empresas, llamadas en los siguientes modelos como proveedores, se enfocan en proporcionar su infraestructura como servicio (IaaS por sus siglas en inglés). El modelo IaaS permite a los usuarios tener acceso a máquinas virtuales, almacenamiento y redes a través de internet [13]. Es decir, el usuario final no administra ni controla la infraestructura base, pero puede controlar dispositivos de almacenamiento, sistemas operativos, aplicaciones desplegadas y opcionalmente componentes de red, como por ejemplo un firewall o un enrutador [14].

Posteriormente los proveedores fueron añadiendo servicios principalmente enfocados a desarrolladores, surge el modelo PaaS (Plataforma como servicio o Platform as a Service). Este incorpora un nivel de abstracción sobre el modelo anterior, donde los proveedores gestionan no sólo los equipos informáticos y las máquinas virtuales, sino también los sistemas operativos y los entornos donde se ejecutan dichas máquinas virtuales. Debido a esto, los desarrolladores pueden crear aplicaciones sobre la plataforma usando las herramientas de software otorgadas por el proveedor sin necesidad de ocuparse de los sistemas subyacentes [13].

Además de los dos modelos antes mencionados, surge un tercero orientado especialmente a los usuarios finales, conocido como modelo SaaS (Software como Servicio o Software as a Service). A través del mismo un usuario individual o una empresa puede hacer uso de aplicaciones de software y servicios, es decir, consiste en ofrecer aplicaciones a través de Internet. Por lo tanto, quien implementa dicho modelo dispone de toda la infraestructura necesaria para soportar las aplicaciones de software que ofrece [15]. Se puede observar Google Docs, para la creación y distribución de documentos; Dropbox, para almacenamiento de archivos; Spotify, para reproducción de música bajo demanda y Netflix, para la visualización de series y películas también bajo demanda en modo streaming [13].

La Figura 2 muestra las diferencias entre los Modelos de Servicios Cloud antes mencionadas.

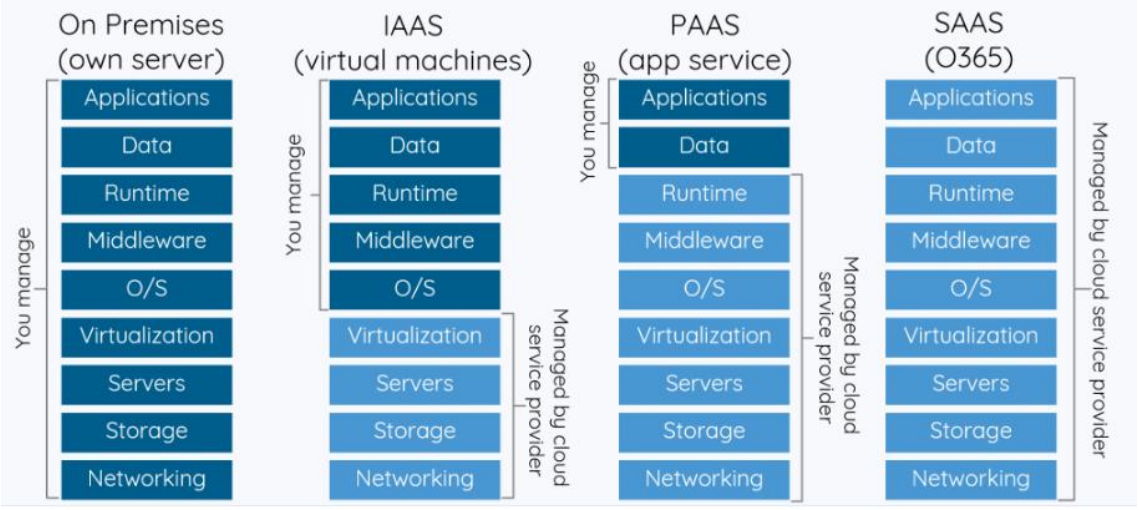


**Figura 2**: Diferencias entre los modelos de servicios Cloud [16]

La nube desde su aparición ha evolucionado mucho, sin embargo, sigue acompañando el elemento referenciado anteriormente, el servidor. Dando lugar a que las empresas, compañías o entidades continúen haciendo hincapié en la administración y funcionamiento de los servidores donde se ejecutan sus aplicaciones. ¿Si el próximo gran paso en la evolución de la nube sea liberarlas de esa preocupación? O aún más ¿Ofrecer toda una arquitectura que los abstraiga?

Surge un nuevo concepto donde las compañías pueden ejecutar sus aplicaciones y servicios sin necesidad de administrar servidores. Con el objetivo de que los desarrolladores puedan enfocarse en el producto principal, en lugar de preocuparse por el funcionamiento, administración y control de los mismos o los tiempos de ejecución, tanto en la nube como en las instalaciones donde se alojen. Otorgando una gran ventaja y solucionando los requisitos funcionales. Este concepto se llama “Serverless”.

Serverless ha dado lugar a una revolución en el desarrollo de servicios web, donde los desarrolladores ya no tienen que preocuparse por el escalado de la infraestructura, pudiendo enfocarse enteramente en sus aplicaciones. Comienza un nuevo modelo llamado FaaS (Funciones como Servicios), incorporando el uso de servicios gestionados por el proveedor, con el objetivo de que el usuario no necesite interactuar y escalar la infraestructura subyacente.

El modelo de servicio FaaS [13] consiste en la definición de funciones en un lenguaje de programación determinado, sobre un Proveedor Cloud. Dichas funciones se ejecutan sobre contenedores creados para responder a peticiones, generalmente provocadas por la interacción de los usuarios, y eliminadas automáticamente cuando dejan de usarse. Debido a esto no es necesario disponer de un servicio constantemente activo, como sucede con los modelos anteriormente mencionados, asegurando una escalabilidad de forma transparente a los usuarios. Ellos solo deben abonar por el tiempo de ejecución de sus funciones en base a los recursos asignados a las mismas.

A continuación, se hará mención de las principales empresas encargadas de ofrecer aplicaciones y servicios sin necesidad de administrar servidores, La Figura 3 muestra los respectivos Logos:

amazon web services

Microsoft Azure

IBM Cloud Functions

Google Cloud Platform

Vercel

**Figura 3:** Ejemplo de Proveedores que Soporta Serverless

## 2.4.1. Amazon Web Services (AWS)

Amazon Web Services es la infraestructura de Amazon, ofrece un amplio conjunto de productos globales basados en la nube, entre los cuales incluye recursos informáticos, de almacenamiento, bases de datos, análisis, redes, dispositivos móviles, herramientas para desarrolladores, entre otros, en diferido, disponibles en segundos, con precios de pago por uso. AWS dispone de más de 200 servicios, desde almacén de datos a herramientas de implementación, y desde directorios hasta entrega de contenido. Además, ofrece una plataforma de infraestructura escalable de alta fiabilidad y de bajo costo que permite trabajar a más de un millón de clientes activos en más de 240 países y territorios.

La infraestructura de Amazon Web Services está compuesta por regiones y zonas. Una región de AWS es una ubicación física en el mundo, compuesta por varias zonas. Las mismas a su vez, constan de uno o varios centros de datos, cada uno de ellos con alimentación, redes y conectividad, que se alojan en instalaciones independientes. Estas zonas ofrecen la capacidad de operar bases de datos y aplicaciones de producción con una disponibilidad, tolerancia a errores y escalabilidad mucho mayores que las que ofrecería un centro de datos local. La nube de AWS funciona en 80 zonas en 25 regiones geográficas de todo el mundo, con planes anunciados para más zonas y regiones.

Cada región de Amazon se ha diseñado para que esté totalmente aislada de las demás regiones. Con ello se consigue la mejor tolerancia a errores y estabilidad posible. Por otro lado, también cada zona está aislada, pero dentro de una región están conectadas a través de conexiones de baja latencia. AWS proporciona la flexibilidad necesaria para colocar las instancias y almacenar los datos en varias regiones geográficas, así como en varias zonas dentro de cada región de AWS. Además del sistema de alimentación ininterrumpida (SAI) independiente y las instalaciones de generación auxiliar in situ, los centros de datos ubicados en diferentes zonas se han diseñado para recibir suministro de subestaciones independientes, a fin de reducir el riesgo de que un suceso en la red eléctrica afecte a más de una zona. [17]

En relación con la seguridad y la conformidad Amazon Web Services posee una política de responsabilidad compartida entre la plataforma y el cliente. Este modelo compartido puede aliviar la carga operativa del cliente, ya que AWS opera, administra y controla los componentes del Sistema Operativo Host y la capa de virtualización, hasta la seguridad física de las instalaciones en las que funcionan los servicios. Por otro lado, el cliente asume la responsabilidad y la administración del

Sistema Operativo Invitado (incluidas las actualizaciones y los parches de seguridad), de cualquier otro software de aplicaciones asociado y de la configuración del firewall del grupo de seguridad que ofrece AWS. [18]

Amazon Web Services ofrece herramientas en las siguientes categorías:

- **Cloud Computing:** Lo necesario para la creación de instancias y el mantenimiento, o el escalado de las mismas, usando el servicio de Amazon EC2.

- **Bases de Datos:** distintos tipos de bases de datos pueden permanecer en la nube mediante el servicio Amazon RDS, que incluye distintos tipos a elegir como MySQL, PosgreSQL, Oracle, SQL Server y Amazon Aurora, o Amazon DynamoDB para NoSQL.

- **Creación de Redes virtuales:** Permite la creación de redes privadas virtuales a través de la nube, gracias principalmente al servicio Amazon VPC.

- **Aplicaciones Empresariales:** Amazon WorkMail es el servicio de correo empresarial que ofrece Amazon, al que pueden unirse otros servicios como Amazon WorkDocs y Amazon WorkSpaces.

- **Almacenamiento y Gestores de Contenido:** Brinda tipos de almacenamiento diferentes, tanto para archivos con acceso regular, poco frecuente o incluso como archivo. Amazon S3 es el servicio principal, aunque complementan la oferta otros como Amazon Glacier o Amazon EBS.

- **Inteligencia de Negocios o Business Intelligence (BI):** Es el sistemas para análisis de datos empresariales a gran escala y otros servicios para la gestión de flujos de datos.

- **Gestión de Aplicaciones Móviles:** Herramientas como Amazon Mobile Hub permiten la gestión, creación, testeo y mantenimiento de aplicaciones móviles a través de la nube.

- **Seguridad y Control de Acceso:** Se pueden establecer autenticaciones en varios pasos para poder proteger el acceso a sus sistemas internos, sea que estén en la nube o instalados de forma local en sus instalaciones. [19]

En la Figura 4 se muestra la Gestión de Recursos Informáticos, incluido el servicio AWS Lambda.

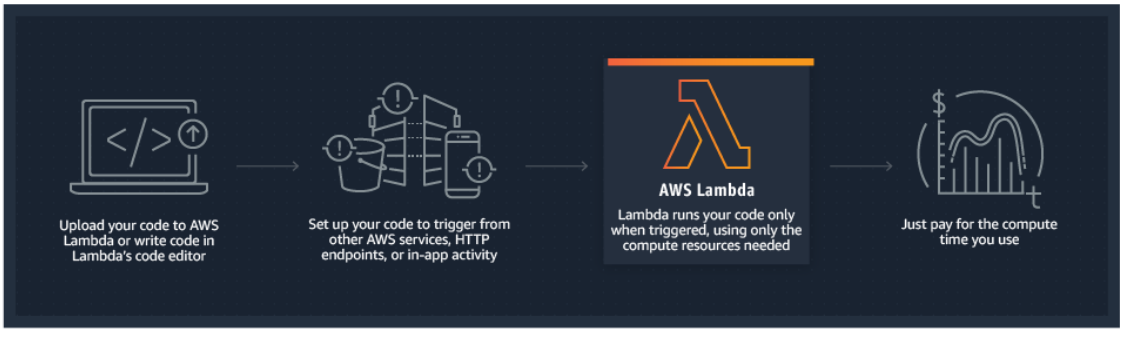


**Figura 4:** AWS. Lambda - Gestión de Recursos Informáticos

Lambda es uno de los servicios más importantes de Amazon Web Service, ya que permite ejecutar código sin administrar servidores. Lambda ejecuta el código en una infraestructura de computación de alta disponibilidad y realiza todas las tareas de administración de los recursos de computación, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como las funciones de registro. Además, permite ejecutar código para prácticamente cualquier tipo de aplicación o servicio de backend.

Cabe resaltar que Lambda ejecuta el código provisto por el desarrollador solo cuando es necesario, y escala de manera automática, desde unas pocas solicitudes por día hasta miles por segundo, en donde solo se abonará por el tiempo informático que se consuma, no se aplican cargos cuando el código no se está ejecutando. Es necesario mencionar que como Lambda administra sus recursos (memoria, CPU, red, entre otros) para poder brindar una combinación equilibrada de los mismos, el desarrollador no puede iniciar sesión en instancias ni personalizar el sistema operativo para ejecutar su código, ya que dicho servicio realiza las actividades operacionales y administrativas, e incluye la administración de la capacidad, la supervisión y el registro de las funciones. Si se necesita administrar los propios recursos informáticos, AWS ofrece otros servicios para satisfacer las necesidades. [20]

## 2.4.2. Microsoft Azure

Azure es una plataforma en la nube diseñada por Microsoft para simplificar el proceso de creación de aplicaciones modernas. Tanto si se decide almacenar de forma completa las aplicaciones, como ampliar las ubicadas localmente en la empresa con los servicios de Azure, ayudando a crear aplicaciones escalables, confiables y fáciles de mantener.

Además, admite los lenguajes de programación más populares que se usan hoy en día, como Python, JavaScript, Java, .NET y Go. Con una completa biblioteca de SDK y una amplia compatibilidad con las herramientas que ya usa, como VS Code, Visual Studio, IntelliJ y Eclipse, la plataforma está diseñada para aprovechar las aptitudes que el desarrollador ya posee y hacer que sean productivas de inmediato.

Azure provee arquitecturas serverless modernas llamadas Azure Functions, donde se busca simplificar la creación de soluciones para administrar flujos de trabajo orientados a eventos, ya sea responder a solicitudes HTTP, controlar cargas de archivos en almacenamiento de blobs o procesar eventos en una cola. El

desarrollador solo debe escribir el código necesario para controlar el evento sin preocuparse por los servidores o el código de la plataforma. Además, puede beneficiarse de más de 250 conectores a otros servicios de Azure y de terceros para abordar los problemas de integración más difíciles.

Por otro lado, Azure le permite al programador u organización que utilice la plataforma, almacenar desde aplicaciones web y API, a bases de datos y servicios de almacenamiento. Admitiendo una gran variedad de modelos de almacenamiento, tanto servicios completamente administrados como contenedores y máquinas virtuales. Al usar los servicios de Azure totalmente administrados, las aplicaciones pueden beneficiarse de las ventajas de la escalabilidad, la alta disponibilidad y la seguridad integradas en Azure.

También se brinda la posibilidad de utilizar los servicios en la nube desde aplicaciones ubicadas localmente en la compañía, es decir, las aplicaciones locales pueden incorporar servicios de Azure para ampliar sus funcionalidades. Por ejemplo, se podría usar Azure Blob Storage para almacenar archivos en la nube, Azure Key Vault para guardar de forma segura datos importantes o sensibles de la aplicación o Azure Cognitive Search para agregar la funcionalidad de búsqueda de texto completo. Estos servicios están totalmente administrados por la plataforma y se pueden incorporar fácilmente a las aplicaciones existentes sin cambiar la arquitectura de aplicación actual ni el modelo de implementación. [21]

Si bien Azure contiene más de 100 servicios, a continuación, se describen algunos de los servicios más usados por los desarrolladores:

- **Almacenamiento y Proceso de Aplicaciones**
  - **Azure App Service**: Permite almacenar aplicaciones web y API de .NET, Java, Node.js y Python en un servicio de Azure totalmente administrado. Solo se tiene que implementar el código en Azure y la plataforma se encarga de toda la administración de la infraestructura, como la alta disponibilidad, el equilibrio de carga y el escalado automático.

  - **Azure Container Instances**: Permite ejecutar contenedores de Docker a petición en un entorno de Azure administrado y sin servidor. Dicho servicio es una solución para cualquier escenario que puede funcionar en contenedores aislados.

- **Datos**
  - **SQL de Azure**: Una versión totalmente administrada basada en la nube de SQL Server.

- **Azure Cosmos DB**: Una base de datos NoSQL basada en la nube totalmente administrada. Este servicio incluye varias API, incluidas las API compatibles con MongoDB, Cassandra y Gremlin.

- **Azure Database para PostgreSQL**: Un servicio de base de datos PostgreSQL totalmente administrado basado en PostgreSQL Community Edition.

- **Servicios cognitivos**
  - **Voz**: Permite transcribir la voz en texto legible, además realizar búsquedas o convertir texto en voz realista para interfaces más naturales.

  - **Form Recognizer**: Es un servicio de extracción de documentos que comprende formularios, permitiendo extraer rápidamente texto y estructura de los documentos.

  - **Servicio Cognitivo para el Lenguaje**: Utiliza el procesamiento de lenguaje natural (NLP) para identificar frases clave y realizar análisis de opiniones a partir del texto. [22]

Cabe destacar que el acceso a los servicios de Azure desde el código de aplicación se puede realizar de dos maneras diferentes, por medio de API REST o SDK. El SDK permite el acceso mediante programación a dichos servicios desde aplicaciones .NET, Java, JavaScript, Python y Go, se recomienda la utilización de este medio siempre que sea posible ya que simplifica el proceso de autenticación de la aplicación en Azure, al crear un objeto de cliente SDK donde se incluyen las credenciales correctas y se encarga de autenticar las llamadas a la plataforma. Además, posee un modelo de programación sencillo, en donde internamente el SDK se comunica con la API REST, y se encuentra integrado el control de errores, la lógica de reintento y la paginación de resultados. [23]

Por otro lado, los lenguajes de programación no compatibles con el SDK de Azure pueden usar directamente la API REST. Las API “Representational State Transfer” (REST) son endpoint de servicio que admiten conjuntos de operaciones HTTP (métodos), que proporcionan acceso de creación, recuperación, actualización o eliminación a los recursos del servicio. [24]

## 2.4.3. IBM Cloud Platform

La Plataforma de IBM llamada IBM Cloud®, combina Plataforma como Servicio (PaaS) con Infraestructura como Servicio (IaaS) para brindar una experiencia totalmente integrada. Permite la posibilidad de escalar y es compatible tanto con pequeños equipos, organizaciones de desarrollo como con grandes empresas. Es una plataforma que se encuentra implementada globalmente en centros de datos de todo el mundo**.**

IBM Cloud proporciona soluciones que posibilita niveles altos de cumplimiento, seguridad y gestión, con métodos y patrones de arquitectura comprobados para una entrega rápida. Esta plataforma se encuentra disponible en centros de datos de todo el mundo, con regiones multizona en América del Norte, América del Sur, Europa, Asia y Australia, donde se pueden implementar aplicaciones localmente con escalabilidad global.

Además, ofrece una nube pública abierta y segura para empresas con una plataforma de nube híbrida de próxima generación, datos avanzados y capacidades de inteligencia artificial, y una profunda experiencia empresarial en 20 industrias. Las soluciones están disponibles según las necesidades del desarrollador u organización para trabajar en la nube, de forma local o en una combinación de ambas:

- **Método Cloud**: Los recursos se ponen a disposición a través de Internet, en donde el hardware y la infraestructura son administrados por IBM®.

- **Nube Híbrida**: Es una combinación de pública y privada, ofreciendo flexibilidad de mover cargas de trabajo entre las dos en función de las necesidades comerciales y tecnológicas. IBM utiliza Red Hat OpenShift en IBM Cloud, una plataforma de contenedores de nube híbrida muy potente que permite desarrollar una aplicación una vez e implementarla en cualquier lugar. Además, con IBM Cloud Satellite, se puede crear un entorno híbrido que aporte la escalabilidad y la flexibilidad bajo demanda de los servicios de nube pública a las aplicaciones, y los datos se ejecutan en su nube privada segura.

- **Nube Privada Virtual (VPC)**: Se encuentra disponible como un servicio de nube pública que permite establecer un entorno informático privado. Con VPC, las empresas pueden definir y controlar una red virtual que está lógicamente aislada de todas las demás personas de la nube pública, creando un lugar privado y seguro dentro del mismo cloud.

Ya sea que el desarrollador necesite migrar aplicaciones a la nube, modernizar sus aplicaciones existentes mediante el uso de servicios en la nube, garantizar la resiliencia de los datos contra fallas regionales o usar nuevos paradigmas y topologías de implementación para innovar y crear sus aplicaciones nativas de la

nube, la arquitectura abierta de la plataforma está diseñada para adaptarse por completo al caso de uso que sea necesario.

Cabe mencionar que la plataforma IBM Cloud se compone de varios componentes que funcionan juntos para proporcionar una experiencia cloud coherente y fiable, alguno de los cuales son:

- Una consola robusta que sirve como interfaz para crear, ver y administrar los recursos en la nube.

- Un componente de gestión de identidades y accesos que autentica de forma segura a los usuarios, para los servicios de la plataforma y controla el acceso a los recursos de forma coherente en IBM Cloud.

- Un catálogo compuesto de cientos de productos compatibles.

- Un mecanismo de búsqueda y etiquetado para filtrar e identificar los recursos.

- Un sistema de gestión de cuentas y facturación para un uso exacto de planes de precios y protección segura contra fraudes con tarjetas de crédito. [25]

Por otro lado, alguna de las características que IBM Cloud Functions ofrece son:

- **Backends Serverless:** Permite exponer la lógica de aplicación al implementar microservicios sin servidor.

- **Backend Móvil:** Facilita a los desarrolladores de telefonía móvil acceder a la lógica en el lado del servidor y externalizar tareas de computación intensas a una plataforma en la nube escalable. Permitiendo implementar funciones en lenguajes como Swift y utilizarlas en el lado del servidor usando el propio SDK de iOS de la Plataforma.

- **Caso de Ejemplo Conversacional:** Permite implementar aplicaciones conversacionales sin servidor, como chatbots, pasando mensajes de conversación a funciones para un procesamiento posterior.

- **Tareas Planificadas:** Ejecutar funciones periódicamente. Donde se definen planificaciones siguiendo una sintaxis cronológica para especificar cuándo se deben ejecutar las mismas. [26]

## 2.4.4. Google Cloud Platform

Google Cloud Platform consiste en un conjunto de recursos físicos, como por ejemplo computadoras, unidades de disco duro, y recursos virtuales, como

máquinas virtuales (VM), los mismos se encuentran en diferentes ubicaciones de América del Norte, América del Sur, Europa, Asia y Australia. Estas a su vez, se dividen en regiones y zonas, pudiendo elegir la ubicación de las aplicaciones para satisfacer los requisitos de latencia, disponibilidad y durabilidad.

Las regiones son áreas geográficas independientes compuestas de zonas, ambas son abstracciones lógicas de los recursos físicos subyacentes que se proporcionan en uno o más centros de datos físicos. Es posible que estos centros de datos sean propiedad de Google (las cuales se encuentran en la página de ubicaciones de Google Cloud) o que puedan alquilarse a proveedores de centros de datos externos. Además, Google Cloud selecciona los centros de datos y diseña su infraestructura para proporcionar un nivel uniforme de rendimiento, seguridad y confiabilidad, sin importar si el centro de datos es propio o alquilado.

Por otro lado, una zona es un área de implementación para los recursos de Google Cloud dentro de una región, y se deben considerar como dominios únicos dentro de la misma. Para implementar aplicaciones tolerantes a errores con alta disponibilidad y ayudar a proteger contra fallas inesperadas, se recomienda implementar las aplicaciones en varias zonas de una región.

Los servicios y recursos de Google Cloud pueden estar administrados en varias regiones o bien ser zonales o regionales:

- **Recursos Multirregionales:** Google administra varios servicios de Google Cloud para que sean redundantes y se distribuyan dentro de las regiones y entre ellas. Estos servicios permiten optimizar la disponibilidad, el rendimiento y la eficiencia de los recursos. Como resultado, estos servicios requieren una compensación entre la latencia o el modelo de coherencia, además, están diseñados para poder funcionar después de la pérdida de una región.

- **Recursos Regionales:** Son recursos que se implementan de manera redundante en varias zonas dentro de una región, por ejemplo, las aplicaciones de App Engine o los grupos de instancias regionales administrados. Esto les permite tener una disponibilidad más alta en comparación con los recursos zonales.

- **Recursos Zonales:** Operan dentro de una sola zona. Las interrupciones zonales pueden afectar a algunos o a todos los recursos de esa zona. Un ejemplo de un recurso zonal es una instancia de máquina virtual (VM) de Compute Engine que reside dentro de una zona específica.

En general, si una región falla solo los clientes de esa región se ven afectados, a menos que posean productos multirregionales. En ese caso, no se ven afectados debido a que dispone de servicios redundantes y distribuidos en varias regiones. Cabe mencionar que Google Cloud tiene una arquitectura significativa con el objetivo de evitar fallas correlacionadas en todas las regiones.

Todos los servicios de Google Cloud se basan en herramientas internas como, herramientas de redes (dentro y fuera de los centros de datos), acceso a centros de datos y sistemas de autorización de identidad. Las mismas son resistentes a las interrupciones regionales, con el objetivo de que una región no se vea afectada si otras regiones no están disponibles. [27]

Google Cloud establece que todos los recursos que el desarrollador asigne a su aplicación deben pertenecer a un proyecto. El proyecto se considera la entidad organizadora de lo que se está compilando, están compuestos por la configuración, los permisos y otros metadatos para describir las aplicaciones. Además, los recursos de un mismo proyecto pueden trabajar en conjunto con facilidad, por ejemplo, pueden comunicarse mediante una red interna, sujetos a las reglas de las regiones y las zonas. No obstante, un proyecto no puede acceder a los recursos de otro proyecto, a menos que se use una VPC compartida o el intercambio de tráfico entre redes de VPC.

La Plataforma Serverless de Google le ofrece al desarrollador u organización tres formas básicas de interactuar con los servicios y recursos que brinda:

- **Consola:** Proporciona una interfaz gráfica de usuario basada en la Web. Al utilizar la consola de Google Cloud, se crea un proyecto nuevo o elige uno existente, y se usan los recursos pertenecientes al contexto de ese proyecto. Se pueden crear varios proyectos y usarlos para separar el trabajo de manera que resulte más conveniente.

- **Interfaz de Línea de Comandos:** Permite administrar el flujo de trabajo de desarrollo y los recursos de Google Cloud en una ventana de la terminal.

- **Bibliotecas Cliente:** Permiten crear y administrar recursos con facilidad. Las bibliotecas cliente de Google Cloud exponen las API para brindar acceso a los servicios y otorgar funciones para la administración de recursos.[28]

Algunos de los productos que Google Cloud Platform incluye son los siguientes:

- **Compute Engine:** Es un servicio de procesamiento seguro y personalizable que permite crear y ejecutar máquinas virtuales en la infraestructura de Google.

- **BigQuery:** Es un almacén de datos empresarial completamente serverless, el cual funciona en todas las nubes e integra, IA, IE y aprendizaje automático.

- **Anthos:** Permite administrar clústeres de GKE y cargas de trabajo que se ejecutan en máquinas virtuales en varios entornos.

- **Cloud Functions:** Ofrece al desarrollador una experiencia intuitiva y sencilla, en donde solo debe escribir el código y Google Cloud se encarga de controlar la infraestructura operativa.

- **Cloud Run:** Permite crear y desplegar aplicaciones en contenedores escalables y programadas en cualquier lenguaje (incluidos Go, Python, Java, Node.js, .NET y Ruby).

- **GPUs de Cloud:** Unidades de procesamiento de gráficos (GPUs) de alto rendimiento que están disponibles en Google Cloud para el aprendizaje automático, la computación científica y la visualización en 3D. [29]

# 2.5. Modelo Tradicional

En el Modelo Local o Tradicional, a diferencia del Modelo Cloud, el software y la tecnología se alojan en la ubicación física de una empresa, en lugar de, en la nube o en servidores ubicados en una instalación remota.

El software local es la forma tradicional de aplicaciones empresariales y de consumo, y generalmente requiere que cada servidor y usuario final tenga una licencia para usar el software. Debido a que estas aplicaciones están almacenadas en el centro de datos de la empresa, este modelo es más costoso que los modelos de servidor alojado o en la nube. El software local requiere una inversión en hardware, licencias de software y personal de TI interno, y la empresa es responsable de la seguridad y la administración del software a lo largo de su ciclo de vida. [30]

Según el servicio que la organización brinde, por ejemplo, almacenar pocas cantidades de datos, el almacenamiento seguro no es un requisito, o si el objetivo del negocio no es escalar, puede que el modelo tradicional represente un costo más rentable frente a la opción del Modelo Cloud.

Además, con este modelo la información se mantiene internamente y terceros no tienen acceso. Un aspecto muy importante ya que no se necesita Internet para acceder a los datos, es decir, los mismos se encuentren disponibles aún si se pierde la conexión a Internet, tanto para una operación de rutina como si fuera necesario recuperar alguna información de alguno de los servidores de respaldo. Además, el acceso a los mismos es rápido debido a que no depende de la conexión a Internet que posea la compañía. Por otro lado, las empresas pueden crear y armar el hardware en función de sus necesidades, permitiendo personalizar sus ofertas en el sitio y administrar las soluciones si cambian las necesidades del mercado o del consumidor.

Sin embargo, al almacenar los datos en el mismo lugar donde se encuentra la empresa corre el riesgo de perderlos en una situación de desastre. Además, si la

organización no realiza copias de seguridad de los datos con frecuencia, no hay garantía de recuperación.

También cabe mencionar que la mayoría de las violaciones de datos en la última década han sido de sistemas convencionales en las instalaciones. Debido a prácticas insuficientes o inexperiencia en los protocolos de seguridad adecuados, llevando a que la información confidencial esté en riesgo independientemente de dónde se almacene.

Por otro lado, si la organización se amplía y requiere espacio de almacenamiento adicional, puede ser un desafío escalar el sistema local rápidamente si fuera necesario que la empresa compre nuevo hardware y dedique personal para armarlo y configurarlo, en el hipotético caso de que el almacenamiento actual se haya agotado.

Los modelos tradicionales requieren personal de TI para administrar y mantener los servidores. Se puede contratar personal nuevo o delegar a algunos de los miembros existentes del equipo para este trabajo. Por lo tanto, se reducirá la eficiencia del personal de TI, ya que tendrá que asumir más tareas o aumentar los costos al contratar personal nuevo. [31]

## 2.6. Vercel

Vercel es la arquitectura serverless con la cual se realizó la comparación de Tecnologías Tradicionales vs Funciones Serverless. Es una plataforma en la nube para framework Front-End y sitios estáticos, brindando características como, un despliegue instantáneo, escalabilidad automática y no requiere supervisión. Creado y fundado como empresa en 2015 por Guillermo Rauch, Vercel ofrece una interfaz de usuario intuitiva con una configuración mínima para almacenar Static Site Generators (Generadores de Sitios Estáticos) como Gatsby o Hugo y varios CMS como Contentful, Prismic o WordPress.

Otra de las principales características es la simplicidad de implementar sitios web o aplicaciones en Vercel. No hay necesidad de preocuparse por problemas como la falta de disponibilidad ni mantenimiento de servidores, recursos limitados o configuraciones del servidor. Además, su desarrollo tiene en cuenta la escalabilidad automática del sitio, por lo que, si empieza a tener más demanda de usuarios, siempre habrá suficientes recursos para soportarlo. Vercel garantiza que la aplicación o sitio web se encuentre funcionando sin problemas, proporcionando contenido personalizado en todo el mundo, mediante la recopilación de información en tiempo real sobre cómo los usuarios de los mismos interactúan con su sitio web o servicio.

Para llevar a cabo aplicaciones en esta plataforma, se debe crear un proyecto que agrupe las implementaciones y los dominios personalizados. Asimismo, permite a los desarrolladores configurar sus proyectos en su dominio personalizado (o una URL gratuita) y un SSL automático gratuito, protegiendo los datos cifrados compartidos entre el servidor y el navegador [32].

Las implementaciones se pueden realizar con Vercel a través de:

- Vercel por GIT
- Deploy Hooks
- Vercel CLI
- Vercel API

El proyecto quedará listo para ser usado en producción mediante un paso de compilación. Una vez completado con éxito, una nueva implementación inmutable estará disponible en la URL como vista previa. Las mismas se conservan indefinidamente a menos que se eliminen de forma manual.

A continuación, se mencionará brevemente cada implementación:

- **GIT**: Se puede realizar implementaciones con Git utilizando Vercel para GitHub, para GitLab o para Bitbucket. Al usar estas integraciones, cada push a una rama le proporcionará una implementación como vista previa para ver sus cambios. Al fusionarse con la rama de producción (comúnmente main), se realizará una implementación a producción.

- **Deploy Hooks**: Los Hooks (ganchos) de implementación le permiten crear direcciones URL que aceptan solicitudes HTTP POST para generar implementaciones, volviendo a ejecutar el paso de compilación, desde fuera de Vercel. Se pueden usar los Hooks por ejemplo para reconstruir el sitio y reflejar cambios en un CMS, programar implementaciones con Cron Jobs.

- **Vercel CLI**: Al usar Vercel CLI, puede implementar proyectos con un solo comando desde su terminal. Para hacer una implementación como vista previa, usa “vercel”, y para producción el comando “vercel --prod”

- **Vercel API**: La API de Vercel se puede utilizar para realizar implementaciones mediante una solicitud HTTP POST al EndPoint correspondiente, incluidos los archivos que desea implementar como body.

Se puede administrar las implementaciones a través del Panel de Control de Vercel o, en casos de uso avanzado, Vercel CLI. A través del Panel de Control, puede encontrar una variedad de configuraciones; incluyendo una pestaña de Dominios donde se puede agregar dominios personalizados al Proyecto. La CLI de Vercel y la API de Vercel brindan formas alternativas de administrar las implementaciones, junto con una lista completa de los comandos disponibles en la documentación de

la CLI de Vercel, además de una sección de implementaciones sobre la API de Vercel.

Vercel asimismo cuenta con tres Tipos de Registros, Build Time, Runtime, y Edge Network. Los registros (Logs) de Tiempo de Compilación (Build Time) contienen información sobre el proceso de compilación y son almacenados indefinidamente. Mientras que los de Red Perimetral (Edge Network) se generan cuando se solicita una ruta desde Edge, y brindan datos sobre misma con detalles como, el nombre de la ruta, el método de solicitud y el código de estado, estos registros no se conservan. Por último, los registros de Tiempo de Ejecución (Runtime) son generados mientras las funciones Serverless son invocadas, son almacenados en memoria y al igual que los de Edge Network, no se conservan.

Otra de las grandes ventajas de Vercel es la utilización de funciones Serverless, las mismas son fragmentos de código escritos con lenguaje de Back-End que toman una solicitud HTTP y brindan una respuesta. El desarrollador puede utilizar dichas funciones para gestionar la autenticación de usuarios, envío de formularios, consultas de bases de datos, entre otras características. Para implementar funciones Serverless sin ninguna configuración adicional, puede colocar archivos con extensiones que coincidan con los lenguajes soportados y las funciones exportadas en el directorio /api, en la raíz de su proyecto. [33]

## 2.6.1. Next.js

Actualmente existe una amplia gama de Framework para que los programadores trabajen, desde Gatsby hasta Angular y Hugo, sin embargo, Vercel está optimizado en una plataforma: Next.js.

Next.js es un framework de código abierto basado en Node.js (entorno de ejecución para JavaScript [34]), y utiliza una biblioteca de JavaScript llamada React. Dicha biblioteca es requerida para crear interfaces de usuario interactivas, es decir, elementos que los usuarios ven y con los que interactúan en pantalla. Por framework, se refiere a que Next.js maneja las herramientas y configuración necesaria para React, proporcionando estructura, características y optimizaciones adicionales para su aplicación. Por lo tanto, el programador puede usar React para crear su interfaz de usuario y luego adoptar funciones de Next.js de manera incremental para crear aplicaciones web totalmente interactivas, altamente dinámicas y de alto rendimiento. Resolviendo los requisitos comunes de las aplicaciones, como el enrutamiento, obtención de datos y las integraciones, todo mientras se mejora tanto la experiencia del desarrollador como la del usuario final. [35]

Una gran ventaja de Next.js y React es el SSR (Server Side Rendering - Renderizado del lado del servidor). Los componentes de React que conforman la parte de un sitio web que está orientada al usuario, se representan inicialmente en el lado del servidor. Esto significa que una vez que el HTML se ha entregado al cliente (el navegador del usuario), no es necesario que suceda nada más para que

el usuario pueda leer el contenido de la página. Esto hace que los tiempos de carga de la página parezcan mucho más rápidos para el usuario.

Esta característica también brinda el beneficio de un sitio web listo para usar, indexable y rastreable, que es esencial para la optimización de motores de búsqueda (SEO), ya que no es necesario ejecutar JavaScript del lado del cliente para ver el contenido de la página. Esencialmente, el cliente se beneficia de un SEO técnico mejorado.

Renderizar los mismos componentes en el lado del servidor que en el lado del cliente (representación universal) significa que el tiempo de desarrollo se reduce, ya que se pueden construir los componentes de React una vez y Next.js se encarga de todo lo relacionado con volver a renderizar esos componentes en el navegador del usuario. Los desarrolladores pueden concentrarse en crear componentes y no tener que dedicar mucho tiempo sobre en qué entorno se representan. [36]

Otra de las principales características de Next.js es la División de Código (Code Splitting). Es una gran ventaja ya que los programadores suelen dividir sus aplicaciones en varias páginas a las que se puede acceder desde diferentes URL. Cada una de estas páginas se convierte en un único punto de entrada a la aplicación. La división de código es el proceso de dividir el paquete de la aplicación en partes más pequeñas requeridas por cada punto de entrada. El objetivo es mejorar el tiempo de carga inicial de la aplicación cargando solo el código necesario para ejecutar esa página.

Dentro del directorio pages/ de Next.js cada archivo se dividirá automáticamente en código, en su propio paquete de JavaScript durante el paso de compilación.

Asimismo Next.js posee un entorno de desarrollo Webpack basado en Hot Module Replacement (HMR). HMR permite a los desarrolladores ver cualquier cambio que hayan realizado durante el desarrollo, en vivo en la aplicación tan pronto como se hayan llevado a cabo. Sin embargo, a diferencia de los métodos tradicionales de Live Reload, solo recarga los módulos que realmente han cambiado, preservando el estado en el que se encontraba la aplicación y reduciendo significativamente la cantidad de tiempo requerido para ver los cambios en acción. [37]

# 2.7. Costos

La evaluación de costos es muy importante cuando se compara arquitecturas totalmente diferentes, dado que además de evaluar las prestaciones, es fundamental para la toma de decisiones en el momento de elegir alguna de ellas, considerar los costos totales del proyecto (compra o alquiler, puesta en marcha, uso de recursos, mantenimiento, etc.).
Las plataformas de funciones como servicio disponen de ciertas limitaciones o desventajas. Por un lado, el tiempo máximo de ejecución de las funciones y memoria asignable a las mismas, y por otro, los entornos de ejecución (lenguajes,

librerías, etc) están determinados por el proveedor. Es importante mencionar que los costos que requieren las Funciones Serverless aumentan proporcionalmente en relación a la cantidad de memoria reservada en el servidor implementado, además, aumentando la cantidad de memoria disminuye el tiempo de ejecución de la función, un factor a tener en cuenta ya que contribuye al costo. Sin embargo, si se lleva a cabo una determinada estrategia para optimizar la cantidad de memoria reservada, para funciones sin servidor, puede disminuir considerablemente los costos [38]. En dicha publicación se aplicó un enfoque propio, con un estudio de caso industrial sobre el uso de los servicios web de Amazon en el contexto de las aplicaciones de Smart Home. Se demostró, que esa estrategia es efectiva para estimar con precisión el impacto de la configuración de la memoria en el rendimiento del tiempo de ejecución, y determinar la configuración óptima que conduce a reducciones significativas de costos.

Según la publicación de Shafiei, Khonsari y Payam, afirman que, muchas grandes empresas de tecnología actualmente ofrecen servicios informáticos Serverless con diferentes especificaciones y precios. A medida que aumenta la popularidad de estos servicios, crecerá la cantidad de empresas y sus opciones de precios, junto con los factores que afectan el precio ofrecido por las mismas. Dichos factores van desde la estrategia de ingresos de la empresa, hasta la plataforma que utiliza y los precios de la energía (según la región o la hora del día durante la cual se ejecuta la función) [39].

Por ejemplo, una solicitud que llega a un servidor a las 2 a. m. (hora local) en invierno suele costar menos en comparación con el de la misma solicitud con el mismo consumo de recursos a las 14 h. en el verano. Otro factor es el nivel de carga que se le impone al proveedor en ese momento, es decir, si el proveedor se acerca a su pico de demanda de energía, se informa que los precios de demanda máxima pueden ser de 200 a 400 veces mayores que de la tasa nominal.

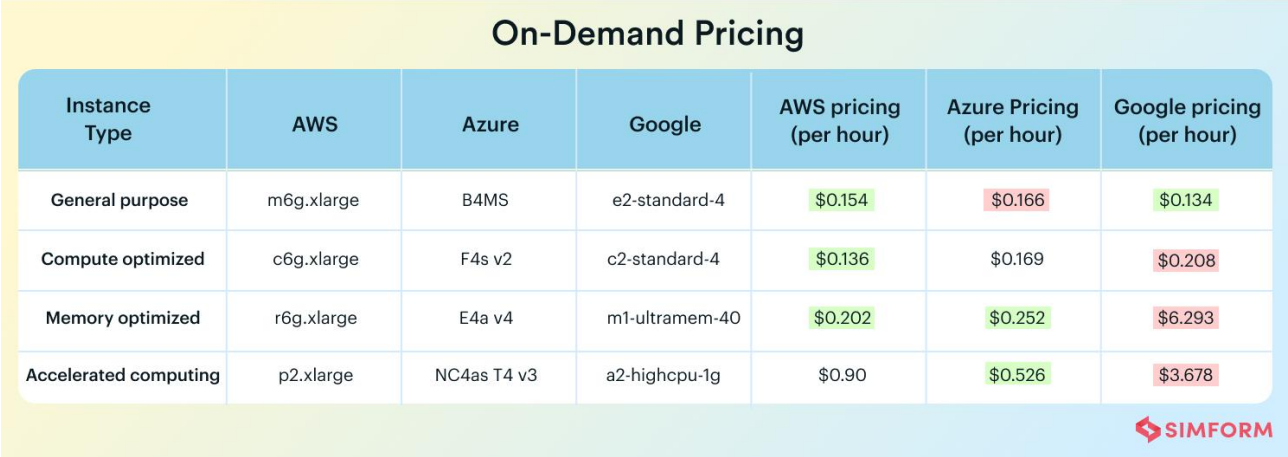
Cabe mencionar que el problema de los precios también es importante para los clientes. Como se discutió anteriormente, la diversidad de los precios conducirá a un entorno competitivo entre los proveedores de servicios. Así, el cliente puede elegir entre varias opciones de precio de forma online para reducir los costes. De esta manera, los clientes colocan sus funciones en múltiples Servicios Serverless y luego, en función de los precios disponibles en línea, la aplicación cliente del usuario decide realizar la solicitud al proveedor de servicios más asequible. El objetivo final de los clientes es reducir sus costos manteniendo la calidad del servicio, siempre teniendo en cuenta que uno de los factores importantes para determinar la calidad del mismo es el tiempo de respuesta. [40]

En función de lo mencionado en el párrafo anterior se realizó una comparación de los precios de las principales Plataformas Serverless. El siguiente artículo presentado por Solanki [41] (Director de Ventas en Simform) compara el precio de

las plataformas AWS, Azure y Google Cloud, teniendo en cuenta los siguientes datos comunes para las tres plataformas:

- Region: US East- North Virginia
- Sistema Operativo: Linux and Windows
- vCPUs/Cores: 4
- Tipos de Instancias/Máquinas Virtuales
  - General purpose (Aplicaciones de Uso General)
  - Compute optimized (Computación Optimizada)
  - Memory optimized (Memoria Optimizada)
  - Accelerated computing (Computación acelerada)

Para la comparación se utilizaron máquinas virtuales de Azure, Amazon EC2 y Google VMs. La Figura 5 muestra la estructura de precios bajo demanda por hora de cada servicio, para cada uno de los tipos de instancias, en donde el rojo indica el precio más alto y el verde representa el precio más bajo.



On-Demand Pricing

| Instance Type | AWS | Azure | Google | AWS pricing (per hour) | Azure Pricing (per hour) | Google pricing (per hour) |
|---|---|---|---|---|---|---|
| General purpose | m6g.xlarge | B4MS | e2-standard-4 | $0.154 | $0.166 | $0.134 |
| Compute optimized | c6g.xlarge | F4s v2 | c2-standard-4 | $0.136 | $0.169 | $0.208 |
| Memory optimized | r6g.xlarge | E4a v4 | m1-ultramem-40 | $0.202 | $0.252 | $6.293 |
| Accelerated computing | p2.xlarge | NC4as T4 v3 | a2-highcpu-1g | $0.90 | $0.526 | $3.678 |

SIMFORM

**Figura 5:** Comparación de Precios Bajo Demanda [42]

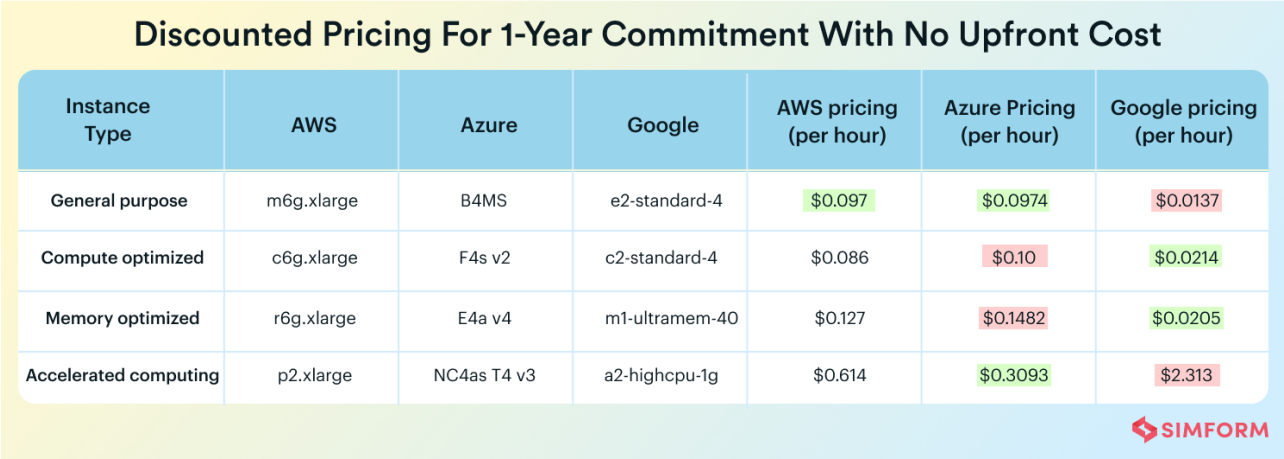
Se puede observar, que los precios de Google Cloud y AWS en relación a los sistemas que funcionan en aplicaciones de uso general (General Purpose) y en los tipos de instancias de memoria optimizada (Memory Optimized), son bastante similares.

Por otro lado, la diferencia de precio entre AWS y Azure es insignificante para los tipos de instancias de computación optimizadas. Sin embargo, Google Cloud Platform tiene el precio más alto en este servicio, debido a sus procesadores escalables y un rendimiento turbo de todos los núcleos. Además, el modelo de precios de Google Cloud para instancias de computación acelerada y memoria

optimizada también es mucho más alto, porque no tienen 4vCPU a diferencia de AWS o Azure, en su lugar, proporcionan 40vCPU y 12vCPU respectivamente.

También cabe mencionar que todos los proveedores de servicios en la nube ofrecen a las empresas descuentos en instancias bajo demanda si se comprometen a usarlas durante 1 año o más. En AWS, se conoce como "Instancias Reservadas" (RI), en Azure "Ahorros Reservados", mientras que en Google Cloud se llama "Precio de compromiso". Estos esquemas alientan a las empresas a comprometerse con un nivel preestablecido de uso durante un período fijo, a cambio de una tarifa por hora con descuento en algunas instancias y máquinas virtuales.

Para calcular los precios con descuento entre AWS, Azure y Google Cloud, fue considerado un período de compromiso de un año sin costo inicial. En la Figura 6 se colocan los valores obtenidos, nuevamente manteniendo el color rojo para el precio más alto y el verde para el menor precio.



**Discounted Pricing For 1-Year Commitment With No Upfront Cost**

| Instance Type | AWS | Azure | Google | AWS pricing (per hour) | Azure Pricing (per hour) | Google pricing (per hour) |
|---|---|---|---|---|---|---|
| General purpose | m6g.xlarge | B4MS | e2-standard-4 | $0.097 | $0.0974 | $0.0137 |
| Compute optimized | c6g.xlarge | F4s v2 | c2-standard-4 | $0.086 | $0.10 | $0.0214 |
| Memory optimized | r6g.xlarge | E4a v4 | m1-ultramem-40 | $0.127 | $0.1482 | $0.0205 |
| Accelerated computing | p2.xlarge | NC4as T4 v3 | a2-highcpu-1g | $0.614 | $0.3093 | $2.313 |

SIMFORM

**Figura 6:** Comparación de Precios con Descuento [42]

Si bien en Google Cloud, el precio bajo demanda de las instancias de memoria optimizadas fue el más alto, su precio de compromiso de 1 año es el más bajo en comparación con los precios de AWS y Azure. Esto se debe a que Google Cloud ofrece un compromiso de 1 año para instancias de memoria optimizadas en dos divisiones: vCPU/hora y GB/hora que se usan por separado según el requisito.

Se puede apreciar que Google Cloud es mucho más económico que AWS y Azure para instancias de computación optimizada. Además, es más caro que otros cuando se trata de los tipos de instancias de computación acelerada.

Una vez más, las instancias de aplicaciones de uso general tanto en AWS, como en Azure son casi similares para los planes de 1 año.

Por otro lado, Laurent Gil (Co-founder y CPO de CAST AI) [42] también realizó una publicación comparando los precios de AWS, Azure y Google Cloud Platform. Establece que seleccionar un proveedor sobre otro se reduce principalmente a saber las necesidades de las aplicaciones y cuáles son las cargas de trabajo que tendrían, es decir, se debe comprender completamente los requisitos antes de explorar el panorama de la nube. Para la comparación se tomaron las siguientes regiones, tanto AWS como Google Cloud Platform Norte de Virginia, EE. UU. y Azure EE. UU. Este, la Figura 7 muestra el precio de las diferentes plataformas teniendo en cuenta el almacenamiento, expresado en GB/Mes.

| Cloud provider | Storage (GB/Month) |
|---|---|
| Amazon S3 | $0.023 |
| Azure | $0.021 |
| Google Cloud Platform | $0.023 |

**Figura 7:** Precios de Almacenamiento GB/Mes

Laurent afirma que los proveedores de servicios en la nube compiten estrechamente entre sí y han establecido rangos de precios similares para los servicios de almacenamiento, en donde Azure se destaca como la alternativa más rentable. Sin embargo, es necesario verificar otras dimensiones de costos, como la transferencia de datos o los cargos por operaciones antes de elegir el servicio de almacenamiento.

Para comprender mejor las diferencias de precios, Laurent utilizó las siguientes características:

- Servicios Analizados
    - AWS: Amazon Elastic Compute Cloud (EC2)
    - Azure: Virtual Machines
    - Google Cloud Platform: Compute Engine
    - Oracle: Virtual Machines

- Ejemplo de configuración
    - Region: AWS US East (Northern Virginia), Azure East US, and Northern Virginia (us-east4) in Google Cloud Platform

    - Operating System: Linux
    - vCPUs: 4

- Tipos de instancias/VMs
    - General purpose (Aplicaciones de Uso General)
    - Compute optimized (Computación Optimizada)

Fueron elegidas instancias con cuatro vCPU y RAM similar, salvo la máquina optimizada para computación de Google Cloud Platform.

En la Figura 8 se encuentran las instancias/VM seleccionadas para la comparación de precios de la nube:

| Cloud provider | Instance type | vCPU | RAM (GB) |
|---|---|---|---|
| AWS general purpose | t4g.xlarge | 4 | 16 |
| AWS compute optimized | c6a.xlarge | 4 | 8 |
| Azure general purpose | B4ms | 4 | 16 |
| Azure compute optimized | F4s v2 | 4 | 8 |
| Google Cloud Platform general purpose | e2-standard-4 | 4 | 16 |
| Google Cloud Platform compute optimized | c2-standard-4 | 4 | 16 |

**Figura 8:** Instancias/VM seleccionadas para la comparación

En el artículo la comparación de precios comienza analizando el costo bajo de demanda por hora de cada una de las plataformas. La Figura 9 muestra cada una de las instancias de aplicaciones de uso general con sus respectivos precios, y La Figura 10 fueron colocados los costos de las instancias de tipo Computación Optimizada:

| Cloud provider | Instance type | Price |
|---|---|---|
| AWS | t4g.xlarge | $0.1344 |
| Azure | B4ms | $0.166 |
| Google Cloud Platform | e2-standard-4 | $0.150924 |

**Figura 9:** Instancias de Aplicaciones de Uso General con Precios

| Cloud provider | Instance type | Price |
|---|---|---|
| AWS | c6a.xlarge | $0.153 |
| Azure | F4s v2 | $0.1690 |
| Google Cloud Platform | c2-standard-4 | $0.2351 |

**Figura 10:** Instancias de Tipo Computación Optimizada

El artículo establece que, si bien Azure es la opción más costosa para las instancias de uso general, es una de las alternativas más rentables para las instancias optimizadas de cómputo. Por otro lado, Google Cloud Platform ofrece el precio más alto para instancias optimizadas de cómputo, pero esta máquina tiene el doble de RAM que las alternativas de AWS y Azure, es interesante la aclaración por si le es más rentable al desarrollador pagar un poco más pero disponer de más RAM.

Al igual que en la publicación analizada anteriormente, se lleva a cabo la comparación de precios de los tres proveedores teniendo en cuenta el descuento en los costos si se establece un compromiso de uso de un año con pago total por adelantado.

La Figura 11 y la Figura 12 contienen los precios en la nube con el descuento de compromiso de 1 año para las Instancias de Aplicaciones de uso General e Instancias de Computación Optimizada respectivamente:

| Cloud provider | Instance type | Price | Discount |
|---|---|---|---|
| AWS | t4g.xlarge | $0.084 | 41% |
| Azure | B4ms | $0.1118 | 32% |
| Google Cloud Platform | e2-standard-4 | $0.095092 | 37% |

**Figura 11:** Precios con un Compromiso de 1 año para Instancias de Aplicaciones de Uso General

| Cloud provider | Instance type | Price | Discount |
|---|---|---|---|
| AWS | c6a.xlarge | $0.1010 | 38% |
| Azure | F4s v2 | $0.1143 | 32% |
| Google Cloud Platform | c2-standard-4 | $0.14072 | 41% |

**Figura 12:** Precios con un Compromiso de 1 año para Instancias de Computación Optimizada

Laurent establece que las instancias de aplicaciones de uso general con un compromiso de 1 año reciben una tasa de descuento bastante similar en AWS y Azure. Aun así, AWS ofrece una alternativa más económica.

Sin embargo, Google Cloud Platform ofrece los mayores descuentos tanto en instancias de aplicaciones de uso general como de computación optimizadas, pero no es la opción más barata. No obstante, se agrega una nota donde dice que la instancia de Google Cloud de computación optimizada elegida tiene 16 GB de RAM, no 8 GB como las otras instancias.

La publicación agrega otra forma de obtener algunos descuentos y reducir los costos de la nube. Los proveedores de la nube venden el exceso de capacidad con buenos descuentos. Las instancias de spot de AWS ofrecen hasta un 90 % de descuento en las tarifas bajo demanda, las máquinas virtuales interrumpibles en Google pueden ser incluso un 80 % más baratas que las regulares.

Nuevamente, se comparan los precios con instancias de spot/máquinas virtuales interrumpibles, donde la Figura 13 y la Figura 14 contienen los costos de las Instancias de Aplicaciones de uso General e Instancias de Computación Optimizada respectivamente:

| Cloud provider | Instance type | Price | Discount |
|---|---|---|---|
| AWS | t4g.xlarge | $0.0436 | ~68% |
| Azure | A4 v2* | $0.0638 | ~67% |
| Google Cloud Platform | e2-standard-4 | $0.045272 | ~70% |

**Figura 13:** Precios de Máquinas Virtuales Interrumpibles para Instancias de Aplicaciones de Uso General

| Cloud provider | Instance type | Price | Discount |
|---|---|---|---|
| AWS | c6a.xlarge | $0.0768 | 50% |
| Azure | F4s v2 | $0.0295 | ~82% |
| Google Cloud Platform | c2-standard-4 | $0.054 | ~77% |

**Figura 14:** Precios de Máquinas Virtuales Interrumpibles para Instancias de Computación Optimizada

AWS ofrece un excelente precio para instancias de aplicaciones de uso general, aunque el descuento no es el más alto.

Por otro lado, Azure ofrece los mayores descuentos para ambas instancias optimizadas para computación. El precio del F4s v2 optimizado para computación es muy atractivo. [42]

# 2.8. Eficiencia de un Sitio Web

La eficiencia (o rendimiento) de un sitio web hace referencia principalmente a la velocidad de carga del mismo. Los sitios web de alto rendimiento mejoran la experiencia del usuario, al acelerar la entrega de contenido. Además, cuando el sitio es más rápido, es más probable que a los usuarios les importe lo que contiene, ningún usuario se preocupa por el contenido de una página que no se carga rápidamente.

Los sitios web lentos también tienen un efecto medible en la participación del usuario. Por ejemplo, en las páginas de comercio electrónico en particular, casi la mitad de los usuarios esperan que se cargue en 2 segundos, y el 40% de los usuarios abandona la página web si tarda más de 3 segundos en cargar. Un retraso de 1 segundo en la respuesta puede significar una reducción del 7 % en la acción de los usuarios, esto significa no sólo una pérdida de tráfico, sino también una pérdida de ingresos.

Además, el rendimiento del sitio web afecta no solo a los usuarios que lo visitan, sino también a la posición del mismo en los resultados de búsqueda de Google. En 2010, Google indicó que la velocidad de la página es un factor en la clasificación de los sitios web en sus resultados de búsqueda. Aunque la relevancia del contenido de la página sigue siendo el factor más importante en el ranking de búsqueda de Google, la velocidad juega un papel importante.

Cabe mencionar que se debe conocer técnicas de mejora del rendimiento para evitar que las experiencias web complejas, opaquen la experiencia del usuario y la capacidad de acceder al contenido. Los problemas de rendimiento generalmente son producidos por fallos en la arquitectura Front End, es decir, si bien algunos pueden originarse en un Back End mal configurado, esos problemas son específicos del lado del cliente. [43]

El tiempo de carga del sitio web puede estar retrasado por factores como:

- Uso intensivo de CSS y JavaScript
- Pobre servidor/plan de alojamiento
- Tamaños de imagen grandes
- No usar caché del navegador
- Demasiados widgets y complementos
- Hotlinking de imágenes y otros recursos de servidores lentos
- Volumen de tráfico
- Navegadores más antiguos
- Conexión de red lenta (dispositivos móviles)

A continuación, se hará mención de una serie de buenas prácticas recomendadas a seguir para poder conseguir un buen tiempo de carga:

**Reducir el Número de Solicitudes HTTP:** El navegador utiliza las solicitudes HTTP para obtener diferentes partes de la página, como imágenes, hojas de estilo y scripts de un servidor web. Cada solicitud, especialmente si se usa HTTP/1.1, tendrá cierta sobrecarga al establecer la conexión entre el navegador y el servidor

web. Además, los navegadores suelen tener un límite en la cantidad de request paralelas, por lo que, si tiene muchos en cola, algunos de ellos se bloquearán si la cola es demasiado larga. Por lo tanto, lo que se debería hacer es eliminar las solicitudes que no son necesarias, las imágenes, archivos de JavaScript, hojas de cálculo, etc, que no se usen desecharlas.

**Optimizar Tamaños de Imagen:** Muchos sitios web utilizan bastantes gráficos, si las imágenes no están comprimidas, o si se usa una resolución demasiado alta, ralentizará el rendimiento del mismo. Por ejemplo, los sitios web a veces usan imágenes con una resolución de 2x o incluso 3x para que se muestren bien en pantallas de alta densidad, como pantallas retina. Pero si sus usuarios no usan una pantalla HiDP, entonces solo está desperdiciando ancho de banda y aumentando el tiempo de carga, especialmente si tienen conexiones de datos móviles lentas. Por lo tanto, se recomienda usar correctamente las imágenes receptivas, es decir, especificar varios tamaños de imagen para que el navegador seleccione la adecuada, según la resolución de la pantalla.

**Utilizar CDN:** Los CDN optimizan la entrega de archivos estáticos como CSS, imágenes, fuentes y JavaScript, además, su configuración generalmente es muy simple. Las CDN utilizan servidores distribuidos geográficamente, esto significa que el servidor más cercano a la persona que este ingresando al sitio es el que enviara los archivos. Entonces, el tiempo de carga por ejemplo, para las imágenes será el mismo, independientemente de dónde se conecte el usuario. Generalmente, cuando se entregan archivos estáticos desde sus propios servidores, el tiempo de carga aumenta cuando los usuarios están físicamente lejos del servidor, por ese motivo es recomendable el uso de CDN.

**Elegir un Plan de Hosting adecuado:** Se debe seleccionar correctamente el proveedor de hosting a fin de maximizar el rendimiento del sitio web. Ya que, si se está utilizando un proveedor compartido, si bien puede ser económico, es muy probable que el rendimiento general sea inferior debido a que los recursos son compartidos. Se recomienda estudiar bien el proveedor a elegir, así no se corre el riesgo de que los usuarios del sitio lo abandonen por la lentitud del mismo en brindar la información.

**Implementar Compresión Gzip**: Se recomienda habilitar la compresión Gzip en los servidores HTTP, la misma minimiza el tamaño de las respuestas HTTP para ciertos tipos de archivos, por lo general, se usa para respuestas textuales. Esto debería reducir los tiempos de carga y ahorrar ancho de banda.

**Minificar y combinar archivos CSS, JavaScript y HTML:** La minificación es el proceso de optimizar el tamaño de los archivos JavaScript y CSS eliminando o acortando los símbolos en el código fuente. La salida es funcionalmente equivalente, pero no completamente legible por humanos, sin embargo, los

navegadores no tienen problemas para leerlo, y los tamaños de archivo más pequeños serán más rápidos de cargar. Por lo tanto, lo que terminan haciendo la mayoría de los sitios web optimizados es, primero minimizar los archivos JavaScript y CSS y luego combinarlos en paquetes únicos.

**Uso de Técnicas Prefetch, Preconnect y Prerender:** Existen diferentes técnicas de prefetching y preloading que se puede utilizar para dar pistas al navegador sobre qué recursos se necesitarán para representar la página antes de que el navegador realmente necesite esos recursos.

- DNS Prefetching: Se le puede informar al navegador que ciertos nombres de dominio deberán resolverse en una dirección IP antes de que vea los recursos de ese nombre de dominio.

- TCP Preconnect. Al igual que el método de DNS Prefetching, preconnect resolverá el DNS, pero también realizará el protocolo de enlace TCP y, opcionalmente, el protocolo de enlace TLS.

- Prefetching: Si un recurso específico será requerido en el futuro, se puede pedirle al navegador que solicite ese elemento y lo almacene en el caché para consultarlo más adelante.

- Prerendering: Se puede indicar al navegador que renderice previamente la página completa, junto con la descarga de todos los activos necesarios.

**Reducir el Número de Plugins:** Si bien se encargan de brindar al sitio funciones adicionales, como análisis o la capacidad de dejar comentarios en publicaciones de blog, tienen un costo, por lo general cada Plugin carga archivos CSS y JavaScript adicionales, incluso algunos también aumentarán el tiempo de TTFB porque requieren un procesamiento adicional en el servidor para cada solicitud de página. Por lo tanto, se recomienda eliminar los Plugin que no sean necesarios y dejar únicamente los que el sitio realmente necesita.

**Usar el almacenamiento en caché del sitio web:** Habilitar el almacenamiento en caché del navegador tiene muchas ventajas, puede reducir el consumo de ancho de banda, aumentar los tiempos de carga, reducir la latencia y la carga de trabajo del servidor. El principal inconveniente es que, básicamente, siempre habrá al menos dos versiones del sitio web en un momento dado. Esto puede causar problemas si se está ejecutando un servicio en tiempo real que se basa en datos precisos, pero incluso esto puede solucionarse hasta cierto punto, obligando a que la subsección de la memoria caché se borre cuando se importan nuevos datos. [44]

El rendimiento web se enfoca en hacer que las páginas web se carguen más rápido, el objetivo principal es reducir el tiempo de carga (latencia).

# Capítulo III

# Metodología

## 3.1. Comparación Tecnologías Tradicionales vs Funciones Serverless

Para la comparación de la eficiencia de Tecnologías Tradicionales de Front versus Funciones Serverless, se tomó como ejemplo dos vistas de la aplicación de acreditación de haberes desarrollada para la Cámara de Diputados de San Juan. En una de las mismas se puede observar el funcionamiento dinámico del sistema (Dynamic Content), obteniendo diferentes datos en función de cada request [45], la siguiente vista tiene un comportamiento estático (Static Content), en el cual el sistema devuelve solo un codigo html previamente generado en tiempos de compilación [46]. El objetivo es evaluar el rendimiento del sistema comportándose de dos maneras diferentes, y poder llegar a una conclusión sobre la eficiencia del mismo, siendo ejecutado con Funciones Serverless y con Tecnologías Tradicionales.

Dicha aplicación fue desplegada en la Plataforma Vercel con el objetivo de usar las Funciones Serverless que provee, además fue utilizado el Sistema de Cloud Computing con Kubernetes que dispone la Cámara de Diputados a fin de poder llevar a cabo la comparación antes mencionada.

Al momento de realizar las evaluaciones, el sistema fue accedido desde una computadora de escritorio de uso doméstico, tomándose como ejemplo una velocidad de conexión de 17 Mbpp (resultado obtenido de FAST [47], como indica la Figura 15).


**Figura 15:** Velocidad de conexión utilizada

La vista sobre contenido dinámico, como se puede apreciar en la Figura 16, es un listado de personas donde a medida que se va avanzando en las páginas del mismo, cada request va obteniendo diferentes resultados. Es decir, el contenido web cambia según el comportamiento, las preferencias y los intereses del usuario, entregando un contenido completamente distinto dependiendo de la solicitud, dichos resultados no se encuentran almacenados en memoria caché. [45]

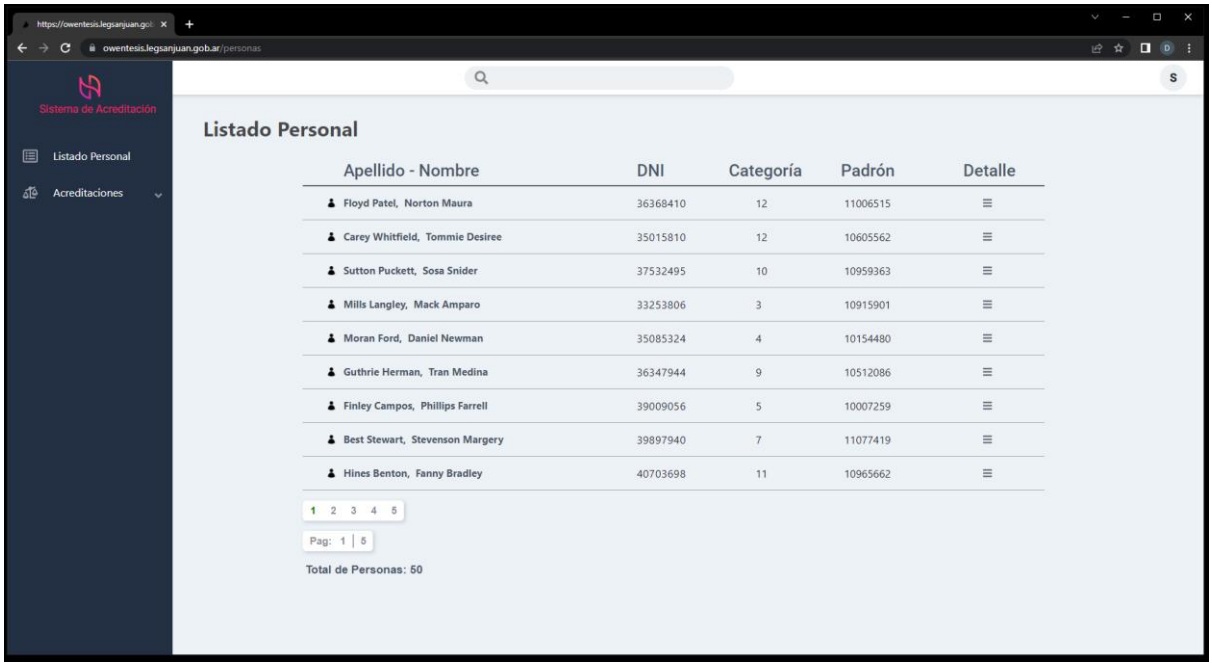



**Figura 16:** Vista sobre Contenido Dinámico

La vista sobre contenido estático, es el Login del Sistema (como indica la Figura 17), ya que es un contenido que se puede entregar a un usuario final sin tener que generarlo, modificarlo o procesarlo y generalmente es almacenado en memoria caché. El servidor entrega el mismo archivo a cada usuario, lo que hace que el contenido estático sea uno de los tipos de contenido más simples y eficientes para transmitir a través de Internet. [46]

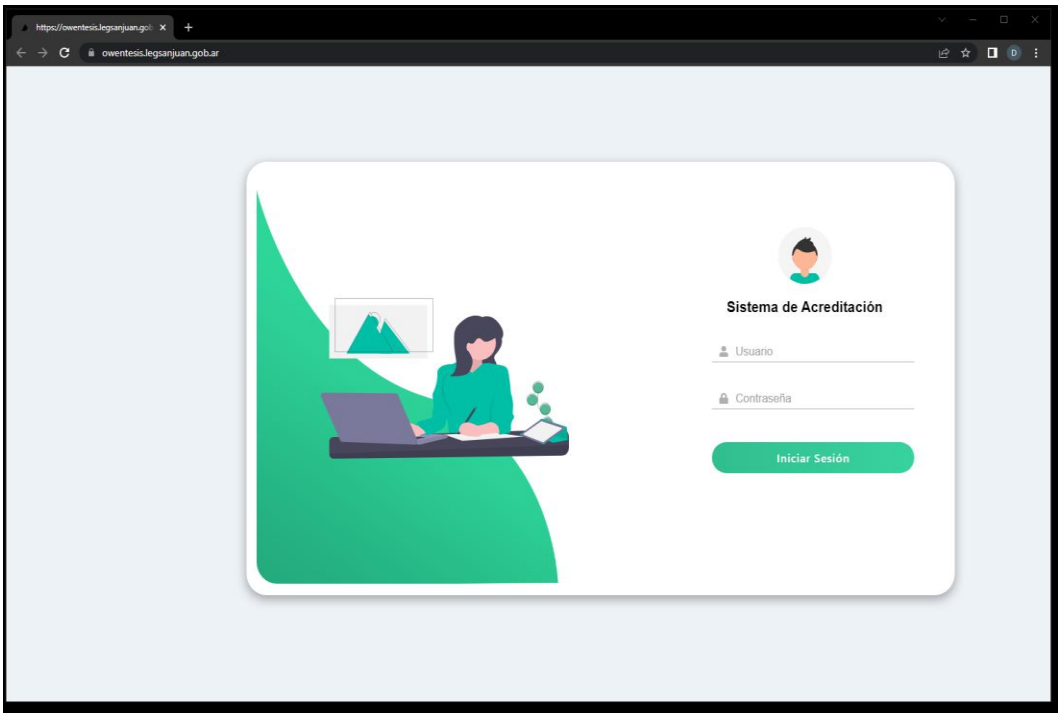



**Figura 17:** Vista sobre Contenido Estático

A continuación, se puede observar en la figura 18 un grafico que indica de forma general el trabajo de investigacion realizado en el siguiente capítulo:

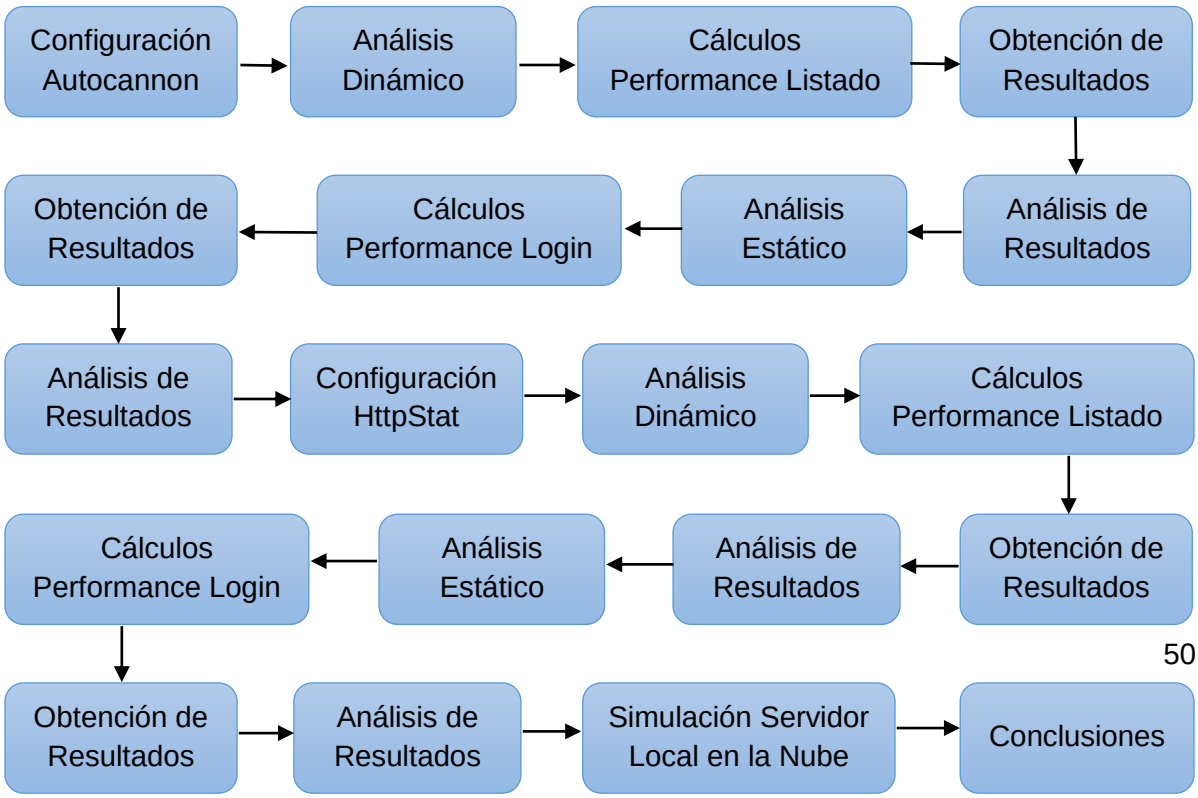


**Figura 18:** Grafico sobre el Trabajo de Investigación Realizado

Para poder comparar la eficiencia del sistema se utilizaron dos herramientas de evaluación comparativa, Auttocannon y go-HttpStat. Se puede apreciar que en ambas se requiere una configuración previa, para luego comenzar con el análisis tanto dinámico como estático, en donde se obtienen resultados y se evalúan los mismos. Finalmente, antes de comenzar con las conclusiones se agrega una simulación del servidor local en la nube a fin de enriquecer la información obtenida y analizada anteriormente, y poder llegar a una conclusión más contundente y precisa.

# Capítulo IV

# Desarrollo

# 4.1. Autocannon

Autocannon es una herramienta desarrollada en Node.js (entorno de ejecució JavaScript orientado a eventos asíncronos, diseñado para crear aplicaciones network escalables [48]) e inspirada en wrk y wrk 2. Dicha herramienta puede simular un alto tráfico en la aplicación, creando solicitudes de acuerdo a una configuración establecida, por ejemplo, número de conexiones HTTP, duración de la evaluación comparativa, número de solicitudes individuales y la concurrencia utilizada para la generación de carga. Dicha herramienta muestra toda la información de latencia requerida por segundo, y además los percentiles (medida de posición usada en estadística) de los tiempos de respuesta. Además, también proporciona estadísticas sobre los Benchmarking Suites (conjunto de herramientas de evaluación comparativa) para ayudar a analizar los resultados [49]. Es una herramienta muy importante para determinar el rendimiento de la aplicación, y mejorarlo si fuera necesario usando diferentes técnicas. [50]

El Front End del Sistema ejecutándose en Serverless y con Tecnologías Tradicionales fue evaluado con el siguiente comando de la herramienta:

**“autocannon -c 100 -d 30 -p 10”**

Utilizando dicho comando se simula 100 conexiones, durante 30 segundos, con 10 solicitudes canalizadas. Se obtuvieron los siguientes resultados:

## 4.1.1. Dynamic Content

### 4.1.1.1. Tecnologías Tradicionales

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 19:

**“autocannon -c 100 -d 30 -p 10 https://owentesis.legsanjuan.gob.ar/personas”**

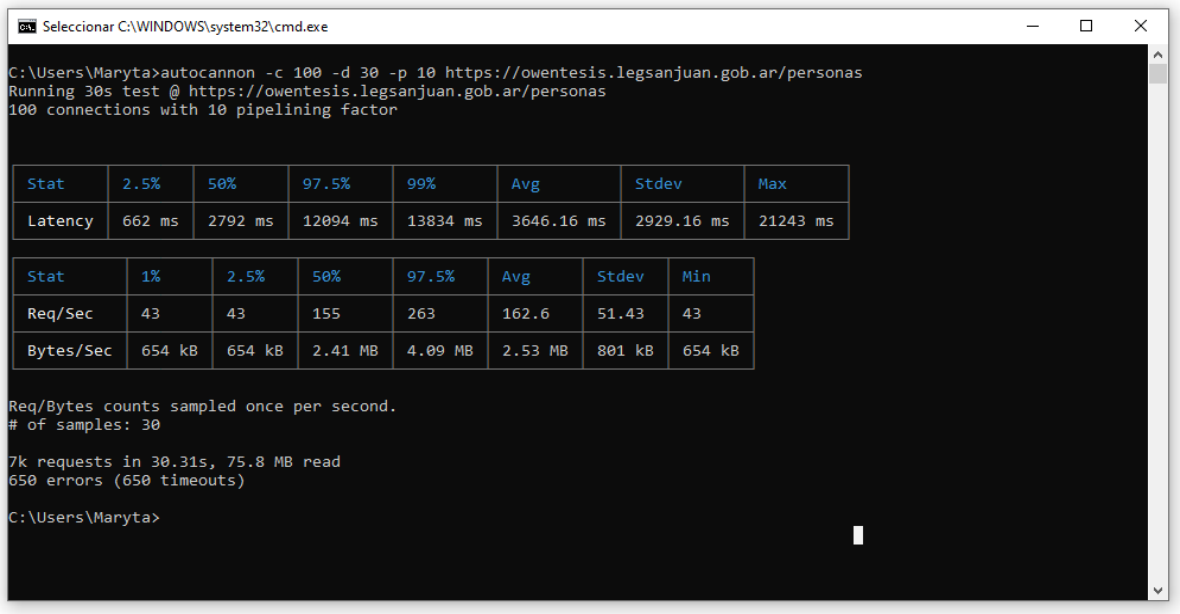



**Figura 19:** Muestra obtenida con Autocannon para Tecnologías Tradicionales (Contenido Dinámico)

## 4.1.1.2. Funciones Serverless

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 20:

**“autocannon -c 100 -d 30 -p 10 https://acreditaciones-front-tesis.vercel.app/personas”**

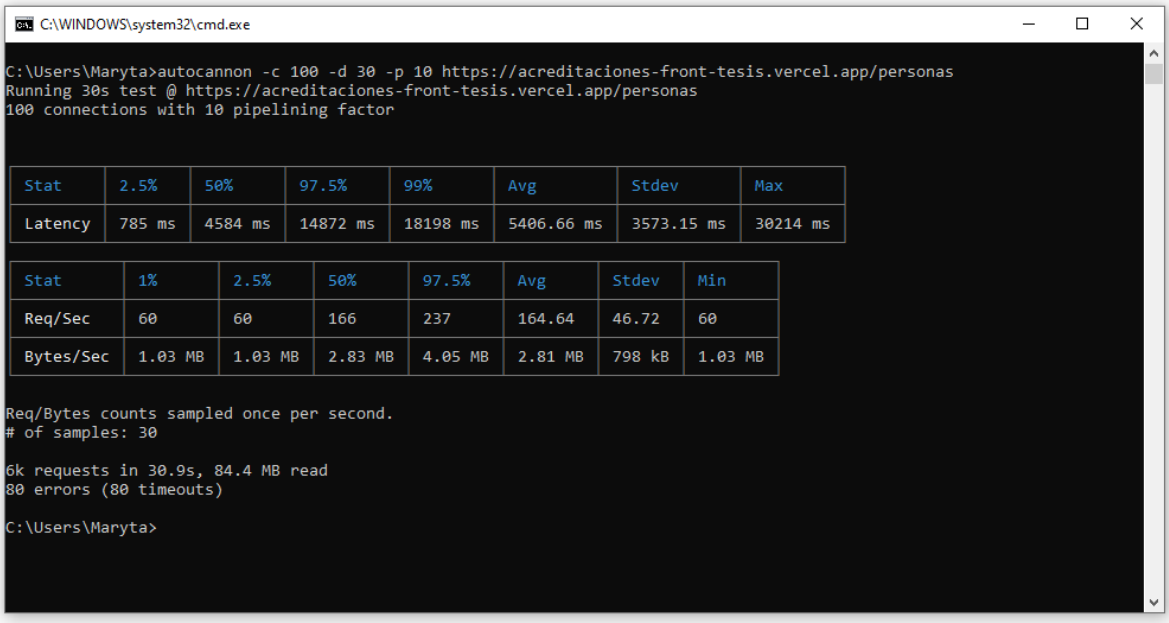



**Figura 20:** Muestra obtenida con Autocannon para Funciones Serverless (Contenido Dinámico)

## 4.1.1.3. Análisis de los resultados

De acuerdo a las Figuras 19 y 20, se pueden observar dos tablas, una contiene la latencia de la solicitud y la otra tabla la cantidad de solicitudes enviadas y la cantidad de bytes descargados.

A continuación, se adiciona un gráfico de barras, a fin de poder ayudar interpretar los datos obtenidos de la primera tabla. Dichos valores se ven representados en la Figura 21, en el eje vertical se observa el Tiempo de la Solicitud (expresado en milisegundos) y el eje horizontal cada uno de los Percentiles:

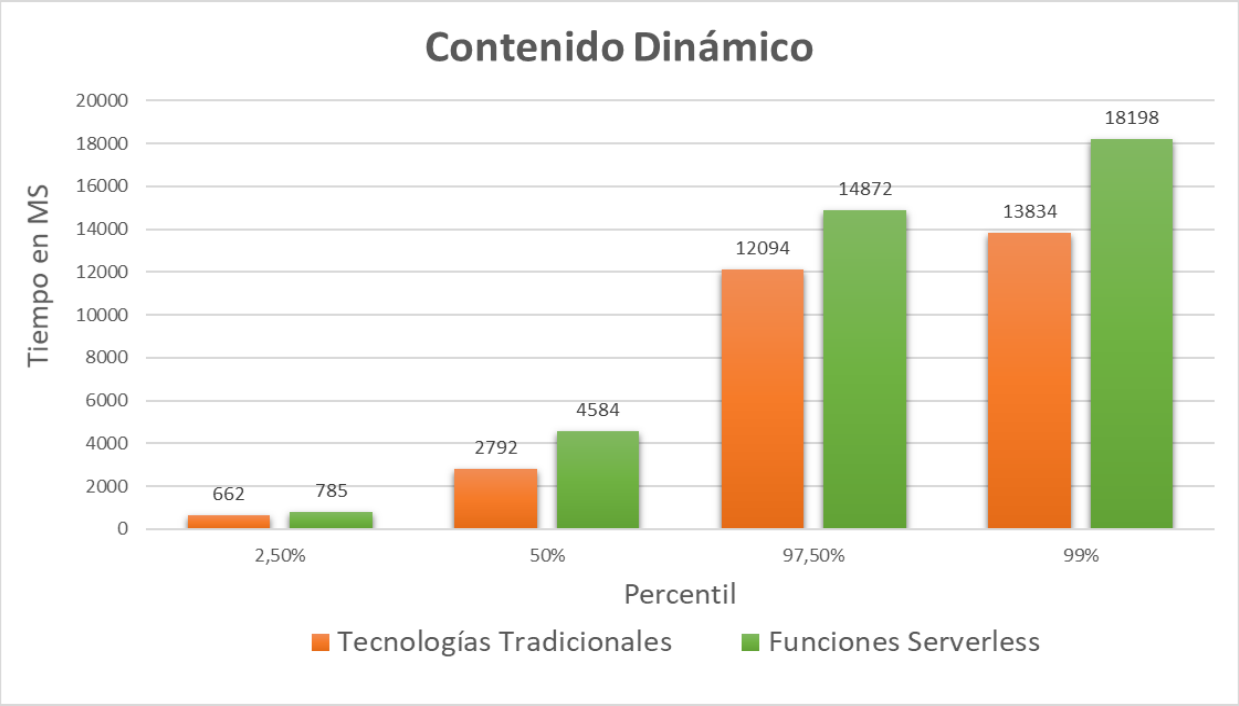



**Figura 21:** Tiempo de Respuesta de la Solicitud

De acuerdo a los valores provistos, se puede establecer que al 50% de las personas que accedan al sistema pueden tener un tiempo de respuesta de 2792 ms (en el caso de Tecnologías Tradicionales), un valor un poco menor que 4584 ms (Funciones Serverless). Además, otro dato interesante es el percentil 99, el mismo indica que al 99% de las personas que ingresen como mucho pueden obtener un tiempo de respuesta de 13834 ms con Tecnologías Tradicionales y 18198 ms con Funciones Serverless.

A continuación, se agregan dos gráficos de barras que representan los valores obtenidos por la segunda tabla. La Figura 22 representa la Cantidad de Solicitudes Enviadas y la Figura 23 la Cantidad de Bytes Descargados. Ambos datos se encuentran en el eje vertical de cada una de las figuras y, nuevamente, los percentiles en el eje horizontal.

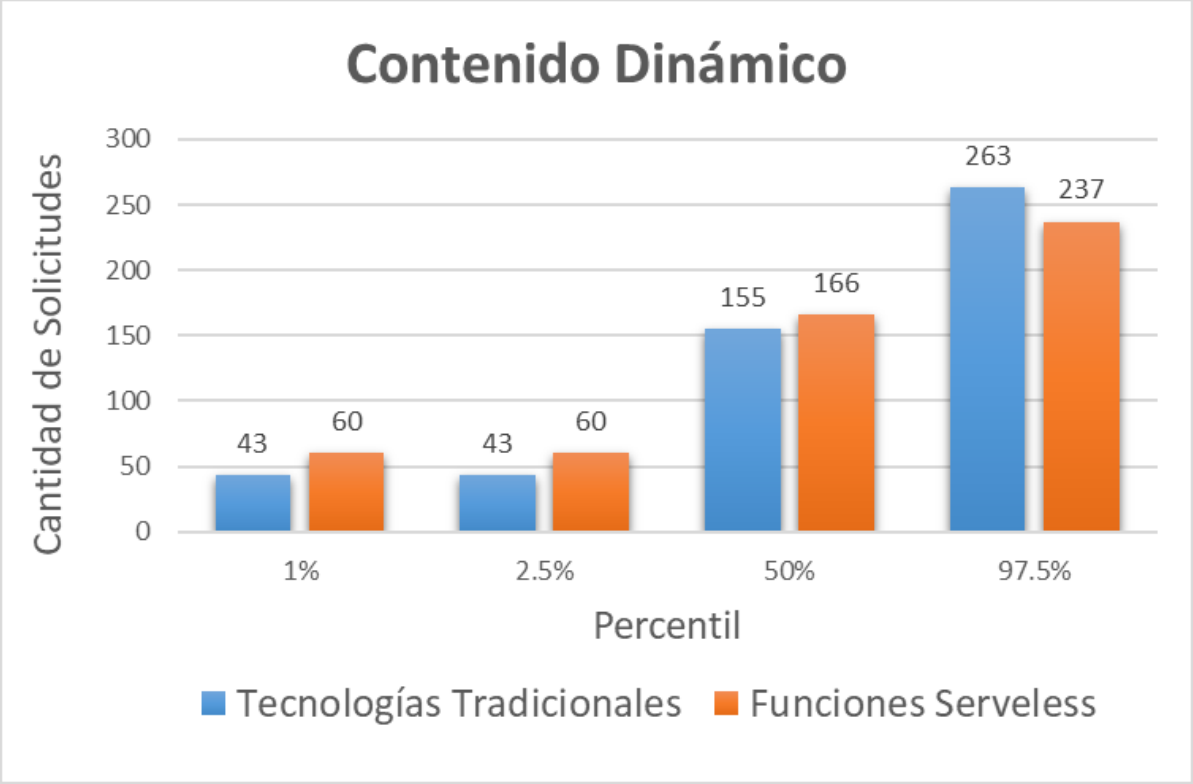



**Figura 22:** Cantidad de Solicitudes Enviadas

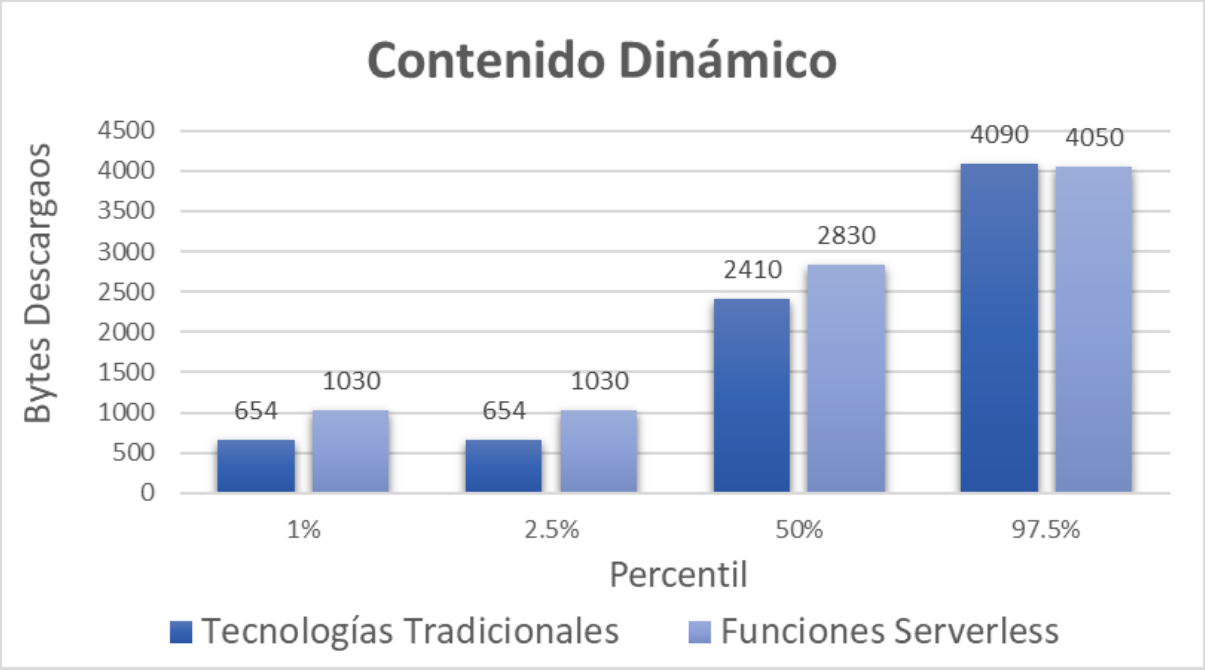



**Figura 23:** Cantidad de Bytes Descargados

Al llevar a cabo el análisis, se puede interpretar que la aplicación siendo ejecutada con Tecnologías Tradicionales es capaz de procesar un promedio de 155

solicitudes por segundo, un máximo de 263 solicitudes por segundo y un mínimo de 43 solicitudes. Estos datos indican que el sistema nunca manejara menos de 43 request por segundo con una descarga mínima de 654 KB y en el mejor de los casos 263 request por segundo permitiendo descargar 4.09 MB. Observando dicha tabla obtenida aplicando Funciones Serverless se puede ver unos resultados un poco mejor en algunos aspectos, si bien la aplicación en el mejor de los casos puede procesar un máximo de 237 request por segundo (un valor un poco menor que con Tecnologías Tradicionales), los datos mejoran en relación a que el sistema nunca manejara menos de 60 solicitudes por segundo con una descarga de 1.03 MB y en promedio es capaz de tratar 166 request por segundo permitiendo descargar 2.83 MB.

## 4.1.2. Static Content

### 4.1.2.1. Tecnologías Tradicionales

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 24:

**“autocannon -c 100 -d 30 -p 10 https://owentesis.legsanjuan.gob.ar/”**

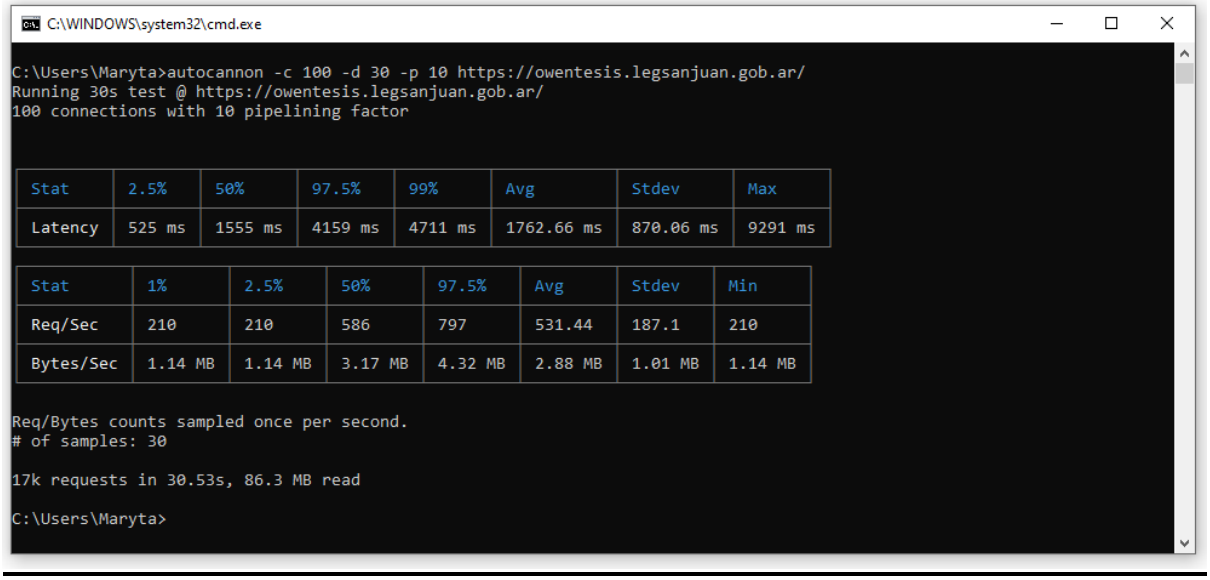



**Figura 24:** Muestra obtenida con Autocannon para Tecnologías Tradicionales (Contenido Estático)

### 4.1.2.2. Funciones Serverless

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 25:

**“autocannon -c 100 -d 30 -p 10 https://acreditaciones-front-tesis.vercel.app/”**

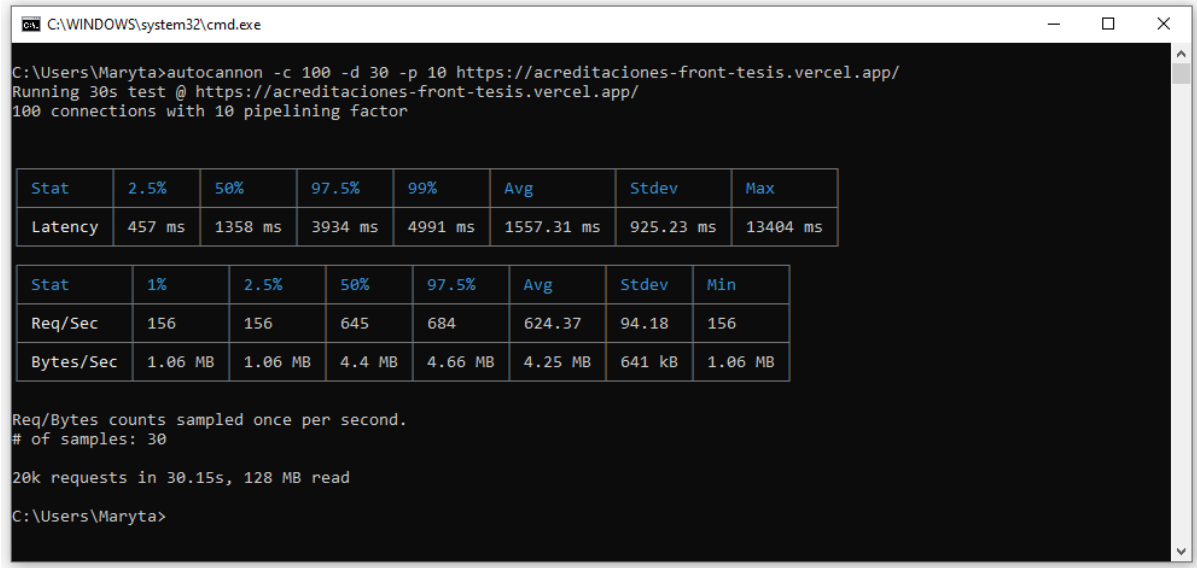



**Figura 25:** Muestra obtenida con Autocannon para Funciones Serverless (Contenido Estático)

## 4.1.2.3. Análisis de los resultados

Al aplicar la herramienta a la vista con contenido estático, según las Figuras 24 y 25 se observan resultados mejores en ambas tablas en relación al contenido dinámico, debido a que (como se mencionó anteriormente) la aplicación tanto en Vercel como en el Sistema de Cloud Computing, entrega siempre el mismo código y generalmente se almacena en memoria caché.

Como en el análisis anterior, se agregan gráficos a fin de ayudar a comprender y entender los valores obtenidos por la herramienta Autocannon.

La Figura 26 representa el contenido de la primer tabla de las Figuras 24 y 25, donde el eje vertical es el Tiempo expresado en Milisegundos, y el eje horizontal cada uno de los Percentiles.

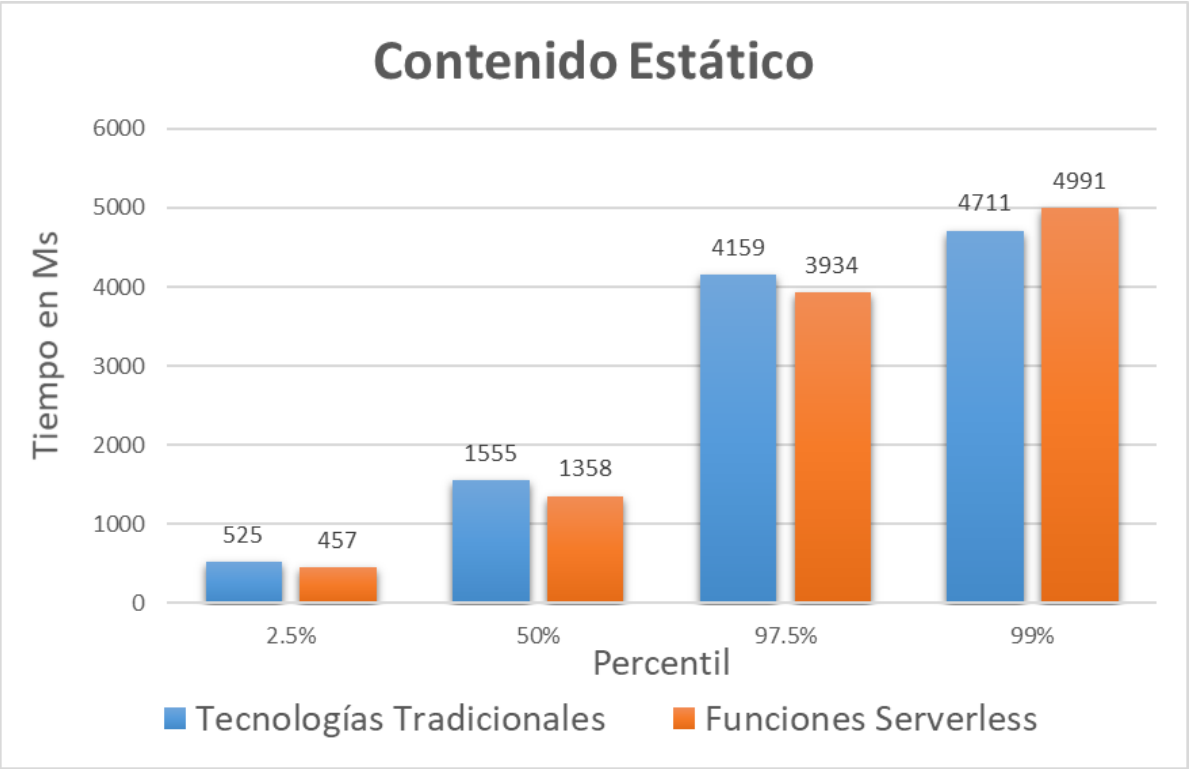



**Figura 26:** Tiempo de Respuesta de la Solicitud

De la primer tabla (con ayuda de la Figura 26) se puede observar que el 99% de las personas que ingresen al sistema si bien tienen un tiempo de respuesta similar, es un poco mejor el obtenido por las Tecnologías Tradicionales, donde los usuarios pueden tener una latencia como mucho de 4711 ms, respecto a 4991 ms usando Funciones Serverless. Si se analiza el percentil 50 de dicha tabla, vemos un resultado diferente, se puede establecer que el 50% de las personas que accedan a la aplicación podrán tener un tiempo de respuesta de 1555 ms, es decir tienen una latencia un poco mayor al obtenido usando Funciones Serverless, ya que su percentil indica una latencia de 1358 ms.

A continuación, la Figura 27 representa la Cantidad de Solicitudes Enviadas y la Figura 28 la Cantidad de Bytes Descargados. Ambos datos son observados en el eje vertical de cada una de las figuras y los percentiles en el eje horizontal.

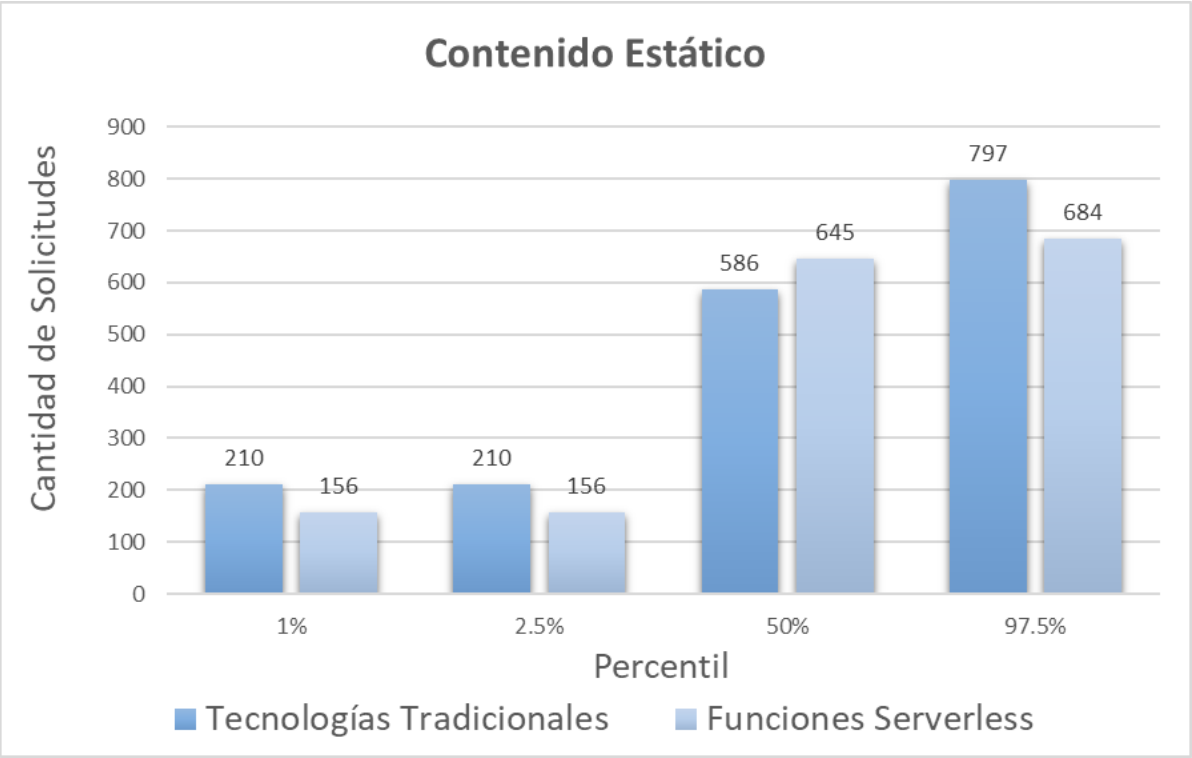



**Figura 27:** Cantidad de Solicitudes Enviadas

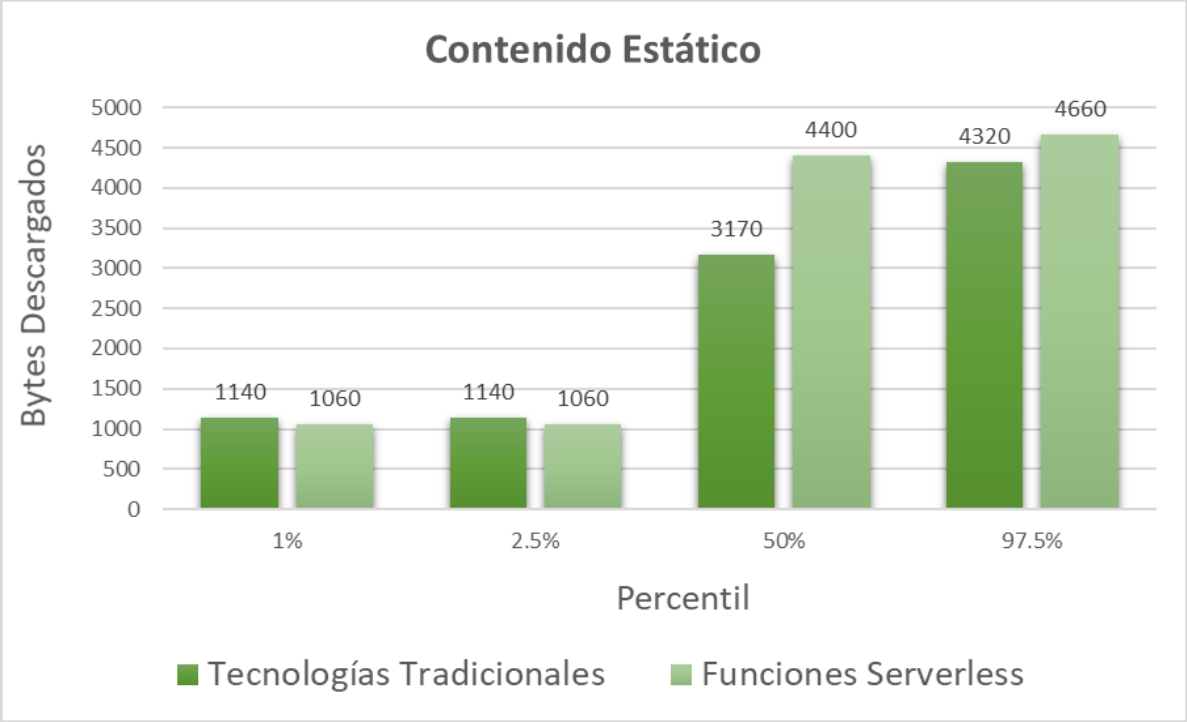



**Figura 28:** Cantidad de Bytes Descargados

Según la segunda tabla, también se pueden apreciar resultados muy diferentes a los vistos en el contenido dinámico (por el mismo motivo explicado anteriormente).

El sistema siendo ejecutado con Tecnologías Tradicionales permite procesar un promedio de 586 solicitudes por segundo, un máximo de 797 y un mínimo de 210 request por segundo. Estos resultados establecen que en el mejor de los casos la aplicación es capaz de procesar 797 solicitudes por segundo, permitiendo una descarga de 4.32 MB, también por segundo, y nunca manejará menos de 210 solicitudes por segundo, con una descarga de 1.14 MB. Al utilizar Funciones Serverless se pueden observar datos un poco menores (en algunos casos), tanto en el percentil 97.5 con 684 solicitudes por segundo, como en el percentil 1 con 156 request por segundo, se ven valores inferiores a los obtenidos con Tecnologías Tradicionales. Solo con el percentil 50 el resultado mejoró, obteniendo 645 solicitudes por segundo, en relación a los 586 request por segundo de las Tecnologías Tradicionales. En función de estos datos se puede deducir que la aplicación con Funciones Serverless y contenido estático, puede procesar un promedio mejor de solicitudes por segundo que con Tecnologías Tradicionales, pero tanto en el mejor de los casos como la cantidad mínima de request por segundo que el sistema es capaz de manejar, se puede ver un mejor desempeño al utilizar las Tecnologías Tradicionales, respecto de las Funciones Serverless.

## 4.2. HttpStat

Continuando con la segunda herramienta de evaluación comparativa, go-HttpStat es un paquete de golang para rastrear la latencia de solicitud HTTP, y muestra gráficamente la sincronización del evento, desde la búsqueda de DNS hasta la transferencia de contenido. Obteniendo como resultado una tabla ASCII, donde se encuentran los tiempos de los siguientes pasos [51]

- DNS Lookup: Proceso de traducir y encontrar qué dirección IP pertenece a qué sitio web (ej. de www.google.com a 192.168.2.1). Por lo tanto, antes de poder ver y descargar todos los recursos con el navegador, se debe realizar un DNS Lookup para cada dominio que proporciona la información. [52]

- TCP Connection: Tiempo en que demora una aplicación de cliente en abrir una conexión TCP/IP a una aplicación de servidor. [53]

- TLS Handshake: Protocolo de encriptación y autenticación diseñado para proteger las comunicaciones en Internet. Durante un protocolo de enlace TLS, las dos partes que se comunican intercambian mensajes para reconocerse y verificarse entre sí, establecer los algoritmos de criptografía que utilizarán y acordar las claves de sesión. [54]

- Server Processing: Un proceso de servidor es una tarea que se realiza en el mismo. Pueden llevar a cabo tareas específicas, iniciarse como un proceso

automatizado o por medio de un comando, algunos procesos incluso pueden generar múltiples procesos de forma simultánea. [55]

- Content Transfer: Proceso de utilizar técnicas y tecnologías informáticas para transmitir o transferir datos electrónicos o analógicos de un nodo informático a otro. Los datos se transfieren en forma de bits y bytes a través de un medio digital o analógico, y el proceso permite las comunicaciones entre dispositivos. [56]

# 4.2.1. Dynamic Content

## 4.2.1.1. Tecnologías Tradicionales

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 29:

**“httpstat https://owentesis.legsanjuan.gob.ar/personas”**

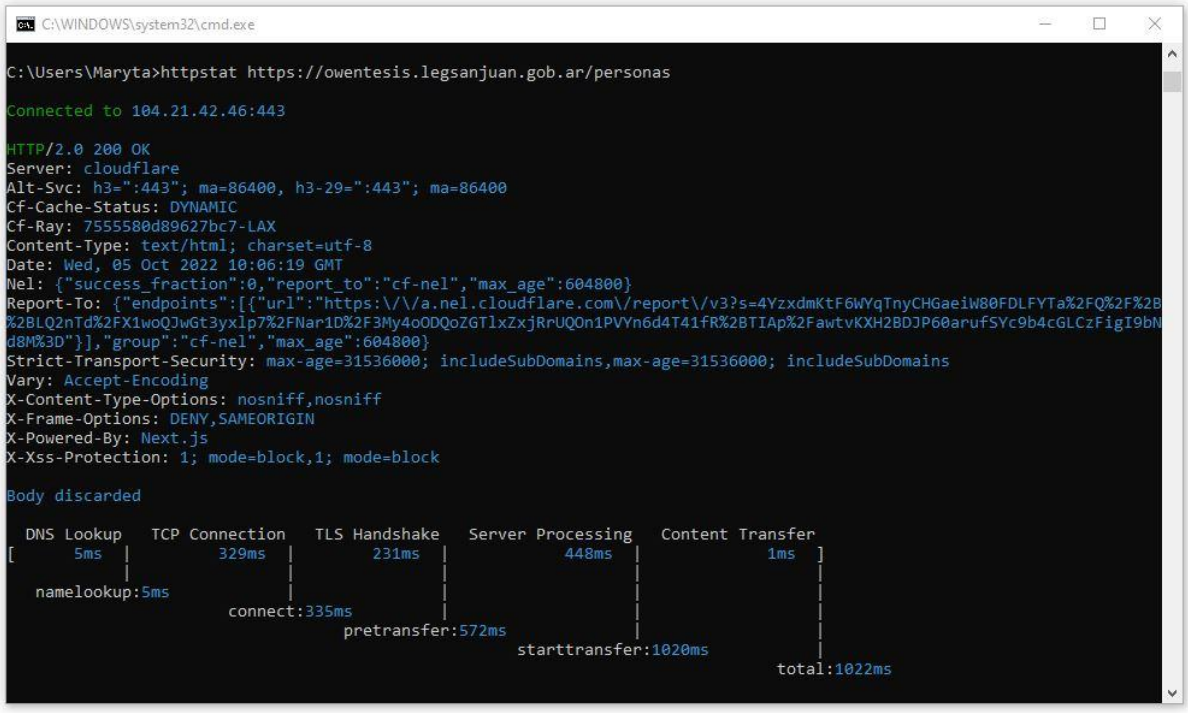



**Figura 29:** Muestra obtenida con HttpStat para Tecnologías Tradicionales (Contenido Dinámico)

## 4.2.1.2. Funciones Serverless

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 30:

**“httpstat https://acreditaciones-front-tesis.vercel.app/personas”**

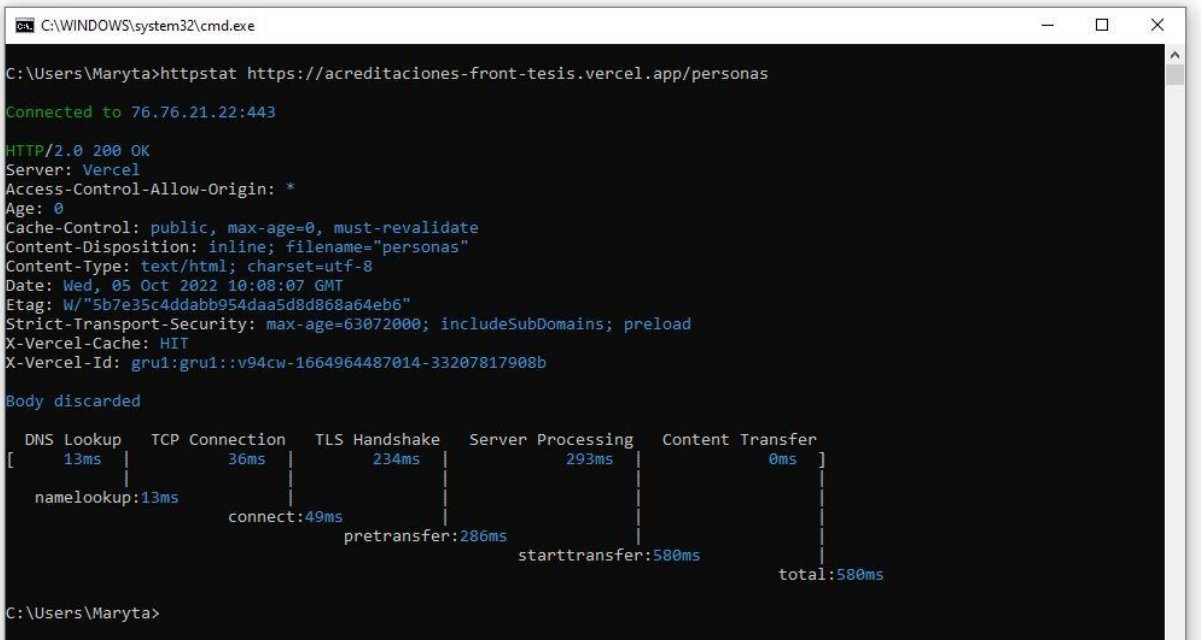



**Figura 30:** Muestra obtenida con HttpStat para Funciones Serverless (Contenido Dinámico)

## 4.2.1.3. Análisis de los Resultados

En función de los resultados obtenidos en las muestras de la Figura 29 y la Figura 30, se pueden observar diferentes Tiempos de Respuesta tanto para Tecnologías Tradicionales como para Funciones Serverless.

En lo que respecta a DNS Lookup se puede apreciar una Latencia bastante buena en ambas muestras, obteniendo un Tiempo de Retardo de 5 ms para Tecnologías Tradicionales y 13 ms para Funciones Serverless. Indicando que ambos fueron capaces de obtener la Dirección IP del Dominio del Sistema tanto en el Servidor de la Cámara de Diputados como en el Servidor de Vercel en un tiempo aceptable. Destacándose el Servidor de la Cámara de Diputados por encima del Servidor de Vercel, debido a un mejor Tiempo de Respuesta.

A continuación, el apartado de TCP Connection indica unos valores diferentes al analizado anteriormente. Obteniendo un Tiempo de Respuesta mejor en Funciones Serverless con 36 ms, en relación a las Tecnologías Tradicionales cuya Latencia fue de 329 ms. Dichos valores informan que (según las muestras analizadas) el

proceso de establecer la conexión TCP entre la computadora donde fueron tomadas y el servidor donde se encuentra almacenado el sistema, fue más rápido el llevar a cabo una conexión exitosa con el Servidor de Vercel, debido a un Tiempo de Retardo mucho menor.

Continuando con TLS Handshake se puede observar nuevamente una Latencia si bien similar, pero mejor en la muestra obtenida para Tecnologías Tradicionales en relación a la de Funciones Serverless. Estos valores establecen que el proceso de intercambio de mensajes entre la computadora cliente y los respectivos servidores, para el acuerdo de las claves de sesión y los algoritmos de criptografía fue más rápido en las Tecnologías Tradicionales con un Tiempo de Respuesta de 231 ms, una Latencia un poco mejor que en la muestra de Funciones Serverless, cuyo Tiempo de Retardo fue de 234 ms.

Finalmente, el resultado obtenido del análisis de Server Processing es muy interesante ya que es donde se procesa la solicitud enviada por el cliente. Es decir, el tiempo de respuesta abarca desde la llegada de la solicitud al servidor, el procesamiento de la misma y finalmente el envío del resultado. Se puede observar una Latencia mucho menor provista por el Servidor de Vercel, en relación al valor obtenido por la herramienta sobre el Servidor de la Legislatura. Estos datos indican que el procesamiento de una solicitud con contenido dinámico, puede ser resuelta en un tiempo menor por Funciones Serverless (en nuestro caso representadas por la plataforma Vercel) que utilizando Tecnologías Tradicionales para almacenar un Sistema.

## 4.2.2. Static Content

### 4.2.2.1. Tecnologías Tradicionales

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 31:

**“httpstat https://owentesis.legsanjuan.gob.ar/”**

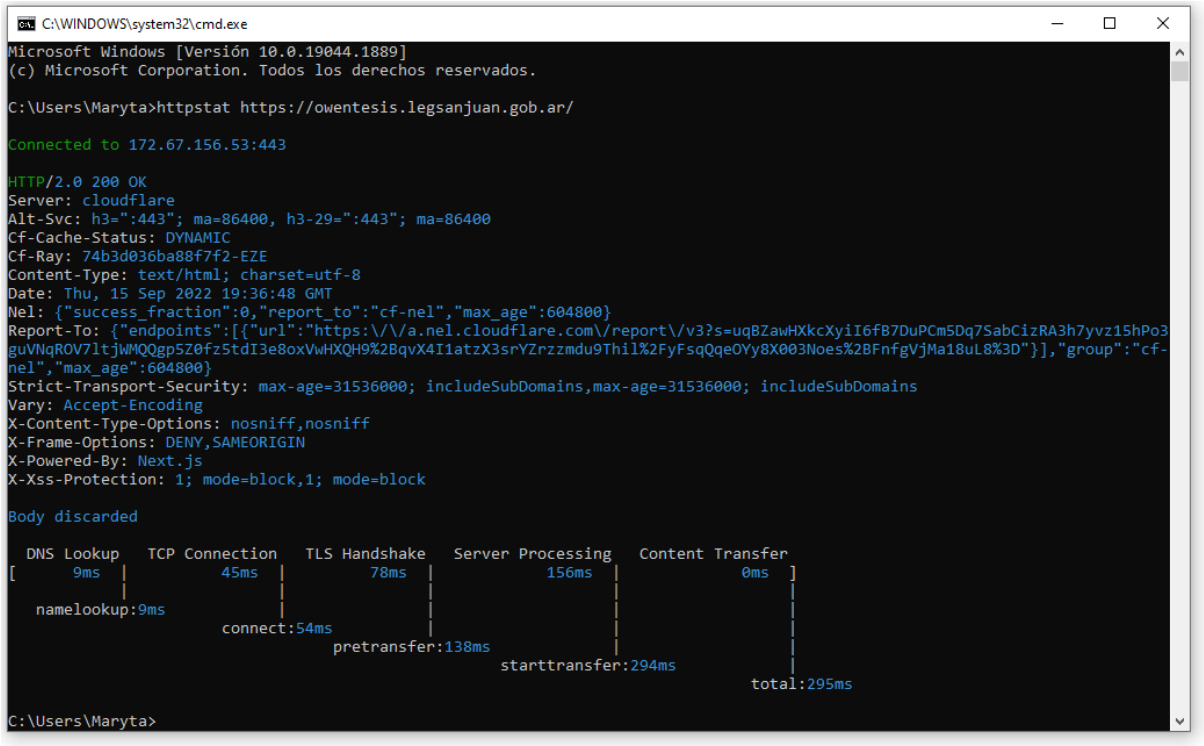



**Figura 31:** Muestra obtenida con HttpStat para Tecnologías Tradicionales (Contenido Estático)

## 4.2.2.2. Funciones Serverless

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 32:

**“httpstat https://acreditaciones-front-tesis.vercel.app/”**

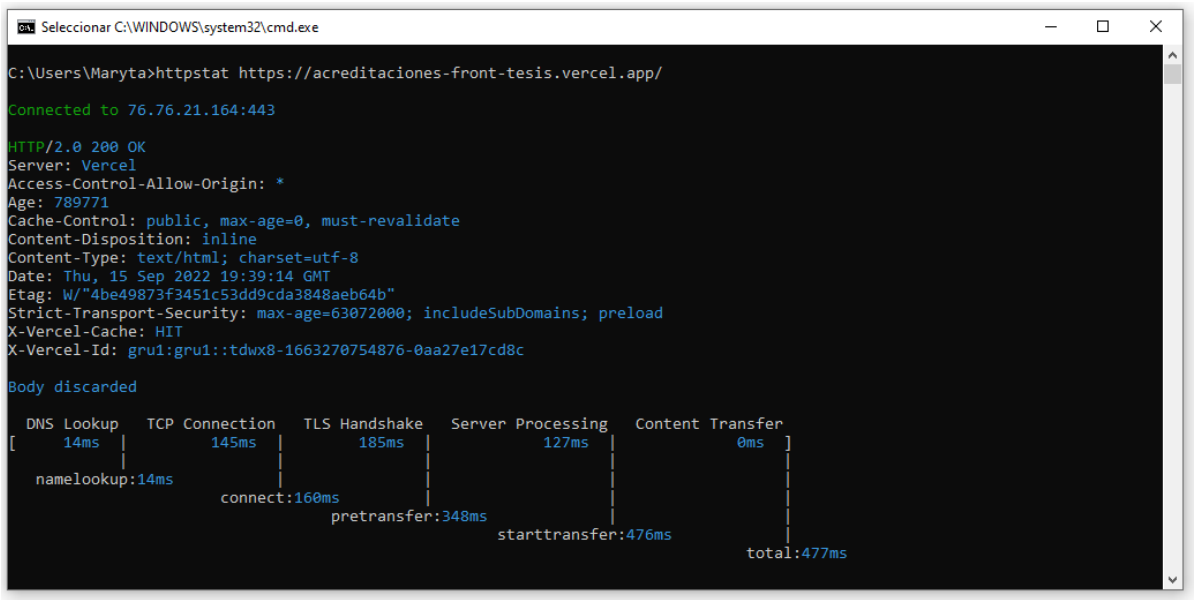



**Figura 32:** Muestra obtenida con HttpStat para Funciones Serverless (Contenido Estático)

## 4.2.2.3. Análisis de los Resultados

Al utilizar la herramienta con la vista con contenido estático, según se muestra en las Figuras 31 y 32, se pueden observar diferentes resultados interesantes. A continuación, se explicarán cada uno de los pasos por los cuales atravesó dicha herramienta.

En DNS Lookup tanto Tecnologías Tradicionales como Funciones Serverless obtienen un tiempo de respuesta bastante bueno. Según la prestigiosa compañía internacional Sematext una latencia óptima en dicho aspecto se encuentra dentro del rango de 20 ms a 120 ms [57]. Se puede observar que en ambas muestras, el resultado es incluso un poco menor a 20 ms, mejorando la recomendación otorgada por la empresa. También se puede ver un tiempo de respuesta un poco mejor en la muestra con Tecnologías Tradicionales, con un resultado de 9 ms, en comparación a los 14 ms obtenidos con Funciones Serverless. Esto significa que (según estas muestras) se pudo obtener la dirección IP del dominio del sistema alojado en el servidor de la Cámara de Diputados (Tecnologías Tradicionales) más rápido que en Vercel (Funciones Serverless).

Tanto en TCP Connection como TLS Handshake se puede observar un tiempo de respuesta mayor en Funciones Serverless, en comparación con la muestra obtenida de Tecnologías Tradicionales. Ese resultado se obtiene debido a que, para establecer la conexión TCP entre la computadora (donde se tomaron dichas muestras) y el servidor de Vercel donde se aloja el sistema, fue necesario atravesar más nodos o routers en comparación con la conexión establecida con la misma aplicación, pero alojada en el Sistema de Cloud Computing ubicado en la Cámara de Diputados de San Juan. Esa cantidad mayor de nodos o routers repercuten en una latencia más elevada, al momento de establecer la conexión TCP entre ambas partes. Luego el proceso de intercambio de mensajes entre la computadora cliente y el servidor donde se encuentra la aplicación, a fin de acordar las claves de sesión y los algoritmos de criptografía a utilizar, para proteger la comunicación según lo establece el Protocolo TLS Handshake, también se ve afectada su latencia (como en TCP connection) a la cantidad mayor de nodos entre la computadora cliente y Vercel. Por eso también se obtiene un tiempo de respuesta mayor en la muestra con Funciones Serverless, comparada con la muestra de Tecnologías Tradicionales.

El siguiente paso en el análisis es Server Processing. Se puede observar una latencia inferior en la métrica obtenida con Funciones Serverless (de 127 ms) en relación a Tecnologías Tradicionales (156 ms). Un resultado favorable para las Funciones Serverless ya que en los otros ítem el tiempo de respuesta fue mayor al obtenido en la métrica de Tecnologías Tradicionales. Se puede interpretar que el servidor de Vercel es capaz de procesar la solicitud enviada por el cliente (en este

caso una computadora de uso doméstico) con una velocidad mayor, al sistema de cloud computing de la Cámara de Diputados, obteniendo una latencia menor. Es decir, en este caso las Funciones Serverless fueron capaces de, una vez llegada la solicitud, interpretarla y entregar resultados en un mejor tiempo, en comparación con las Tecnologías Tradicionales al demorar más en brindar el mismo resultado.

Antes de establecer conclusiones, se llevará a cabo una simulación del servidor de la Legislatura como si el mismo se encontrará en la nube, se ejecutarán las herramientas de análisis de rendimiento y los resultados serán comparados con los ya obtenidos sobre el sistema desplegado en la plataforma Vercel, a fin de enriquecer la información obtenida y analizada anteriormente, y poder llegar a una conclusión más contundente y precisa.

## 4.3. Simulación Servidor Local en la Nube

Dicha simulación puede realizarse de dos maneras diferentes. La opción número uno es utilizar el Comando de Windows llamado Tracert, y la opción número dos es crear una máquina virtual, por ejemplo, en la plataforma Amazon Web Service. A continuación, se desarrollarán ambas opciones.

### 4.3.1. Opción 1 Comando Tracert

La primera de ellas, como se mencionó es utilizar el comando de Windows **Tracert**, el mismo determina la ruta a un destino mediante el envío de paquetes de eco de Protocolo de Mensajes de Control de Internet (ICMP). En estos paquetes, Tracert usa valores de Período de Vida (TTL) IP variables. Dado que los routers de la ruta deben disminuir el TTL del paquete como mínimo una unidad antes de reenviar el paquete, el TTL en realidad es un contador de saltos. Cuando el TTL de un paquete alcanza el valor cero (0), el router devuelve al equipo de origen un mensaje ICMP de Tiempo agotado.

Comienza enviando el primer paquete de eco con un TTL de 1 y lo aumenta en 1 en cada transmisión posterior, hasta que el destino responde o hasta que se alcanza el TTL máximo. Los mensajes ICMP Tiempo agotado que devuelven los routers intermedios muestran la ruta. Se observa, sin embargo, que algunos routers colocan paquetes que han agotado el TTL sin avisar, los mismos son invisibles para TRACERT.

Finalmente se imprime por pantalla una lista ordenada de los routers intermedios que devuelven mensajes ICMP Tiempo agotado. La opción “-d” con el comando tracert le indica a que no efectúe una búsqueda de DNS en todas las direcciones IP, de manera que se devuelve la dirección IP de la interfaz del lado cercano de los routers. [58]

Desde la computadora de uso doméstico donde se tomaron las muestras, usando la Línea de Comandos de Windows (o Símbolo del Sistema), se ejecutó Tracert a los dos destinos, owentesis.legsanjuan.gob.ar y acreditaciones-front-tesis.vercel.app.


**Figura 33:** Ruta obtenida con Tracert al destino owentesis.legsanjuan.gob.ar

Se puede observar en la Figura 33 que desde la computadora donde fue ejecutado el comando al destino owentesis.legsanjuan.gob.ar, fueron necesarios 7 saltos y los tres paquetes al llegar a destino dieron un Tiempo de Respuesta entre 25 ms y 27 ms.

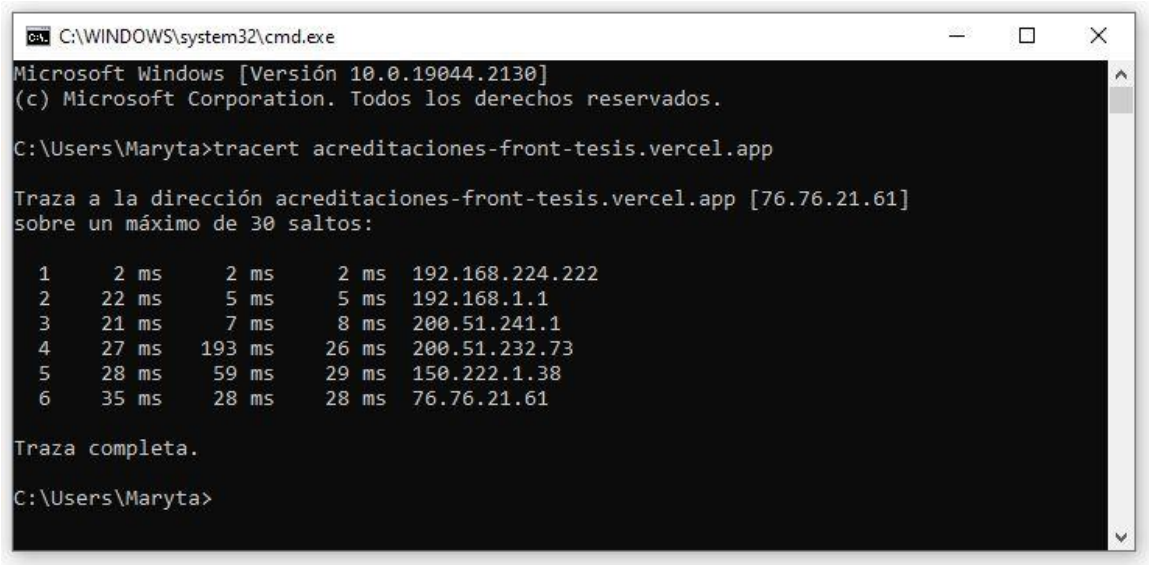



**Figura 34:** Ruta obtenida con Tracert al destino acreditaciones-front-tesis.vercel.app

Sin embargo, en la Figura 34 al realizar Tracert al destino acreditaciones-front-tesis.vercel.app nos indicó una cantidad de 6 saltos hasta llegar al mismo, dando los tres paquetes una Latencia entre 28 ms y 35 ms en el salto número 6 (dicho salto indica que se llegó al destino indicado).

A continuación, se tomó el promedio de los tres Tiempos de Respuesta obtenidos para cada uno de los dos sitios:

- 26 ms de Latencia a **owentesis.legsanjuan.gob.ar**
- 30 ms de Latencia a **acreditaciones-front-tesis.vercel.app**

Estos valores indican que desde la computadora donde fueron tomadas las muestras se tiene un Tiempo de Retardo promedio de 26 ms a donde se encuentra almacenado el sistema en la Legislatura, y un Tiempo un poco mayor a donde el mismo sistema está alojado en Vercel, con 30 ms promedio de Latencia.

Finalmente, para realizar la simulación del Servidor de la Legislatura en la nube según esta opción, se deberían tomar de referencia los valores obtenidos por la herramienta HttpStat, a los mismos simplemente restarles a cada uno de los ítem el Tiempo obtenido por el comando Tracert a dicho servidor, y luego sumarle el tiempo obtenido por el mismo comando a Vercel. Es decir, lo que se estaría haciendo es, a los tiempos obtenidos por la herramienta, quitarles el tiempo de respuesta que existe desde la computadora donde se tomaron las muestras correspondientes al servidor donde se encuentra alojado el sistema en la Legislatura, y luego al adicionales el tiempo de respuesta de dicho sistema desplegado en Vercel. De esta manera se estaría simulando el Servidor de la Legislatura en la nube.

Lamentablemente esta opción no puede llevarse a cabo debido a que, por ejemplo, en el caso del Tiempo de Respuesta obtenido por la herramienta para el DNS Lookup no se puede realizar la operación establecida anteriormente, ya que la resolución del DNS se hace contra otro servidor, y el comando Tracert nos devuelve el la Latencia que hay hacia la Legislatura, no al servidor donde se lleva a cabo el DNS Lookup.

## 4.3.2. Opción 2 Plataforma Amazon

La opción número dos para llevar a cabo la simulación del servidor local en la nube es (como se mencionó al comienzo del apartado) por medio de la creación y uso de una máquina virtual, por ejemplo, en la Plataforma de Amazon, bajo el servicio de Amazon Web Service. El procedimiento es acceder al Sistema alojado en el servidor de la legislatura desde la máquina virtual y aplicar la herramienta HttpStat.

La simulación es correcta ya que de esta manera las muestras obtenidas al servidor de la Legislatura no son de una computadora ubicada en la misma localidad, debido a que dicha máquina virtual puede incluso estar alojada fuera del País. Lo mismo sucede con las muestras del Sistema almacenado en el Servidor de Vercel, en el camino inverso, es decir se utiliza una computadora física (de uso doméstico) ubicada localmente en la provincia de San Juan (Argentina), y se aplican las herramientas al Sistema alojado en un Servidor cuya ubicación, si bien no se sabe con exactitud, pero se encuentra fuera del País.

La Máquina Virtual creada en la Plataforma de Amazon contiene el Sistema Operativo **Microsoft Windows Server 2022** con una arquitectura de 64 bits. La misma se encuentra situada en **US East (N. Virginia).**

Luego, al igual que en la computadora de uso doméstico donde se obtuvieron las muestras analizadas anteriormente, se instaló la herramienta HttpStat y a continuación analizado el Sistema alojado en la Legislatura, tanto el contenido Dinámico como el Estático.

Los resultados obtenidos son volcados posteriormente en una tabla comparativa, junto con los datos del Sistema almacenado en Vercel a fin de poder analizarlos.

Se comenzará con el análisis y comparación de Dynamic Content.

## 4.3.2.1. Dynamic Content

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 34:

**“httpstat https://owentesis.legsanjuan.gob.ar/personas”**

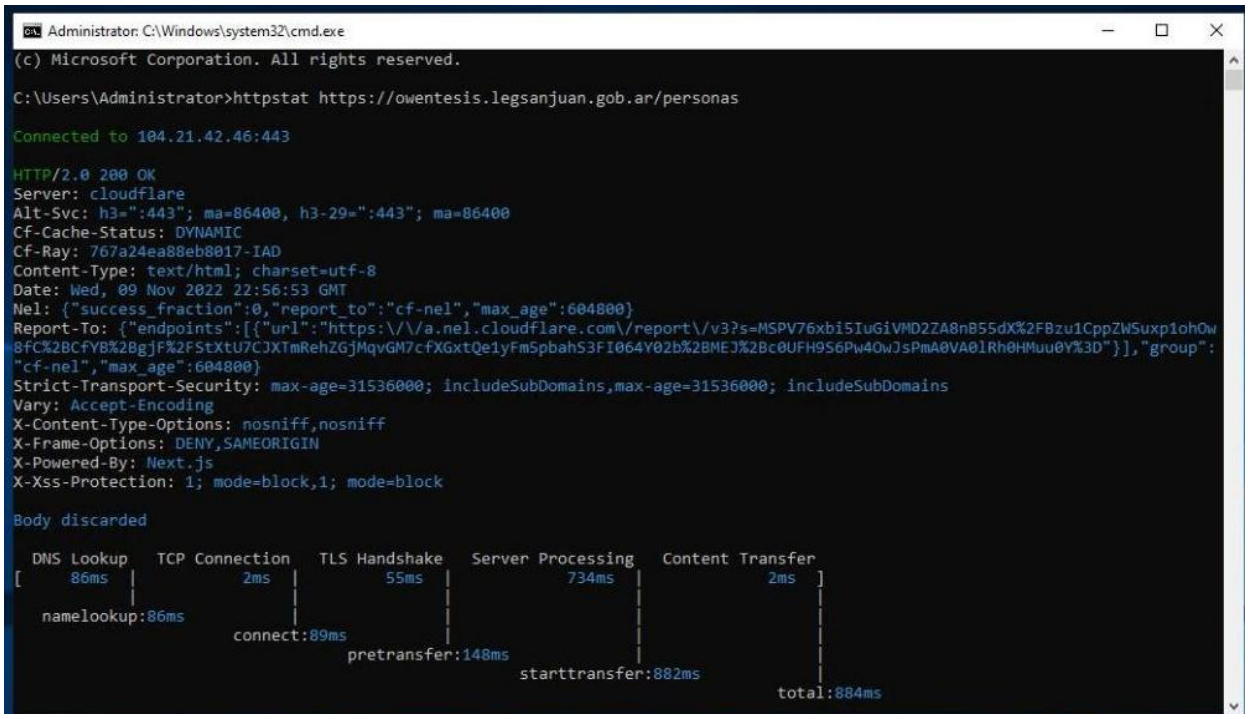



**Figura 35:** Muestra obtenida con HttpStat para Tecnologías Tradicionales (Contenido Dinámico)

A continuación, se llevará a cabo una tabla comparativa donde se agregarán los valores obtenidos de la Figura 35 y los resultados de la muestra de Funciones Serverless para contenido dinámico con la herramienta httpstat, como indica la Figura 30, a fin de poder ayudar a estudiar ambos resultados.

| | Vercel | Legislatura |
|---|---|---|
| DNS Lookup | 13 ms | 86 ms |
| TCP Connection | 36 ms | 2 ms |
| TLS Handshake | 234 ms | 55 ms |
| Server Processing | 293 ms | 734 ms |
| Content Transfer | 0 ms | 2 ms |

**Tabla 1:** Comparación de Contenido Dinámico obtenido con HttpStat

Según La tabla 1 se puede observar un Tiempo de Respuesta del DNS Lookup mucho mayor en Tecnologías Tradicionales con 86 ms de latencia, en relación a Funciones Serverless donde se obtuvo 13 ms de retardo. Esto indica que al ser utilizada la máquina virtual ubicada en **US East (N. Virginia)** el proceso de obtención de la Dirección IP del Dominio del Sistema alojado en el servidor de la

Legislatura, se llevó a cabo en un periodo de tiempo mayor, en comparación con el mismo procedimiento realizado desde la computadora de uso doméstico al mismo sistema almacenado en el servidor de vercel.

En los dos ítems siguientes, TCP Connection y TLS HandShake se puede analizar una situación diferente, ambos procesos en la muestra de Vercel tuvieron más demora en realizarse, en comparación con esta muestra de Tecnologías Tradicionales. Se puede interpretar que tanto el proceso de establecer la conexión TCP entre la computadora cliente y el servidor donde se encuentra almacenado el sistema, como el intercambio de mensajes entre ambas partes para acordar las claves de sesión y los algoritmos de criptografía, fueron realizados en un tiempo mucho menor en el caso de Tecnologías Tradicionales, a pesar de ser accedido el Sistema desde la máquina virtual.

El último apartado a analizar es Server Processing, es muy importante ya que hace referencia al tiempo que le lleva al servidor realizar la tarea establecida y brindar una respuesta. Se pueden observar unos resultados favorables en la muestra de Vercel, en comparación con la simulación realizada de Tecnologías Tradicionales en la Nube. Gracias a estos valores se puede analizar que el Servidor de Vercel fue capaz de interpretar la solicitud enviada por el cliente, procesarla y llevar a cabo una respuesta en solo 293 ms. Mientras que el Servidor de la Legislatura le llevó mucho más tiempo realizar la misma petición, 734 ms.

A continuación, se llevará a cabo el análisis y comparación del Contenido Estático, también utilizando la máquina virtual para simular el Servidor de la Legislatura en la nube.

## 4.3.2.2. Static Content

Se usó el siguiente Comando y se obtuvieron los resultados observados en la Figura 36:

**“httpstat https://owentesis.legsanjuan.gob.ar/”**

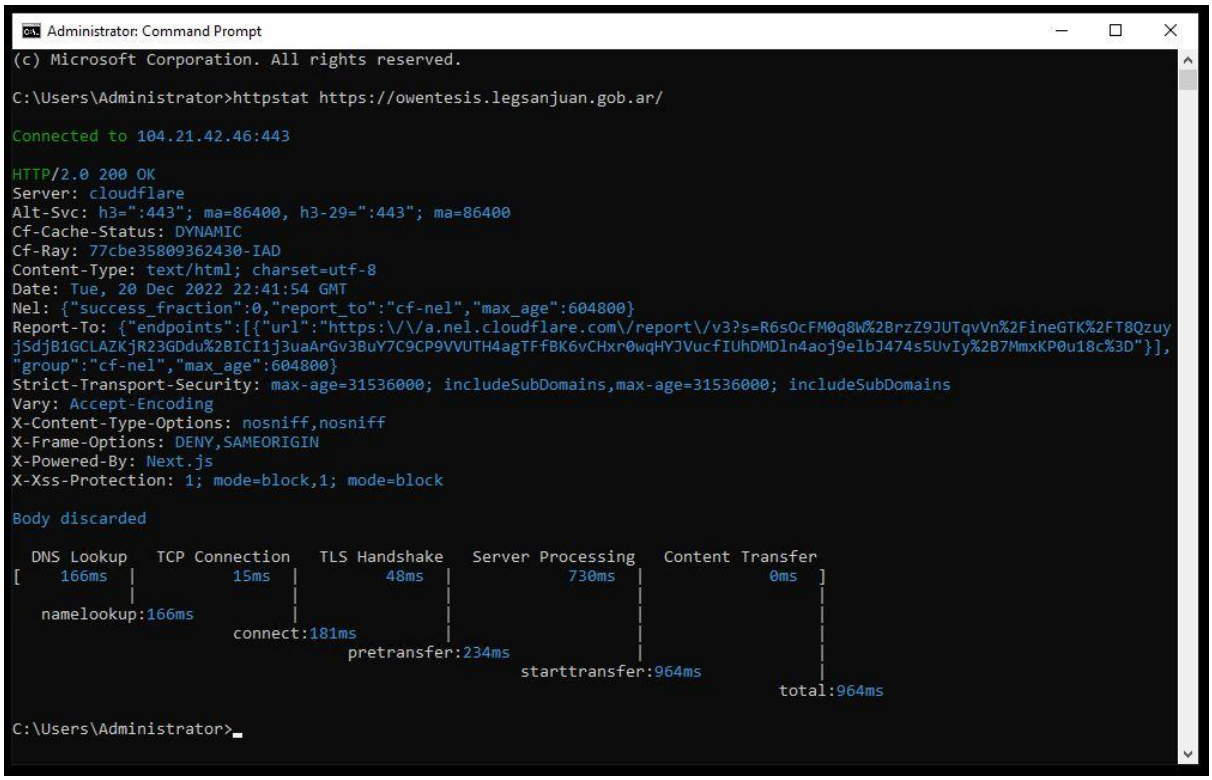



**Figura 36:** Muestra obtenida con HttpStat para Tecnologías Tradicionales (Contenido Estático)

Al igual que en el apartado anterior se crea una tabla comparativa a fin de ayudar a estudiar los valores obtenidos. En esta ocasión, además de agregar los resultados de la Figura 36, se incorporan a la Tabla 2 los datos de Funciones Serverless obtenidos por la herramienta httpstat para contenido estático, observados en la Figura 32.

| | Vercel | Legislatura |
|---|---|---|
| DNS Lookup | 14 ms | 166 ms |
| TCP Connection | 145 ms | 15 ms |
| TLS Handshake | 185 ms | 48 ms |
| Server Processing | 127 ms | 730 ms |
| Content Transfer | 0 ms | 0 ms |

**Tabla 2:** Comparación de Contenido Estático obtenido con HttpStat

Nuevamente el primer apartado a analizar es DNS Lookup. Según la Tabla 2, se puede observar una situación muy similar a la obtenida sobre el contenido dinámico, es decir hubo una demora mayor en obtener la Dirección IP del dominio

del sistema almacenado en el servidor de la legislatura, en relación al mismo proceso, pero estando el sistema alojado en el servidor de Vercel, donde hubo más de 100 ms de diferencia.

A continuación, tanto en el ítem de TCP Connection como en TLS Handshake, se puede apreciar una situación favorable para las Tecnologías Tradicionales, donde el Tiempo de Respuesta para ambos apartados fue mucho menor en comparación con las Funciones Serverless. Es decir, para las Funciones Serverless (en este caso utilizando la plataforma Vercel) fueron necesarios 145 ms para llevar a cabo el proceso de establecer la conexión TCP entre ambas partes, y posteriormente 185 ms para acordar las claves de sesión y algoritmos de criptografía. Una Latencia mucho mayor a la obtenida en la muestra de Tecnologías Tradicionales, a pesar de haberse agregado el uso de la máquina virtual ubicada en **US East (N. Virginia),** con una diferencia de más de 100 ms.

Finalmente, en el apartado de Server Processing, se puede observar que el Servidor de Vercel (al igual que en el contenido dinámico) es capaz de procesar la solicitud y brindar una respuesta en un tiempo mucho menor que el Servidor de la Legislatura. Es decir, tanto en la muestra analizada para contenido dinámico como la muestra sobre el contenido estático, se obtienen resultados favorables para las Funciones Serverless, en relación al procesamiento de la solicitud enviada por el cliente.

# Capítulo V

# Conclusión

## 5.1. Evaluación y Conclusiones

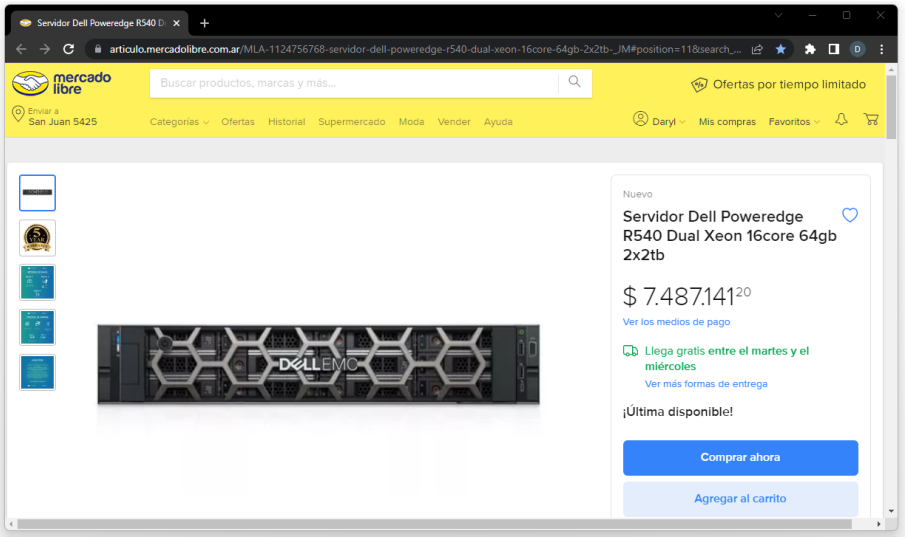
El presente trabajo ha permitido comparar la eficiencia de la estructura del lado del Cliente de una Página Web, mejor conocida como Front End, utilizando Funciones Serverless y Tecnologías Tradicionales, a fin de poder establecer cuál es la mejor opción. Se llevó a cabo la creación de una Página Web y posteriormente el despliegue de su Front End tanto en un Servidor Local como en una Plataforma Serverless.

Se han mostrado los resultados de las Herramientas de Análisis de Performance Autocannon y HttpStat, sobre Contenido Dinámico y Contenido Estático, de la Página Web desarrollada para esta finalidad. Mediante el estudio de los datos obtenidos, se ha podido determinar que el Tiempo de Respuesta es mucho mejor al utilizar Tecnologías Tradicionales, frente a la aplicación de Funciones Serverless,

cuyo punto a favor fue solamente la velocidad con la cual el Servidor pudo procesar las solicitudes. Si bien un servidor rápido en el procesamiento de peticiones es un punto importante, solo con ese dato no podríamos llegar a una conclusión sólida.

De acuerdo a todos los datos analizados, estudiados, y tomando como referencia algunas partes del Capítulo de Desarrollo, como por ejemplo lo evaluado en la Figura 21 y la Figura 24 (entre otras) se podría afirmar que el uso de Tecnologías Tradicionales es la mejor opción para un desarrollador al momento de desplegar el Front End de la Página Web que está programando, debido a un Tiempo de Respuesta, en general, mucho más rápido. Sin embargo, hay otros factores muy importantes a tener en cuenta antes de dar el veredicto final.

Por un lado, el dinero, adquirir un Servidor con las prestaciones como el que dispone la Legislatura es sumamente costoso, en la Figura 37 se coloca una publicación en Mercadolibre de un Servidor cuyas características son similares al utilizado para realizar las pruebas de Tecnologías Tradicionales. El mismo es la versión R540 Poweredge de un Servidor Dell, cuenta con un procesador Dual Xeon Silver 4208, 2 TB de Disco duro y 64 GB de




memoria Ram [59].



**Figura 37:** Publicación de Servidor en Mercado Libre

Se puede observar que el desarrollador debería contar con una gran cantidad de dinero para poder comprarlo. Además, una vez obtenido, el programador necesita tener los conocimientos suficientes para poder instalarlo, configurarlo y posteriormente llevar a cabo el mantenimiento correspondiente. Esto requiere tiempo adicional al necesario para el desarrollo del Sistema, teniendo en cuenta que él es encargado de realizar todo el trabajo, ya que si dispone sólo con estudios de programación debería buscar otras personas encargadas de realizar esta tarea, otra desventaja ya que se sumaría un nuevo costo a la adquisición del Servidor. Sin mencionar que aumente la cantidad de accesos al Sistema por parte de un incremento en la demanda de clientes, y la respuesta del mismo empiece a ser lenta, se deberían agregar más recursos al Servidor para poder solventar dicha demanda, otro costo adicional para el desarrollador. Incluso si la demanda crece tanto se puede llegar al punto en donde dicho Servidor no cuenta con espacio físico para seguir agregando recursos y se tendría que optar por cambiarlo o agregar un segundo servidor, para no perder clientes debido a un tiempo de respuesta lento, al carecer de recursos.

Estas desventajas se ven solucionadas si el programador opta por la utilización de Funciones Serverless, donde la elasticidad y escalabilidad es prácticamente inmediata, solo basta con abonar una cuota mensual y hasta quizás simplemente cambiar de plan.

Además, vale la pena mencionar que la naturaleza de los Servicios Serverless hace que los precios en línea sean más factibles en comparación con otros servicios en la nube. En esos servicios como IaaS, el costo de mover y mantener varias máquinas virtuales en varios proveedores de servicios es más alto en comparación con el de la ubicación de funciones en varios proveedores serverless.

Sin embargo, en la práctica, los principales proveedores serverless solo ofrecen precios estáticos, ya que es más simple para los clientes, al evitar la confusión. Este esquema de precios ha dado lugar a algunos argumentos sobre que las aplicaciones sin servidor en realidad pueden costar más en comparación con otros enfoques heredados, y por lo tanto se tiene que cambiar [10].

No obstante, el actual esquema de precios estáticos, hace que la tarea de predecir los costos del cliente sea sencilla, ya que dichos costos solo dependen del uso de los recursos y no de otros costos algo más difíciles para predecir parámetros (como el tiempo de uso) [11].

Por ejemplo, en la Figura 38, se puede observar que en la Plataforma Vercel, el desarrollador puede optar por una opción gratuita, con la finalidad de probar ciertas

características básicas para desplegar su sistema. Además, también dispone de una versión llamada Pro, donde abonando 20 dólares al mes puede acceder a características como 1 TB de ancho de banda, solicitudes de clientes ilimitadas, revisar las interfaces de usuario con un equipo de trabajo, 1,000 GB-hours de ejecución de funciones sin servidor, entre otras tantas ventajas de la versión Pro [60].

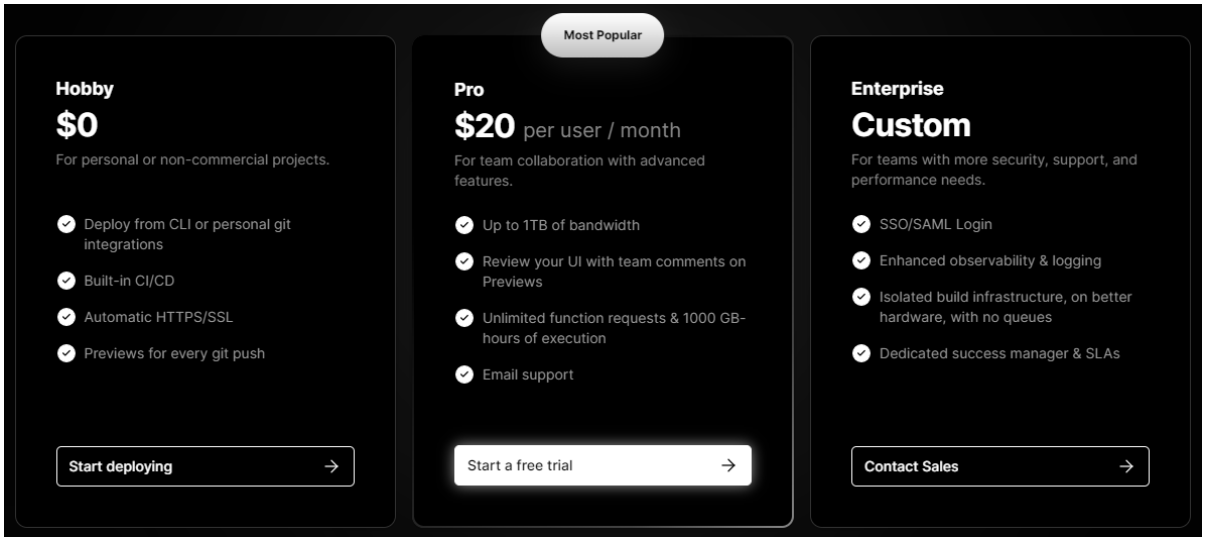



**Figura 38:** Opciones de Planes

Se puede apreciar que el abono mensual representa una cantidad de dinero mínima si se compara con la compra de un Servidor. Además, al utilizar la Plataforma Serverless el programador no necesita disponer de conocimientos sobre Servidores de ningún tipo, tanto la instalación y configuración de los mismos como el mantenimiento requerido se encarga dicha Plataforma, el programador solo se centra en el desarrollo de su Sistema. Incluso si el mismo necesitará más cantidad de recursos ya que fue creciendo el número de clientes que acceden, sólo se debería cambiar de plan (por uno personalizado de acuerdo a sus necesidades) y la aplicación automáticamente cuenta con los recursos para un buen funcionamiento acorde a la demanda, el poder incorporar más recursos es muy sencillo a diferencia del uso de Tecnologías Tradicionales.

Debido a lo expuesto anteriormente y que la diferencia de performance no impacta directamente en la experiencia del usuario, se puede establecer que el uso de Funciones Serverless para montar el Front End de una Página web es la opción más conveniente para un desarrollador, frente a la utilización de Tecnologías Tradicionales.

## 5.2. Futuros Trabajos

- Llevar a cabo el análisis de la comparación de la eficiencia de Tecnologías Tradicionales de Front End versus Funciones Serverless usando la Plataforma Serverless de Microsoft (Microsoft Azure), de Amazon (Amazon Web Services), de Google (Google Cloud Platform) y de IBM (IBM Cloud Platform).

# Bibliografía

[1] CNCF Serverless Working Group, 2018. CNCF WG-Serverless Whitepaper v1.0. Accessed 8.4.2019. Available:
https://github.com/cncf/wg-serverless/raw/master/whitepapers/serverless-overview/cncf_serverless_whitepaper_v1.0.pdf

[2] Al-Ameen, M., & Spillner, J. (2018). Systematic and open exploration of FaaS and Serverless Computing research. In *ESSCA@ UCC* (pp. 30-35).

[3] Pons, D. B., Ollobarren, A. R., Pinto, D. A., & López, P. G. (2018). Studying the feasibility of serverless actors. In *ESSCA@ UCC* (pp. 25-29).

[4] Yussupov, V., Breitenbücher, U., Leymann, F., & Wurster, M. (2019, December). A Systematic Mapping Study on Engineering Function-as-a-Service Platforms and Tools. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing* (pp. 229-240).

[5] Jangda, A., Pinckney, D., Brun, Y., & Guha, A. (2019). Formal foundations of serverless computing. *Proceedings of the ACM on Programming Languages*, *3*(OOPSLA), 1-26.

[6] Meissner, D., Erb, B., Kargl, F., & Tichy, M. (2018, June). Retro-λ: An Event-sourced Platform for Serverless Applications with Retroactive Computing Support. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems* (pp. 76-87).

[7] Moczurad, P., & Malawski, M. (2018, December). Visual-textual framework for serverless computation: a Luna Language approach. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)* (pp. 169-174). IEEE.

[8] Historia del HTML: Los Inicios. Available: http://www.manualweb.net/html/historia-html-inicios/

[9] Breve historia de CSS. Available: https://uniwebsidad.com/libros/css/capitulo-1/breve-historia-de-css

[10] Manual de CSS 3. Available: https://desarrolloweb.com/manuales/css3.html

[11] Qué es Javascript. Available: https://openwebinars.net/blog/que-es-javascript/

[12] What is cloud computing? Available: https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud

[13] Plataforma Serverless Híbrida de Procesado de Datos. Available: https://riunet.upv.es/handle/10251/125892

[14] Servicio de Infraestructura (IaaS). Available:
http://repositorio.unap.edu.pe/handle/UNAP/2304

[15] Desarrollo De Software Orientado a Prestación De Servicios - Universidad de San Carlos de Guatemala.
https://scholar.google.com/scholar?hl=es&as_sdt=0%2C5&q=Desarrollo+De+Software+Orientado+a+Prestaci%C3%B3n+De+Servicios&btnG=&lr=lang_es

[16] Ensi Maria. IaaS, PaaS, SaaS – What Do They Mean? | CloudOnMove
http://cloudonmove.com/iaas-paas-saas-what-do-they-mean/

[17] Información general sobre Amazon Web Services. Available:
https://docs.aws.amazon.com/es_es/whitepapers/latest/aws-overview/aws-overview.pdf

[18] Conformidad de AWS. Available: https://aws.amazon.com/es/compliance/

[19] Amazon Web Services. Available:
https://www.ticportal.es/temas/cloud-computing/amazon-web-services

[20] ¿Qué es AWS Lambda? Available:
https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html

[21] Introducción a Azure para desarrolladores. Available:
https://learn.microsoft.com/es-es/azure/developer/intro/azure-developer-overview

[22] Servicios clave de Azure para desarrolladores. Available:
https://learn.microsoft.com/es-es/azure/developer/intro/azure-developer-key-services

[23] Conectar la aplicación a los servicios de Azure. Available:
https://learn.microsoft.com/es-es/azure/developer/intro/connect-to-azure-services

[24] Azure REST API reference. Available:
https://learn.microsoft.com/es-es/rest/api/azure

[25] What is the IBM Cloud platform? Available: https://cloud.ibm.com/docs/overview?topic=overview-whatis-platform

[26] IBM Cloud Functions. Available: https://cloud.ibm.com/functions/

[27] Geography and regions. Available: https://cloud.google.com/docs/geography-and-regions

[28] Google Cloud overview. Available: https://cloud.google.com/docs/overview

[29] Productos de Google Cloud. Available: https://cloud.google.com/products

[30] On-Premises. Available: https://www.suse.com/suse-defines/definition/on-premises/

[31] Cloud Computing vs. On-Premises: Advantages, Disadvantages, and Cost Comparison. Available: https://s-pro.io/blog/cloud-computing-vs-on-premises-advantages-disadvantages-and-cost-comparison

[32] What Is VERCEL? Is It The Right Platform For Front-End Developers? https://webo.digital/blog/what-is-vercel-is-it-the-right-platform-for-front-end-developers/

[33] Introduction to Vercel. Available: https://vercel.com/docs

[34] Node.js. Available: https://nodejs.org/es/

[35] What is Next.js? Available: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs

[36] Next.js; what is it and why do we use it? Available: https://www.clock.co.uk/insight/next-js-what-is-it-and-why-do-we-use-it

[37] How Next.js Works. Available: https://nextjs.org/learn/foundations/how-nextjs-works/code-splitting

[38] Özgür Sedefoğlu, Hasan Sözer. “Cost Minimization for Deploying Serverless Functions”. SAC ’21, March 22–26, 2021, Virtual Event, Republic of Korea. ACM ISBN978-1-4503-8104-8/21/03. Available: https://doi.org/10.1145/3412841.3442069

[39] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2022. Serverless Computing: A Survey of Opportunities, Challenges, and Applications. ACM Comput. Surv. 54, 11s, Article 239 (November 2022), 32 pages. Available: https://doi.org/10.1145/3510611

[40] Eoin Shanaghy. 2021. Why AWS Lambda Pricing Has to Change for the Enterprise. Retrieved September 24, 2021. Available: https://www.infoq.com/articles/aws-lambda-price-change/

[41] Cloud Pricing Comparison 2023: AWS vs Azure vs Google Cloud: Available: https://www.simform.com/blog/compute-pricing-comparison-aws-azure-googlecloud/

[42] Cloud Pricing Comparison: AWS vs. Azure vs. Google Cloud Platform in 2023. Available: https://cast.ai/blog/cloud-pricing-comparison-aws-vs-azure-vs-google-cloud-platform/

[43] Chapter 1. Understanding web performance. Available: https://livebook.manning.com/book/web-performance-in-action/chapter-1/

[44] 14 Website Speed Optimization Tips: Techniques to Improve Performance and User Experience. Available: https://sematext.com/blog/improve-website-performance/

[45] What is… Dynamic content. Available: https://www.omniconvert.com/what-is/dynamic-content/

[46] WHAT IS STATIC CONTENT? Available: https://www.stackpath.com/edge-academy/what-is-static-content/

[47] FAST. Available: https://fast.com/es/

[48] Acerca de Node.js®. Available: https://nodejs.org/es/about/

[49] Autocannon is an Apache Bench alternative in Node.js. Available: https://react-etc.net/entry/autocannon-is-an-apache-bench-alternative-in-node-js

[50] Autocannon. Available: https://www.npmjs.com/package/autocannon

[51] README, go-httpstat. Available: https://pkg.go.dev/github.com/tcnksm/go-httpstat#section-readme

[52] ¿Cómo reducir los DNS Lookup? Available: https://www.hostinger.com.ar/tutoriales/reducir-dns-lookup

[53] TCP Connections. Available: https://www.oreilly.com/library/view/http-the-definitive/1565925092/ch04s01.html

[54] ¿Qué ocurre durante un protocolo de enlace TLS? | Protocolo de enlace SSL. Available: https://www.cloudflare.com/es-es/learning/ssl/what-happens-in-a-tls-handshake/

[55] IBM Tivoli Storage Manager, Version 7.1. Available: https://www.ibm.com/docs/en/tsm/7.1.0?topic=operations-processes-that-run-server

[56] Data Transfer. Available: https://www.techopedia.com/definition/18715/data-transfer

[57] What Is Response Time? Available: https://sematext.com/glossary/response-time/

[58] Cómo usar TRACERT para solucionar problemas de TCP/IP en Windows. Available: https://support.microsoft.com/es-es/topic/c%C3%B3mo-usar-tracert-para-solucionar-problemas-de-tcp-ip-en-windows-e643d72b-2f4f-cdd6-09a0-fd2989c7ca8e

[59] Servidor Dell Poweredge R540 Dual Xeon 16core 64gb 2x2tb. Available: https://articulo.mercadolibre.com.ar/MLA-1124756768-servidor-dell-poweredge-r540-dual-xeon-16core-64gb-2x2tb-_JM#position=11&search_layout=stack&type=item&tracking_id=032ad6e1-0791-4caa-b7fd-2f4029f3a237

[60] Find a plan to power your projects. Available: https://vercel.com/pricing