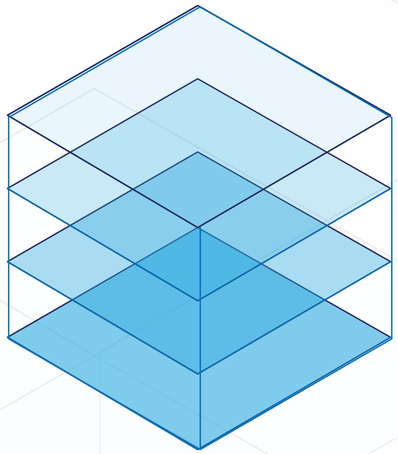


MISO

Maestría en Ingeniería de Software

Diseño de experimento - gRPC

Proyecto Final 1

Titulo del experimento	**Mejora de tiempos de respuestas con gRPC**
Propósito del experimento	El objetivo del experimento es validar la mejora en cuanto a tiempo de respuesta en la comunicación interna entre microservicios, todas las comunicaciones internas entre microservicios se van a realizar utilizando gRPC y se va a realizar la comparación comunicación HTTP/Rest.
Resultados esperados	Se espera que en el 100% de las veces se logre evidenciar que los tiempos de respuestas de comunicaciones gRPC son menores a los tiempos de respuestas de un API Rest tradicional.
Recursos requeridos	**Software:** NestJS para la implementación del backend, herramientas de pruebas como Postman para simular acciones de usuario, y una base de datos PostgreSQL para almacenamiento de registros. **Librerías:** Librerías para soporte de comunicación con gRPC **Hardware:** Instancias de CloudRun en la nube para desplegar el sistema y simular diferentes condiciones de carga.
Elementos de arquitectura involucrados	Componentes **API Gateway, Mediator e Incident Process Service**. Ubicados en la vista de desarrollo en los modelos de mediator y Business Logic.
Esfuerzo estimado	Aproximamente 15 Horas/hombre distribuidas entre 4 ingenieros de desarrollo.

Hipótesis de diseño	
Punto de sensibilidad	Este experimento se enfoca en la latencia y los tiempos de respuesta como punto de sensibilidad, ya que se busca validar la mejora en el rendimiento de las comunicaciones internas entre microservicios utilizando gRPC en lugar de HTTP/Rest. El punto de sensibilidad clave es el tiempo de respuesta de las comunicaciones, el cual debe ser consistentemente menor con gRPC para ser considerado una mejora significativa.
Historia de arquitectura asociada	Como **usuario final**, cuando **se envia una peticion para registrar un incidente** dado que **el sistema se encuentra en operacion normal**, quiero que **se registre el incidente para brindarle efectivamente el servicio**. Esto debe suceder en **menos de 2 segundos el 95% fe las veces.**
Nivel de incertidumbre	**Medio:** Aunque se espera que gRPC proporcione mejores tiempos de respuesta debido a su diseño más eficiente para la comunicación de microservicios, hay factores de implementación y condiciones de carga que podrían afectar los resultados. Estos incluyen la configuración específica de los microservicios, la latencia de red en Cloud Run y la forma en que se simulan las cargas de trabajo. Por lo tanto, hay cierta incertidumbre sobre si gRPC siempre superará a HTTP/Rest en todos los escenarios de carga.

Estilos de Arquitectura asociados al experimento	Análisis (Atributos de calidad que favorece y desfavorece)
Microservicios	**Favorece:** • Escalabilidad: Cada microservicio puede escalarse de forma independiente según la demanda. • Mantenibilidad: Al estar desacoplados, los microservicios permiten un desarrollo y despliegue independientes, facilitando actualizaciones y mantenimiento. • Disponibilidad: Falla en un microservicio no implica la caída de todo el sistema, permitiendo una mayor tolerancia a fallos. **Desfavorece:** • Modificabilidad: Introduce complejidad en la gestión del sistema debido al número de servicios y la necesidad de monitoreo y orquestación adecuados. • Consistencia de Datos: Asegurar la consistencia de datos entre microservicios puede ser un desafío, especialmente cuando se requieren transacciones distribuidas.
Uso de gRPC en el contexto de microservicios	**Favorece:** • Latencia: gRPC aporta a este atributo al proporcionar un protocolo de comunicación binario eficiente. **Desfavorece:** • Mantenibilidad: Introduce cierta complejidad en la configuración y mantenimiento comparado con enfoques más sencillos como HTTP/REST.

Tácticas de Arquitectura asociadas al experimento	Descripción
Latencia: Increase Efficiency of Resource Usage	La táctica de “Increase Efficiency of Resource Usage” se centra en optimizar los recursos y mejorar el perormance de nuestros componentes, para implementar esta táctica nos vamos a enfocar en la comunicación entre componentes utilizando el protocolo **gRPC** en este caso para reducir la latencia y mejorar el rendimiento general del sistema, esta tactica aplica para la comunicación entre componentes internos del sistema.

Listado de componentes (Microservicios) involucrados en el experimento		
Microservicio	Propósito y comportamiento esperado	Tecnología Asociada
API Gateway	Orquestar y balancear la carga de todas las peticiones que reciba el sistema de ABCall. Debe redirigir la petición hacia el mediator.	NestJS, gRPC,
Mediator	Se encarga de validar, orquestar y redirigir las peticiones hacia las capacidades expuestas en la capa de business logic del sistema de ABCall. Debe recibir la petición y redirigir la petición hacia el Incident Process Service.	NestJS, gRPC
Incident Process Service	Es el responsable de toda la gestión relacionada a los incidentes del sistema, desde su registro hasta la resolución del mismo. Debe recibir la petición y registrar correctamente el incidente en la base de datos.	NestJS, PostgreSQL

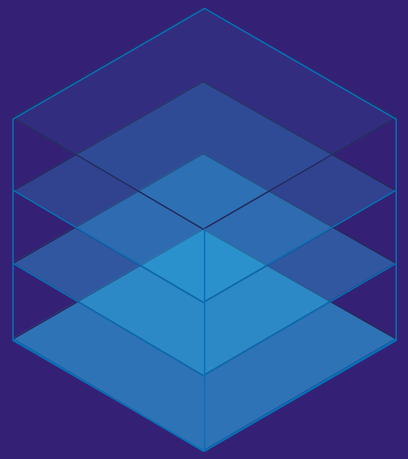
Listado de conectores involucrados en el experimento

Conector	Comportamiento deseado en el experimento	Tecnología Asociada
HTTP/Rest in API Gateway	Todas las peticiones del experimento tendrán como punto de entrada será el API Gateway, estas peticiones serán redirigidas hacia el mediator indicado.	NestJs
gRPC in Mediator	Gestionara las peticiones recibidas por gRPC desde el API Gateway y las redirigirá hacia el Incident Process Service.	NestJs, gRPC
gRPC in Incident Process Service	Se encarga de registrar los incidentes en la base de datos del sistema.	NestJs, gRPC, PostgreSQL

Tecnología asociada con el experimento (Desarrollo, infraestructura, almacenamiento)	Justificación
Lenguajes de programación	**TypeScript**: Utilizado con **NestJS** para el desarrollo del backend, proporciona un tipado estático que mejora la robustez y mantenibilidad del código.
Plataforma de despliegue	**Google Cloud Run**: Permite desplegar contenedores en la nube con escalabilidad automática y gestión simplificada, ideal para microservicios desarrollados en NestJS.
Bases de datos	**PostgreSQL**: Elegido por su robustez, soporte de transacciones complejas, y capacidad para manejar un gran volumen de registros de auditoría de manera eficiente.
Herramientas de análisis	Jmeter, Apache AB Benchmark.
Librerías	**gRPC:** Se utilizara las librerías necesarias para poder utilizar Protocol Buffers con gRPC dentro de NestJS.
Frameworks de desarrollo	**NestJS**: Un framework progresivo de Node.js que facilita la creación de aplicaciones escalables y mantenibles mediante el uso de patrones de diseño robustos y buenas prácticas.

Distribución de actividades por integrante

Integrante	Tareas a realizar	Esfuerzo Estimado
Jesús Henríquez	- Desarrollo del API Gateway en NestJ y su implementación HTTP/Rest y con gRPC. - Desarrollo del Mediator en NestJs y su implementación HTTP/Rest y con gRPC. - Desarrollo del Incident Process Service en NestJS y su implementación HTP/Rest y con gRPC.	15h
Daniel Jimenez	- Configuración de la infraestructura en **Google Cloud Run** para despliegue de microservicios. - Configuración de **Google Cloud Pub/Sub** para comunicación asíncrona. - Configuración de bases de datos en **PostgreSQL** para el almancenamiento de los Incidentes.	
Carlos Castillo	Realizar pruebas de carga para evaluar los tiempos de respuestas de gRPC y HTTP/Rest con Jmeter.	1h
Santiago Begambre	Realizar pruebas de carga para evaluar los tiempos de respuestas de gRPC y HTTP/Rest con Apache AB Benchmark	



MISO

Maestría en Ingeniería de Software