UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Web services

Sommaire

1. Introduction & SOAP Web Services
2. Web Services RESTful
3. REST avancé, sécurité et documentation
4. gRPC et GraphQL
5. Les microservices

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction & SOAP Web Services

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Qu'est-ce qu'un web service ?

- Un **web service** est une application accessible via un **réseau (souvent Internet)**, permettant à différentes applications — écrites dans des langages ou exécutées sur des plateformes différentes — de **communiquer et d'échanger des données**.

Objectif principal :

- Rendre des fonctionnalités d'un système disponibles à d'autres systèmes via des interfaces standardisées.

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Qu'est-ce qu'un web service ?

-**Exemple simple :**

Un service météo accessible via Internet :

- Entrée : nom d'une ville → `{"city": "Paris"}`
- Sortie : température → `{"temperature": 19}`

Propriétés clés :

- Interopérabilité
- Communication via des protocoles standard (HTTP, HTTPS, TCP, etc.)
- Utilisation de formats d'échange universels (XML, JSON, Protobuf, etc.)

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Architecture SOA (Service-Oriented Architecture)

- L'**architecture orientée services (SOA)** repose sur la notion de **services indépendants** et **réutilisables** qui interagissent via des interfaces bien définies.

Principes de SOA :

- **Faible couplage** : les services ne dépendent pas fortement les uns des autres.
- **Interopérabilité** : les services peuvent communiquer malgré des technologies différentes.
- **Réutilisabilité** : un même service peut être consommé par plusieurs applications.
- **Découverte dynamique** : possibilité de publier et de découvrir les services via un registre (ex. UDDI).

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Architecture SOA (Service-Oriented Architecture)

- L'**architecture orientée services (SOA)** repose sur la notion de **services indépendants** et **réutilisables** qui interagissent via des interfaces bien définies.

Principes de SOA :

- **Faible couplage** : les services ne dépendent pas fortement les uns des autres.
- **Interopérabilité** : les services peuvent communiquer malgré des technologies différentes.
- **Réutilisabilité** : un même service peut être consommé par plusieurs applications.
- **Découverte dynamique** : possibilité de publier et de découvrir les services via un registre (ex. UDDI).

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Architecture SOA (Service-Oriented Architecture)

Exemple :

Une architecture d'entreprise peut avoir :

- un service `Facturation`,
- un service `Livraison`,
- un service `Stock`,
  tous accessibles via un bus d'intégration (ESB).

Évolution :

SOA a inspiré les architectures **microservices**, plus légères et découplées.

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Protocoles utilisés

Les web services utilisent différents **protocoles de transport** selon le contexte et les besoins :

Protocole	Type	Usage principal	Caractéristiques
HTTP	Application	REST, GraphQL, SOAP	Basé sur requêtes/réponses (GET, POST, PUT, DELETE)
HTTPS	Application	REST, SOAP, gRPC	Version sécurisée de HTTP via SSL/TLS
TCP	Transport	gRPC, WebSockets	Connexion persistante, fiable, orientée flux
UDP	Transport	Streaming, IoT	Rapide, sans garantie de livraison

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Protocoles utilisés

Exemple :

- REST → HTTP/HTTPS
- gRPC → HTTP/2 (donc basé sur TCP)
- SOAP → HTTP ou SMTP (selon le contexte)

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Formats d'échange de données

Format	Type	Description	Exemple
XML	Texte structuré	Format verbeux, typé, extensible	`<person><name>Ali</name></person>`
JSON	Texte léger	Format clé-valeur, lisible et efficace	`{"name": "Ali"}`
Protobuf	Binaire	Format compact et rapide utilisé par gRPC	Encodage binaire, non lisible humainement

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Formats d’échange de données

Format	Type	Description	Exemple
XML	Texte structuré	Format verbeux, typé, extensible	`<person><name>Ali</name></person>`
JSON	Texte léger	Format clé-valeur, lisible et efficace	`{"name": "Ali"}`
Protobuf	Binaire	Format compact et rapide utilisé par gRPC	Encodage binaire, non lisible humainement

Introduction & SOAP Web Services

1. Concepts fondamentaux

◆ Formats d'échange de données

Comparaison :

Critère	XML	JSON	Protobuf
Lisibilité	★★★	★★★★★	★
Poids des messages	⚖️ Lourd	⚖️ Léger	⚖️ Très léger
Performance	⚡ Moyenne	⚡ Rapide	⚡⚡ Très rapide
Typage fort	Oui	Non	Oui
Standard courant	SOAP	REST	gRPC

Introduction & SOAP Web Services

1. Concepts fondamentaux

- **Différence entre SOAP, REST, GraphQL et gRPC**

Critère	SOAP	REST	GraphQL	gRPC
Protocole	HTTP, SMTP	HTTP	HTTP	HTTP/2
Format	XML	JSON	JSON	Protobuf
Type	Contrat rigide (WSDL)	Flexible	Requêtes dynamiques	Contrat rigide (.proto)
Performance	Moyenne	Bonne	Très bonne	Excellente

Introduction & SOAP Web Services

1. Concepts fondamentaux

- **Différence entre SOAP, REST, GraphQL et gRPC**

Critère	SOAP	REST	GraphQL	gRPC
Interopérabilité	Très élevée	Très élevée	Bonne	Moyenne (nécessite support Protobuf)
Streaming	Non	Non	Oui (via WebSockets)	Oui (natif bidirectionnel)
Usage typique	Services bancaires, gouvernementaux	API web classiques	API front modernes	Microservices internes, performance

Introduction & SOAP Web Services

1. Concepts fondamentaux

SOAP : ancien, structuré, robuste → pour systèmes critiques
REST : universel, simple, lisible → pour API web
GraphQL: flexible, orienté client → pour front-end moderne
gRPC : rapide, binaire, typé → pour communication interservices

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction à gRPC

Introduction à gRPC

Qu’est-ce que gRPC ?

- **gRPC (Google Remote Procedure Call)** est un **framework RPC** (Remote Procedure Call) développé par **Google**.
- Il permet à deux programmes de **communiquer entre eux via le réseau** en appelant des **méthodes distantes** comme si elles étaient locales.

Introduction à gRPC

Qu’est-ce que gRPC ?

Objectif :

Plutôt que d’envoyer des requêtes HTTP avec des JSON (comme dans REST),
👉 tu appelles directement une **fonction distante** avec des **objets typés** (Protobuf).

Exemple :

```
// REST
GET /api/users/42 → {"id":42, "name":"Ali"}

// gRPC
UserService.getUser(UserRequest{id:42}) → UserResponse{id:42, name:"Ali"}
```

➡️ Pour le développeur, ça ressemble à un appel de méthode **locale**, mais la communication passe par le réseau.

Introduction à gRPC

Qu'est-ce que gRPC ?

Composants principaux :

Élément	Rôle
Service	Définit les méthodes RPC disponibles
Message	Définit les structures de données échangées
.proto	Fichier de définition du contrat (service + messages)
Stub	Code client généré automatiquement à partir du .proto
Skeleton (Impl)	Code serveur généré automatiquement
Channel	Connexion réseau gérée par gRPC
StreamObserver	Interface Java utilisée pour gérer les flux de données (streaming)

Introduction à gRPC

Comment ça marche en interne ?

1 Étape 1 : Le contrat (.proto)

C’est la base du système : un **fichier de description** partagé entre le client et le serveur.
Il définit :

- les **messages échangés**,
- les **services** (méthodes RPC),
- et les **options de génération de code**.

Introduction à gRPC

Comment ça marche en interne ?

1️⃣ Étape 1 : Le contrat (.proto)

Exemple :

```
syntax = "proto3";
package user;

option java_multiple_files = true;
option java_package = "com.example.user";
option java_outer_classname = "UserProto";

service UserService {
  rpc GetUser (UserRequest) returns (UserResponse);
}
message UserRequest {
  int32 id = 1;
}
message UserResponse {
  int32 id = 1;
  string name = 2;
  string email = 3;
}
```

Introduction à gRPC

Comment ça marche en interne ?

2️⃣ Étape 2 : Compilation Protobuf

Le compilateur **protoc** transforme le `.proto` en classes pour différents langages :

```
protoc --java_out=. --grpc-java_out=. user.proto
```

→ Génère :

```
UserServiceGrpc.java
UserRequest.java
UserResponse.java
```

Introduction à gRPC

Comment ça marche en interne ?

3 Étape 3 : Implémentation du serveur

Côté serveur, tu **implémentes les méthodes RPC** définies dans ton fichier `.proto`.

```
@GrpcService
public class UserServiceImpl extends UserServiceGrpc.UserServiceImplBase {

  @Override
  public void getUser(UserRequest request, StreamObserver<UserResponse> responseObserver) {
      UserResponse response = UserResponse.newBuilder()
          .setId(request.getId())
          .setName("Faïza Guène")
          .setEmail("faiza.guene@example.com")
          .build();
      responseObserver.onNext(response);
      responseObserver.onCompleted();
  }
}
```

Introduction à gRPC

Comment ça marche en interne ?

Étape 4 – Implémentation client

```
ManagedChannel channel = ManagedChannelBuilder
    .forAddress("localhost", 6565)
    .usePlaintext()
    .build();

UserServiceGrpc.UserServiceBlockingStub stub =
    UserServiceGrpc.newBlockingStub(channel);

UserRequest req = UserRequest.newBuilder().setId(1).build();
UserResponse res = stub.getUser(req);

System.out.println(res.getName());
channel.shutdown();
```

Introduction à gRPC

Comment ça marche en interne ?

Étape 5 – Transmission (interne)

1. Le client sérialise `UserRequest` en **Protobuf binaire**.
2. Le message est transmis via **HTTP/2** (connexion persistante).
3. Le serveur désérialise, exécute la méthode, renvoie la réponse.
4. Le client reçoit et désérialise en `UserResponse`.

Introduction à gRPC

HTTP/2 et performance

gRPC exploite **HTTP/2** pour :

- le **multiplexage** : plusieurs flux sur une seule connexion TCP,
- la **compression des entêtes**,
- le **streaming bidirectionnel**,
- la **sécurité TLS native**.

👉 Résultat :
gRPC est **5 à 10 fois plus rapide** que REST (grâce à Protobuf + HTTP/2).

Introduction à gRPC

Types d’appels RPC

Type	Description	Exemple d’usage
Unary	1 requête → 1 réponse	Authentification
Server streaming	1 requête → plusieurs réponses	Envoi d’un flux de données
Client streaming	plusieurs requêtes → 1 réponse	Upload de fichiers
Bidirectional streaming	plusieurs requêtes ↔ plusieurs réponses	Chat, streaming temps réel

Introduction à gRPC

Syntaxe complète du fichier `.proto`

En-tête

```
syntax = "proto3"; // Version du langage Protobuf
package bank;      // Namespace logique
```

Options de compilation (annotations)

```
option java_multiple_files = true;
option java_package = "com.example.bank";
option java_outer_classname = "BankProto";
```

> Ces options indiquent au compilateur comment générer les classes.

Introduction à gRPC

Syntaxe complète du fichier `.proto`

◆ Messages

```
message Account {
  int32 id = 1;
  double balance = 2;
  string owner = 3;
  repeated string tags = 4; // Liste
}
```

Élément	Signification
`message`	Déclare une structure de données
`int32`, `double`, `string`	Types primitifs
`repeated`	Collection (liste)
`= 1`	Numéro unique du champ (pour la sérialisation binaire)

Introduction à gRPC

Syntaxe complète du fichier `.proto`

- **Types supportés**

Type	Java	Description
`int32`, `int64`	int, long	Entier
`float`, `double`	float, double	Décimal
`bool`	boolean	Booléen
`string`	String	Texte
`bytes`	ByteString	Binaire
`map<K,V>`	Map<K,V>	Dictionnaire
`repeated`	List	Liste d'éléments

Introduction à gRPC

Syntaxe complète du fichier `.proto`

- **Champs imbriqués et messages internes**

```
message BankAccount {
  int32 id = 1;
  double balance = 2;

  message Transaction {
    int32 id = 1;
    double amount = 2;
  }

  repeated Transaction history = 3;
}
```

Introduction à gRPC

Syntaxe complète du fichier `.proto`

🔹 Numérotation des champs

- Chaque champ a un identifiant **unique et permanent**.
- Les numéros **1–15** sont plus rapides à encoder (1 octet).
- ⚠️ Ne jamais réutiliser un ancien numéro supprimé.

```
reserved 3, 5;
reserved "old_field";
```

Introduction à gRPC

Syntaxe complète du fichier `.proto`

Imports et modularisation

```
import "google/protobuf/empty.proto";
import "google/protobuf/timestamp.proto";
```

Exemple :

```
message LogEntry {
  string message = 1;
  google.protobuf.Timestamp created_at = 2;
}
```

Introduction à gRPC

Syntaxe complète du fichier `.proto`

Imports et modularisation

```
import "google/protobuf/empty.proto";
import "google/protobuf/timestamp.proto";
```

Exemple :

```
message LogEntry {
  string message = 1;
  google.protobuf.Timestamp created_at = 2;
}
```

Introduction à gRPC

Syntaxe complète du fichier `.proto`

- **Services RPC**

```
service BankService {
  rpc GetAccountBalance (BalanceCheckRequest) returns (AccountBalance);
  rpc Withdraw (WithdrawRequest) returns (stream Money);
  rpc Deposit (stream DepositRequest) returns (AccountBalance);
  rpc Transfer (stream TransferRequest) returns (stream TransferResponse);
}
```

Mot-clé	Description
`rpc`	Méthode distante
`stream`	Active le streaming
`returns`	Type de réponse
`service`	Conteneur de RPCs

Introduction à gRPC

Syntaxe complète du fichier `.proto`

🔹 Bonnes pratiques de nommage

- 🔠 Nom des messages : **PascalCase** (`UserRequest`, `BankAccount`)
- 🔡 Nom des champs : **snake_case** (`account_number`, `created_at`)
- 🔷 Nom des services : **PascalCase** (`BankService`)
- 🧩 Nom des méthodes : **CamelCase** (`GetBalance`, `WithdrawMoney`)

Introduction à gRPC

Protobuf – Types avancés et bien connus

Google fournit des **types standards** réutilisables :

Type	Description
`google.protobuf.Empty`	Message vide (équivalent `void`)
`google.protobuf.Timestamp`	Représente une date/heure
`google.protobuf.Duration`	Durée
`google.protobuf.Any`	Type générique (polymorphisme)

Exemple :

```
import "google/protobuf/timestamp.proto";

message Transaction {
  double amount = 1;
  google.protobuf.Timestamp date = 2;
}
```

Introduction à gRPC

Les Stubs (clients)

Après compilation du `.proto`, le code généré contient 3 types de stubs :

Type	Description
BlockingStub	Appel synchrone
FutureStub	Asynchrone (basé sur Future)
AsyncStub	Basé sur callback `StreamObserver`

Exemple :

```
BankServiceGrpc.BankServiceBlockingStub stub = BankServiceGrpc.newBlockingStub(channel);
AccountBalance res = stub.getAccountBalance(req);
```

Introduction à gRPC

Sécurité dans gRPC

Niveau	Mécanisme
Transport	TLS / SSL intégré à HTTP/2
Authentification	Intercepteurs (`ServerInterceptor`, `ClientInterceptor`)
Autorisation	Jetons JWT ou métadonnées (`Metadata`)

Exemple :

```
Metadata md = new Metadata();
md.put(Metadata.Key.of("authorization", Metadata.ASCII_STRING_MARSHALLER), "Bearer <token>");
```

Introduction à gRPC

Écosystème gRPC

Outil	Rôle
protoc	Compilateur Protobuf
grpcurl	Tester les services depuis le terminal
BloomRPC / Insomnia	Interface graphique pour gRPC
grpc-gateway	Convertit une API gRPC en REST
Spring Boot gRPC Starter	Intégration facile avec `@GrpcService`, `@GrpcClient`

Introduction à gRPC

Avantages majeurs

Avantage	Détail
⚡ **Performance**	Encodage binaire rapide + HTTP/2
🧠 **Typage fort**	Contrat partagé via `.proto`
🌍 **Interopérabilité**	Multi-langages (Java, Go, Python…)
🔄 **Streaming**	Bidirectionnel natif
💬 **Évolutif**	Ajout de champs sans casser l'existant

Introduction à gRPC

Limites

Limite	Description
🌐 Pas accessible via navigateur	Pas de support HTTP/1.1
🧩 Moins intuitif que REST	Format binaire moins lisible
💬 Versioning manuel	Gestion des `.proto` requise
🔧 Outils de test limités	grpcurl ou BloomRPC nécessaires

Introduction à gRPC

Cas d’usage typiques

Cas	Pourquoi gRPC ?
Microservices internes	Performance + typage
IoT / streaming temps réel	Support du flux bidirectionnel
Systèmes distribués	Stabilité et scalabilité
Backend multi-langages	Java ↔ Python ↔ Go
Migration REST → gRPC	Gain de bande passante