Trabajo Final gRPC: Java vs Ruby

Enunciado

1. Implementar lo mismo que se hizo en las prácticas con gRPC en Java pero en Ruby.
2. La comparación debe incluir:
    2.1. Manejo de infraestructura de software con maven vs. lo que uses con Ruby. Más específicamente: cómo se definen las dependencias en maven-java y cómo las definís en lo que uses para Ruby. Esto incluye dar todos los detalles del .xml en el caso de maven y dependiendo de lo que uses en Ruby cómo se resuelven/proveen todas las bibliotecas.
    2.2. Rendimiento: dado que vas a hacer lo mismo que hiciste en la práctica, comparar los tiempos de transferencia de un archivo de 1k, 10k, 100k, 1M, 10M.
    2.3. Software propiamente dicho, es decir qué programaste en Java y qué programás en Ruby. Podés hacerlo con un ejemplo/operación en particular, y a lo sumo lo veo y te pido alguna otra función si veo que es necesario.
3. Como siempre, entregar código fuente e informe, incluyendo lo anterior + instrucciones de instalación y uso de lo que desarrolles.

1. Manejo de gRPC en Ruby

Las herramientas gRPC de Ruby incluyen el compilador de búfer de protocolo protoc y el complemento especial para generar código de servidor y cliente a partir de las .proto definiciones de servicio.

Con gRPC podemos definir nuestro servicio una vez en un .proto archivo y generar clientes y servidores en cualquiera de los lenguajes soportados por gRPC, que a su vez pueden ejecutarse en entornos que van desde servidores dentro de un gran centro de datos hasta su propia tableta: toda la complejidad de la comunicación entre gRPC maneja diferentes idiomas y entornos.

2.1 Manejo de infraestructura de software

Con Java podemos utilizar Maven como una herramienta de gestión de proyectos que ofrece una serie de beneficios significativos para el desarrollo de software, especialmente en entornos Java, aunque no exclusivamente. Algunas razones para utilizar Maven son:

1. **Gestión de dependencias:** Maven simplifica la gestión de las bibliotecas y dependencias de tu proyecto. Maneja la descarga automática de bibliotecas desde repositorios remotos, lo que elimina la necesidad de descargarlas manualmente.

2. **Ciclo de vida del proyecto:** Maven define un ciclo de vida predefinido con fases específicas (compilación, prueba, empaquetado, etc.). Esto permite estandarizar y automatizar tareas comunes en el desarrollo.

3. **Estructura del proyecto:** Proporciona una estructura predefinida para los proyectos, lo que facilita la organización y el mantenimiento del código. Esto asegura que los proyectos sigan una estructura común, lo que ayuda a los nuevos desarrolladores a orientarse más rápidamente.

4. **Construcciones coherentes:** Maven permite la creación de construcciones coherentes y repetibles. Esto asegura que todos los miembros del equipo estén utilizando las mismas configuraciones y dependencias, lo que minimiza los problemas de compatibilidad y errores relacionados con el entorno de desarrollo.

5. **Integración con repositorios:** Facilita la publicación de artefactos en repositorios remotos para compartir con otros equipos o proyectos. Esto simplifica la distribución y el intercambio de componentes de software entre proyectos y equipos.

6. **Plugins y extensibilidad:** Maven es altamente extensible a través de plugins, lo que permite adaptar su funcionalidad para satisfacer necesidades específicas del proyecto.

Dependencias

En el ecosistema de Ruby, el uso del Gemfile es una convención establecida por Bundler, una herramienta que simplifica la gestión de dependencias para proyectos Ruby. El Gemfile actúa como un descriptor de dependencias, especificando qué gemas (paquetes o bibliotecas de Ruby) necesita tu proyecto y en qué versiones.

La elección de un archivo específico como el Gemfile se debe a la filosofía de convención sobre configuración en Ruby. Bundler proporciona un flujo de trabajo estandarizado para la gestión de dependencias, lo que facilita la reproducibilidad del entorno de desarrollo en diferentes máquinas. Al compartir un proyecto Ruby, otros desarrolladores pueden utilizar el Gemfile para instalar las mismas dependencias y versiones específicas, asegurando una consistencia en el ambiente de desarrollo.

En Maven, las dependencias se definen en el archivo pom.xml. Aquí está un ejemplo de cómo se definen las dependencias para gRPC en Maven:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>tu.grupo.id</groupId>
    <artifactId>nombre-proyecto</artifactId>
    <version>1.0.0</version>

    <dependencies>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-netty-shaded</artifactId>
            <version>1.42.0</version>
        </dependency>
        <!-- Otras dependencias pueden ir aquí -->
    </dependencies>
</project>
```

Las dependencias se especifican dentro del bloque <dependencies>. Cada dependencia tiene un groupId, artifactId, y version. Maven se encarga de resolver estas dependencias descargando los artefactos necesarios (bibliotecas) de repositorios remotos.

2.2 Rendimiento:

Comparación de tiempos de transferencia de archivos de diferentes tamaños (1k, 10k, 100k, 1M, 10M) entre las implementaciones en Java y en Ruby de gRPC. Esto implica medir y comparar los tiempos que toma realizar estas transferencias utilizando ambos entornos.

	1k	10k	100k	1M	10M
Ruby	3.18 ms	12.72 ms	98.74 ms	980.02 ms	9937.45 ms
Java	3 ms	6 ms	8 ms	36 ms	209 ms

- **Tiempos de ejecución en Ruby:** Las siguientes filas muestran los tiempos de ejecución, en milisegundos, de ciertas operaciones o programas en Ruby para cada tamaño de datos. Por ejemplo, para el tamaño de datos de 1 kilobyte, el tiempo de ejecución en Ruby es de 3.18 milisegundos. A medida que aumenta el tamaño de los datos, los tiempos de ejecución en Ruby también aumentan significativamente. Para el tamaño de datos de 10 millones de bytes, el tiempo de ejecución en Ruby es de 9937.45 milisegundos.

- **Tiempos de ejecución en Java:** Las columnas en "Java" muestran los tiempos de ejecución, en milisegundos, de las mismas operaciones o programas, pero esta vez en Java. En general, los tiempos de ejecución en Java son más bajos que en Ruby para los mismos tamaños de datos. Por ejemplo, para el tamaño de datos de 1 kilobyte, el tiempo de ejecución en Java es de 3 milisegundos, mientras que en Ruby es de 3.18 milisegundos. Sin embargo, a medida que aumenta el tamaño de los datos, los tiempos de ejecución en Java también aumentan, aunque en general son menores que en Ruby.

2.3 Software

El software empleado en las pruebas de rendimiento está disponible en un repositorio de GitHub, junto con su archivo README que detalla los pasos de instalación y los comandos necesarios.

LINK: https://github.com/DelleDonneMati/final-PDyTR/tree/main