

Projet gRPC HelloWorld Multilingue

Test de communication entre PC et Smartphone

Safwen Essayes ST-iot

21 octobre 2025

Table des matières

1	Introduction	2
2	Présentation du projet gRPC	2
2.1	Qu'est-ce que gRPC?	2
2.2	Protocol Buffers (Protobuf)	2
2.3	Structure du projet	3
3	Configuration de l'environnement	3
3.1	Serveur PC	3
3.2	Smartphone avec Termux	3
3.3	Environnement virtuel Python	4
3.4	Transfert des fichiers client	4
4	Configuration de la connexion	4
4.1	Adresse IP du serveur	4
4.2	Configuration du client	4
5	Démarrage du serveur	5
6	Tests de communication gRPC	5
6.1	Test 1 : Appel simple (Unary RPC)	5
6.2	Test 2 : Streaming serveur (Server Streaming RPC)	5
6.3	Test 3 : Streaming client (Client Streaming RPC)	5
6.4	Test 4 : Streaming bidirectionnel	6
7	Résultats et conclusion	6
7.1	Synthèse des résultats	6
7.2	Conclusion	6

1 Introduction

Ce projet implémente une application gRPC HelloWorld multilingue établissant une communication entre un serveur Java (PC) et un client Python (smartphone). Il démontre les quatre modèles de communication gRPC et offre des salutations en anglais, français et arabe.

2 Présentation du projet gRPC

2.1 Qu'est-ce que gRPC ?

gRPC est un framework RPC open source développé par Google permettant d'appeler des méthodes sur un serveur distant comme s'il s'agissait d'appels locaux, utilisant Protocol Buffers pour la définition d'interface et la sérialisation.

2.2 Protocol Buffers (Protobuf)

Protocol Buffers est un mécanisme de sérialisation de données. Notre fichier `hello_service.proto` définit :

```
1 syntax = "proto3";
2
3 service MultilingualGreeter {
4     // Unary RPC - Simple request/response
5     rpc SayHello (HelloRequest) returns (HelloResponse) {}
6
7     // Server streaming RPC
8     rpc SayHellosServerStreaming (HelloRequest) returns (stream
9         HelloResponse) {}
10
11    // Client streaming RPC
12    rpc SayHellosClientStreaming (stream HelloRequest) returns (
13        HelloResponse) {}
14
15    // Bidirectional streaming RPC
16    rpc SayHellosBidirectional (stream HelloRequest) returns (stream
17        HelloResponse) {}
18 }
19
20 message HelloRequest {
21     string first_name = 1;
22     string last_name = 2;
23     string language_code = 3;
24 }
25
26 message HelloResponse {
27     string greeting = 1;
28 }
```

Ce fichier proto :

- Définit le contrat d'API entre client et serveur
- Spécifie les structures de données échangées
- Déclare les méthodes RPC disponibles
- Permet la génération automatique de code

De ce fichier sont générés :

- `hello_service_pb2.py` : Classes de messages
- `hello_service_pb2_grpc.py` : Code client/serveur pour la communication

2.3 Structure du projet

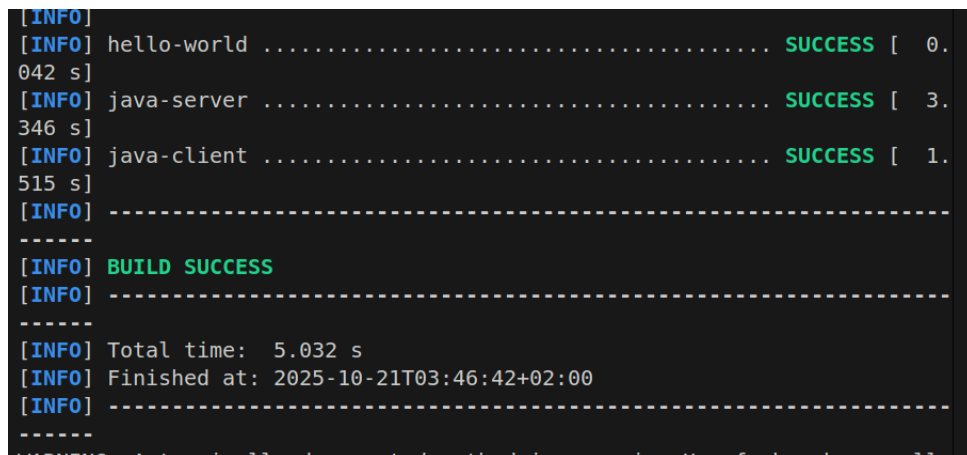
- `proto/` : Définition du service gRPC
- `java-server/` : Serveur Java
- `java-client/` : Client Java pour tests sur PC
- `python-client/` : Client Python pour smartphone

3 Configuration de l'environnement

3.1 Serveur PC

Prérequis :

- Java JDK 11+
- Maven 3.6+
- Protocol Buffers (protoc)



```
[INFO] hello-world ..... SUCCESS [ 0.042 s]
[INFO] java-server ..... SUCCESS [ 3.346 s]
[INFO] java-client ..... SUCCESS [ 1.515 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.032 s
[INFO] Finished at: 2025-10-21T03:46:42+02:00
[INFO] -----
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
```

FIGURE 1 – Compilation du serveur Java avec Maven

Comme le montre la Figure 1, nous compilons le projet Java avec Maven qui génère automatiquement les classes Java à partir des définitions Protocol Buffers et compile le serveur gRPC.

3.2 Smartphone avec Termux

Installation de Termux :

```
1 pkg update && pkg upgrade
2 pkg install python python-pip
3 pip install grpcio grpcio-tools protobuf
```

Installation d'Ubuntu sur Termux :

```
1 pkg install proot-distro
2 proot-distro install ubuntu
3 proot-distro login ubuntu
```

3.3 Environnement virtuel Python

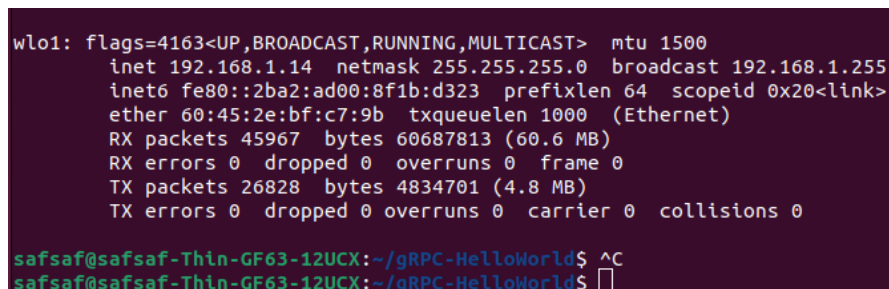
```
1 # Mettre à jour Ubuntu
2 apt update
3 apt upgrade -y
4
5 # Installer Python et dépendances
6 apt install -y python3 python3-pip python3-venv build-essential python3-dev
7
8 # Créer et activer l'environnement virtuel
9 python3 -m venv grpc_env
10 source grpc_env/bin/activate
11
12 # Installer packages gRPC
13 pip install grpcio grpcio-tools
```

3.4 Transfert des fichiers client

```
1 # Copier les fichiers vers Ubuntu dans Termux
2 cp ~/downloads/* /data/data/com.termux/files/usr/var/lib/proot-distro/
   installed-rootfs/ubuntu/root/
```

4 Configuration de la connexion

4.1 Adresse IP du serveur



```
wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.14 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::2ba2:ad00:8f1b:d323 prefixlen 64 scopeid 0x20<link>
    ether 60:45:2e:bf:c7:9b txqueuelen 1000 (Ethernet)
    RX packets 45967 bytes 60687813 (60.6 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 26828 bytes 4834701 (4.8 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

safsaf@safsaf-Thin-GF63-12UCX:~/gRPC-HelloWorld$ ^C
safsaf@safsaf-Thin-GF63-12UCX:~/gRPC-HelloWorld$
```

FIGURE 2 – Identification de l'adresse IP du serveur

La Figure 2 montre l'exécution de la commande `ipconfig` sur le PC serveur pour identifier l'adresse IP WiFi (10.136.70.244) qui sera utilisée par le client pour se connecter au serveur.

4.2 Configuration du client

```
1 parser.add_argument('--target', default='10.136.70.244:50051',
2                       help='Server address in format host:port')
```

```
oct. 21, 2025 3:46:42 AM com.grpc.hello.server.HelloWorldServer start
INFO: Server started, listening on 50051
```

FIGURE 3 – Démarrage du serveur gRPC

5 Démarrage du serveur

Comme illustré dans la Figure 3, le serveur Java est lancé avec succès et écoute sur le port 50051, prêt à recevoir des connexions entrantes de clients gRPC.

```
oct. 21, 2025 3:46:42 AM com.grpc.hello.server.HelloWorldServer start
INFO: Server started, listening on 50051
oct. 21, 2025 3:47:33 AM com.grpc.hello.server.MultilingualGreeterImpl sayHelloServerStreaming
INFO: Received server streaming request from: saf
oct. 21, 2025 3:47:47 AM com.grpc.hello.server.MultilingualGreeterImpl sayHelloServerStreaming
INFO: Received server streaming request from: saf
oct. 21, 2025 3:48:11 AM com.grpc.hello.server.MultilingualGreeterImpl sayHelloServerStreaming
INFO: Received server streaming request from: saf
oct. 21, 2025 3:48:26 AM com.grpc.hello.server.MultilingualGreeterImpl sayHello
INFO: Received unary request from: saf saf
oct. 21, 2025 3:49:15 AM com.grpc.hello.server.MultilingualGreeterImpl$1 onNext
INFO: Received client streaming request part from: ali
oct. 21, 2025 3:49:15 AM com.grpc.hello.server.MultilingualGreeterImpl$1 onNext
INFO: Received client streaming request part from: ali ali
oct. 21, 2025 3:49:45 AM com.grpc.hello.server.MultilingualGreeterImpl sayHello
INFO: Received unary request from: saf sayes
```

FIGURE 4 – Serveur gRPC en cours d'exécution

La Figure 4 montre le serveur en cours d'exécution, affichant les détails de la configuration et confirmant qu'il est opérationnel.

6 Tests de communication gRPC

6.1 Test 1 : Appel simple (Unary RPC)

La Figure 5 illustre un test réussi d'appel unaire (simple requête/réponse) depuis notre client sur smartphone. On peut y observer l'interface interactive permettant de saisir le prénom, le nom et la langue choisie, ainsi que la réponse du serveur.

6.2 Test 2 : Streaming serveur (Server Streaming RPC)

La Figure 6 démontre le streaming côté serveur, où le serveur envoie plusieurs réponses à une seule requête client. On peut observer les multiples messages de salutation reçus en anglais et en arabe, confirmant le support multilingue de l'application.

6.3 Test 3 : Streaming client (Client Streaming RPC)

La Figure 7 montre le streaming côté client en action, où le client envoie plusieurs requêtes et le serveur répond avec une seule réponse consolidée. On peut également voir sur cette capture un exemple d'appel simple en français, démontrant la flexibilité linguistique du système.

6.4 Test 4 : Streaming bidirectionnel

Notre application prend également en charge le streaming bidirectionnel, où client et serveur peuvent échanger simultanément des flux de messages.

7 Résultats et conclusion

7.1 Synthèse des résultats

Notre système démontre :

- Une communication réussie entre serveur Java (PC) et client Python (smartphone)
- L'implémentation des quatre modèles de communication gRPC
- Le support multilingue (anglais, français, arabe)

Comme le montrent les Figures 5, 6 et 7, tous les types de communication gRPC fonctionnent correctement entre le PC et le smartphone, et le système répond bien dans les trois langues implémentées.

7.2 Conclusion

Ce projet démontre l'efficacité de gRPC pour les applications distribuées multilingues. Protocol Buffers offre une définition claire des interfaces et une sérialisation efficace. L'intégration entre Java et Python illustre la capacité de gRPC à faciliter la communication inter-langages et multi-plateformes.

```
2:50 [signal] [battery]

=== Menu gRPC Client Multilingue ===
1. Appel simple (Unary RPC)
2. Streaming Serveur (Server Streaming RPC)
3. Streaming Client (Client Streaming RPC)
4. Streaming Bidirectionnel (Bidirectional Streaming RPC)
5. Quitter

Choisissez une option (1-5): 1

=== Appel Unary RPC ===
Entrez votre prénom: saf
Entrez votre nom: saf
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): fr
Making unary call with saf saf in language fr
Server response: Bonjour, saf saf!

=== Menu gRPC Client Multilingue ===
1. Appel simple (Unary RPC)
2. Streaming Serveur (Server Streaming RPC)
3. Streaming Client (Client Streaming RPC)
4. Streaming Bidirectionnel (Bidirectional Streaming RPC)
5. Quitter

Choisissez une option (1-5): 3

=== Client Streaming RPC ===
Vous allez envoyer plusieurs salutations (streaming client)
Entrez les informations pour plusieurs personnes. Tapez
'fin' comme prénom pour terminer.

Prénom (ou 'fin' pour terminer): ali
Nom: salah
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): ar
Personne ajoutée: ali salah (ar)

Prénom (ou 'fin' pour terminer): ali ali
Nom: sayes
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): ar
Personne ajoutée: ali ali sayes (ar)

Prénom (ou 'fin' pour terminer): fin
```

FIGURE 5 – Test d'appel simple (Unary RPC)

```
2:50
!
=== Menu gRPC Client Multilingue ===
1. Appel simple (Unary RPC)
2. Streaming Serveur (Server Streaming RPC)
3. Streaming Client (Client Streaming RPC)
4. Streaming Bidirectionnel (Bidirectional Streaming RPC)
5. Quitter

Choisissez une option (1-5): 2

=== Server Streaming RPC ===
Entrez votre prénom: saf
Entrez votre nom: saf
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): ar
Making server streaming call with saf saf
Server streaming response: أهلا وسهلا بك! saf saf!
Server streaming response: أهلا وسهلا بك! saf saf!
Server streaming response: أهلا وسهلا بك! saf saf!
Server streaming response: أهلا وسهلا بك! saf saf!

=== Menu gRPC Client Multilingue ===
1. Appel simple (Unary RPC)
2. Streaming Serveur (Server Streaming RPC)
3. Streaming Client (Client Streaming RPC)
4. Streaming Bidirectionnel (Bidirectional Streaming RPC)
5. Quitter

Choisissez une option (1-5): 2

=== Server Streaming RPC ===
Entrez votre prénom: saf
Entrez votre nom: saf
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): en
Making server streaming call with saf saf
Server streaming response: Hello, saf saf!
Server streaming response: Welcome! , saf saf!
Server streaming response: Nice to meet you! saf saf!
Server streaming response: Have a great day! saf saf!

=== Menu gRPC Client Multilingue ===
1. Appel simple (Unary RPC)
2. Streaming Serveur (Server Streaming RPC)
3. Streaming Client (Client Streaming RPC)
4. Streaming Bidirectionnel (Bidirectional Streaming RPC)
5. Quitter
```

FIGURE 6 – Test de streaming serveur en arabe et en anglais


```
2:50
'fin' comme prénom pour terminer.
Prénom (ou 'fin' pour terminer): ali
Nom: salah
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): ar
Personne ajoutée: ali salah (ar)

Prénom (ou 'fin' pour terminer): ali ali
Nom: sayes
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): ar
Personne ajoutée: ali ali sayes (ar)

Prénom (ou 'fin' pour terminer): fin
Envoi des demandes au serveur...
Envoi de la demande pour ali salah...
Envoi de la demande pour ali ali sayes...
Réponse du streaming client: عي ج ل ا ب آ ح ر م : ة ع ا م ج ة ح ت
ali salah, ali ali sayes!

=== Menu gRPC Client Multilingue ===
1. Appel simple (Unary RPC)
2. Streaming Serveur (Server Streaming RPC)
3. Streaming Client (Client Streaming RPC)
4. Streaming Bidirectionnel (Bidirectional Streaming RPC)
5. Quitter

Choisissez une option (1-5): 1

=== Appel Unary RPC ===
Entrez votre prénom: saf
Entrez votre nom: sayes
Choisissez une langue:
en - Anglais
fr - Français
ar - Arabe
Code de langue (en/fr/ar): fr
Making unary call with saf sayes in language fr
Server response: Bonjour, saf sayes!

=== Menu gRPC Client Multilingue ===
1. Appel simple (Unary RPC)
2. Streaming Serveur (Server Streaming RPC)
3. Streaming Client (Client Streaming RPC)
4. Streaming Bidirectionnel (Bidirectional Streaming RPC)
5. Quitter

Choisissez une option (1-5):
```

FIGURE 7 – Test de streaming client