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

GRPC COMO MEDIO DE COMUNICACIÓN ENTRE MICROSERVICIOS, UNA ALTERNATIVA A *REST*, CARACTERISTICAS Y VENTAJAS

Pavel Alexander Vásquez Flores
Asesorado por el Ingeniero Jorge Mario Gutierrez Ovando

Guatemala, marzo de 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

GRPC COMO MEDIO DE COMUNICACIÓN ENTRE MICROSERVICIOS, UNA ALTERNATIVA A *REST*, CARACTERISTICAS Y VENTAJAS.

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

PAVEL ALEXANDER VÁSQUEZ FLORES
ASESORADO POR EL INGENIERO JORGE MARIO GUTIERREZ OVANDO

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MARZO DE 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

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

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. Álvaro Giovanni Longo Morales
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. César Rolando Batz Saquimux
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

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

GRPC COMO MEDIO DE COMUNICACIÓN ENTRE MICROSERVICIOS, UNA ALTERNATIVA A *REST*, CARACTERISTICAS Y VENTAJAS.

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha mayo de 2022.

Pavel Alexander Vásquez Flores

Guatemala, 25 de septiembre de 2022

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

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **PAVEL ALEXANDER VASQUEZ FLORES** con carné **201503611 y CUI 3020 39937 0101** titulado **"GRPC COMO MEDIO DE COMUNICACIÓN ENTRE MICROSERVICIOS, UNA ALTERNATIVA A REST, CARACTERISTICAS Y VENTAJAS"**, luego de corroborar que el mismo se encuentra finalizado, lo he revisado y doy fé de que el mismo cumple con los objetivos propuestos en el respectivo protocolo, por consiguiente, procedo a la aprobación correspondiente.

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

Atentamente,

Ing. Jorge Mario Gutierrez Ovando
Colegiado No. 18735

Jorge Mario Gutierrez Ovando
Ingeniero en Ciencias y Sistemas
Colegiado No. 18735

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

Guatemala 28 de septiembre de 2022

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

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **PAVEL ALEXANDER VÁSQUEZ FLORES** con carné **201503611** y CUI **3020 39937 0101** titulado **"GRPC COMO MEDIO DE COMUNICACIÓN ENTRE MICROSERVICIOS, UNA ALTERNATIVA A REST, CARACTERISTICAS Y VENTAJAS"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

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

Atentamente,

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

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

LNG.DIRECTOR.049.EICCSS.2023

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **GRPC COMO MEDIO DE COMUNICACIÓN ENTRE MICROSERVICIOS, UNA ALTERNATIVA A REST, CARACTERISTICAS Y VENTAJAS**, presentado por: **Pavel Alexander Vásquez Flores** , procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

"ID Y ENSEÑAD A TODOS"

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

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

Guatemala, marzo de 2023

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

LNG.DECANATO.OI.256.2023

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **GRPC COMO MEDIO DE COMUNICACIÓN ENTRE MICROSERVICIOS, UNA ALTERNATIVA A REST, CARACTERÍSTICAS Y VENTAJAS,** presentado por: **Pavel Alexander Vásquez Flores,** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANA
FACULTAD DE INGENIERÍA

Inga. Aurelia Anabela Cordova Estrada
Decana

Guatemala, marzo de 2023

AACE/gaoc

ACTO QUE DEDICO A:

Dios	Porque me ha regalado la vida
Mis padres	Imelda Flores y Walter Vásquez, por todo su apoyo, por invertir en mis estudios y darme todo lo necesario para alcanzar este día.
Mis hermanos	Denis, Nayeli Vásquez y Francisco Santos, por su apoyo y amor incondicional.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser mi alma mater, y que me permitió convertirme en el profesional que soy hoy en día.
Jeralmy de León	Por su apoyo incondicional, compañía durante todo este tiempo.
Mis amigos	Por traer a mi vida gran felicidad y ser la motivación para seguir adelante.
Ing. Jorge Mario Gutierrez	Por todo su apoyo y asesoría en la realización del presente trabajo de graduación.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
$	Dólar
GiB	Gibibyte
Gb	Gigabyte
MiB	Mebibyte
Mb	Megabyte
m	Metros
ms	Milisegundos
%	Porcentaje

GLOSARIO

api	Interfaz de programación de aplicaciones.
aws	*Amazon Web Services*, se refiere al conjunto de servicios de computación en la nube bajo demanda que ofrece *AWS*.
ec2	*Elastic Cloud Computin*g, es un servicio de cómputo que ofrece *AWS*.
http	Es el protocolo de transferencia de hipertexto que permite la comunicación a través de la *World Wide Web*.
json	Es la notación de objeto para *Javascript*.
npm	Es el gesto de paquetes de *NodeJs*.
rpc	Se refiere a la llamada a procedimientos remotos.
s3	*Simple Storage Service*, es un servicio de almacenamiento de objetos que ofrece *AWS*.
www	Se refiere a la *World Wide Web*.
xml	Es un lenguaje de marcado extensible.

RESUMEN

Generalmente los nuevos sistemas que surgen, están pensados para funcionar con una arquitectura basada en microservicios, esta práctica se volvió popular, ya que provee las siguientes ventajas: bajo acoplamiento, escalabilidad, independencia de servicios, reutilización, agilidad en el desarrollo, entre otros.

La facilidad de implementar microservicios viene debido al hecho de que la comunicación entre microservicios es agnóstica al lenguaje de programación, por lo que la forma más sencilla de comunicar microservicios fue mediante *API Rest*, esto relacionado al hecho de que solo es necesario manejar una interfaz de definición de lenguaje común para que los microservicios entiendan las peticiones y respuestas entre cliente y servidor. En la actualidad existen otros métodos para comunicar microservicios, para este caso se hace énfasis en gRPC, el cual es una implementación de las llamadas a procedimientos remotos en el que se permite realizar una comunicación entre cliente y servidor.

En gRPC existen cuatro tipos de comunicación entre los cuales se encuentran las llamadas unarias (uno a uno), llamadas unarias a transmisión, llamadas de transmisión a unarias y llamadas bidireccionales, estas últimas permiten una comunicación en tiempo real.

Ya que las *API Rest* y gRPC son tecnologías para comunicar microservicios es necesario determinar ventajas y casos de uso entre ambas, esto se logra realizando pruebas de carga concurrentes, secuenciales, con archivos, con el objetivo de identificar cual posee tiempos de respuesta y procesamiento menores bajo ciertas condiciones, además se debe verificar cual

es el consumo de recursos de *hardware* y cual es más propensa a rechazar peticiones. Luego de aplicar procedimientos estadísticos como hipótesis, pruebas de normalidad, pruebas de varianza a los resultados obtenidos se pudo determinar que en la mayoría de los casos utilizar gRPC proporciona mejores tiempos de respuesta y procesamiento en general, asimismo el monitoreo de los sistemas mostro que gRPC consume una cantidad de recursos relativamente mayor a *Rest*.

Otro punto importante fue la transmisión de archivos entre microservicios, cuando se envían archivos completos del cliente al servidor ambas tecnologías manejan tiempos de respuesta y procesamiento similares, pero *Rest* posee los tiempos más bajos, el otro punto a destacar es el hecho de que enviar un archivo por partes utilizando gRPC hace que el tiempo de respuesta y procesamiento aumente, dando a entender que el caso de uso principal para la comunicación bidireccional es para entablar una conexión de una larga duración y realizar un procesamiento de los mensajes entrantes sin haber completado la transmisión del resto de mensajes, ejemplos claros de esto son los servicios de chats o plataformas de reproducción de videos.

OBJETIVOS

General

Identificar las características y ventajas de gRPC sobre *REST* para la comunicación entre microservicios, por medio de pruebas en *Elastic Cloud Computing* (*EC2*) de la plataforma en la nube *Amazon Web Services* (*AWS*).

Específicos

1. Identificar las características de gRPC y *Rest*.

2. Diseñar las pruebas de comparación entre gRPC y *Rest*.

3. Ejecutar una serie de pruebas entre gRPC y *Rest* en la plataforma *AWS* utilizando el servicio *EC2*.

4. Almacenar los resultados de las pruebas utilizando el servicio *Simple Storage Service* (*S3*) de la plataforma *AWS*.

5. Analizar estadísticamente por medio de métodos cuantitativos los resultados obtenidos.

INTRODUCCIÓN

Las Interfaces de programación de aplicaciones (*API*) se han vuelto parte vital en el desarrollo de aplicaciones tanto móviles como web, son la base para comunicar aplicaciones desarrolladas en el mismo o diferente lenguaje de programación. Los esquemas utilizados pueden ir desde cliente a servidor hasta servidor a servidor.

Inicialmente el protocolo simple de acceso a objetos (*SOAP*) era muy utilizado, ya que era sencillo de utilizar y contaba con el lenguaje de descripción de servicios web (*WSDL*) para explorar los servicios disponibles, esto facilitaba el trabajo de los desarrolladores cuando tenían que utilizar *SOAP*. Cabe mencionar que las *API SOAP* se comunican mediante *XML* lo que en un inicio puede parecer ideal, si se analizan las entradas y salidas que se envían como carga útil en las solicitudes y respuestas de las aplicaciones estas tienden a ser pesadas, lo que genera un mayor tiempo de respuesta, mayor tiempo para procesar la información, entre otros.

Con el auge de la transferencia de estado representacional (*REST*) el uso de *SOAP* fue decayendo, *REST* utiliza los métodos *http GET*, *POST*, *PUT*, *PATCH*, *DELETE*, entre otros y un formato de transferencia de información llamado *javascript object notation* (*JSON*), este formato es muy sencillo de entender tanto para máquinas como para personas y es compatible con casi todos los lenguajes de programación. Una desventaja de *REST* es que no contiene nativamente una exploración de servicios como *WSDL* en *SOAP*, aunque para las *APIs* desarrolladas bajo *REST* se utiliza la especificación *OpenAPI* para producir archivos de interfaz en la cual se pueden describir,

visualizar y probar los servicios desarrollados en *REST*, esto se conoce como un proceso de documentación del *API*. En la actualidad existen otras tecnologías para comunicar microservicios, entre ellas se puede mencionar a gRPC el cual es una implementación de las llamadas a procedimientos remotos en el que se permite realizar una comunicación entre cliente y servidor, posee múltiples implementaciones de comunicación como unario, unario a transmisión, transmisión a unario y bidireccional, cada una con un caso de uso diferente, además posee la capacidad de enviar la carga útil de las peticiones en formato binario, lo que ayuda en el tamaño de la petición.

1. MICROSERVICIOS

La arquitectura de microservicios es un enfoque de arquitectura y organización para la estructura de aplicaciones, consiste en dividir la aplicación en pequeños componentes independientes permitiendo un desarrollo ágil, frecuente y escalable para aplicaciones complejas, los componentes son un conjunto de servicios con las siguientes características:

- Independientes entre microservicios: El concepto de microservicios se basa en múltiples componentes que funcionan de forma independiente, esto quiere decir que se puede desarrollar, implementar, desplegar y escalar sin afectar a otros microservicios del mismo sistema. Al ser independientes cada microservicio es único y esto permite una mayor flexibilidad durante el desarrollo, el microservicio 1 puede estar desarrollado en *NodeJs* utilizando gRPC mientras que el microservicio 2 puede estar desarrollado en Python utilizando un *API Rest*.

    - Poco acoplados: Ser poco acoplados permite que sean independientes, en otras palabras, la modificación de un componente afecta en poco o nada la funcionalidad o rendimiento de otro componente. En muchos casos los microservicios pueden comunicarse entre sí, creando una forma de dependencia, esto no es algo malo, siempre y cuando se mantenga el acoplamiento bajo para mantener las características de los microservicios.
    - Escalable: Cuando la demanda en el sistema aumenta es muy común utilizar un enfoque de escalado, ya sea horizontal o vertical. El escalado horizontal consiste en crear nuevas instancias de la

aplicación mientras que el escalado vertical consiste en aumentar las características del *hardware*, en cualquiera de los casos en una arquitectura de microservicios el escalado se aplica únicamente al componente que tiene una alta demanda, por lo que los otros componentes mantienen su estado.

- Reusable: Cada componente posee una funcionalidad, por lo que no es necesario realizarla de nuevo para otra aplicación, de esta manera con la arquitectura de microservicios solamente se utiliza el componente que posee la funcionalidad.

1.1. Monolito

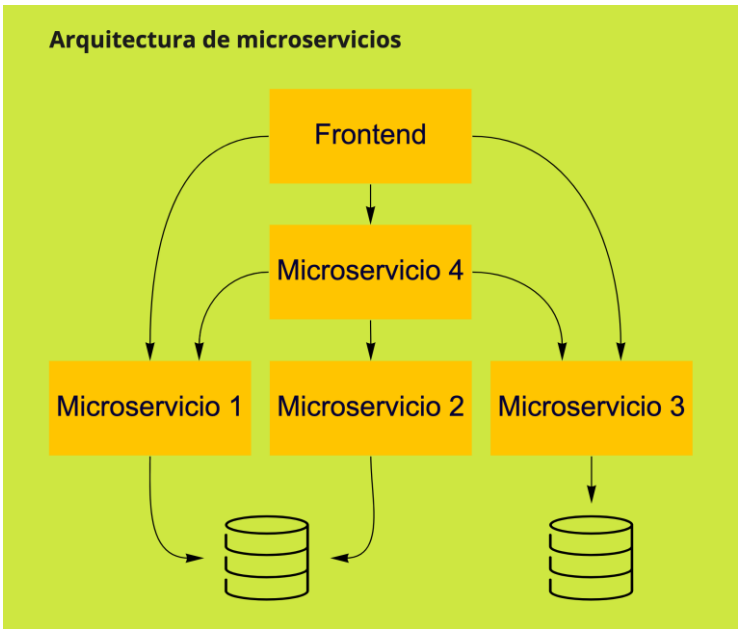
La arquitectura monolítica es un estilo de arquitectura en el cual se construye un sistema altamente acoplado, se puede considerar a este sistema como una única unidad que contiene todas las funcionalidades y posee las siguientes características:

- Todos los miembros del equipo trabajan sobre el mismo proyecto.
- El lenguaje de programación es único.
- Altamente acoplado.
- Un problema en el sistema puede afectar a toda la aplicación.

Figura 1. **Diagrama de arquitectura monolítica**

Fuente: elaboración propia, realizado en Miro.

Figura 2. **Diagrama de arquitectura de microservicios**



Fuente: elaboración propia, realizado en Miro.

1.2. Escalabilidad

La escalabilidad es una manera de aumentar o disminuir el rendimiento, tamaño y disponibilidad de una aplicación en la nube, este concepto es muy común para aplicaciones en la nube en donde la escalabilidad se consigue de forma sencilla, además la escalabilidad se puede aplicar para arquitecturas monolíticas y de microservicios. Existen varios tipos de escalamiento en la nube, en esta sección nos vamos a enfocar en el escalado vertical y horizontal.

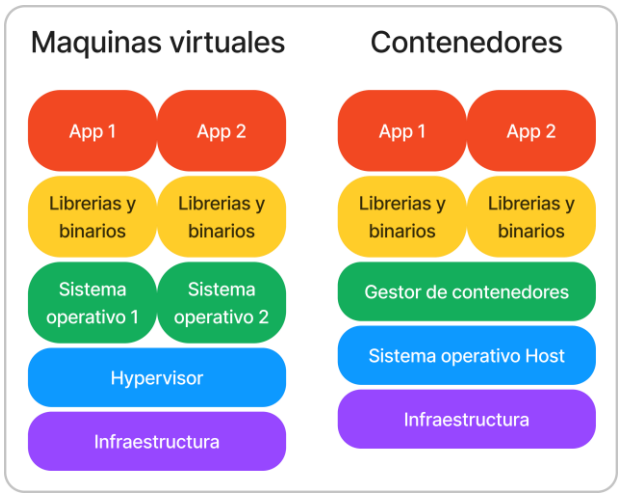
- Escalado vertical: Este tipo de escalado se produce cuando se actualiza un solo recurso tecnológico de tal forma en que se aumenta su memoria, almacenamiento, ancho de banda, entre otras.

- Escalado horizontal: Este tipo de escalado se produce cuando se crea una nueva o varias instancias del recurso tecnológico, creando múltiples recursos del mismo tipo logrando de esta manera aumentar los recursos de *hardware* con otros recursos.

1.3. Contenedores

Los contenedores son unidades pequeñas de *software* en donde se empaquetan y ejecutan aplicaciones de forma aislada. Los contenedores virtualizan a nivel de sistema operativo, en otras palabras, la capa de virtualización se ejecuta como una aplicación en el sistema operativo del host, además comparten el *kernel* con el *host* y el uso de memoria es reducido. En los contenedores se pueden ejecutar bases de datos, microservicios y demás procesos. Ejecutar aplicaciones en contenedores permite mantener un entorno idéntico para desarrollo y producción.

Figura 3. **Diferencia entre máquinas virtuales y contenedores**



Fuente: elaboración propia, realizado en FigJam.

Entre las ventajas del uso de contenedores se encuentran las siguientes:

- Portabilidad: el uso de contenedores permite que las aplicaciones sean portables, ya que se pueden ejecutar en cualquier sistema operativo de forma sencilla.
- Aislamiento: los contenedores aprovechan los recursos del sistema operativo para controlar el almacenamiento, memoria, *CPU* y aislar los procesos, esto permite una separación lógica entre contenedores.
- Despliegue: el uso de contenedores permite que las aplicaciones se desplieguen de forma ágil, ya que el proceso se puede automatizar utilizando integración y despliegue continuo.

1.4. Pruebas y monitoreo

Las pruebas y monitoreo son una parte importante en el desarrollo, despliegue y seguimiento de aplicaciones, en este capítulo se profundiza en el uso de herramientas para pruebas y monitoreo enfocado en microservicios.

Inicialmente se debe verificar el correcto funcionamiento del código, esto se puede realizar mediante pruebas unitarias y utilizando herramientas especializadas para *APIs*, estas herramientas están diseñadas con el fin de realizar pruebas de los múltiples servicios que el *API* expone y brindar información relevante, por ejemplo, el tiempo de respuesta, si la petición fue exitosa u ocurrió alguna inconveniente.

El monitoreo de sistemas es clave para el seguimiento de la aplicación luego de su publicación, asimismo monitorear los sistemas permite verificar si los servicios funcionan de forma adecuada. Otro punto importante es verificar el rendimiento de la aplicación mediante el uso de recursos de *hardware*, además

es necesario rastrear las peticiones de los clientes a través de los microservicios y de esta forma resolver los problemas que surjan de forma eficiente.

1.4.1. *Postman*

Postman es una herramienta para construir y realizar pruebas de *APIs*, *Postman* permite crear peticiones para probar cada uno de los servicios expuestos por un *API*, de tal forma que se pueden configurar diferentes escenarios de prueba y verificar el correcto funcionamiento del servicio.

Postman permite crear y almacenar colecciones con peticiones de tipo *http* y gRPC, de esta manera se puede reutilizar la petición e incluso ejecutar un proceso para probar todas las peticiones almacenadas en una colección y verificar el funcionamiento los servicios. *Postman* posee varias funcionalidades entre las cuales se puede mencionar:

- Diseño y desarrollo colaborativo de *APIs*
- Creación de colecciones
- Creación de peticiones *http* y gRPC
- Pruebas unitarias de las peticiones
- Métricas de ejecución de peticiones
- Ejecución de lote de peticiones
- Creación de monitores para verificar el funcionamiento de los servicios
- Notificaciones de monitores cuando ocurra algún inconveniente con los servicios
- Creación de *Mock servers*
- Creación de flujos de trabajo para la ejecución de peticiones

1.4.2. *Prometheus*

Es un conjunto de herramientas para monitoreo, *Prometheus* recopila, almacena la información de las métricas como series de tiempo junto con un conjunto opcional de etiquetas, cada serie de tiempo se identifica con un nombre de métrica y especifica una característica general de medición. *Prometheus* posee cuatro tipos de métricas principales para representar información, estas métricas se alimentan de los datos recopilados y son las siguientes:

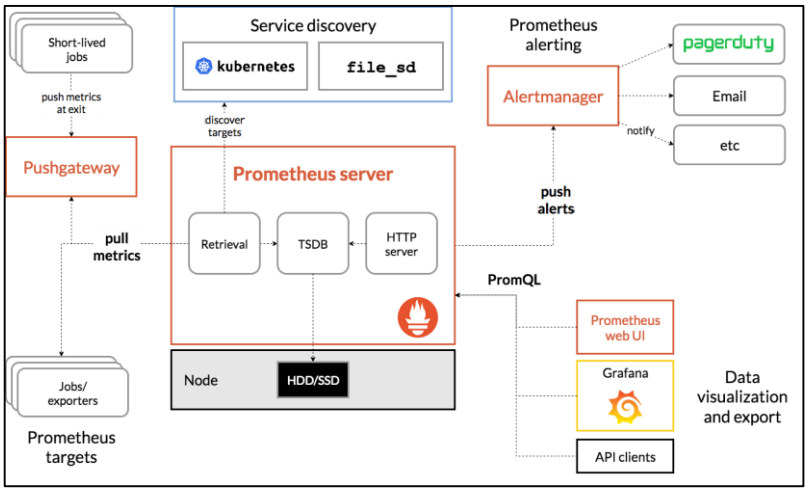
- Contador: es un valor numérico que avanza de forma ascendente desde cero, puede ser utilizado para contar la cantidad de peticiones exitosas y fallidas ocurridas en cierto rango de tiempo
- Medidor: es un valor numérico que puede aumentar o disminuir con base a los valores recolectados, puede ser utilizado para medir el uso de *CPU* o la cantidad de memoria utilizada por el sistema
- Histograma: es un conjunto de contadores en los cuales se mide la frecuencia de los valores en las observaciones, por ejemplo, la cantidad de peticiones fallidas durante cada hora del día
- Resumen: similares a los histogramas, su procesamiento conlleva mayor esfuerzo en el cliente al contrario de los histogramas donde el mayor esfuerzo lo hace el servidor, además los resultados en la métrica de resumen son valores poco precisos

Prometheus posee varias maneras de visualizar la información y realizar consultas, aunque una característica importante es que se puede integrar con otras herramientas visualización como Grafana. Para utilizar *Prometheus* es necesario exportar datos de la aplicación al servidor de *Prometheus*, esto se logra utilizando librerías de instrumentación, estas librerías permiten la exportación de forma manual o automática de los datos, de tal forma que *Prometheus* recolecta

y procesa los datos. Adicionalmente *Prometheus* posee las siguientes características:

- Capacidad de enviar alertas con base en ciertos eventos
- Un modelo de datos multidimensional
- Un lenguaje de consultas llamado *PromQL*
- Recopilación de datos a través *http*
- Múltiples modos de soporte para *dashboards*

Figura 4. **Arquitectura de Prometheus y componentes**



Fuente: *Prometheus Authors* (2022). *Prometheus vista general*. Consultado el 10 de Julio de 2022. Recuperado de https://prometheus.io/docs/introduction/overview/.

1.4.3. Grafana

Grafana es un *software* de código abierto que permite realizar consultas, visualizaciones, alertar y explorar métricas, logs y trazabilidad en microservicios. Grafana proporciona herramientas para transformar los datos en información y gráficos que proporcionan valor. Grafana soporta diferentes fuentes de datos,

cada fuente de datos posee su propio lenguaje de consulta, aunque esto no impide combinar múltiples fuentes de datos en un único *dashboard* separando cada fuente en paneles. Entre las fuentes de datos más comunes se encuentran los siguientes:

- Alertmanager
- *AWS* CloudWatch
- Azure Monitor
- Elasticsearch
- Google Cloud Monitoring
- *Prometheus*
- Jaeger
- Zipkin

Los *dashboards* en Grafana son un conjunto de paneles. Los paneles son bloques básicos de visualización, cada panel tiene la opción de ser personalizado y puede contener una fuente de datos especifica por lo que cada panel tendrá su propio editor de consultas. Grafana posee muchos tipos de visualizaciones entre los cuales se pueden mencionar los siguientes:

- Series de tiempo
- Gráfico de barras
- Histogramas
- Mapas de calor
- Gráfico de pie
- Estadísticas
- Medidores
- Tablas
- Trazabilidad

Grafana también permite enviar alertas, dichas alertas se envían luego de que se cumple la condición y la política de notificación, por ejemplo, que un servicio falle durante cierta cantidad de tiempo y la severidad de esta acción sea critica, de esta forma se puede configurar un canal de alerta como correo electrónico o Slack.

1.5. Microservicios con *Nodejs*

La arquitectura de microservicios es un enfoque de arquitectura y organización para la estructura de aplicaciones, la implementación de microservicios se puede llevar a cabo en muchos lenguajes de programación, el uso de *frameworks* es una práctica muy común, ya que proporcionan las herramientas necesarias para la construcción de sistemas distribuidos. Otro punto de interés en los microservicios es la flexibilidad que existe en el desarrollo, un microservicio puede ser desarrollado con *JavaScript* y otro con Golang, esta independencia del lenguaje de programación permite explorar en soluciones enfocadas al rendimiento, curva de aprendizaje y documentación.

1.5.1. *JavaScript*

Es un lenguaje de programación interpretado, imperativo, declarativo y de un único hilo. Una de las características más notables de *JavaScript* es el soporte para tipado dinámico, esto hace posible que el lenguaje se interpretado y la curva de aprendizaje sea baja. Al igual que otros lenguajes de programación, *JavaScript* maneja una definición léxica, sintáctica y semántica, este estándar se conoce como ECMASCRIPT.

JavaScript es un lenguaje ampliamente utilizado en muchos entornos de desarrollo, la flexibilidad de *JavaScript* permite ser utilizado de las siguientes

maneras:

- Desarrollo de páginas web
- Desarrollo de aplicaciones web
- Desarrollo de aplicaciones móviles
- Desarrollo de aplicaciones en entornos *backend*
- Implementaciones de aprendizaje automático

1.5.2. *NodeJs*

NodeJs es un entorno de ejecución de *JavaScript*, muy popular para el desarrollo de aplicaciones y especialmente para el desarrollo de servicios web, *NodeJs* está orientado a eventos asíncronos y funciona con un único hilo de ejecución, pero esto no impide el manejo de la ejecución de tareas en paralelo. *NodeJs* como sus homólogos en otros lenguajes de programación posee un gestor de paquetes llamado *NPM* (*Node Package Manager*), este gestor de paquetes es de los más populares a nivel mundial y posee una gran cantidad de librerías listas para ser utilizadas.

El lenguaje principal que utiliza *NodeJs* es *JavaScript*, pero también se puede integrar con *TypeScript*, agregando sus características de tipado estático, un patrón común es utilizar *TypeScript* para desarrollar y mediante un proceso de transpilación el cual resulta en código *JavaScript* listo para ejecutar. A pesar de que *NodeJs* maneja un único hilo de ejecución, se puede manejar la concurrencia, esto es gracias al bucle de eventos, la cola de eventos, la pila de llamadas, entre otras estructuras y *APIs* que proporciona *NodeJs*, los términos concurrencia y asíncrono son muy importantes en *NodeJs*, ya que apoyan al desarrollo de *API Rest* las cuales contienen funciones asíncronas y soportan concurrencia.

El uso por defecto de *NodeJs* permite la creación de microservicios, aunque es recomendable utilizar *frameworks* web para desarrollar aplicaciones pensadas en ser escalables y para lanzar en producción. Los *frameworks* web facilitan el desarrollo, ya que simplifican el desarrollo de código a comparación de no utilizar un *framework*, además proporcionan herramientas para el manejo de peticiones *http*, middlewares, gestores de rutas, motores de renderizado de vistas, gestores de autenticación, entre otros. *NodeJs* permite generar aplicaciones de alto nivel, ligeras y escalables, esto es un aliciente para la implementación de sistemas con una arquitectura de microservicios.

2. INTERFACES DE PROGRAMACIÓN DE APLICACIONES

Las interfaces de programación de aplicaciones también conocidas como *APIs* son un conjunto de métodos o funciones que permiten la comunicación entre sistemas. Se han vuelto parte vital en el desarrollo de aplicaciones tanto móviles como web, son la base para comunicar aplicaciones desarrolladas en el mismo o diferente lenguaje de programación. Los esquemas utilizados pueden ir desde cliente a servidor hasta servidor a servidor.

2.1. *API Rest*

Por sus siglas en inglés *representational state transfer* (*Rest*) es una forma de arquitectura para sistemas distribuidos la cual posee una serie de principios enfocados a una mejor implementación y buenas prácticas. Cuando se desarrolla un servicio web y bajo los principios *Rest* se puede decir que se está construyendo un *API Rest* o *Restful*.

2.1.1. Métodos *Http*

Los métodos o verbos *http* son utilizados cuando se hace una petición a un servidor, estos métodos se utilizan en el desarrollo de *API Rest* y permiten indicar la acción a realizar para un recurso determinado. Esta acción brinda un contexto semántico abstracto de la funcionalidad del servicio.

- *Get*: Este método indica una acción para obtener información o para ejecutar una función que no necesita de un cuerpo en la petición.
- *Post*: Este método indica una acción para guardar información.

- *Put*: Este método indica una acción para actualizar información.
- *Patch*: Este método indica una acción para actualizar parcialmente un recurso.
- *Delete*: Este método indica una acción para eliminar información.

2.1.2. Lenguaje de definición

Debido a que las *API Rest* comunican múltiples sistemas en diferentes lenguajes de programación es necesario definir un esquema común entre los sistemas, debido a ello surge el lenguaje de definición el cual es una interfaz de datos comprensible para los sistemas que utilizan el *API Rest*.

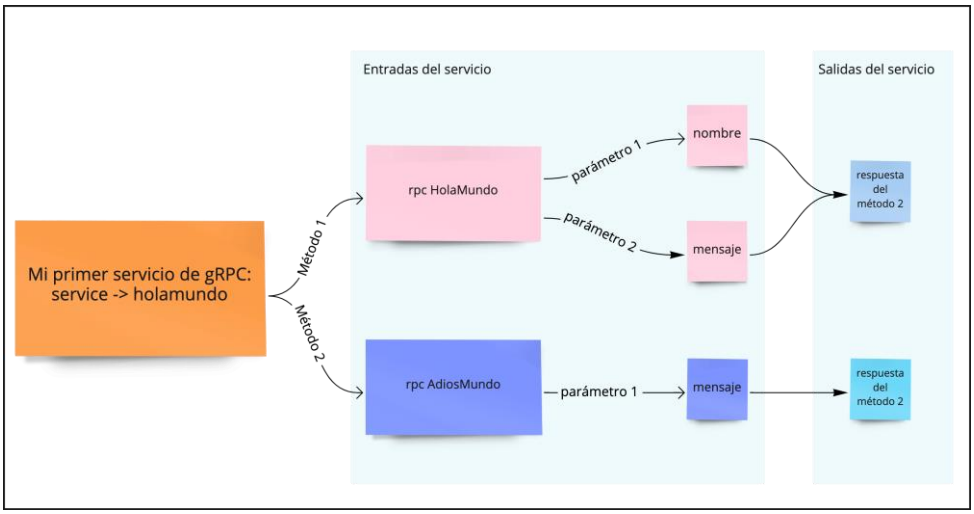
En *API Rest* es muy común utilizar *JSON* para las entradas y salidas de cada uno de los recursos, esta definición se puede utilizar en casi cualquier lenguaje de programación. Otras alternativas a *JSON* pueden ser *XML* o *YAML*.

2.2. gRPC

Es una implementación de las llamadas a procedimientos remotos (*rpc*) creado por Google y posteriormente cedido a la *Cloud Native Computing Foundation*, es una herramienta *open source* la cual permite la comunicación cliente servidor por medio de llamadas a métodos desde el cliente hacia el servidor, esto facilita la creación e implementación de aplicaciones y servicios distribuidos.

Un servicio en gRPC contiene uno o varios métodos con sus respectivas entradas y salidas, estos servicios se exponen al cliente por medio de la interfaz de definición, la cual indica los parámetros de entrada y el objeto de respuesta.

Figura 5. **Flujo simple para definición de servicio en *gRPC***



Fuente: elaboración propia, realizado en Miro.

Con gRPC se pueden conectar servicios de forma eficiente, entre las principales características se encuentra el hecho de que la información se transmite de forma binaria entre el cliente y el servidor, además tiene la posibilidad de enviar y recibir mensajes al mismo tiempo, esto se conoce como multiplexación y es posible gracias a que gRPC utiliza el protocolo de red *http*/2.

2.2.1. Lenguaje de definición de interfaz

Actualmente los lenguajes de programación poseen librerías, *frameworks* o entornos de ejecución que permiten la creación de servicios web, los estándares actuales para el desarrollo de sistemas se basan en arquitecturas orientadas a microservicios, esto da paso a que cada microservicio sea independiente y por lo tanto cada microservicio de un sistema pueda estar desarrollado en diferentes lenguajes de programación, gRPC utiliza el lenguaje

de definición de interfaz (*IDL*) para centralizar la comunicación entre varios lenguajes de programación.

Por defecto gRPC utiliza *protocol buffers* el cual es un lenguaje neutral, de plataforma neutral, extensible para serializar estructuras de datos, funciona como *XML* o *JSON*, pero es más rápido.

Los *protocol buffers* comparado con *JSON* manejan muchos más tipos de datos, entre los cuales se pueden mencionar los siguientes:

- Double
- Float
- Int32/64
- bool
- string
- bytes

Adicionalmente los *protocol buffers* poseen una serie de características las cuales proporcionan una mejor semántica para la definición de interfaces, algunas características son las siguientes:

- Valores por defecto: Es la capacidad de establecer un valor preestablecido a una propiedad.
- Valores opcionales: Es la capacidad de que el cliente decida si establecer o no un valor para la propiedad.
- Enumeradores: Es una propiedad con una lista de valores predefinidos.
- Repetidores: Es la implementación de arreglos en los *protocol buffers*

- Objetos: En los *protocol buffers* existen los mensajes, un mensaje posee múltiples propiedades, estos mensajes pueden utilizarse como tipo de dato en otra propiedad.

Los *protocol buffers* poseen muchas más características de las mencionadas anteriormente, dichas características se pueden implementar en los lenguajes de programación en donde los *protocol buffers* estén soportados. Los *protocol buffers* se serializan de forma binaria, lo que permite una carga útil más ligera y un procesamiento más rápido, esta es una característica que gRPC aprovecha cuando se invocan los métodos de un servicio.

2.2.2. Ciclo de vida de gRPC

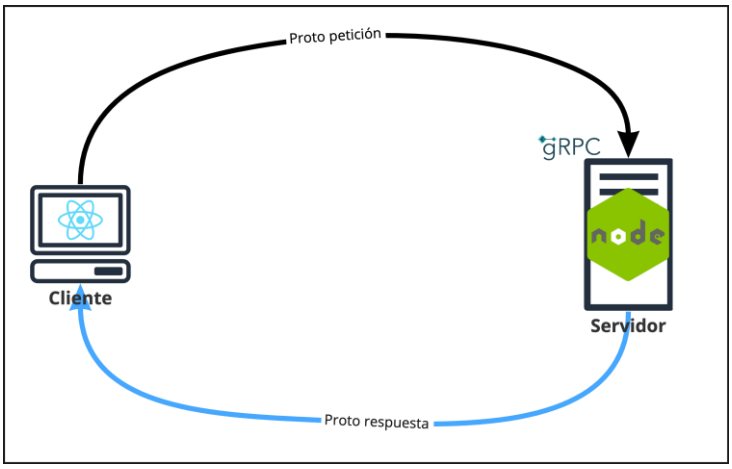
En gPRC existen varios escenarios de comunicación distintos a los convencionales utilizando *Rest*, en un entorno normal se encuentra al cliente enviando una única petición y el servidor respondiendo una única respuesta, *gRPC* puede hacer esto y con el agregado de una característica muy interesante llamada *streaming*. El *streaming* en gRPC permite enviar o recibir una serie de respuestas, esto permite un manejo de memoria eficiente, ya que la información se va cargando por partes, también es eficiente en tiempo, ya que al recibir la información por partes el procesamiento es más rápido.

Un caso de uso muy común se da para los videos, ya que estos se cargan conforme los datos llegan al cliente, permitiendo un procesamiento continuo, ligero y eficiente para el cliente.

2.2.2.1. *RPC* Unario

En gRPC usar un *RPC* unario significa invocar un método de un servicio que tiene un mensaje simple como parámetro y un mensaje simple como respuesta del método. Esto es similar a lo que ocurre al invocar un servicio utilizando *Rest*.

Figura 6. **Flujo para *RPC* Unario**

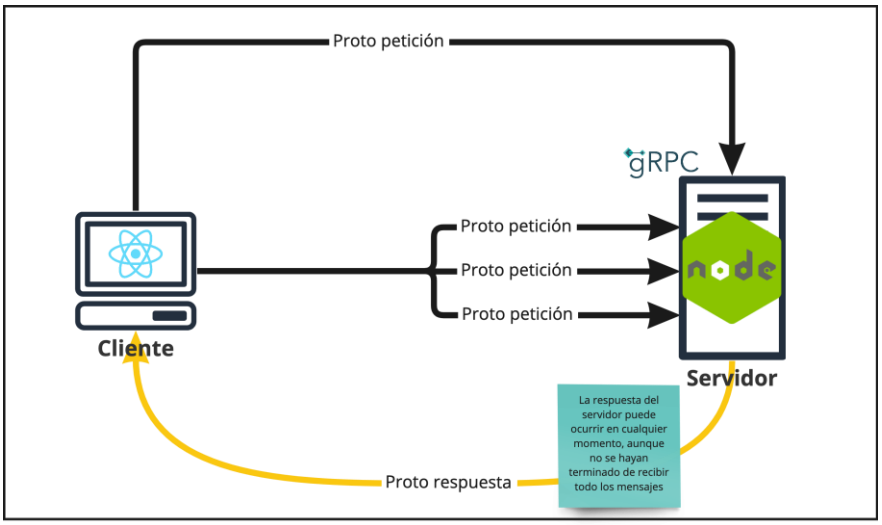


Fuente: elaboración propia, realizado en Miro.

2.2.2.2. Cliente transmitiendo *RPC*

Este flujo existe cuando la configuración de un método en los parámetros de entrada permite que el cliente envié una serie de mensajes hacia el servidor, y el servidor responde con un único mensaje. Esto es posible con la ayuda de la palabra reservada *stream* antes del tipo del mensaje de entrada. Cuando el cliente finaliza la transmisión de mensajes, el servidor puede o no esperar a procesar todos los mensajes para enviar la respuesta.

Figura 7. **Flujo cliente transmitiendo *RPC***

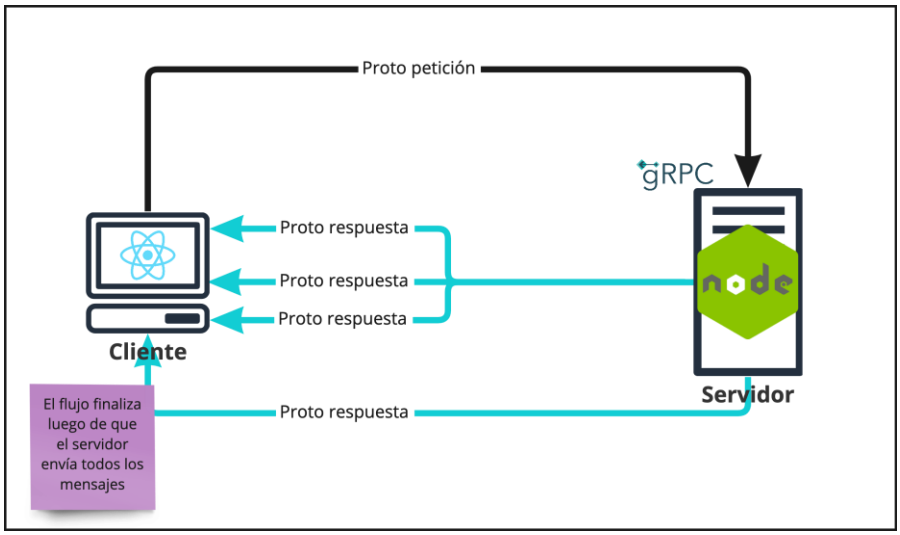


Fuente: elaboración propia, realizado en Miro.

2.2.2.3. Servidor transmitiendo *RPC*

Este flujo se da cuando el cliente envía un mensaje simple y el servidor responde con una serie de mensajes hacia el cliente, el flujo finaliza luego de que el cliente termine el procesamiento de todos los mensajes. Esta configuración se logra colocando la palabra reservada *stream* antes del tipo del mensaje de salida.

Figura 8. **Flujo servidor transmitiendo *RPC***

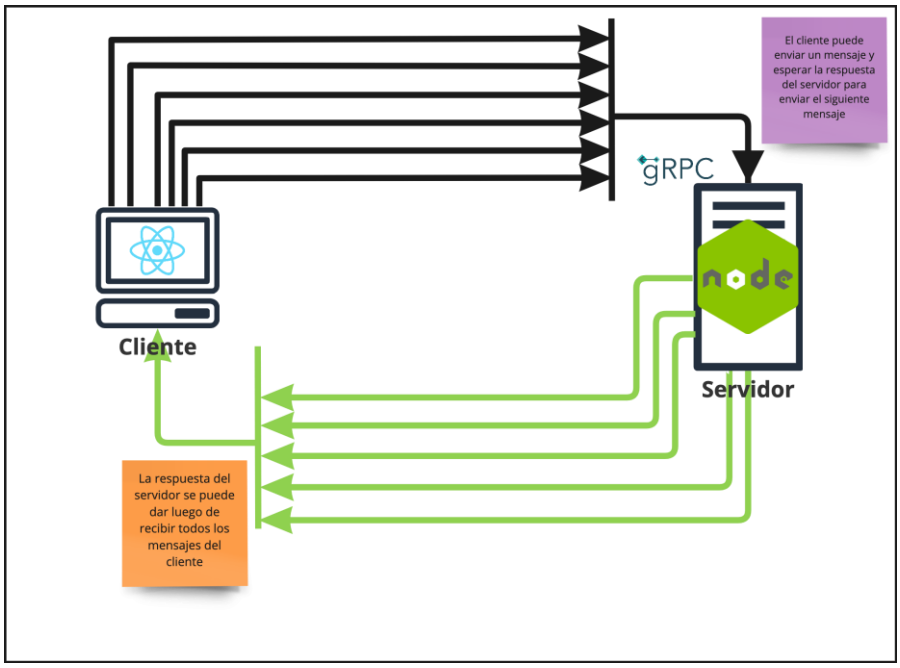


Fuente: elaboración propia, realizado en Miro.

2.2.2.4. Transmisión bidireccional

En este flujo se debe configurar el método para recibir y responder con una serie de mensajes utilizando la palabra reservada *stream* antes de los tipos de los mensajes de entrada y salida. La lógica de envío y recepción puede variar según las necesidades, por ejemplo, cuando el cliente comience a enviar mensajes, el servidor podría esperar para procesar todos los mensajes o podría procesar un mensaje y responder a cada uno de los mensajes del cliente, otro caso de uso se da cuando el cliente envía un nuevo mensaje cada vez que el servidor responde.

Figura 9. **Flujo de transmisión bidireccional**



Fuente: elaboración propia, realizado en Miro.

2.2.3. Conceptos clave de gRPC

Durante el desarrollo de aplicaciones utilizando gRPC es necesario el uso de otros conceptos distintos al ciclo de vida, existen conceptos similares a *Rest*, pero gRPC también posee sus propias implementaciones, entre ellas encontramos:

- Sincrónico y asincrónico
- *Timeouts*
- *Metadata*
- Canales
- Manejo de errores

- Interceptores

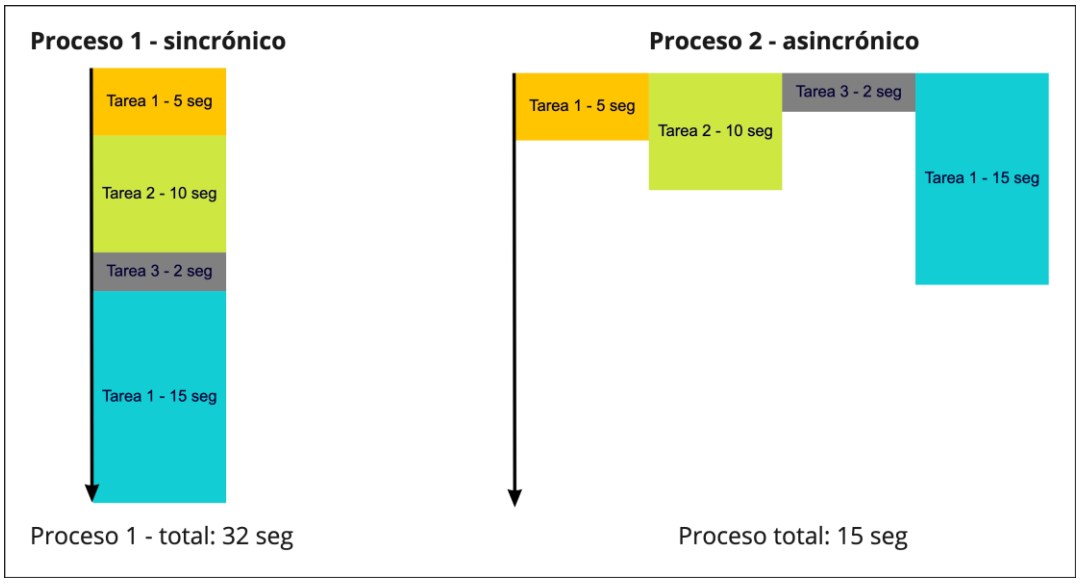
Adicional a esto se pueden hacer otras configuraciones cuando se realiza el despliegue de la aplicación, por ejemplo, agregar balanceadores de carga, servicios de monitoreo y análisis, entre otros.

2.2.3.1. Sincrónico y Asincrónico

En términos generales las operaciones sincrónicas son las más comunes en los lenguajes de programación, las instrucciones se ejecutan de forma secuencial, una a la vez, por lo tanto, para ejecutar la siguiente instrucción la anterior debe finalizar completamente para continuar. Si una *API* fuera desarrollada con un paradigma completamente sincrónico, los clientes serian atendidos de uno en uno.

El paradigma asincrónico permite ejecutar instrucciones en paralelo, por ello la mayoría de los lenguajes programación proporcionan soporte para operaciones asíncronas, durante el desarrollo de *APIs* el paradigma asíncrono es el más utilizado, esto debido a que por defecto las aplicaciones atienden a los clientes en paralelo, evitando de esta manera las colas, caso contrario al que ocurre con el paradigma sincrónico.

Figura 10. **Proceso sincrónico y asincrónico**

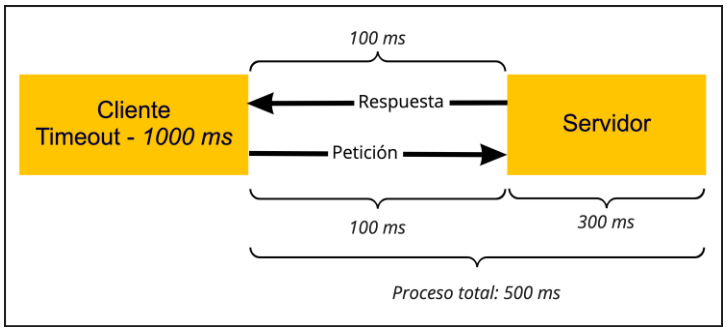


Fuente: elaboración propia, realizado en Miro.

2.2.3.2. Tiempo limite

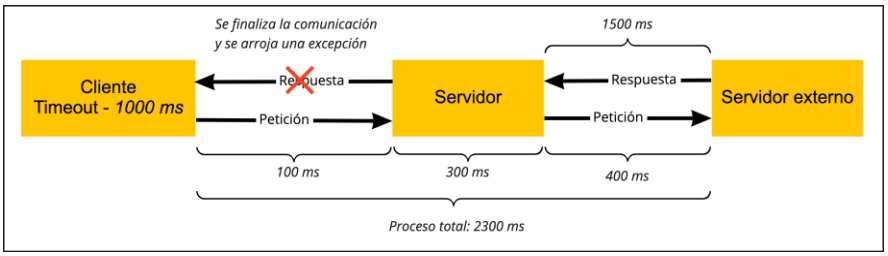
Los *timeouts* o tiempos limites indican cuanto tiempo debe esperar el cliente para obtener una respuesta del servidor, si se alcanza el tiempo máximo de espera, se finaliza la operación y un error es lanzado. En gRPC el servidor puede saber de antemano si la petición alcanzo su tiempo límite o el tiempo restante. El *timeout* en el cliente se aplica a lo largo del proceso invocado, por ejemplo, con un *timeout* de 500 ms, el servidor tiene ese tiempo para proveer una respuesta, esto sin importar si el servidor debe invocar otros métodos de otro servidor.

Figura 11. **Proceso sin *timeout***



Fuente: elaboración propia, realizado en Miro.

Figura 12. **Proceso con *timeout***



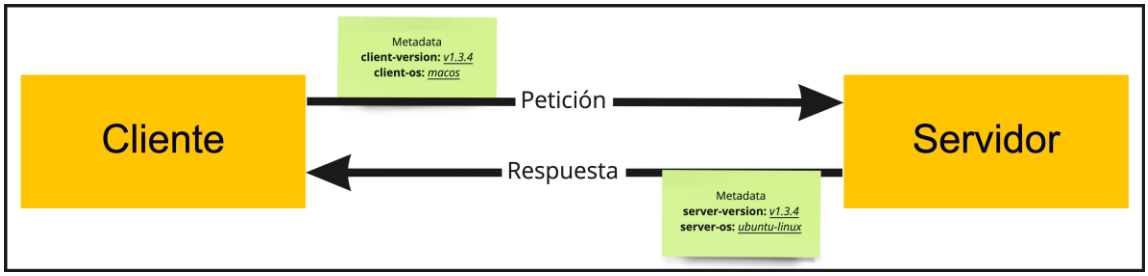
Fuente: elaboración propia, realizado en Miro.

2.2.3.3. Metadatos

En gRPC se comparte información entre el cliente y el servidor por medio de las llamadas *RPC*, esto normalmente se lleva a cabo por medio de los parámetros en los métodos de los servicios. Pero esta no es la única opción para compartir información, en ciertas ocasiones en necesario compartir información que no tiene relación con la lógica del método a invocar, para estos casos existen los metadatos, la *metadata* es información relevante que se envía del cliente al

servidor y viceversa, esta información es comúnmente una lista de pares clave-valor de tipo cadena.

Figura 13. **Envió de *metadata* entre cliente y servidor**



Fuente: elaboración propia, realizado en Miro.

2.2.3.4. Manejo de errores

Durante las llamadas a métodos de un servicio en gRPC existen múltiples escenarios de respuesta, pueden ser exitosos o erróneos. El manejo de errores con gRPC es similar a como funciona en *Rest*, una de las diferencias son los códigos de estados de respuesta, por ejemplo, durante una respuesta exitosa en *Rest* el código de estado es 200 y en gRPC es 0. En gRPC existe un conjunto de errores preestablecidos que deben ser manejados por el cliente y el servidor. Cuando un error ocurre, gRPC retorna un código de error y un mensaje opcional para dar más detalles del error, adicional a esto se pueden incluir otras descripciones del error en la *metadata* de la petición.

Tabla I. **gPRC códigos de error**

Número	Código de error	Descripción
0	*OK*	Petición exitosa
1	*CANCELLED*	Operación cancelada por el cliente
2	*UNKNOWN*	Error desconocido
3	*INVALID_ARGUMENT*	Parámetro invalido
4	*DEADLINE_EXCEEDED*	Se ha alcanzado el tiempo límite antes de completar la operación
5	*NOT_FOUND*	El recurso solicitado no ha sido encontrado
6	*ALREADY_EXISTS*	El recurso ya existe
7	*PERMISSION_DENIED*	Permisos denegados para ejecutar la operación
8	*RESOURCE_EXHAUSTED*	Se han alcanzado los límites de uso del recurso
9	*FAILED_PRECONDITION*	El sistema no posee el estado requerido para ejecutar la operación
10	*ABORTED*	La operación fue detenida
11	*OUT_OF_RANGE*	La operación fue ejecutada más allá del rango permitido
12	*UNIMPLEMENTED*	El método solicitado no ha sido implementado
13	*INTERNAL*	Error interno
14	*UNAVAILABLE*	Servidor apagado
15	*DATA_LOSS*	Se ha perdido información durante la llamada al método
16	*UNAUTHENTICATED*	El cliente no se ha autenticado para invocar al método

Fuente: elaboración propia, realizado en Microsoft Word.

3. COMPUTACIÓN EN LA NUBE

La computación en la nube es una solución que ofrece servicios de tecnologías de la información bajo de demanda por medio de internet, ofrece precios más reducidos, ya que no es necesario mantener centros de datos en las instalaciones. La computación en la nube proporciona agilidad en las nuevas implementaciones, da paso a la innovación y permite escalar de forma simple.

Los servicios que se ofrecen en la nube se pueden clasificar en:

- Infraestructura como servicio (*IaaS*): la infraestructura como servicio permite una alta gestión de componentes básicos como máquinas virtuales, redes, almacenamiento, entre otros.
- Plataforma como servicio (*PaaS*): la plataforma como servicio se refiere a todos aquellos servicios que proporcionan la infraestructura y los elementos necesarios para ejecutar una aplicación, por ejemplo, un servicio para publicar aplicaciones de *NodeJs* donde solo se debe cargar el código fuente.
- *Software* como servicio (*SaaS*): el *software* como servicio es muy común y no está atado a la computación en la nube, este se refiere a el *software* listo para ser utilizado por el usuario final en el que la infraestructura la gestiona quien ofrece el servicio, por ejemplo, servicios de bases de datos.

La computación en la nube se ha extendido en los últimos años, esto ha hecho que las empresas inicien una migración de sistemas en las instalaciones hacia la nube, una de las razones es la reducción de costos y la facilidad de implementación, pero una de las razones más importantes es que las compañías

pueden iniciar un proceso de transformación digital utilizando los servicios de los proveedores de la nube. Existen varias compañías que ofrecen servicios en la nube, entre las más importantes se encuentran:

- *Amazon Web Services* (*AWS*)
- *Google Cloud Computing* (*GCP*)
- Microsoft Azure

3.1. Servicios de Amazon

Amazon Web Services (AWS) es una plataforma en la nube, que ofrece decenas de servicios bajo demanda, es la plataforma con mayor popularidad a nivel mundial. *AWS* trabaja con el modelo de responsabilidad compartida, este modelo se fija entre el cliente y *AWS*, se basa en el hecho de dividir la responsabilidad de ambas partes de la siguiente forma:

- Responsabilidad de *AWS*: se encarga de la gestión de la infraestructura, *software* y seguridad en los centros de datos de las regiones y zonas de disponibilidad.
- Responsabilidad del cliente: se encarga de gestionar los datos de los clientes, accesos a los servicios utilizados, seguridad en los servicios expuestos.

AWS se compone de regiones que a su vez poseen zonas de disponibilidad, una zona de disponibilidad puede tener uno o varios centros de datos. Las regiones se encuentran distribuidas alrededor del mundo y a una gran distancia entre sí, esto permite cumplir con regulaciones requeridas por ciertos países, por ejemplo, que los datos personales de las personas solo puedan estar almacenados en servidores dentro del país, una región, además las zonas de

disponibilidad se encuentran a una distancia estratégicamente pensada para mantener una latencia relativamente baja.

AWS ofrece el modelo de pago por uso, pero también ofrece otras variantes para los clientes que desean una reducción de costos, estos son los modelos de pago que *AWS* ofrece:

- Pruebas gratuitas: existen ciertos servicios que permiten el uso gratuito para que los clientes se puedan familiarizar y aprender del servicio sin ningún costo, las pruebas gratuitas pueden ser por tiempo o por uso dependiendo del servicio, una vez se completa la prueba gratuita se pasa a un modelo de pago por uso.
- Pago por uso: el cliente solamente paga por el uso del servicio y se deja cobrar hasta que se cancela el uso del servicio, el uso puede ser por tiempo desde que se inició el uso del servicio, cantidad de almacenamiento o datos transferidos.
- Pago por reserva: *AWS* ofrece costos más bajos en ciertos servicios al reservar un servicio de 1 a 3 años, este modelo de pago se recomiendo para compañías que ya tienen una visibilidad del uso de sus sistemas con base a la experiencia previa en *AWS*.
- Por un alto uso: *AWS* ofrece descuentos en ciertos servicios al alcanzar cierta cantidad de uso, esto es especialmente útil cuando las cargas de trabajo son grandes.

3.2. Servicio de cómputo de Amazon

Amazon Elastic Cloud Computing (*EC2*) es un servicio de *AWS* catalogado como infraestructura como servicio, ofrece máquinas virtuales más conocidas como instancias, en las que se puede personalizar el almacenamiento, memoria

RAM, *CPUs* y sistema operativo, esto gracias a que ofrecen plantillas para la creación de instancias de *EC2*. Estas máquinas virtuales se pueden crear de forma sencilla y eliminar en cualquier momento, posee una versión de prueba gratuita aplicable al utilizar una plantilla específica de pocos recursos, además también ofrecen los modelos de pago bajo demanda, por reserva, entre otros.

El acceso a estas instancias se lleva a cabo por medio de una consola y una conexión *ssh*, permitiendo de esta manera gestionar y trabajar sobre el sistema operativo instalado. El concepto de instancias se puede emplear a la construcción de una arquitectura de microservicios, en donde una serie de máquinas virtuales pueden ser los anfitriones de un microservicio e intercomunicarse, de esta manera cada instancia puede tener solo los recursos necesarios para la ejecución del microservicio, con esto se evita tener una alta cantidad de recursos de cómputo que no van a ser explotados por la aplicación.

3.3. Servicio de almacenamiento de Amazon

Amazon Simple Storage Service (*S3*) es un servicio de *AWS* el cual permite el almacenamiento de objetos a un bajo costo, el almacenamiento puede ser de archivos de texto, imágenes, videos, entre otros.

Posee varios tipos de almacenamiento, como la capa inteligente que gestiona los objetos por su frecuencia de acceso, el servicio estándar de acceso poco frecuente que funciona para objetos que se acceden de vez en cuando, entre otros, pero el que se utiliza en el presente trabajo es el servicio *S3* Standard, el cual proporciona alta disponibilidad de los objetos y una serie de *APIs* listas para implementar en las cuales se permite obtener o subir objetos de forma rápida.

4. CONCEPTOS ESTADÍSTICOS

Los datos que se recolectan durante las mediciones necesitan ser procesados para poder realizar inferencias a partir de esos datos, los conceptos y métodos que se presentan a continuación son especialmente útiles, ya que al realizar una implementación proporcionan la información necesaria para realizar una comparativa, determinar ventajas y desventajas bajo diferentes escenarios de prueba.

4.1. Tipos de variables

Durante la recolección de datos pueden resultar muchos valores que se pueden clasificar de diferentes maneras, esta clasificación resulta luego de identificar las propiedades y agruparlos en variables, por ejemplo, una recolección de datos de personas puede indicar los siguientes datos: la persona uno posee 1.75 m de altura y color de cabello rojo; la persona dos posee 1.60 m de altura y color de cabello negro. Por lo tanto, se puede extraer en forma de variable la altura y el color de cabello, estas son propiedades de la persona y son variables porque para cada persona los valores pueden cambiar.

4.1.1. Variables cualitativas

Las variables cualitativas expresan cualidades de la muestra o población, estas variables son observables y permiten clasificar, algunos ejemplos de variables cualitativas pueden ser: color de cabello, clase social, nacionalidad de una persona, entre otras.

4.1.2. Variables cuantitativas

Las variables cuantitativas expresan valores medibles numéricos en los cuales se pueden realizar cálculos, estas variables se pueden clasificar en dos tipos:

- Variables cuantitativas discretas: estas variables expresan valores numéricos enteros como la cantidad de veces que una persona visito cierto lugar.
- Variables cuantitativas continuas: estas variables expresan valores numéricos definidos bajo cierto rango y que pueden ser valores decimales, un ejemplo de este tipo de variables es el peso de una persona.

4.2. Parámetros estadísticos

Los parámetros estadísticos proporcionan una vista general de la muestra o población de estudio, se pueden clasificar en medidas de posición central, medidas de posición no central y medidas de dispersión, de esta manera los datos se pueden presentar en tablas o gráficos en los cuales se puede inferir rápidamente cual es el promedio de los datos, la desviación estándar, la moda, entre otros resultados.

4.2.1. Medidas de posición centrales

Las medidas de tendencia central también conocidas como medidas de posición centrales, son una serie de valores derivados que surgen luego de procesar los datos de la muestra o población de tal forma que el resultado representa información sobre el centro de la distribución, cada medida de

tendencia central tiene su propia forma de calcularse y posee una interpretación diferente.

- Media aritmética: representa un valor promedio de la serie de datos perteneciente a la muestra o población.
- Mediana: representa el valor intermedio de la serie de datos, se calcula de dos maneras.
    - Si la cantidad de datos es par se deben tomar los dos valores más al centro de la serie de datos y obtener la media aritmética, el resultado será la mediana.
    - Si la cantidad de datos es impar el valor al centro de la serie de datos será la mediana.
- Moda: representa el valor con mayores apariciones en la serie de datos, comúnmente utilizado para variables cualitativas.

4.2.2. Medidas de posición no centrales

Las medidas de posición no centrales representan los valores divididos en partes iguales en los que se encuentran limites inferiores, medios y superiores de una serie de datos, estos también se conocen como cuantiles y se pueden interpretar como rangos de valores y porcentajes.

- Cuartil: son 3 valores que representan una posición en la serie de datos y que dividen los datos en 4 partes iguales.
- Deciles: son 9 valores que representan una posición en la serie de datos y que dividen los datos en 10 partes iguales.
- Percentiles: son 99 valores que representan una posición en la serie de datos y que dividen los datos en 100 partes iguales.

4.2.3. Medidas de dispersión

Las medidas de dispersión absoluta permiten determinar la variabilidad de los datos tomando de referencia la media aritmética, conocer la variabilidad permite encontrar valores atípicos o visualizar el comportamiento de los datos de forma gráfica.

- Rango: representa la diferencia entre el valor final y el valor inicial del conjunto de datos.
- Desviación media: representa la distancia media que existe entre el conjunto de datos y la media aritmética.
- Varianza: representa la variabilidad de un conjunto de datos con respecto a la media aritmética.
- Desviación estándar: representa la dispersión que existe entre el conjunto de datos y la media aritmética.

4.3. Distribución de frecuencias

Cuando existe una cantidad grande de datos es necesario expresar la información de forma resumida, el resumen de la información se puede realizar de tal manera en que, el resultado pueda describir los datos de cierta manera con base a sus características generales, se puede decir que una distribución de frecuencias es una manera de agrupar y representar datos cualitativos o cuantitativos en el cual se puede observar la frecuencia de cada grupo descrito.

4.3.1. Frecuencia absoluta

La frecuencia absoluta representa la cantidad de apariciones de un valor individual o que se encuentre en un intervalo de datos, la sumatoria de todos los

registros de frecuencia absoluta debe ser equivalente a la cantidad de elementos de la muestra.

Tabla II. **Ejemplo frecuencia absoluta**

Valor	Frecuencia absoluta (f)
5	38
9	59
13	12
Total	109

Fuente: elaboración propia, realizado en Microsoft Word.

4.3.2. Frecuencia relativa

La frecuencia relativa es una relación entre la frecuencia absoluta de cada registro en la tabla y la cantidad de datos de la muestra, la sumatoria de todos los registros de frecuencia relativa debe ser equivalente a 1.

Tabla III. **Ejemplo de frecuencia relativa**

Valor	Frecuencia absoluta (f)	Frecuencia relativa (fr)
5	38	0.35
9	59	0.54
13	12	0.11
Total	109	1

Fuente: elaboración propia, realizado en Microsoft Word.

4.3.3. Frecuencia acumulada

La frecuencia acumulada representa la cantidad de valores acumulados para cada registro y sus predecesores, si se hablara de una frecuencia

acumulada el último registro contiene la cantidad de los datos de la muestra y si fuera una frecuencia acumulada relativa el último registro contiene el valor equivalente a 1.

Tabla IV. **Frecuencia acumulada absoluta y relativa**

Valor	Frecuencia absoluta (f)	Frecuencia acumulada (F)	Frecuencia relativa (fr)	Frecuencia acumulada relativa (Fr)
5	38	38	0.35	0.35
9	59	97	0.54	0.89
13	12	109	0.11	1
Total	109		1	

Fuente: elaboración propia, realizado en Microsoft Word.

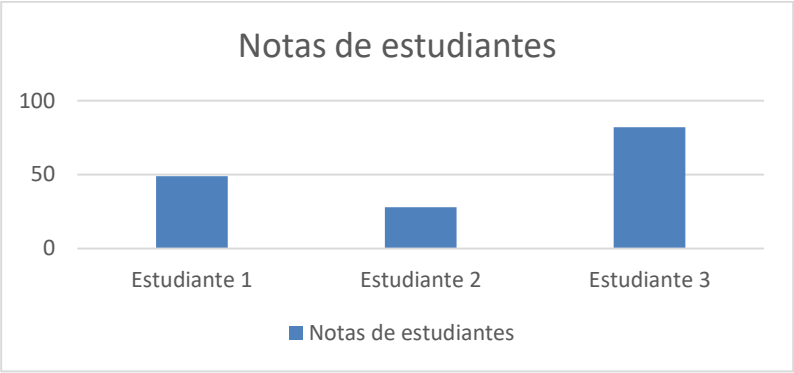
4.4. Representación de datos

El procesamiento de los datos es una etapa en la que se obtiene información relevante de los datos, pero esta información necesita ser presentada de forma visual, ordenada y comprensible para las personas, para realizar esto es necesario utilizar tablas y graficas que correspondan con los tipos de datos procesados.

4.4.1. Gráfica de barras

La gráfica de barras es una representación visual de un conjunto de datos utilizando barras rectangulares de forma horizontal o vertical, este tipo de grafica puede representar tipos de datos cualitativos, además permite identificar como se comparan los datos entre diferentes intervalos por medio de las barras.

Figura 14. **Gráfica de barras**

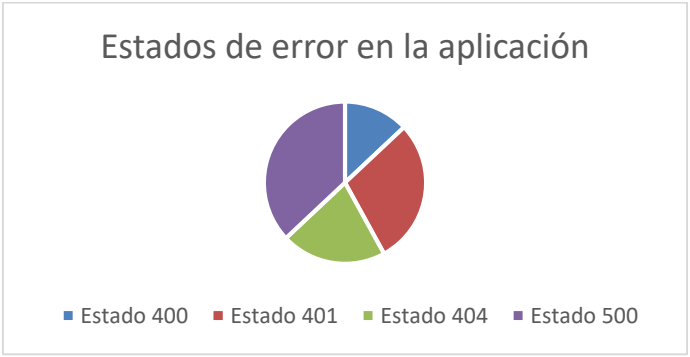


Fuente: elaboración propia, realizado en Microsoft Word.

4.4.2. Gráfico de pie

El gráfico de pie es una representación circular en la que cada uno de sus segmentos indica una porción del porcentaje de los datos representados, el circulo completo representa el 100% de los datos, en este tipo de gráfica se pueden utilizar datos cualitativos.

Figura 15. **Ejemplo gráfico de pie**

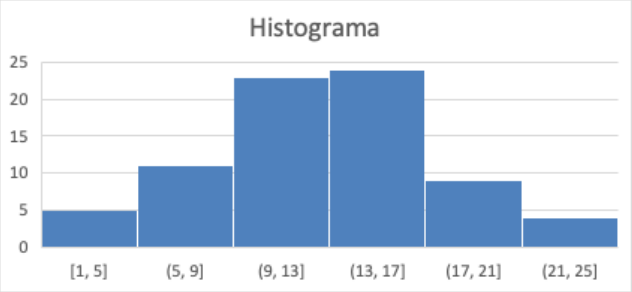


Fuente: elaboración propia, realizado en Microsoft Word.

4.4.3. Histograma

Una forma de representar datos cualitativos es el histograma, el cual permite mostrar de forma visual las distribuciones de frecuencia, similar a la gráfica de barras, está formada por una serie de rectángulos en el cual sus extremos indican el límite inferior y superior respectivamente, cada uno de los rectángulos al centro representa un valor de la distribución.

Figura 16. **Ejemplo histograma**



Fuente: elaboración propia, realizado en Microsoft Word.

4.5. Hipótesis

La hipótesis es una suposición que puede ser verdadera o falsa, las hipótesis son útiles en los métodos de científicos, ya que es necesario experimentar para verificar y comprobar o refutar la premisa.

En estadística existen dos hipótesis que se contradicen entre sí, la hipótesis nula denotada por Hn, es la pretensión de que inicialmente se supone cierta. La hipótesis alternativa denotada por Ha, es la aseveración

> contradictoria a Hn. La hipótesis nula será rechazada en favor de la hipótesis alternativa sólo si la evidencia muestral sugiere que Hn es falsa. Si la muestra no contradice fuertemente a Hn, se continuará creyendo en la verdad de la hipótesis nula. Las dos posibles conclusiones derivadas de un análisis de prueba de hipótesis son entonces rechazar Hn o no rechazar Hn. (Devore, 2008, p. 285)

Para verificar las hipótesis se utilizan las pruebas de hipótesis, por ejemplo, una compañía de tecnología desea actualizar el método actual de transmisión de archivos que utiliza *API Rest* por la alternativa gRPC, inicialmente con *API Rest* el tiempo promedio que tarda la transmisión de un archivo multimedia de 500MB es de 3 minutos, por lo que la hipótesis nula es Hn >= 3 minutos y la hipótesis alternativa Ha < 3, siendo en caso de rechazar la hipótesis nula, la compañía procedería a utilizar gRPC.

> En cuanto al procedimiento de prueba de hipótesis, Devore (2008) especifica:
>
> - Un estadístico de prueba, un conjunto de datos recolectados sobre los cuales se tomará la decisión.
> - Una región de rechazo, el conjunto de todos los valores de prueba para rechazar la hipótesis nula.
>
> La hipótesis nula se rechaza si el estadístico de prueba está dentro de la región de rechazo. (p. 287)

4.5.1. Pruebas de normalidad

La prueba de normalidad intenta determinar la hipótesis de que cierta variable posea una distribución normal, tomando en cuenta sus dos hipótesis contradictorias:

- Hipótesis nula (Hn): la variable posee una distribución normal.
- Hipótesis alternativa (Ha): la variable no posee una distribución normal.

4.5.2. Pruebas de independencia

Las pruebas chi cuadrado de independencia se aplican para variables con tipos de datos cualitativos y permiten verificar si las variables son independientes.

- Hipótesis nula (Hn): las variables tienen independencia entre sí.
- Hipótesis alternativa (Ha): las variables son dependientes entre sí.

4.6. Análisis de varianza

> El análisis de la varianza se refiere en general a un conjunto de situaciones experimentales y procedimientos estadísticos para el análisis de respuestas cuantitativas de unidades experimentales e implica el análisis de datos muestreados de más de dos muestras numéricas o de datos de experimentos en los cuales se utilizaron más de dos tratamientos. La característica que diferencia los tratamientos o poblaciones una de otra se llama factor en estudio y los distintos tratamientos o poblaciones se conocen como niveles del factor. (Devore, 2008, p. 369)

Las hipótesis asociadas al análisis de varianza corresponden a:

- Hipótesis nula (Hn): las medias de las poblaciones son iguales.
- Hipótesis alternativa (Ha): al menos una media de las poblaciones es diferente.

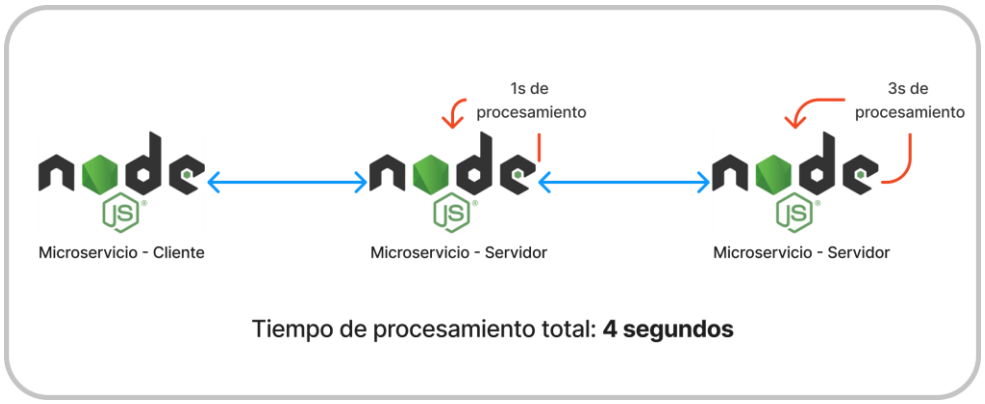
5. DEFINICIÓN DE MEDIDAS Y METRICAS DE COMPARACIÓN

Las medidas y métricas son un medio cuantificable que se obtiene a partir de un muestreo con el objetivo de realizar inferencias a partir de los resultados. A continuación, se definen una serie de medidas y métricas con el objetivo de observar el comportamiento de gRPC frente a una *API Rest* realizando las mismas tareas de procesamiento y bajo las mismas condiciones de *hardware* para determinar las ventajas y casos de uso efectivos para gRPC.

5.1. Tiempo de procesamiento de la respuesta

El tiempo de procesamiento de la respuesta se calcula en el servidor, este tiempo abarca desde el primer momento en que la petición llega hasta que se finaliza el procesamiento y se envía al cliente, si el servidor se comunica con otro microservicio servidor este también tendrá otro tiempo de procesamiento, esto hace que los tiempos de procesamiento de respuesta se acumulen hasta llegar al cliente. Dependiendo de cómo se gestione la recepción de la petición en los servidores el tiempo de procesamiento puede variar mucho o poco entre *API Rest* y gRPC.

Figura 17. **Explicación del tiempo de procesamiento de la respuesta**

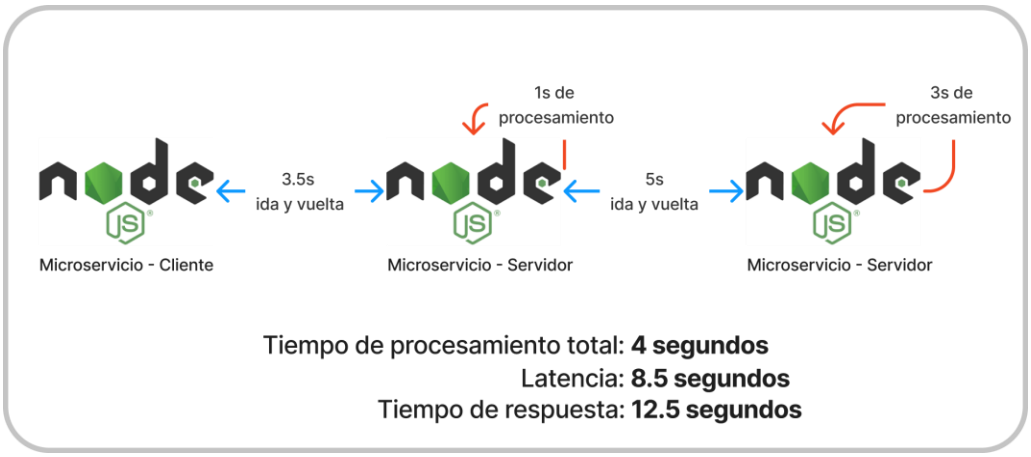


Fuente: elaboración propia, realizado en FigJam.

5.2. Tiempo de respuesta del servidor

El tiempo de respuesta del servidor se calcula tomando en cuenta el tiempo en que la petición viaja del cliente al servidor, el tiempo de procesamiento de la respuesta y el tiempo en que la respuesta viaja del servidor al cliente. Para gRPC y *API Rest* los tiempos en los que la petición viaja del cliente al servidor y viceversa se pueden ver afectados por el formato en el que la carga útil se envía y por tipo de comunicación utilizado.

Figura 18. **Explicación del tiempo de respuesta**



Fuente: elaboración propia, realizado en FigJam.

5.3. Uso de *CPU*

Esta medición indica cuanto poder de procesamiento está siendo utilizado por el sistema y se expresa en forma de porcentaje. El *CPU* posee varios modos y en cada uno de estos modos se recolectan datos del *CPU*, para esta prueba se hace uso de todos los modos a excepción del modo inactivo. Para facilitar la recolección se utilizan las herramientas *Prometheus* y Grafana, se hace una medición por cada iteración durante las pruebas, este tipo de medición es general para observar el comportamiento frente a diferentes cargas de trabajo utilizando gRPC y *API Rest*.

5.4. Uso de memoria

Esta medición indica el porcentaje de memoria *RAM* utilizada. Para facilitar la recolección se utilizan las herramientas *Prometheus* y Grafana se hace una medición por cada iteración durante las pruebas, este tipo de medición es general

para observar el comportamiento frente a diferentes cargas de trabajo utilizando gRPC y *API Rest*.

5.5. Rendimiento (*Throughput*)

La métrica de rendimiento representa la cantidad de trabajo realizado en cierta unidad de tiempo, el cálculo se obtiene a partir de dividir la cantidad de peticiones exitosas con la sumatoria del tiempo de respuesta de cada petición. El objetivo de esta métrica es identificar el rendimiento de gRPC y *API Rest* para las mismas cargas de trabajo a evaluar.

5.6. Peticiones exitosas y fallidas

Una petición es catalogada de exitosa cuando la respuesta posee un código de respuesta para *API Rest* en el rango de 200 a 299 y para gRPC *OK*. Una petición es catalogada de fallida cuando la respuesta posee un código de respuesta para *API Rest* en el rango de 400 a 599 y para gRPC cualquier código distinto a *OK*. Esta medición representa cuantas peticiones de forma concurrente soporta la aplicación en cada una de las pruebas realizadas.

6. DISEÑO DE ARQUITECTURA Y PRUEBAS A IMPLEMENTAR

El diseño y planteamiento de la arquitectura permite visualizar los componentes de una aplicación a bajo nivel, el diagrama de arquitectura proporciona una idea general o detallada de cómo estará construida y distribuida la aplicación a nivel de componentes, asimismo un diseño correcto ayuda a identificar componentes nuevos o eliminar elementos que no son necesarios.

Con los elementos de arquitectura seleccionados se puede realizar un análisis de integración entre los componentes, esto con el objetivo de que el sistema funcione correctamente, además permite evaluar el precio de cada componente a utilizar y determinar si se encuentra en el rango del presupuesto.

6.1. Diseño de arquitectura

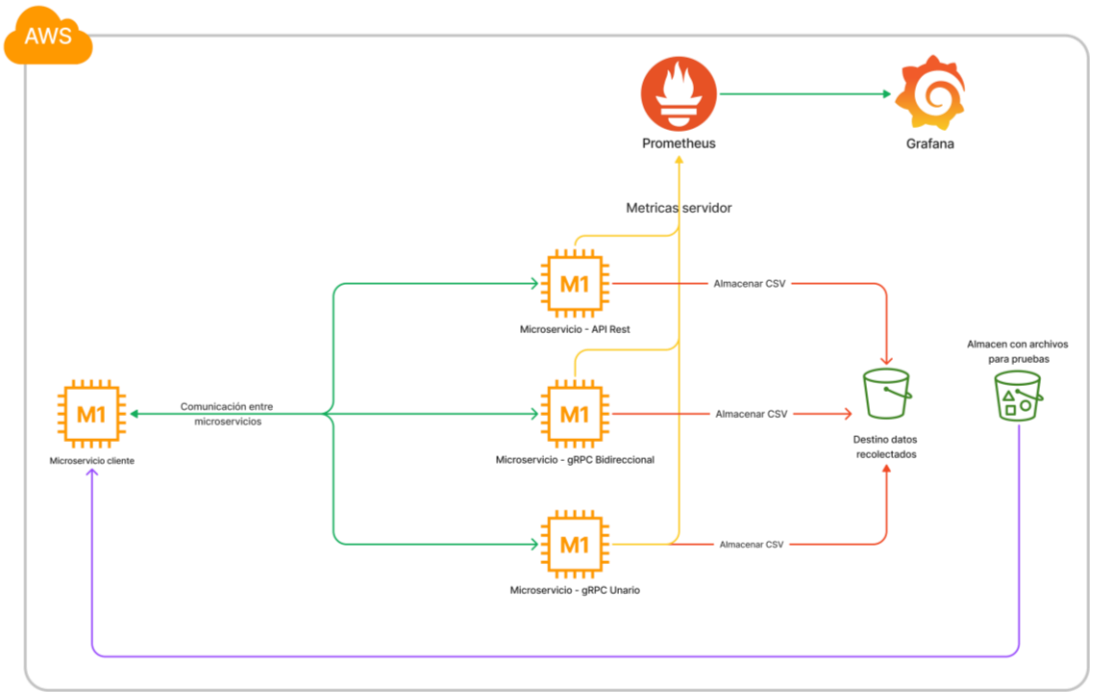
El diseño de arquitectura está pensado desde un punto de vista enfocado en pruebas de estrés y monitoreo, es por ello que se utilizan herramientas como *Prometheus* y Grafana, para la captura de datos automática y creación de tableros para visualizar las gráficas.

A continuación, se presentan las herramientas, servicios y *software* para el desarrollo de las pruebas:

- Instancias *EC2*: este servicio de cómputo de *AWS* es la base y parte importante de la infraestructura, en él se publican las aplicaciones sobre las cuales se harán las pruebas comparativas entre *Rest* y gRPC.

- Almacenamiento en *S3*: este servicio de almacenamiento de *AWS* permite almacenar los resultados de las pruebas realizadas y permite obtener ciertos archivos que se utilizan en las pruebas.
- *Prometheus*: se utiliza para monitorear el servidor y obtener métricas de uso de *CPU* y memoria durante cada iteración de las pruebas.
- Grafana: se utiliza para visualizar las métricas que se extraen de *Prometheus*, además se pueden realizar otro tipo de consultas y generar sus propias gráficas.
- Docker: se utiliza para facilitar la creación y despliegue de microservicios.
- Lenguaje de programación: se utiliza *JavaScript* con *NodeJs* por su flexibilidad en la implementación de tecnologías para la comunicación entre microservicios.
- Pruebas: se utiliza *Postman* para la ejecución de pruebas y de esta forma no intervenir manualmente en el servidor.
- Comunicación: para la comunicación entre microservicios se utiliza *Rest* y gRPC unario y bidireccional.

Figura 19. **Diagrama de arquitectura**



Fuente: elaboración propia, realizado en FigJam.

6.1.1. Detalle de componentes de arquitectura

Como se observa en la figura 17, se están utilizando los siguientes componentes de esta manera:

- Instancia 1: posee la aplicación del cliente que hará las peticiones a los otros microservicios servidores.
- Instancia 2: posee el servidor *Api Rest*, este atiende las peticiones del cliente.
- Instancia 3: posee el servidor gRPC bidireccional, este atiende las peticiones del cliente.

- Instancia 4: posee el servidor gRPC unario, este atiende las peticiones del cliente.
- Instancia 5: posee el servicio de *Prometheus* y Grafana para el procesamiento de las métricas obtenidas en cada microservicio servidor.
- Almacenamiento: el almacenamiento se maneja en *S3* y contendrá los archivos de resultados de las pruebas, además contendrá archivos precargados para realizar pruebas.

Cada instancia cliente y servidor posee una instancia de tipo M1.xlarge, esta es una instancia de uso general con altos recursos de memoria y *CPU* permitiendo evitar reinicios de la instancia frente a una alta carga de trabajo, el objetivo es mantener las instancias de prueba con las mayores similitudes posibles, permitiendo una comparación justa y adecuada.

La instancia que posee el servicio de *Prometheus* y Grafana es de tipo T2.micro, la razón del porque esta instancia posee menos recursos de *hardware* es porque, es un componente aislado que, aunque posea pocos recursos no afectara el rendimiento de las otras instancias y el procesamiento de la información no es necesario que sea realizado inmediatamente, ya que será extraído posteriormente al finalizar todas las pruebas.

Cada servidor posee un agente de *Prometheus* para la extracción de métricas de recursos del servidor, este agende se ejecuta como un proceso más del servidor y consume pocos recursos, por lo que este no afectará el funcionamiento de la aplicación durante las pruebas, asimismo este componente solo envía la información recolectada al servicio de *Prometheus*, de esta forma se evita un alto acoplamiento entre ambos elementos que podría llevar a un cuello de botella en el procesamiento. Por otra parte, todas las instancias cliente y servidor están desarrolladas con el entorno de ejecución *NodeJs* y el lenguaje

Javascript. Cada componente mencionado anteriormente estará publicado en *AWS*, tomando en cuenta el precio bajo demanda de cada servicio utilizado.

Tabla V. ***Hardware* para las instancias a utilizar**

Microservicio	Tipo de servidor	Sistema operativo	Disco (GB)	*Ram (Gib)*	***vCpu***	Precio por hora (USD)
Cliente	*M1.xlarge*	Amazon Linux 2	30	15	4	0.3790
gRPC unario	*M1.xlarge*	Amazon Linux 2	30	15	4	0.3790
gRPC bidireccional	*M1.xlarge*	Amazon Linux 2	30	15	4	0.3790
API ***Rest*** (express)	*M1.xlarge*	Amazon Linux 2	30	1	4	0.3790
Prometheus + Grafana	*T2.micro*	Amazon Linux 2	8	1	1	0.0138

Fuente: elaboración propia, realizado en Microsoft Word.

6.2. Definición de las pruebas a realizar

A continuación, se presentan las pruebas a implementar para determinar las ventajas y casos de uso entre cada tecnología, el primer paso previo a iniciar las pruebas es generar un ambiente estable y constante para cada servidor, por ejemplo, mantener el mismo sistema operativo, recursos de *hardware* (*RAM*, *CPU*). Cuando el ambiente esté construido se debe asegurar que el sistema operativo posea todas sus actualizaciones y demás, de esta forma se evita que se ejecuten procesos externos de actualización mientras se corren las pruebas.

Se definen 5 pruebas entre las cuales se pretende medir el comportamiento concurrente, secuencial, frente a archivos de tamaño medio-alto. Para cada escenario se plantea realizar la misma prueba con las tecnologías

propuestas, en cada escenario se realizan 10 iteraciones, esto con el objetivo de obtener datos confiables y reducir el factor de incertidumbre que pudiera ocurrir en los servidores por alguna ejecución de un proceso externo, entre otros. Al finalizar cada prueba se genera un archivo con el resultado, dicho resultado será almacenado en la nube utilizando el servicio *S3* de *AWS*, además durante todas las iteraciones de las pruebas se mide de forma constante el uso de memoria y *CPU*, esto con el fin de mostrar el comportamiento de los recursos de *hardware* de forma gráfica.

Los resultados principales de las pruebas serán el tiempo de respuesta y tiempo de procesamiento en cada iteración por cada valor de carga y tecnología. El tiempo de respuesta se obtiene calculando la diferencia entre el tiempo en que se recibe la respuesta del servidor y el tiempo en que se envía la petición, por otra parte, el tiempo de procesamiento se calcula realizando la diferencia entre el tiempo en el que el servidor responde y el tiempo en que la petición llego al controlador del servidor.

Otros resultados de la prueba son el uso de *CPU* y memoria obtenidos por cada iteración, este resultado se obtiene con las herramientas Grafana y *Prometheus*.

6.2.1. Prueba 1 – concurrente

El primer escenario de la primera prueba está enfocado en la concurrencia y exigir la capacidad de cómputo y memoria de los servidores. El punto de partida de la prueba inicia en el cliente, se envía una carga útil pequeña con ciertos parámetros configurables, cuando la petición llegue al servidor se extrae la carga útil y sobre los parámetros enviados se realizan ciertas operaciones que en su mayoría son recursivas para forzar el procesamiento del servidor, y de esta

manera mientras la carga de peticiones aumenta la capacidad de respuesta debería disminuir, asimismo de la forma en que cada tecnología está diseñada, la compresión y el lenguaje de definición pueden afectar el tiempo de inicio en el procesamiento de la petición, el tiempo de finalización y envió de la respuesta.

Tabla VI. **Casos escenario 1 - prueba 1 - concurrente**

Cantidad iteraciones	Carga (Peticiones concurrentes)	Tecnología
10	*1*	*Rest* gRPC *unario* gRPC *streaming*
10	*10*	
10	*50*	
10	*100*	
10	*500*	
10	*1,000*	
10	*5,000*	

Fuente: elaboración propia, realizado en Microsoft Word.

6.2.2. Prueba 1 – secuencial

El segundo escenario de la primera prueba es una alteración del primer escenario, cambiando el enfoque de concurrencia por una ejecución secuencial, esto quiere decir que, las peticiones se envían de uno en uno y se tiene que finalizar la petición actual previo a enviar la siguiente. De esta manera los recursos no serán exigidos como en la prueba anterior, lo que evita un cuello de botella y tiene como objetivo obtener tiempos de respuesta inferiores por petición, pero con la salvedad de que la prueba tomara más tiempo en completarse.

Tabla VII. **Casos escenario 2 – prueba 1 – secuencial**

Cantidad iteraciones	Carga (Peticiones concurrentes)	Tecnología
10	*1*	*Rest* gRPC *unario* gRPC *streaming*
10	*10*	
10	*50*	
10	*100*	
10	*500*	
10	*1,000*	

Fuente: elaboración propia, realizado en Microsoft Word.

6.2.3. Prueba 2 – concurrente

El primer escenario de la segunda prueba está enfocado en la concurrencia y en él envió de una carga útil compleja y pesada, a diferencia de la primera prueba, esta no pretende forzar la capacidad de cómputo y memoria, sino que afectar el tiempo de procesamiento, ya que al modificar la carga útil de las peticiones el inicio del procesamiento entre las tecnologías puede variar significativamente.

Tabla VIII. **Casos escenario 1 – prueba 2 – concurrente**

Cantidad iteraciones	Carga (Peticiones concurrentes)	Tecnología
10	*1*	*Rest* gRPC *unario* gRPC *streaming*
10	*10*	
10	*50*	
10	*100*	
10	*500*	
10	*1,000*	
10	*5,000*	

Fuente: elaboración propia, realizado en Microsoft Word.

6.2.4. Prueba 2 – secuencial

El segundo escenario de la segunda prueba es similar al escenario 1, pero al igual que en la primera prueba se reduce la carga en los recursos al enviar de uno en uno las peticiones. Para esta prueba cada petición tardara menos tiempo en completarse a comparación a la prueba concurrente, pero en la vista general de la prueba el tiempo en completar todas las iteraciones es mayor, por el simple hecho de ser secuencial, por ese motivo en los escenarios secuenciales solo se llega a un máximo de 1,000 peticiones de carga.

Tabla IX. **Casos escenario 2 - prueba 2 - secuencial**

Cantidad iteraciones	Carga (Peticiones concurrentes)	Tecnología
10	*1*	*Rest* gRPC *unario* gRPC *streaming*
10	*10*	
10	*50*	
10	*100*	
10	*500*	
10	*1,000*	

Fuente: elaboración propia, realizado en Microsoft Word.

6.2.5. Prueba 3 – manejo de archivos

La tercera prueba está enfocada en archivos, el objetivo principal es verificar el comportamiento entre tecnologías cuando se varían los tamaños de los archivos, la forma de enviarlos y el cómo se procesan en el servidor. Esta prueba se lleva a cabo de forma secuencial, se modifica el factor de la carga por un factor de tamaño de archivo y se modifica el uso de las tecnologías para enviar los archivos. Para *Rest*, gRPC unario y *streaming* se envía el archivo completo desde el cliente al servidor, pero también se agrega una implementación para enviar archivos por partes utilizando gRPC *streaming* y de esta manera verificar

como cambia el tiempo de respuesta y procesamiento cuando se envía por partes y completo.

Tabla X. **Casos prueba 3**

Cantidad iteraciones	Tamaño de archivo (MB)	Extensión	Tecnología
5	*20*	*mp4*	*Rest* gRPC unario gRPC *streaming* gRPC *streaming full*
5	*40*	*mp4*	
5	*50*	*zip*	
5	*50*	*txt*	
5	*55*	*zip*	
5	*70*	*mp4*	
5	*130*	*mp4*	
5	*512*	*avi*	

Fuente: elaboración propia, realizado en Microsoft Word.

7. RESULTADOS

A continuación, se presentan los resultados obtenidos en cada escenario de prueba, durante las pruebas se mantuvieron las condiciones lo más similar posible para no alterar los resultados, así mismo se hicieron varias iteraciones por cualquier alteración que llegara a ocurrir.

7.1. Resultados prueba 1 – concurrente

La primera prueba en el primer escenario tuvo un total de 210 iteraciones y 199,830.00 peticiones entre todas las iteraciones de esta prueba. A continuación, se presentan los resultados de tiempo de procesamiento y respuesta los cuales fueron procesados realizando pruebas de normalidad, análisis de varianza con los factores de carga y tecnología.

7.1.1. Normalidad

Para verificar la normalidad de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, el resultado es normal

Ha = p < 0.05, el resultado no es normal

Tabla XI. **Resultados prueba de normalidad tiempo de respuesta medio y tiempo de procesamiento medio vs carga, prueba 1 – escenario 1**

	Tiempo de respuesta		Tiempo de procesamiento	
Carga	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
1	4.2563e-10	No	0.0001	No
10	0.0130	No	0.5371	Si
50	0.0032	No	0.8058	Si
100	0.0646	Si	0.2508	Si
500	0.4771	Si	0.0328	No
1,000	0.0022	No	0.0016	No
5,000	0.0013	No	5.7619e-07	No

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XII. **Resultados prueba de normalidad tiempo de respuesta y tiempo de procesamiento vs tecnología, prueba 1 – escenario 1**

	Tiempo de respuesta		Tiempo de procesamiento	
Tecnología	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
Rest	1.1434e-12	No	1.9615e-12	No
gRPC Unario	2.4882e-12	No	5.8808e-09	No
gRPC Streaming	5.1372e-12	No	5.2771e-09	No

Fuente: elaboración propia, realizado en Microsoft Word.

7.1.2. Homocedasticidad

Para verificar la igualdad de las varianzas (homocedasticidad) de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, Las varianzas son iguales para todos los tipos de carga/tecnología

Ha = p < 0.05, Las varianzas no son iguales para todos los tipos de carga/tecnología.

Tabla XIII. **Resultados de la prueba de homocedasticidad para la carga y tecnología contra el tiempo de procesamiento y tiempo de respuesta, prueba 1 – escenario 1**

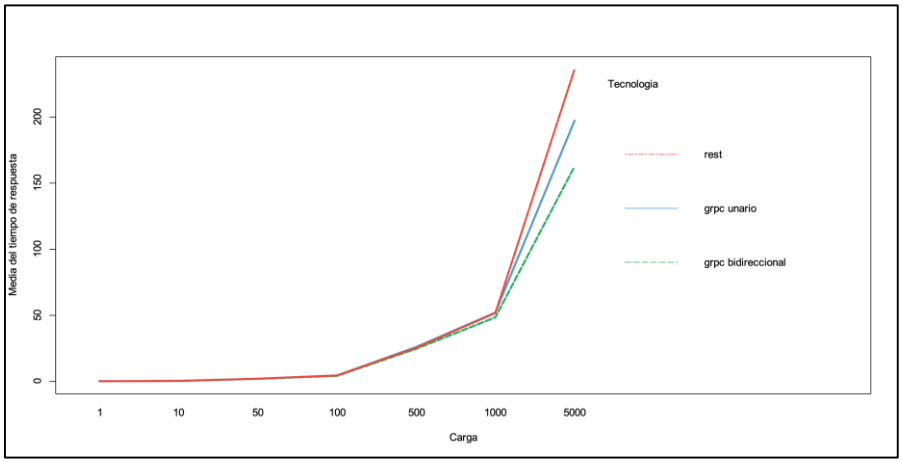
	Tiempo de procesamiento		**Tiempo de respuesta**	
	Carga	Tecnología	Carga	Tecnología
K-Cuadrado	1684.7	153.32	1412.1	9.8174
Valor p	p < 2.2e-16	p < 2.2e-16	p < 2.2e-16	p = 0.007382

Fuente: elaboración propia, realizado en Microsoft Word.

7.1.3. Comparación de medias

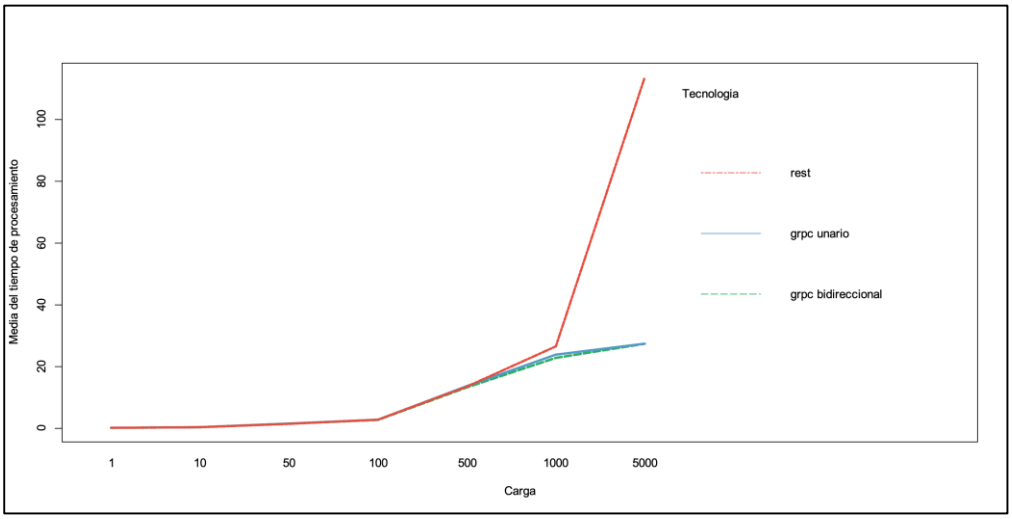
Se presenta una comparación entre el tiempo de respuesta y procesamiento contra la carga, esta comparativa es de utilidad porque muestra como la carga afecta directamente los resultados del tiempo de respuesta y procesamiento.

Figura 20. **Media del tiempo de respuesta vs carga**



Fuente: elaboración propia, realizado en DataSpell.

Figura 21. **Media del tiempo de procesamiento vs carga, prueba 1 – escenario 1**



Fuente: elaboración propia, realizado en DataSpell.

7.1.4. Grupos y significancia entre comparaciones

La representación de los grupos y significancia permite identificar la similitud de los resultados, de esta forma si en dado caso ciertos valores se encuentran en grupos iguales se puede determinar si existe o no una diferencia significativa.

Tabla XIV. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la carga, prueba 1 – escenario 1**

Carga	Tiempo de respuesta (s)	Grupo	Tiempo de procesamiento (s)	Grupo
1	0.2014	D	0.1799	D
10	0.4405	D	0.3905	D
50	2.0765	D	1.5245	D
100	4.4764	D	2.7932	CD
500	25.4810	C	13.5107	BC
1,000	50.8336	B	24.4206	B
5,000	198.6380	A	56.1911	A

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XV. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la tecnología, prueba 1 – escenario 1**

Tecnología	Tiempo de respuesta	Grupo	Tiempo de procesamiento	Grupo
Rest	45.8244	A	22.6718	A
gRPC Unario	40.4098	B	9.9976	B
gRPC *Streaming*	34.6862	C	9.7636	B

Fuente: elaboración propia, realizado en Microsoft Word.

7.1.5. Recursos y rendimiento general

A continuación, se presenta el uso de recursos a lo largo de toda la prueba, este es un punto de partida para determinar si algún factor afecta en el consumo de recursos. Además, se presenta el rendimiento en general de la tecnología y como la carga puede afectar en la disponibilidad de los servicios.

Figura 22. **Uso de Memoria durante la ejecución de la prueba 1 y escenario 1**

Fuente: elaboración propia, realizado en Grafana.

Figura 23. **Uso de *CPU* durante la ejecución de la prueba 1 y escenario 1**

Fuente: elaboración propia, realizado en Grafana.

Tabla XVI. **Rendimiento de tecnología entre cargas, prueba 1 – escenario 1**

Peticiones/ segundo	Carga						
Tecnología	1	10	50	100	500	1,000	5,000
Rest	5.2052	2.2843	0.4787	0.2182	0.0391	0.0193	0.0043
gRPC Unario	4.9647	2.2312	0.4746	0.2189	0.0383	0.0192	0.0051
gRPC ***Streaming***	4.7484	2.2963	0.4917	0.2338	0.0404	0.0206	0.0062

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XVII. **Peticiones fallidas y exitosas, prueba 1 – escenario 1**

Tecnología	Peticiones exitosas	Peticiones fallidas
Rest	64189	2421
gRPC Unario	66610	0
gRPC *Streaming*	66610	0

Fuente: elaboración propia, realizado en Microsoft Word.

7.2. Resultados prueba 1 – secuencial

La primera prueba en el segundo escenario tuvo un total de 180 iteraciones. A continuación, se presentan los resultados de tiempo de procesamiento y respuesta los cuales fueron procesados realizando pruebas de normalidad, análisis de varianza con los factores de carga y tecnología.

7.2.1. Normalidad

Para verificar la normalidad de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, el resultado es normal

Ha = p < 0.05, el resultado no es normal

Tabla XVIII. **Resultados prueba de normalidad tiempo de respuesta medio y tiempo de procesamiento medio vs carga, prueba 1 – escenario 2**

	Tiempo de respuesta		Tiempo de procesamiento	
Carga	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
1	1.5101e-09	No	5.9907e-09	No
10	0.4543	Si	0.5061	Si
50	0.0625	Si	0.0435	No
100	0.9680	Si	0.5899	Si
500	0.6323	Si	0.2941	Si
1,000	0.6263	Si	0.0214	No

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XIX. **Resultados prueba de normalidad tiempo de respuesta y tiempo de procesamiento vs tecnología, prueba 1 – escenario 2**

	Tiempo de respuesta		Tiempo de procesamiento	
Tecnología	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
Rest	1.6908e-14	No	3.0420e-15	No
gRPC Unario	1.2846e-16	No	6.4942e-06	No
gRPC ***Streaming***	4.0271e-14	No	2.6573e-11	No

Fuente: elaboración propia, realizado en Microsoft Word.

7.2.2. Homocedasticidad

Para verificar la igualdad de las varianzas (homocedasticidad) de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, Las varianzas son iguales para todos los tipos de carga/tecnología

Ha = p < 0.05, Las varianzas no son iguales para todos los tipos de carga/tecnología.

Tabla XX. **Resultados de la prueba de homocedasticidad para la carga y tecnología contra el tiempo de procesamiento y tiempo de respuesta, prueba 1 – escenario 2**

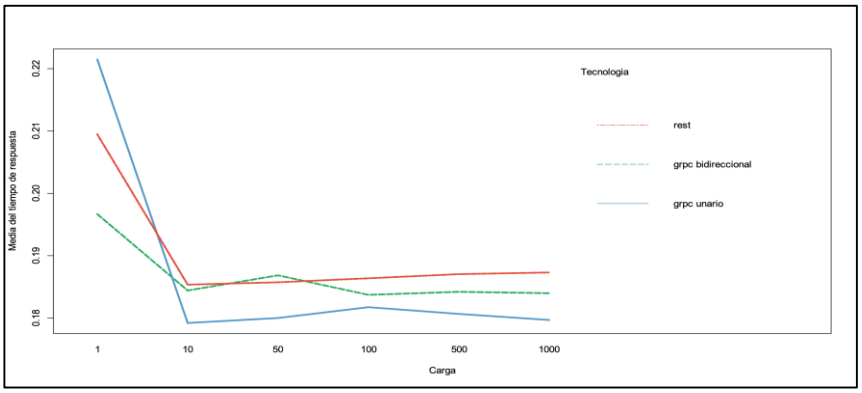
	Tiempo de procesamiento		**Tiempo de respuesta**	
	Carga	Tecnología	Carga	Tecnología
K-Cuadrado	218.1	90.033	489.33	104.57
Valor p	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16

Fuente: elaboración propia, realizado en Microsoft Word.

7.2.3. Comparación de medias

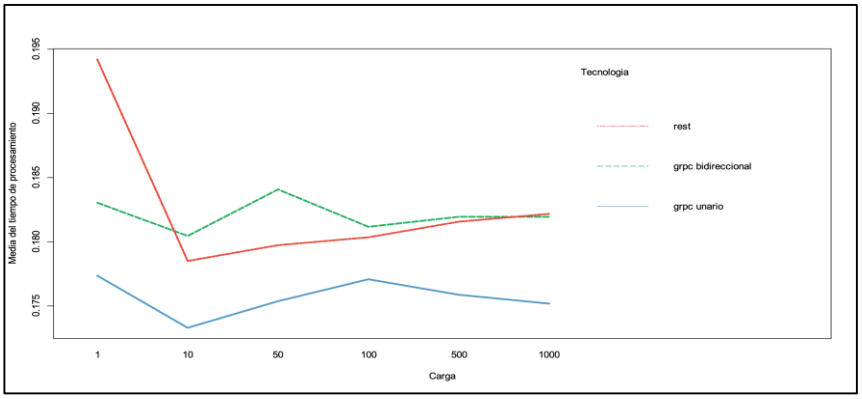
Se presenta una comparación entre el tiempo de respuesta y procesamiento contra la carga, esta comparativa es de utilidad porque muestra como la carga afecta directamente los resultados del tiempo de respuesta y procesamiento.

Figura 24. **Media del tiempo de respuesta vs carga, prueba 1 – escenario 2**



Fuente: elaboración propia, realizado en DataSpell.

Figura 25. **Media del tiempo procesamiento vs carga, prueba 1 – escenario 2**



Fuente: elaboración propia, realizado en DataSpell.

7.2.4. Grupos y significancia entre comparaciones

La representación de los grupos y significancia permite identificar la similitud de los resultados, de esta forma si en dado caso ciertos valores se encuentran en grupos iguales se puede determinar si existe o no una diferencia significativa.

Tabla XXI. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la carga, prueba 1 – escenario 2**

Carga	Tiempo de respuesta (s)	Grupo	Tiempo de procesamiento (s)	Grupo
1	0.2092	A	0.1849	A
10	0.1830	B	0.1774	B
50	0.1842	B	0.1797	AB
100	0.1839	B	0.1795	AB
500	0.1840	B	0.1798	AB
1,000	0.1837	B	0.1798	AB

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXII. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la tecnología, prueba 1 – escenario 2**

Tecnología	Tiempo de respuesta (s)	Grupo	Tiempo de procesamiento (s)	Grupo
Rest	0.1902	A	0.1828	A
gRPC Unario	0.1871	A	0.1757	B
gRPC *Streaming*	0.1866	A	0.1821	A

Fuente: elaboración propia, realizado en Microsoft Word.

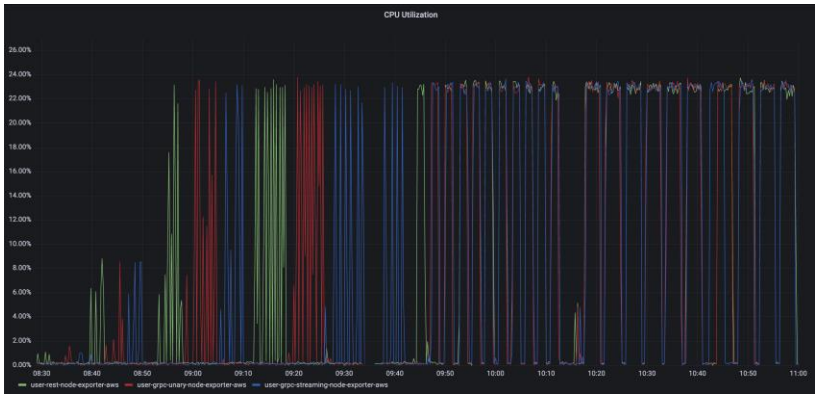
7.2.5. Recursos y rendimiento general

A continuación, se presenta el uso de recursos a lo largo de toda la prueba, este es un punto de partida para determinar si algún factor afecta en el consumo de recursos. Además, se presenta el rendimiento en general de la tecnología y como la carga puede afectar en la disponibilidad de los servicios.

Figura 26. **Uso de Memoria, prueba 1 – escenario 2**

Fuente: elaboración propia, realizado en Grafana.

Figura 27. **Uso de *CPU*, prueba 1 – escenario 2**



Fuente: elaboración propia, realizado en Grafana.

Tabla XXIII. **Rendimiento de tecnología entre cargas, prueba 1 – escenario 2**

Peticiones/segundo	Carga					
Tecnología	1	10	50	100	500	1,000
Rest	4.7738	5.3953	5.3841	5.3657	5.3467	5.339
gRPC Unario	4.515	5.5805	5.5557	5.5026	5.5354	5.5654
gRPC ***Streaming***	5.0845	5.4227	5.3520	5.4432	5.4288	5.4354

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXIV. **Peticiones fallidas y exitosas, prueba 1 – escenario 2**

Tecnología	**Peticiones exitosas**	**Peticiones fallidas**
Rest	16610	0
gRPC Unario	16610	0
gRPC *Streaming*	16610	0

Fuente: elaboración propia, realizado en Microsoft Word.

7.3. Resultados prueba 2 – concurrente

La segunda prueba en el primer escenario tuvo un total de 210 iteraciones y 199,830.00 peticiones entre todas las iteraciones de esta prueba. A continuación, se presentan los resultados de tiempo de procesamiento y respuesta los cuales fueron procesados al realizar pruebas de normalidad, análisis de varianza con los factores de carga y tecnología.

7.3.1. Normalidad

Para verificar la normalidad de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, el resultado es normal

Ha = p < 0.05, el resultado no es normal

Tabla XXV. **Resultados prueba de normalidad tiempo de respuesta medio y tiempo de procesamiento medio vs carga, prueba 2 – escenario 1**

	Tiempo de respuesta		Tiempo de procesamiento	
Carga	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
1	3.9726e-09	No	2.4869e-07	No
10	3.4447e-07	No	0.1668	Si
50	0.0049	No	0.0001	No
100	0.0039	No	1.3777e-05	No
500	0.0008	No	2.8450e-07	No
1,000	0.0002	No	1.4921e-07	No
5,000	2.2987e-05	No	5.1911e-07	No

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXVI. **Resultados prueba de normalidad tiempo de respuesta y tiempo de procesamiento vs tecnología, prueba 2 – escenario 1**

	Tiempo de respuesta		Tiempo de procesamiento	
Tecnología	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
Rest	2.7014e-14	No	2.5823e-14	No
gRPC Unario	1.5921e-13	No	0.0008	No
gRPC ***Streaming***	3.2183e-13	No	2.4067e-06	No

Fuente: elaboración propia, realizado en Microsoft Word.

7.3.2. Homocedasticidad

Para verificar la igualdad de las varianzas (homocedasticidad) de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, Las varianzas son iguales para todos los tipos de carga/tecnología

Ha = p < 0.05, Las varianzas no son iguales para todos los tipos de carga/tecnología.

Tabla XXVII. **Resultados de la prueba de homocedasticidad para la carga y tecnología contra el tiempo de procesamiento y tiempo de respuesta, prueba 2 – escenario 1**

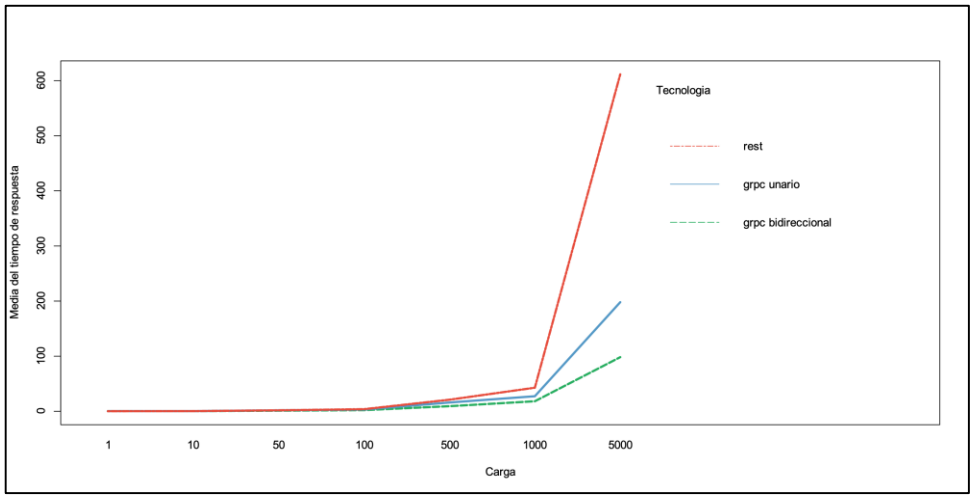
	Tiempo de procesamiento		**Tiempo de respuesta**	
	Carga	Tecnología	Carga	Tecnología
K-Cuadrado	1673.1	1787.9	1527.7	211.63
Valor p	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16

Fuente: elaboración propia, realizado en Microsoft Word.

7.3.3. Comparación de medias

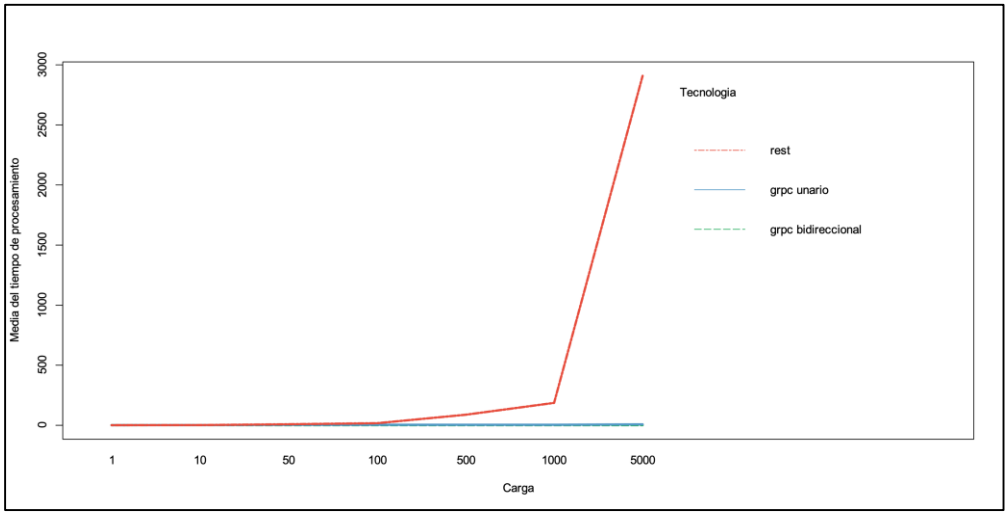
Se presenta una comparación entre el tiempo de respuesta y procesamiento contra la carga, esta comparativa es de utilidad porque muestra como la carga afecta directamente los resultados del tiempo de respuesta y procesamiento.

Figura 28. **Media del tiempo de respuesta vs carga, prueba 2 – escenario 1**



Fuente: elaboración propia, realizado en DataSpell.

Figura 29. **Media del tiempo de procesamiento vs carga, prueba 2 – escenario 1**



Fuente: elaboración propia, realizado en DataSpell.

7.3.4. Grupos y significancia entre comparaciones

La representación de los grupos y significancia permite identificar la similitud de los resultados, de esta forma si en dado caso ciertos valores se encuentran en grupos iguales se puede determinar si existe o no una diferencia significativa.

Tabla XXVIII. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la carga, prueba 2 – escenario 1**

Carga	Tiempo de respuesta (s)	Grupo	Tiempo de procesamiento (s)	Grupo
1	0.2224	B	1.0912	B
10	0.3689	B	1.7722	B
50	1.5276	B	4.7808	B
100	3.0783	B	7.9195	B
500	15.4092	B	31.3405	B
1,000	29.4529	B	64.3113	B
5,000	302.5484	A	972.5511	A

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXIX. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la tecnología, prueba 2 – escenario 1**

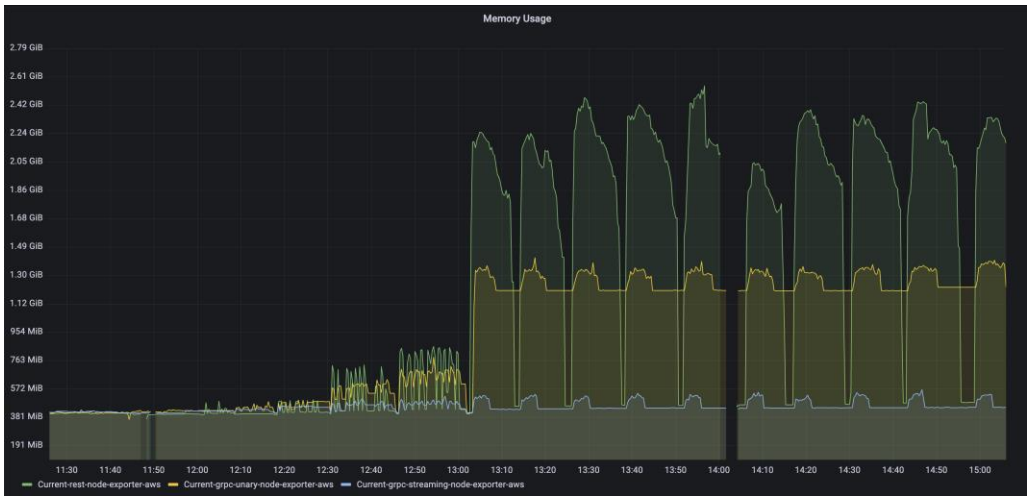
Tecnología	Tiempo de respuesta (s)	Grupo	Tiempo de procesamiento (s)	Grupo
Rest	97.3571	A	458.8686	A
gRPC Unario	35.2822	B	4.1243	B
gRPC *Streaming*	18.4782	B	1.4786	B

Fuente: elaboración propia, realizado en Microsoft Word.

7.3.5. Recursos y rendimiento general

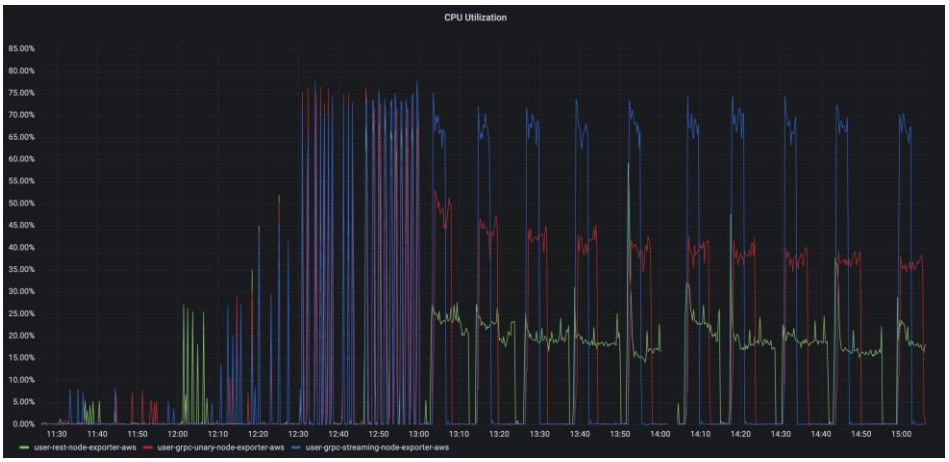
A continuación, se presenta el uso de recursos a lo largo de toda la prueba, este es un punto de partida para determinar si algún factor afecta en el consumo de recursos. Además, se presenta el rendimiento en general de la tecnología y como la carga puede afectar en la disponibilidad de los servicios.

Figura 30. **Uso de memoria, prueba 2 – escenario 1**



Fuente: elaboración propia, realizado en Grafana.

Figura 31. **Uso de *CPU*, prueba 2 – escenario 1**



Fuente: elaboración propia, realizado en Grafana.

Tabla XXX. **Rendimiento de tecnología entre cargas, prueba 2 – escenario 1**

Peticiones/segundo	Carga						
Tecnología	1	10	50	100	500	1,000	5,000
Rest	6.8489	2.9205	0.5561	0.2565	0.0475	0.0234	0.0016
gRPC Unario	4.0282	2.3093	0.5653	0.3075	0.0628	0.0366	0.0051
gRPC ***Streaming***	3.6658	3.0194	0.9846	0.4798	0.1083	0.0550	0.0102

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXXI. **Peticiones fallidas y exitosas, prueba 2 – escenario 1**

Tecnología	**Peticiones exitosas**	**Peticiones fallidas**
Rest	66245	365
gRPC Unario	66610	0
gRPC *Streaming*	66610	0

Fuente: elaboración propia, realizado en Microsoft Word.

7.4. Resultados prueba 2 – secuencial

La segunda prueba en el segundo escenario tuvo un total de 180 iteraciones. A continuación, se presentan los resultados de tiempo de procesamiento y respuesta los cuales fueron procesados realizando pruebas de normalidad, análisis de varianza con los factores de carga y tecnología.

7.4.1. Normalidad

Para verificar la normalidad de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, el resultado es normal

Ha = p < 0.05, el resultado no es normal

Tabla XXXII. **Resultados prueba de normalidad tiempo de respuesta medio y tiempo de procesamiento medio vs carga, prueba 2 – escenario 2**

	Tiempo de respuesta		Tiempo de procesamiento	
Carga	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
1	2.1938e-10	No	1.3171e-09	No
10	4.6992e-08	No	1.2855e-05	No
50	0.0344	No	0.0139	No
100	0.0193	No	0.0916	Si
500	0.0195	No	0.0899	Si
1,000	0.0427	No	0.0322	No

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXXIII.**Resultados prueba de normalidad tiempo de respuesta y tiempo de procesamiento vs tecnología, prueba 2 – escenario 2**

	Tiempo de respuesta		Tiempo de procesamiento	
Tecnología	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
Rest	7.9961e-13	No	0.0297	No
gRPC Unario	1.2252e-16	No	5.2298e-15	No
gRPC ***Streaming***	8.4438e-17	No	1.7019e-05	No

Fuente: elaboración propia, realizado en Microsoft Word.

7.4.2. Homocedasticidad

Para verificar la igualdad de las varianzas (homocedasticidad) de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, Las varianzas son iguales para todos los tipos de carga/tecnología

Ha = p < 0.05, Las varianzas no son iguales para todos los tipos de carga/tecnología.

Tabla XXXIV. **Resultados de la prueba de homocedasticidad para la carga y tecnología contra el tiempo de procesamiento y tiempo de respuesta, prueba 2 – escenario 2**

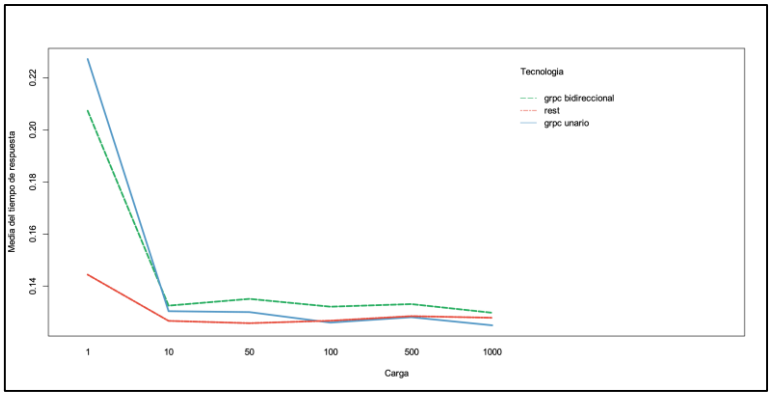
	Tiempo de procesamiento		Tiempo de respuesta	
	Carga	Tecnología	Carga	Tecnología
K-Cuadrado	239.36	180.84	716.11	185.06
Valor p	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16

Fuente: elaboración propia, realizado en Microsoft Word.

7.4.3. Comparación de medias

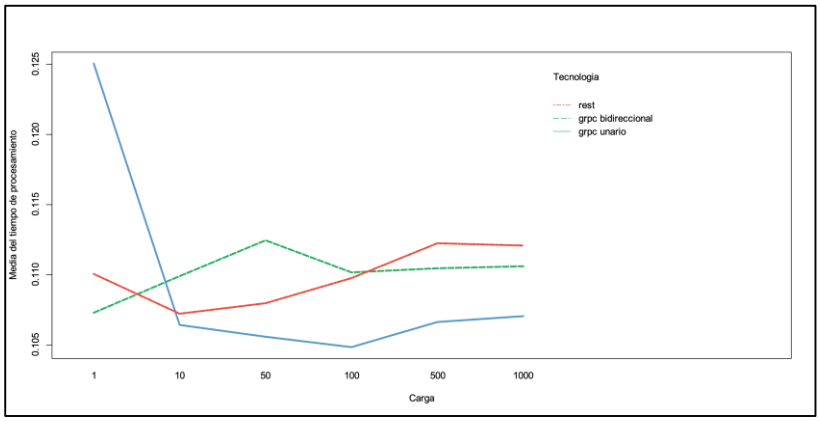
Se presenta una comparación entre el tiempo de respuesta y procesamiento contra la carga, esta comparativa es de utilidad porque muestra como la carga afecta directamente los resultados del tiempo de respuesta y procesamiento.

Figura 32. **Media del tiempo de respuesta vs carga, prueba 2 – escenario 2**



Fuente: elaboración propia, realizado en DataSpell.

Figura 33. **Media tiempo de procesamiento vs carga, prueba 2 – escenario 2**



Fuente: elaboración propia, realizado en DataSpell.

7.4.4. Grupos y significancia entre comparaciones

La representación de los grupos y significancia permite identificar la similitud de los resultados, de esta forma si en dado caso ciertos valores se encuentran en grupos iguales se puede determinar si existe o no una diferencia significativa.

Tabla XXXV. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la carga, prueba 2 – escenario 2**

Carga	Tiempo de respuesta	Grupo	Tiempo de procesamiento	Grupo
1	0.1930	A	0.1141	A
10	0.1298	B	0.1079	A
50	0.1303	B	0.1087	A
100	0.1282	B	0.1083	A
500	0.1299	B	0.1098	A
1,000	0.1275	B	0.1099	A

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXXVI. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio vs la tecnología, prueba 2 – escenario 2**

Tecnología	Tiempo de respuesta	Grupo	Tiempo de procesamiento	Grupo
Rest	0.1299	A	0.1099	A
gRPC Unario	0.1444	A	0.1093	A
gRPC *Streaming*	0.1450	A	0.1102	A

Fuente: elaboración propia, realizado en Microsoft Word.

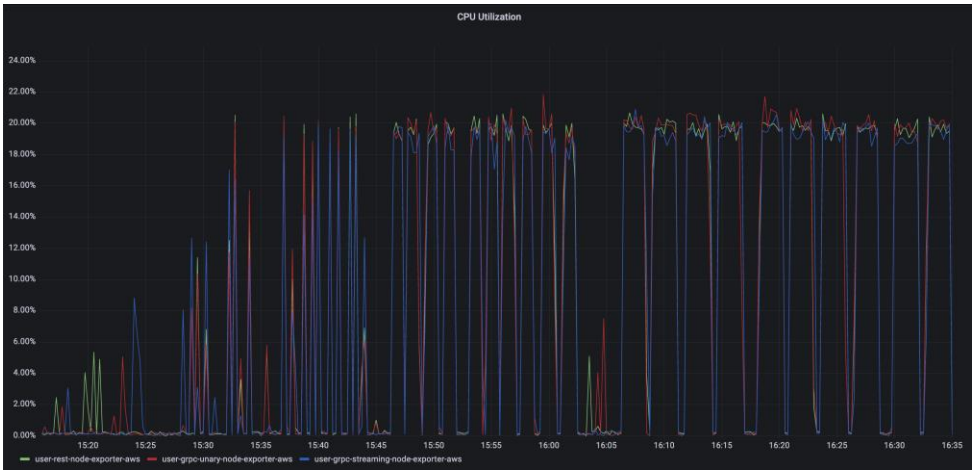
7.4.5. Recursos y rendimiento general

A continuación, se presenta el uso de recursos a lo largo de toda la prueba, este es un punto de partida para determinar si algún factor afecta en el consumo de recursos. Además, se presenta el rendimiento en general de la tecnología y como la carga puede afectar en la disponibilidad de los servicios.

Figura 34. **Uso de Memoria, prueba 2 – escenario 2**

Fuente: elaboración propia, realizado en Grafana.

Figura 35. **Uso de *CPU*, prueba 2 – escenario 2**



Fuente: elaboración propia, realizado en Grafana.

Tabla XXXVII. **Rendimiento de tecnología entre cargas, prueba 2 – escenario 2**

Peticiones/segundo	Carga					
Tecnología	1	10	50	100	500	1,000
Rest	6.9254	7.8976	7.9532	7.8935	7.7857	7.8237
gRPC Unario	4.3998	7.6717	7.6925	7.9394	7.8100	8.0062
gRPC ***Streaming***	4.8221	7.5484	7.4037	7.5718	7.5145	7.7081

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XXXVIII. **Peticiones fallidas y exitosas, prueba 2 – escenario 2**

Tecnología	**Peticiones exitosas**	**Peticiones fallidas**
Rest	16610	0
gRPC Unario	16610	0
gRPC *Streaming*	16610	0

Fuente: elaboración propia, realizado en Microsoft Word.

7.5. Resultados prueba 3

La tercera prueba tuvo un total de 160 iteraciones. A continuación, se presentan los resultados de tiempo de procesamiento y respuesta los cuales fueron procesados realizando pruebas de normalidad, análisis de varianza con los factores de tamaño de archivo y tecnología.

7.5.1. Normalidad

Para verificar la normalidad de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, el resultado es normal

Ha = p < 0.05, el resultado no es normal

Tabla XXXIX. **Resultados prueba de normalidad tiempo de respuesta medio y tiempo de procesamiento medio vs tamaño de archivo, prueba 3**

	Tiempo de respuesta		Tiempo de procesamiento	
Tamaño de archivo (MB)	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
20	3.7033e-05	No	1.7804e-06	No
40	0.0008	No	1.5160e-06	No
50.zip	4.1175e-05	No	1.7441e-06	No
50.txt	2.3554e-05	No	1.8610e-06	No
55	7.4529e-05	No	1.7751e-06	No
70	1.2826e-05	No	1.6107e-06	No
130	7.4547e-06	No	1.1486e-06	No
512	3.1827e-05	No	9.0454e-07	No

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XL. **Resultados prueba de normalidad tiempo de respuesta y tiempo de procesamiento vs tecnología, prueba 3**

	Tiempo de respuesta		Tiempo de procesamiento	
Tecnología	**Valor p**	**¿Es normal?**	**Valor p**	**¿Es normal?**
Rest	2.7610e-09	No	1.3183e-09	No
gRPC Unario	1.8368e-08	No	1.4717e-09	No
gRPC ***Streaming***	1.5623e-09	No	1.5727e-09	No
gRPC ***Streaming Full***	2.5018e-10	No	5.8213e-10	No

Fuente: elaboración propia, realizado en Microsoft Word.

7.5.2. Homocedasticidad

Para verificar la igualdad de las varianzas (homocedasticidad) de los resultados se establece una prueba de hipótesis en la que se define lo siguiente:

Ho = p >= 0.05, Las varianzas son iguales para todos los tipos de tamaño de archivo/tecnología

Ha = p < 0.05, Las varianzas no son iguales para todos los tipos de tamaño de archivo /tecnología.

Tabla XLI. **Resultados de la prueba de homocedasticidad para el tamaño de archivo y la tecnología contra el tiempo de procesamiento y tiempo de respuesta, prueba 3**

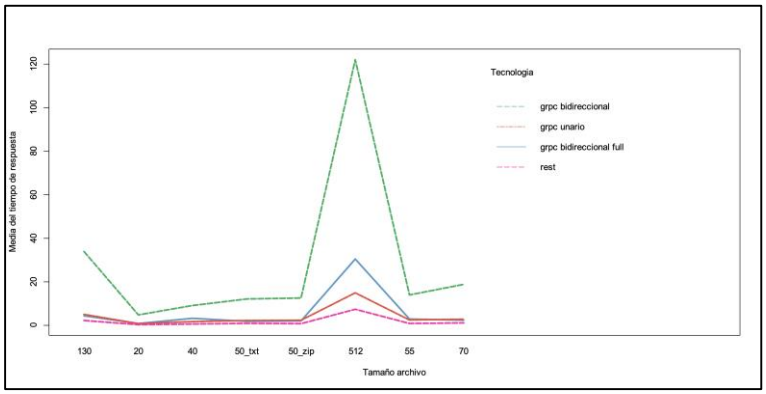
	Tiempo de procesamiento		**Tiempo de respuesta**	
	Tamaño	Tecnología	Tamaño	Tecnología
K-Cuadrado	295.98	1424.6	294.7	282.11
Valor p	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16	Valor p < 2.2e-16

Fuente: elaboración propia, realizado en Microsoft Word.

7.5.3. Comparación de medias

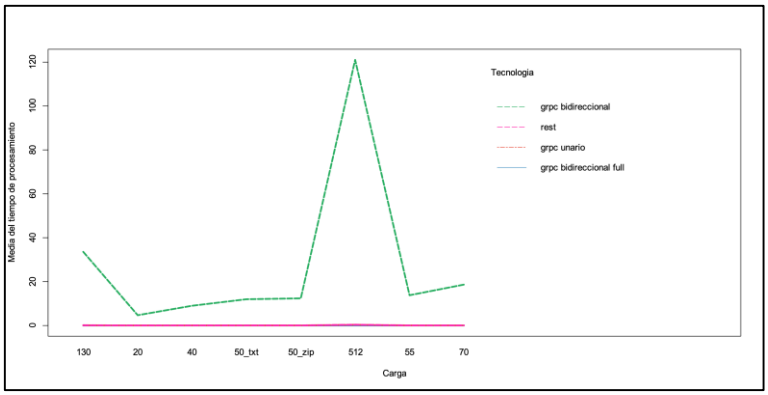
Se presenta una comparación entre el tiempo de respuesta y procesamiento contra la carga, esta comparativa es de utilidad porque muestra como la carga afecta directamente los resultados del tiempo de respuesta y procesamiento.

Figura 36. **Media del tiempo de respuesta vs tamaño de archivo, prueba 3**



Fuente: elaboración propia, realizado en DataSpell.

Figura 37. **Media del tiempo de procesamiento vs tamaño de archivo, prueba 3**



Fuente: elaboración propia, realizado en DataSpell.

7.5.4. Grupos y significancia entre comparaciones

La representación de los grupos y significancia permite identificar la similitud de los resultados, de esta forma si en dado caso ciertos valores se encuentran en grupos iguales se puede determinar si existe o no una diferencia significativa.

Tabla XLII. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio contra el tamaño de archivo, prueba 3**

Tamaño de archivo (MB)	Tiempo de respuesta	Grupo	Tiempo de procesamiento	Grupo
20	1.6963	B	1.1816	B
40	3.6795	B	2.2643	B
50.zip	4.4264	B	3.1111	B
50.txt	4.2692	B	3.0109	B
55	5.0531	B	3.4683	B
70	6.2937	B	4.6833	B
130	11.3991	B	8.4405	B
512	43.7026	A	30.4266	A

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XLIII. **Grupos y significancia entre comparaciones para el tiempo de respuesta y procesamiento medio contra la tecnología, prueba 3**

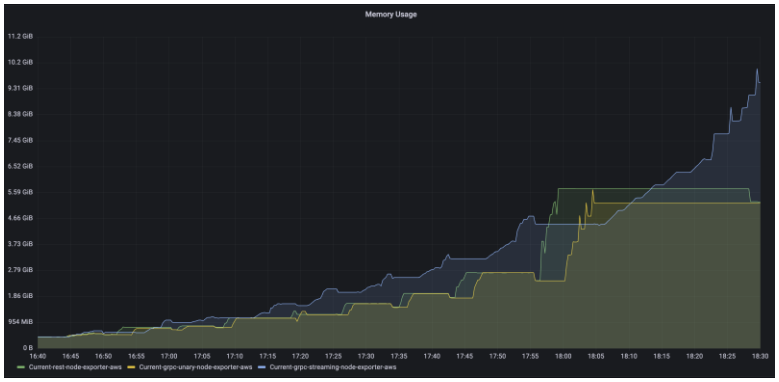
Tecnología	Tiempo de respuesta	Grupo	Tiempo de procesamiento	Grupo
Rest	1.7807	B	0.0813	B
gRPC Unario	4.0469	B	0.0804	B
gRPC *Streaming*	28.4210	A	28.1295	A
gRPC *Streaming Full*	6.0113	B	0.0021	B

Fuente: elaboración propia, realizado en Microsoft Word.

7.5.5. Recursos y rendimiento general

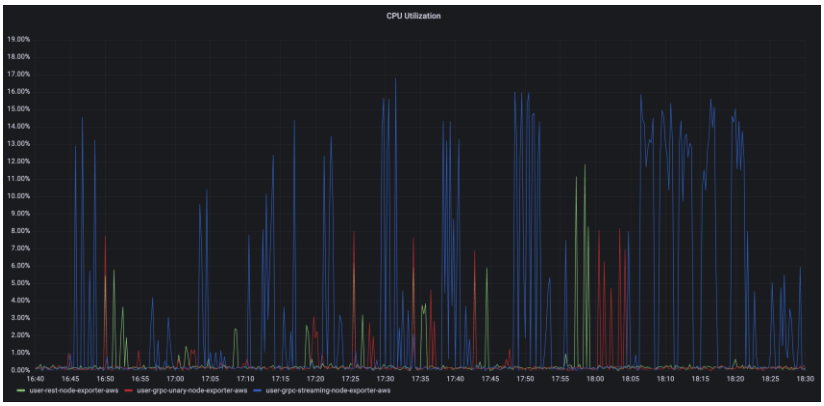
A continuación, se presenta el uso de recursos a lo largo de toda la prueba, este es un punto de partida para determinar si algún factor afecta en el consumo de recursos. Además, se presenta el rendimiento en general de la tecnología y como la carga puede afectar en la disponibilidad de los servicios.

Figura 38. **Uso de Memoria, prueba 3**



Fuente: elaboración propia, realizado en Grafana.

Figura 39. **Uso de *CPU*, prueba 3**



Fuente: elaboración propia, realizado en Grafana.

Tabla XLIV. **Rendimiento de tecnología entre el tamaño de archivo, prueba 3**

Peticiones/ segundo	Carga							
Tecnología	20	40	50zip	50txt	55	70	130	512
Rest	3.14	1.61	1.31	1.12	1.15	0.88	0.45	0.14
gRPC Unario	1.24	0.58	0.43	0.45	0.41	0.35	0.20	0.07
gRPC ***Streaming***	0.20	0.11	0.08	0.08	0.07	0.05	0.03	0.01
gRPC ***Streaming Full***	1.18	0.31	0.49	0.55	0.35	0.42	0.23	0.03

Fuente: elaboración propia, realizado en Microsoft Word.

Tabla XLV. **Peticiones fallidas y exitosas, prueba 3**

Tecnología	Peticiones exitosas	Peticiones fallidas
Rest	40	0
gRPC Unario	40	0
gRPC *Streaming*	40	0
gRPC *Streaming Full*	40	0

Fuente: elaboración propia, realizado en Microsoft Word.

8. INTERPRETACIÓN DE RESULTADOS

A continuación, se plantea una explicación de los resultados obtenidos en las pruebas para ambas tecnologías.

8.1. Prueba 1 – Escenario 1

A continuación, se describen los resultados obtenidos para cada una de las medidas y métricas de comparación definidas en el capítulo 5, bajo las condiciones de la primera prueba de forma concurrente utilizando cada tecnología.

8.1.1. Normalidad y Homocedasticidad

Como se puede visualizar en tabla XI, solamente para los valores de carga 100 y 500 los tiempos de respuesta se comportan de manera normal, pero el resto no tienen una distribución normal. Por su parte el tiempo de procesamiento solo muestra normalidad para las cargas 10, 50 y 100.

En la tabla XII, se aprecia que ningún resultado para el tiempo de respuesta y procesamiento posee una distribución normal.

Para la varianza únicamente el tiempo de respuesta contra la tecnología son iguales, pero para el resto existe una alta variabilidad entre las varianzas.

8.1.2. Comparación de medias

Como se observa en la figura 20 y 21, el tiempo de respuesta y procesamiento de *Rest* a frente a gRPC es aproximadamente igual hasta llegar a un valor de carga de 500, en el cual su rendimiento empieza a empeorar, y dando como resultado altos tiempos de respuesta y procesamiento en sistemas concurrentes.

8.1.3. Grupos y significancia

Los grupos y valores de significancia según la tabla XIV, indican que a partir de 500 la carga representa una diferencia significativa en el tiempo de respuesta y procesamiento medio.

Los grupos y valores de significancia según la tabla XV, indican que la tecnología si representa una diferencia significativa en el tiempo de respuesta y procesamiento medio.

8.1.4. Recursos y rendimiento general

Durante las pruebas se puede notar una diferencia para el nivel de carga 1 a comparación del resto, esto se puede deber a que durante la primera petición los servidores inicializan las conexiones, por lo que en las siguientes iteraciones el tiempo de respuesta es menor al de la primera petición.

El uso de *CPU* en cada tecnología fue aproximadamente igual, tal y como se puede observar en la figura 23. Por otra parte, el uso de memoria fue considerablemente mayor en gRPC *Streaming* como se observa en la figura 22, esto se pudo deber a que la conexión entre el cliente y servidor siempre estaba

abierta y pudo generar mayor uso de memoria, por lo que a mayor carga se utiliza más memoria.

Según la tabla XVI, el rendimiento entre tecnologías es similar, aunque mientras aumenta el factor de carga, disminuye el rendimiento para todas las tecnologías, pero entre todas *Rest* es la más deficiente.

En sistemas concurrentes *Rest* es más propenso a fallar cuando se aumenta la carga del sistema, esto se aprecia claramente en la tabla XVII, cuando el nivel de carga es de 5,000 peticiones, gRPC no rechazo peticiones durante la prueba.

8.2. Prueba 1 – Escenario 2

A continuación, se describen los resultados obtenidos para cada una de las medidas y métricas de comparación definidas en el capítulo 5, bajo las condiciones de la primera prueba de forma secuencial utilizando cada tecnología.

8.2.1. Normalidad y Homocedasticidad

Durante las pruebas se puede notar una diferencia para el nivel de carga 1 a comparación del resto, esto se puede deber a que durante la primera petición los servidores inicializan las conexiones, por lo que en las siguientes iteraciones el tiempo de respuesta es menor al de la primera petición.

Según la tabla XVIII, se observa que el tiempo de respuesta se comporta de forma normal a lo largo de los niveles de carga superiores a 1, esto se debe al hecho de que las peticiones se realizaron de forma secuencial. Por otra parte, el tiempo de procesamiento solo para ciertos niveles de carga existe una

distribución normal. Mientras tanto en la tabla XIX, los tiempos de respuesta y procesamiento no se comportan de forma normal para la tecnología.

En la tabla XX, se puede observar que las varianzas son diferentes para los tiempos de respuesta y procesamiento, por lo que no existe homocedasticidad, ya que la varianza no es constante.

8.2.2. Comparación de medias

Como se observa en las figuras 24 y 25, la diferencia entre las medias de las tecnologías es mínima y poseen un comportamiento similar a lo largo de los diferentes valores de carga.

A comparación del escenario 1, los resultados no son tan variantes. Esto se debe al hecho de que al ser peticiones de forma secuencial no existe una gran carga al servidor, permitiendo que un procesamiento similar entre cada iteración.

8.2.3. Grupos y significancia

Los grupos y valores de significancia indican según la tabla XXI, que el nivel de carga 1 es un valor distinto al resto de niveles, por lo que se puede decir que el nivel de carga no representa una diferencia significativa en el tiempo de respuesta y procesamiento.

Los grupos y valores de significancia indican según la tabla XXII, que la tecnología no representa una diferencia significativa en el tiempo de respuesta, pero en el tiempo de procesamiento, gRPC unario si representa una diferencia significativa en el tiempo de procesamiento.

8.2.4. Recursos y rendimiento general

El uso de *CPU* en cada tecnología fue aproximadamente igual, tal y como se puede observar en la figura 27. Por otra parte, en la figura 26, el uso de memoria en cada tecnología fue aproximadamente igual, pero gRPC *Streaming* tuvo algunos picos en ciertas ocasiones.

El rendimiento entre tecnologías es similar y mientras aumenta la carga se pueden observar resultados consistentes.

En sistemas secuenciales todas las tecnologías se comportan adecuadamente y para una carga de 1,000 peticiones no presentan ninguna petición rechazada.

8.3. Prueba 2 – Escenario 1

A continuación, se describen los resultados obtenidos para cada una de las medidas y métricas de comparación definidas en el capítulo 5, bajo las condiciones de la segunda prueba de forma concurrente utilizando cada tecnología.

8.3.1. Normalidad y Homocedasticidad

Como se observa en las tablas XXV y XXVI, los tiempos de respuesta y procesamiento no tienen un comportamiento normal para la carga y tecnología.

Por otra parte, los resultados del valor p para los tiempos de respuesta y procesamiento contra la carga y tecnología no poseen homocedasticidad, por lo que la varianza no es constante.

8.3.2. Comparación de medias

Como se puede observar en las figuras 28 y 29, gRPC se comporta de forma constante para el tiempo de respuesta y procesamiento mientras aumenta la carga, pero los resultados de *Rest* se disparan luego de una carga de 5,000 peticiones concurrentes.

8.3.3. Grupos y significancia

Los grupos y valores de significancia de la tabla XXVIII, indican que a partir de 5,000 la carga representa una diferencia significativa en el tiempo de respuesta y procesamiento medio.

Por otra parte, en la tabla XXIX, los grupos y valores de significancia indican que la tecnología si representa una diferencia significativa en el tiempo de respuesta y procesamiento medio.

8.3.4. Recursos y rendimiento general

El uso de *CPU* fue similar durante el inicio de la prueba para todas las tecnologías, pero mientras se aumentó la carga, el uso de *CPU* por parte de gRPC aumento considerablemente, esto se aprecia de mejor manera en la figura 31.

Como se muestra en la figura 30, el uso de memoria fue mayor en *Rest*, esto se puede deber a la alta concurrencia del sistema.

El rendimiento es mejor en *Rest* al inicio de la prueba, pero cuando aumenta la carga el rendimiento de *Rest* empieza a decrecer.

La cantidad de peticiones fallidas en *Rest* es mayor cuando el nivel de carga es 5,000, esto se debe a la alta concurrencia, el servidor rechaza las peticiones porque no las puede procesar, mientras que gRPC no rechazo ninguna petición.

8.4. Prueba 2 – Escenario 2

A continuación, se describen los resultados obtenidos para cada una de las medidas y métricas de comparación definidas en el capítulo 5, bajo las condiciones de la segunda prueba de forma secuencial utilizando cada tecnología.

8.4.1. Normalidad y Homocedasticidad

Según la tabla XXXII, se observa que el tiempo de respuesta y procesamiento no se comporta de forma normal a lo largo de los niveles de carga, pero cuando aumenta la carga se observa una tendencia a una distribución normal, esto se debe a que el sistema se estabiliza cuando la muestra aumenta. Mientras tanto en la tabla XXXIII, los tiempos de respuesta y procesamiento no se comportan de forma normal para la tecnología.

En la tabla XXXIV, se puede observar que las varianzas son diferentes para los tiempos de respuesta y procesamiento, por lo que no existe homocedasticidad, ya que la varianza no es constante.

8.4.2. Comparación de medias

A comparación del escenario 1, los resultados no son tan variantes. Esto se debe al hecho de que al ser peticiones de forma secuencial no existe una gran carga al servidor, permitiendo un procesamiento similar entre cada iteración.

Como se observa en las figuras 32 y 33, el tiempo de respuesta y procesamiento tiene un comportamiento similar en los distintos niveles de carga.

8.4.3. Grupos y significancia

Según la tabla XXXV, los grupos y valores de significancia indican que la carga no representa una diferencia significativa en el tiempo de respuesta y procesamiento.

Por otra parte, como se observa en la tabla XXXVI, los grupos y valores de significancia indican que la tecnología no representa una diferencia significativa en tiempo de respuesta y procesamiento.

8.4.4. Recursos y rendimiento general

Como se observa en la figura 35, el uso de *CPU* tiene un comportamiento constante a lo largo de la duración de la prueba, aunque gRPC es la que presenta picos en ciertas ocasiones.

En la figura 34 se observa que el uso de memoria aumenta a lo largo de la prueba, pero en general el comportamiento para ambas tecnologías es similar.

El rendimiento entre tecnologías es similar y mientras aumenta la carga se pueden observar resultados consistentes.

En sistemas secuenciales todas las tecnologías se comportan adecuadamente y para una carga de 1,000 peticiones no presentan ninguna petición rechazada.

8.5. Prueba 3

A continuación, se describen los resultados obtenidos para cada una de las medidas y métricas de comparación definidas en el capítulo 5, bajo las condiciones de la tercera prueba de forma secuencial utilizando cada tecnología.

8.5.1. Normalidad y Homocedasticidad

Para las pruebas de transferencia de archivos, según la tabla XXXIX, el tiempo de respuesta y procesamiento no se comporta de forma normal para la carga. De igual manera en la tabla XL, el tiempo de respuesta y procesamiento no se comporta de forma normal para la tecnología.

En la tabla XLI, se puede observar que las varianzas son diferentes para los tiempos de respuesta y procesamiento, por lo que no existe homocedasticidad, ya que la varianza no es constante.

8.5.2. Comparación de medias

En líneas generales la transferencia de archivos completos se comporta de forma similar, si observan las figuras 36 y 37, se determina que *Rest* posee los tiempos de respuesta más bajos.

Por otra parte, se observa que enviar un archivo por partes conlleva un mayor tiempo de procesamiento, ya que es necesario procesar todas las partes antes de finalizar la conexión. Se debe tomar en cuenta que, dividir un archivo por partes conlleva cierto procesamiento, por lo que, utilizar esta forma de transferir archivos se debe llevar a cabo para necesidades específicas.

8.5.3. Grupos y significancia

Los grupos y valores de significancia según la tabla XLII, indican que cuando el tamaño del archivo es grande, si se representa una diferencia significativa en el tiempo de respuesta y procesamiento.

Los grupos y valores de significancia según la tabla XLIII, indican que enviar un archivo por partes si tiene una diferencia significativa en el tiempo de respuesta y procesamiento, a comparación de enviar un archivo completo el cual no tiene un impacto entre tecnologías.

8.5.4. Recursos y rendimiento general

El uso de *CPU* y memoria se comporta de forma similar para los casos de *Rest* y gRPC Unario, pero con gRPC *Streaming* se intensifica el uso, esto se debe a la conexión constante que se mantiene provocando un incremento en el uso de *CPU* y memoria. El uso de *CPU* y memoria aumenta cuando el tamaño del archivo se incrementa.

Como se observa en la tabla XLIV, *Rest* muestra un mejor rendimiento con cualquier tamaño de archivo, mientras que gRPC al enviar el archivo por partes muestra el peor rendimiento, esto se debe a que el proceso de extraer partes de

los archivos conlleva cierto procesamiento, por lo que este proceso está pensado para otros tipos de carga de trabajo.

Todas las tecnologías procesaron las peticiones sin ningún rechazo.

8.6. Comentarios generales

Bajo las condiciones de las pruebas realizadas gRPC maneja tiempos de respuesta y procesamiento menores a comparación a *Rest*, esto aplica para una implementación completamente unaria o bidireccional.

Con gRPC en su configuración bidireccional completa se consume una mayor cantidad de recursos de *CPU* y memoria a comparación de *Rest* y gRPC unario, esto se debe a la conexión que existe entre el cliente y el servidor durante el tiempo en que se ejecuta la petición o el proceso y hasta que se finaliza la conexión.

Con gRPC se logra un entorno de alta disponibilidad, ya que para un sistema concurrente de por lo menos 5,000 peticiones no se encontró un punto de fallo o rechazo en las peticiones, a comparación de *Rest*, el cual es más propenso a rechazar las peticiones de los clientes cuando el servidor se encuentra bajo una alta demanda.

En general para sistemas secuenciales el nivel de carga y la tecnología no representan una diferencia significativa en el tiempo de respuesta y procesamiento de las peticiones, es decir, el uso de *Rest* y gRPC no mejoran o empeoran el rendimiento en cualquier nivel de carga para peticiones secuenciales.

En general la transmisión de archivos es similar entre *Rest*, gRPC unario y bidireccional enviando el archivo completo, pero de estos tres escenarios, *Rest* posee los tiempos más bajos.

Durante las pruebas se determinó que la forma de enviar archivos afecta proporcionalmente al tiempo de respuesta y procesamiento, este comportamiento se observó en el escenario de gRPC bidireccional enviando el archivo por partes, siendo que al extraer partes del archivo y enviarlas se debe tomar en cuenta la penalización en procesamiento que conlleva extraer partes del archivo en el cliente o servidor, así mismo el procesamiento del lado del servidor o cliente inicia más rápido, pero tarda más en finalizar porque se deben procesar todas las partes del archivo.

En general la transmisión de archivos hace que el uso de recursos de *hardware* aumente considerablemente para cualquier tecnología, pero con gRPC bidireccional el uso de recursos aumenta.

CONCLUSIONES

1. *Rest* es una de las tecnologías más utilizadas actualmente para la comunicación entre microservicios, posee una amplia gama de métodos que agregan semántica en la construcción de servicios y es ampliamente soportado en casi cualquier lenguaje de programación. gRPC por su parte, maneja una comunicación de forma más eficiente, al funcionar bajo el protocolo *HTTP* 2.0 permite realizar operaciones de transmisión de mensajes entre cliente y servidor, es soportado por una gran variedad de lenguajes de programación, maneja una lista reducida y comprensible de códigos de estado.

2. Las pruebas se diseñaron con el objetivo de verificar el comportamiento de forma concurrente y paralela. Asimismo, se agregó el enfoque de estrés al servidor por medio de funciones recursivas y verificar si una alta carga útil afecta en el procesamiento de las peticiones. Por último, se diseñaron una serie de pruebas para verificar el comportamiento de las tecnologías frente a la transmisión de archivos de forma completa y por partes utilizando gRPC.

3. Se ejecutaron las pruebas utilizando el servicio *EC2* de *AWS*, el uso de una plataforma en la nube proporciono un ambiente estable, libre de agentes anómalos como pueden ocurrir en un ambiente local. Además, el uso de la red era 100% dedicado a las pruebas, por lo que no ocurrió una reducción en el ancho de banda y la probabilidad de apagones que pudieran afectar la realización de las pruebas era muy baja.

4. Al finalizar cada iteración de las pruebas era necesario almacenar los resultados, esto se llevó a cabo utilizando el servicio *S3* de *AWS*, el uso de este servicio facilito la automatización de las pruebas, esto debido a que, al finalizar una prueba, el archivo se transfería automáticamente al *bucket* seleccionado. Por lo que, al finalizar todas las iteraciones de todas las pruebas, los archivos se encontraban en un lugar centralizado y listos para ser procesados.

5. Se llevaron a cabo diferentes procesos estadísticos descriptivos y pruebas de hipótesis para verificar el comportamiento de los datos, estas operaciones permitieron la obtención de gráficas para facilitar la comprensión en interpretación de resultados, esto se puede visualizar principalmente en las gráficas de interacción y tablas de agrupamiento y significancia.

RECOMENDACIONES

1. Enfocar la transmisión de archivos por partes en casos donde el archivo debe ser procesado rápidamente y no es dependiente de funcionar si no están todas las partes completas, un ejemplo de esto es la reproducción de contenido multimedia que se lleva a cabo mientras cada parte llega al destino, permitiendo que el cliente no deba esperar mucho tiempo para procesar el contenido.

2. Tomar en cuenta que, la transmisión de archivos por partes se recomienda en casos donde la prioridad no sea sobrecargar el servidor, esto con el fin de recibir pequeñas partes del archivo e irlo completando mientras llegan, reduciendo de esta manera la cantidad de memoria que se utiliza a comparación de enviar el archivo completo.

3. Iniciar con gRPC utilizando la forma unaria es un buen punto de partida, ya que su implementación es sencilla y tiene muchas similitudes con *Rest*, pero agregando las ventajas de gRPC.

4. Introducir un enfoque más robusto a las pruebas que se presentaron en este proyecto se recomienda para aquellos que quisieran realizar un trabajo similar y tomando en cuenta el escalamiento para aplicaciones que utilicen comunicación mediante *API Rest*, esto con el objetivo de reducir la tasa de peticiones fallidas.

5. Considerar el uso de gRPC para aquellas personas que trabajan de manera común con sockets o necesitan un tipo de comunicación en

tiempo real, la comunicación bidireccional con gRPC es ideal, ya que posee todos los requisitos para este caso de uso, por ejemplo, una aplicación de chat o un sistema que realice cálculos que no deben estar expuestos en el cliente, entre otras.

REFERENCIAS

1. Devore, J. (2008). *Probabilidad y Estadística para Ingeniería y Ciencias.* Boston, Estados Unidos: Cengage Learning.

2. *Elastic compute cloud (EC2) de capacidad modificable en la nube* (2022). *AWS*. Recuperado de https://aws.amazon.com/es/ec2/.

3. *Grafana OSS* (2022). Grafana. Recuperado de https://grafana.com/docs/grafana/latest/introduction/.

4. *Introduction to gRPC* (2022). gRPC. Recuperado de https://grpc.io/docs/what-is-grpc/introduction/.

5. *Prometheus Metrics: A Practical Guide to Types, Uses, Functions, Exporters, and More* (2022). Tigera. Recuperado de https://www.tigera.io/learn/guides/prometheus-monitoring/prometheus-metrics/.

6. *Prometheus Overview* (2022). Prometheus. Recuperado de https://prometheus.io/docs/introduction/overview/.

7. *Protocol Buffers Overview* (2022). Developers Google. Recuperado de https://developers.google.com/protocol-buffers/docs/overview.

8. *¿Qué son los contenedores?* (2022). Cloud Google. Recuperado de https://cloud.google.com/learn/what-are-containers?hl=es.

9. *¿Qué son los microservicios?* (2022). *AWS*. Recuperado de https://aws.amazon.com/es/microservices/.

10. *What is Postman?* (2022). *Postman*. Recuperado de https://www.postman.com/product/what-is-postman/.