



# REST, gRPC, GraphQL, Webhooks

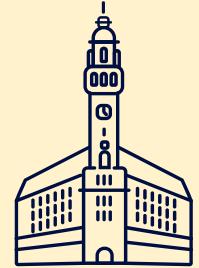
Dans quelles situations ?

# \$ whoami

François Delbrayelle (@fdelbrayelle)   

Cht'Ippon chez **Ippon** Lille depuis 2019

- **Développeur Full-Stack**
- **Tech Lead**
- **Architecte Solutions** 😊
- **Formateur**
- Référent [blog.ippon.fr](http://blog.ippon.fr)







27/9/21  
LILLE

CHALEUREUX!

ARTHUR, TAS  
GAGNE!  
VAN  
STRATEGY  
OFFICER

ACCOMPAGNEMENT

ON FOUT DES BAFFES!

CRAFT

ADEO

NEOLIA

VEROY  
MERLIN

EDF

NEXITY

CAMPÉS  
LAFAYETTE

FORMATION

SITE WEB

CONTRIBUTION

MEEETUP

DENFEST

ACCUEIL  
SEGAIGRES

COMMUNAUTES

CLOUD  
NORD

JAVA

PARAGE  
DE  
DESIGN

CONNAISSANCES

POURQUOI?

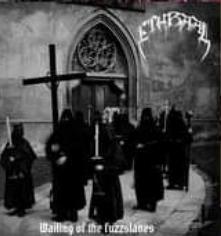
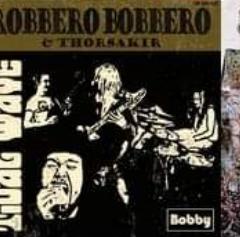
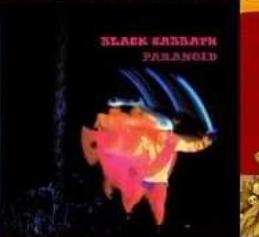
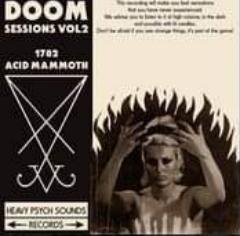
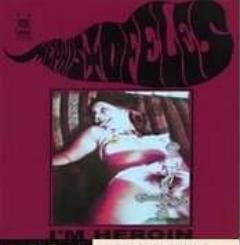












# Disclaimer

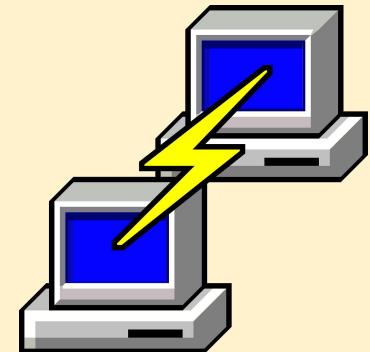
-  Pas de mise en avant d'une techno par rapport aux autres
-  Pas une présentation détaillée de chaque techno
-  Pas d'ordre logique entre les technos abordées
-  Pas de biais cognitifs (au moins essayer)
-  Pas abordé : *pub/sub* et *queues*
-  Pas de *silver bullets*





# Faire communiquer des acteurs

📜 Une brève histoire de la communication client / serveur



# Communication Client / Serveur

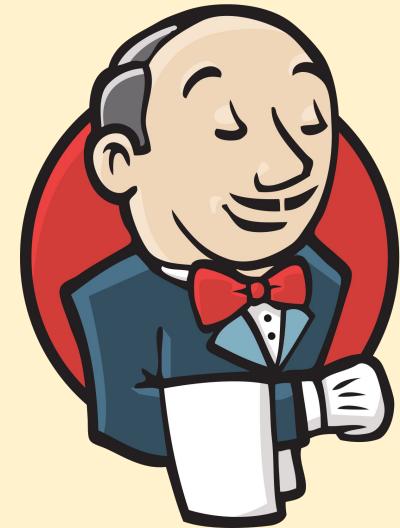
**Client / Serveur** : architecture réseau

**Sockets**

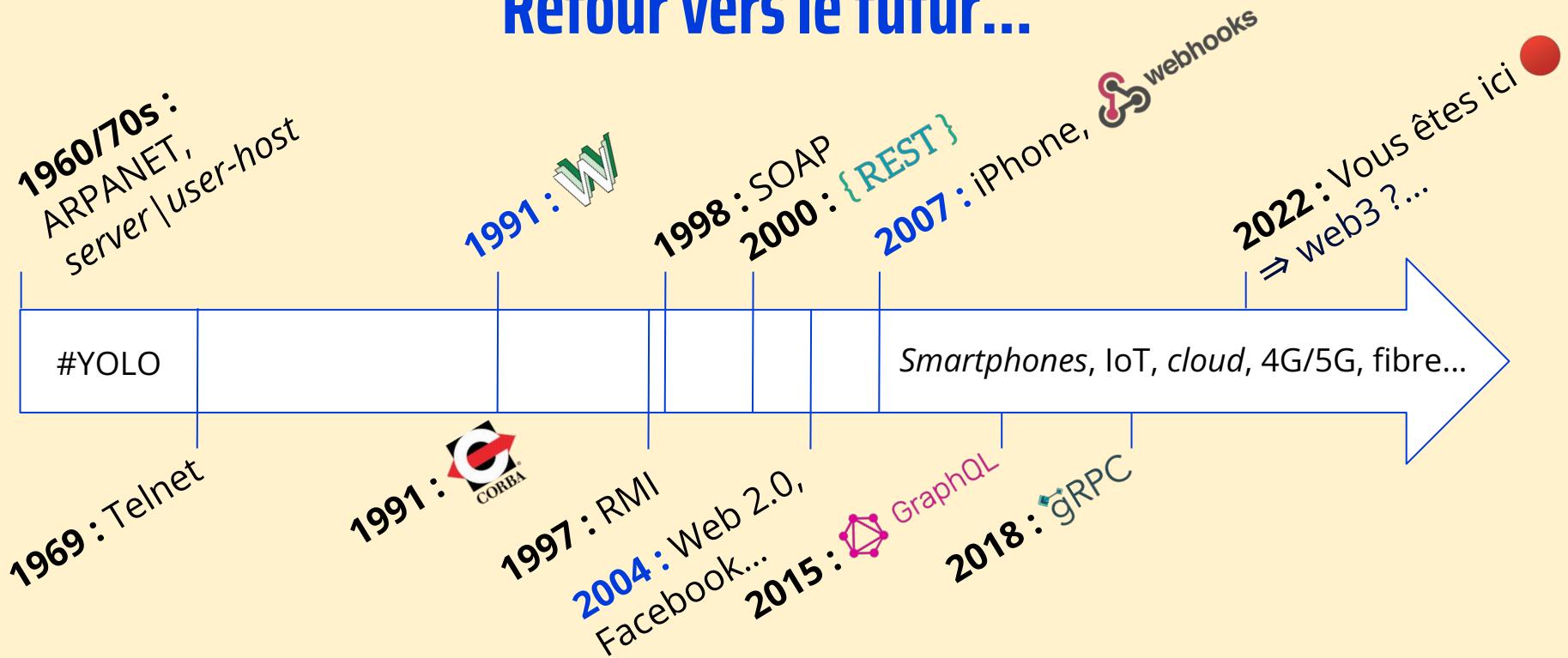
**TCP/IP** : protocole de transport commun

**Communication inter-processus (IPC)**

**Besoins non fonctionnels**



# Retour vers le futur...



```
<?xml version="1.0"?>

<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

    <soap:Header>
    ...
    </soap:Header>

    <soap:Body>
    ...
        <soap:Fault>
        ...
        </soap:Fault>
    </soap:Body>

    </soap:Envelope>
```



**2010...**



**REST IS THE NEW SOAP**

**2015...**

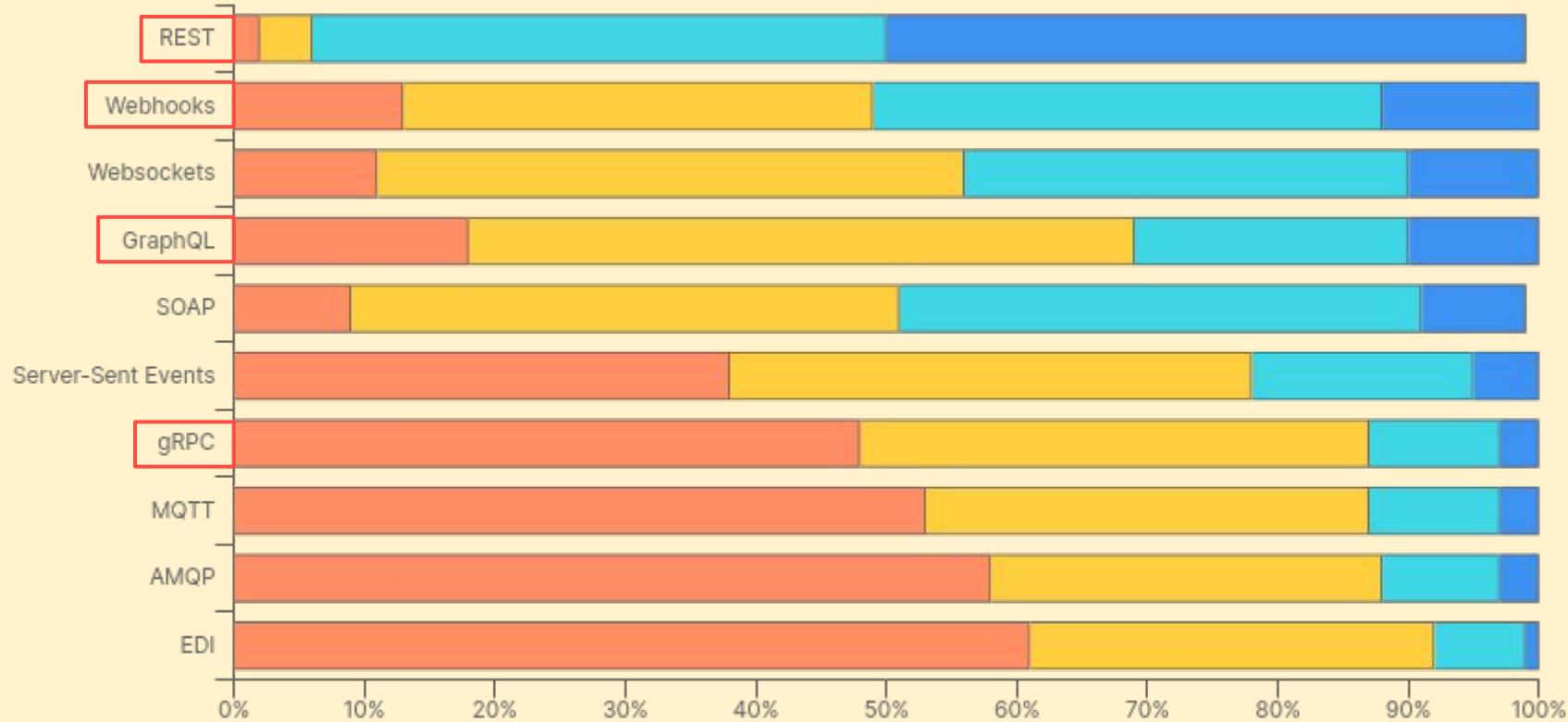


**GRAPHQL IS THE NEW REST  
(IT'S MORE VERSATILE!)**

**2018...**



**GRPC IS THE NEW GRAPHQL  
(IT'S FASTER!)**



Source : Postman's 2021 State of the API Report (<https://www.postman.com/state-of-api>)

Légende : En bleu = utilise et adore - En cyan = utilise - En jaune = connaît mais n'utilise pas - En orange = jamais entendu



C'EST UNE BONNE SITUATION ÇA ?

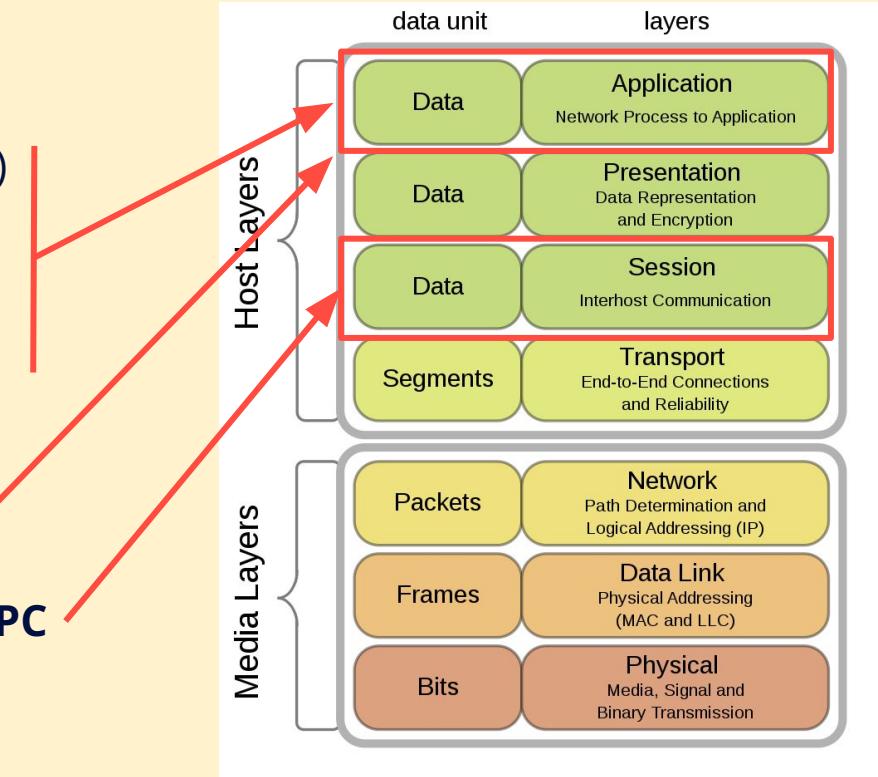
# Différentes typologies

**REST** : architecture (souvent avec **HTTP**)

**GraphQL** : langage (**HTTP POST**)

**Webhooks** : **HTTP POST**

**gRPC** : framework basé sur **HTTP/2 et RPC**



Source : Wikipédia

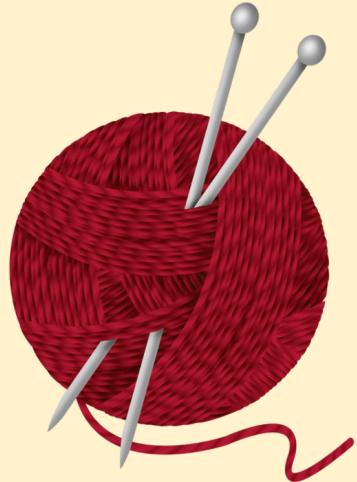


# Produit fil rouge

📦 Extrait du SI d'un grand compte *retail* avec :

*Applications web / mobiles*

*PIM (Product Information Management)*

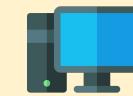




E-Commerce



Mobile



Caisse



Magasin



Marketplace



API Product



API Price



API Stock



API Asset



# API - C'est quoi ?

*Application Programming Interface*

**Client / Serveur**

**Échange d'informations**

**Utilise des standards**

**Cycle propre**



# { REST }



# C'est quoi ?

*REpresentational State Transfer*

**Architecture** pour systèmes distribués à hypermédias

Pensée par **Roy Fielding** dès **2000** dans sa thèse :

*Architectural Styles and the Design of Network-based Software Architectures*

Peut reposer sur le **protocole HTTP (RFC 2616)**



# Concepts clés

## Ressource

- Entité, modèle, concept identifiable, (im)matérielle
- URI (*Uniform Resource Identifier*) = identité
- Service REST associe des URLs à URIs

**Verbe HTTP** : GET - PUT - DELETE (idempotents) - POST - PATCH...

**Représentation** de l'état de la ressource :

Format d'échange avec (dé)sérialisation (JSON, XML, HTML, texte brut...)



# Contraintes architecturales

Séparation **client / serveur**

Approche **sans état** (*stateless*)

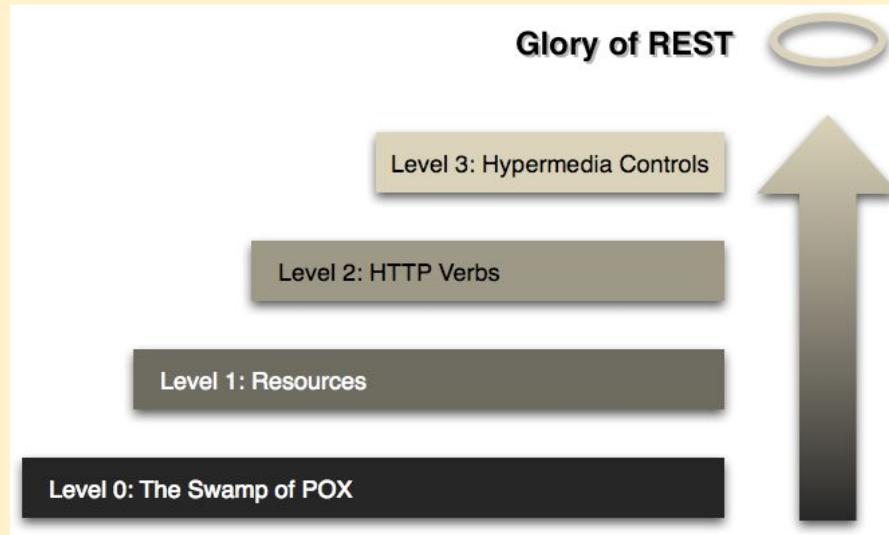
Mise en **cache** (GET et POST)

Architecture en **couches** (*layered system*)

**Interface uniforme**



# Modèle de maturité de Richardson



Source : <https://www.martinfowler.com/articles/richardsonMaturityModel.html>





# Exposer de la valeur métier avec les hypermédias

Focus métier

Attention aux APIs anémiques

Ne pas exposer directement la base de données

⇒ Utiliser les **hypermédias** (**HATEOAS** = *Hypermedia as the Engine of Application State*)

# Exemple d'appel

```
GET /characters/42 HTTP/1.1  
Host: api.h2g2.com  
Content-Type: application/json
```



```
{  
  "id": 42,  
  "name": "Arthur Dent",  
  "links": [  
    {  
      "href": "42/messages",  
      "rel": "messages",  
      "type": "GET"  
    }  
  ]  
}
```



# { REST }

## Avantages

- Basé sur des standards du web
- Support universel (navigateurs)
- Beaucoup de cas d'utilisation
- Bien connu par les développeurs
- Modélisation d'un domaine
- CRUD
- HATEOAS

## Inconvénients

- *Over|underfetching*
- Tendance à faire du 1 pour 1
- HTTP/1.1 limité parfois
- Taille des *payloads*





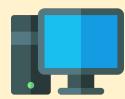
# Pourquoi choisir REST ?

## Motivations :

- Support des navigateurs web
- Exposition de données / comportements métier
- *Headless (front-end mis à jour plus souvent que back-end)*
- Actions relativement simples (CRUD)

## Cas d'utilisation :

- *Resource-driven applications*
- Applications web
- Applications mobiles
- Microservices



E-Commerce



Mobile



Caisse



Magasin



Marketplace



API Product



API Price

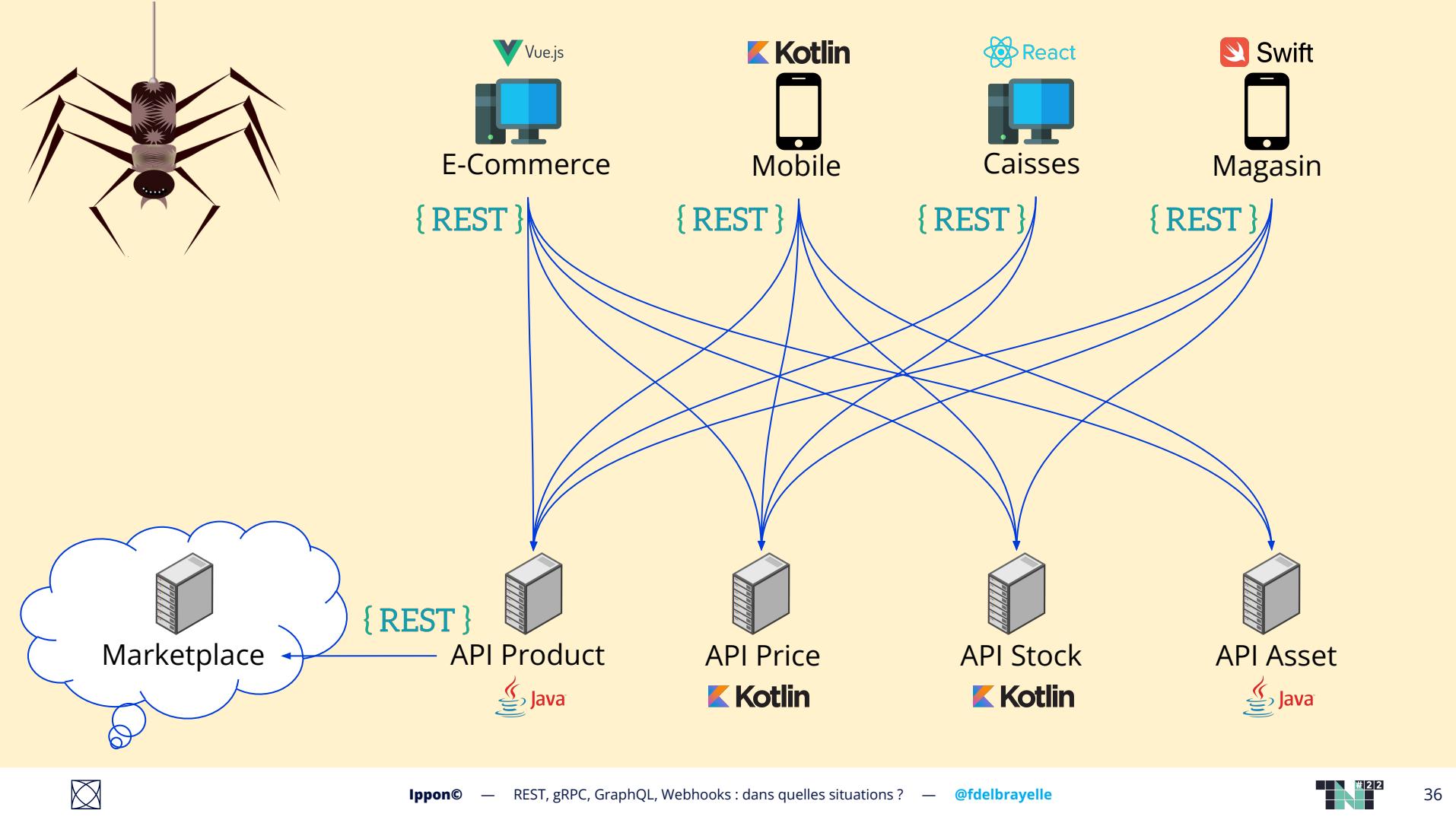


API Stock



API Asset







# C'est quoi ?

Framework créé par **Google** en **2016**

Rendu **open-source** (licence Apache 2.0) : <https://github.com/grpc/grpc>

Intégré à la **fondation CNCF**

Basé sur le **protocole RPC (*Remote Procedure Call*)**

**3 implémentations principales** : GRPC-JAVA, GRPC-GO et GRPC-C (dont découlent C++, Python, Ruby, Objective C, PHP et C#)

**En Java** : <https://github.com/grpc/grpc-java> (génération avec plugin Maven/Gradle)





# HTTP/2

**Standard depuis 2015 (RFC 7540)**

**Utilisé pour le transport**

**Streams** formés de **paquets** (*frames* avec format spécifique et taille)

**Multiplexing** : Plusieurs *frames* dans une requête / connexion

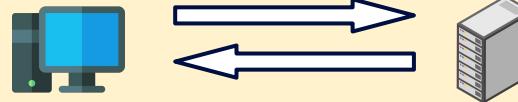
**< 46,9 % des sites l'utilisent en janvier 2022**

<https://w3techs.com/technologies/details/ce-http2>

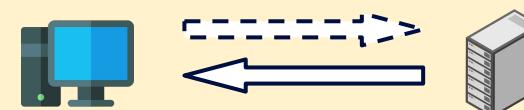


# Modes de communication

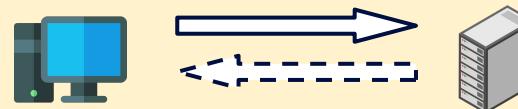
*Unary*



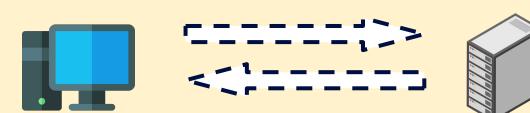
*Client Streaming*



*Server Streaming*



*Bidirectional Streaming*





# Protocol Buffers

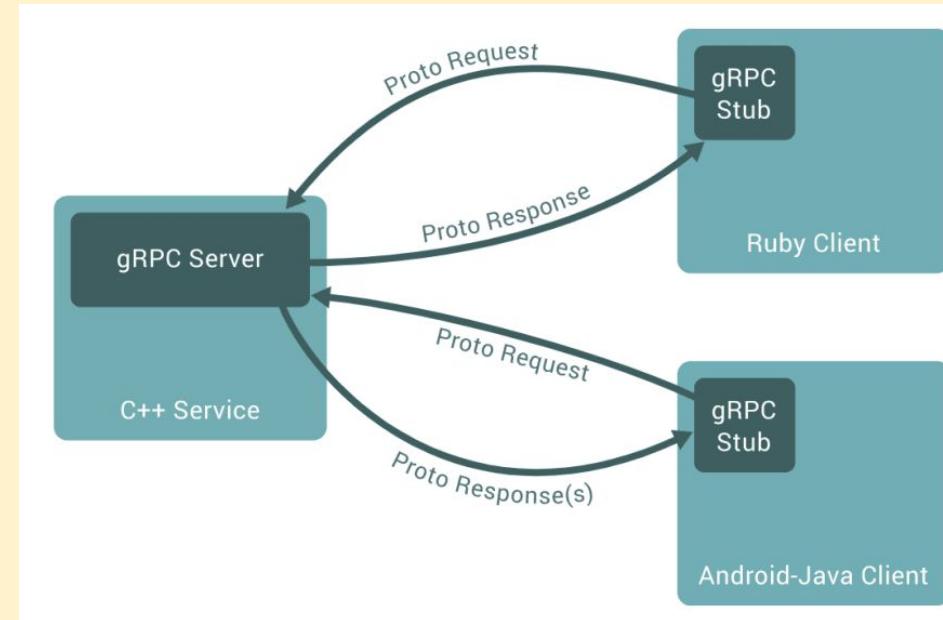
Langage de description d'interface (IDL) pour définir **messages** et **services**  
**Agnostique**

Interface indépendante de l'implémentation

Données binaires et (dé)sérialisation (plus efficace que JSON) :

- moins de bande passante (binaire = petits *payloads*)
- plus proche de la représentation machine des données
- moins consommateur de CPU

# Exemple client / serveur



Source du schéma :

<https://grpc.io/docs/what-is-grpc/introduction/>



```
// todo.proto
syntax = "proto3";

package todoPackage;

service Todo {
    rpc createTodo(TodoItem) returns (TodoItem);
    rpc readTodos(voidNoParam) returns (TodoItems);
    rpc readTodosStream(voidNoParam) returns (stream TodoItem);
}

message voidNoParam {}

message TodoItem {
    int32 id = 1;
    string text = 2;
}

message TodoItems {
    repeated TodoItem items = 1;
}
```



```
// server.js

const grpc = require("grpc");
const protoLoader = require("@grpc/proto-loader");
const packageDef = protoLoader.loadSync("todo.proto", {});
const grpcObject = grpc.loadPackageDefinition(packageDef);
const todoPackage = grpcObject.todoPackage;

const server = new grpc.Server();
server.bind("0.0.0.0:40000", grpc.ServerCredentials.createInsecure());
server.addService(todoPackage.Todo.service,
{
    "createTodo": createTodo,
    "readTodos": readTodos,
    "readTodosStream": readTodosStream
});
server.start();

const todos = [];
function createTodo(call, callback) {
    // Impl
}
```



```
// client.js
const client = new todoPackage.Todo("localhost:40000",
  grpc.credentials.createInsecure());

const text = process.argv[2];
client.createTodo({
  "id": -1,
  "text": text
}, (err, response) => {
  console.log("Received from server " + JSON.stringify(response))
});
```





# CRUD (REST) vs Actions (RPC)

POST /users/501/messages HTTP/1.1

Host: api.example.com

Content-Type: application/json



POST /SendUserMessage HTTP/1.1

Host: api.example.com

Content-Type: application/json

{"message": "Hello!"}

{"userId": 501, "message": "Hello!"}



## Avantages

- HTTP/2 (performances, *streaming*...)
- ProtoBuf (*payloads* légers...)
- Une librairie pour les gouverner tous
- *Feedback* progressif
- Actions
- Procédures plus complexes

## Inconvénients

- Schéma (pas pour tout le monde)
- ProtoBuf difficile à lire sur le réseau
- Courbe d'apprentissage
- Pas (encore) prêt pour le web



	{ REST }	
<b>Protocoles</b>	<b>HTTP/1.1</b> HTTP/2 mais... (requête / réponse)	<b>HTTP/2 + RPC</b> (client / réponse)
<b>Schéma</b>	Non mais...	<b>Oui</b> (.proto)
<b>Formats</b>	<b>JSON   XML   PDF   HTML ...</b>	<b>Binaire (Protocol Buffers)</b>
<b>Versioning</b>	Oui	Oui
<b>Streaming/bidirection</b>	Non mais...	<b>Oui</b>
<b>Support navigateurs</b>	<b>Oui</b>	<b>Non</b> mais gRPC-web + proxy
<b>Génération de code</b>	Swagger / OpenAPI / Postman ...	<b>protoc</b> (natif à gRPC)
<b>Sécurité</b>	JWT / OAuth	SSL/TLS / ALTS / JWT / OAuth





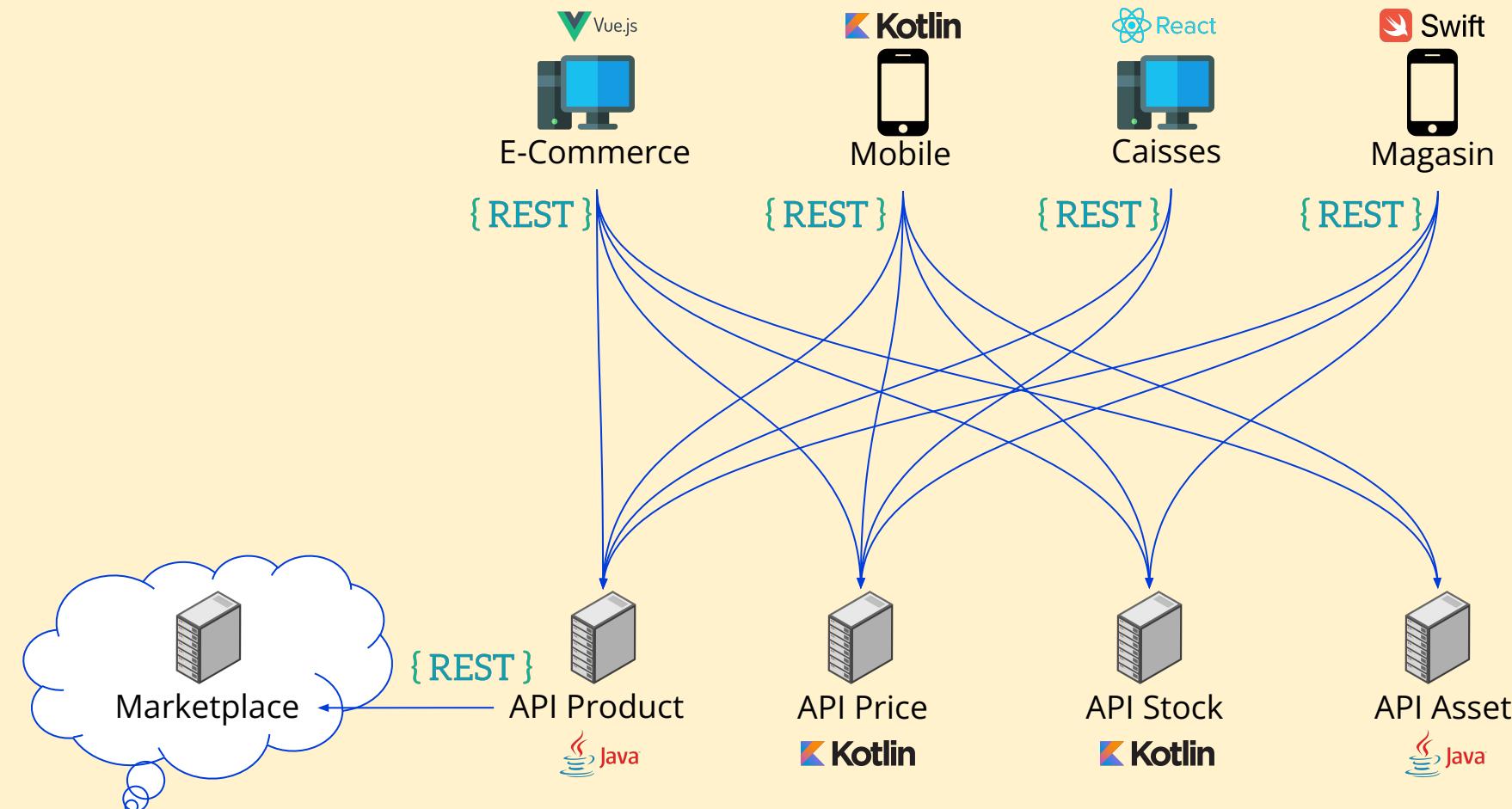
# Pourquoi choisir gRPC ?

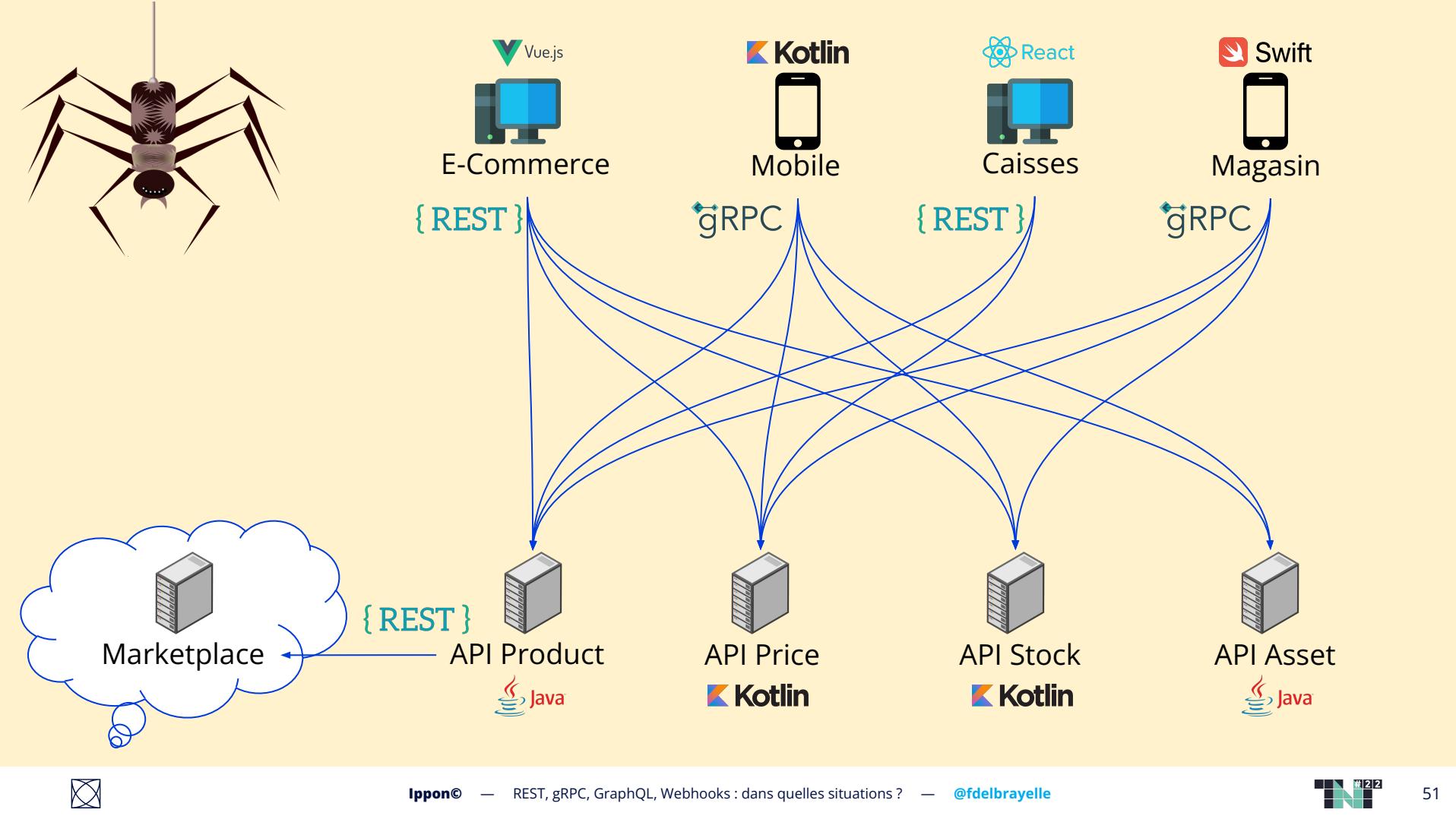
## Motivations :

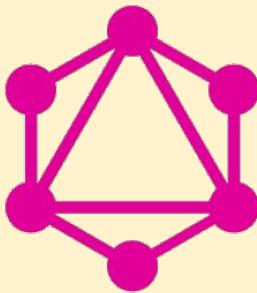
- Performances
- Légereté des *payloads*
- Communication bilatérale et *streaming* en temps-réel
- Actions plus complexes

## Cas d'utilisation :

- Microservices
- Applications mobiles
- *Internet of Things* (IoT)
- Systèmes multi-langages







# GraphQL



# C'est quoi ?

**Langage** créé par **Facebook** en **2012**

Rendu **open-source** en **2015** : <https://github.com/graphql>

Extension .graphql (s)

Réponses au format **JSON**

**Spécification** : <https://spec.graphql.org>

*Peut être transporté par **HTTP POST***

**Client / Serveur** (Apollo Server pour JS/TS, Java / Spring, ...)





# Notions de base

**Schéma** (+ introspection)

**Type**, interface, input, champs, arguments

Type scalaire (`Int`, `Float`, `String`, `Boolean`, `ID`, `scalar` `Date...`), enum, fragment

Opération : **query** (lecture : *read* en parallèle), **mutation** (écriture : *create/update/delete* en série), subscription (streaming) + variables

Directive : `@include | skip (if: $isAwesome)`

*Non-nullable* supporté avec !





# Patterns d'utilisation

**Composite pattern** : Composer plusieurs appels (API, service externe, *in-memory*...) pour agréger des données - API Gateway / BFF (*Back-end for Front-end*)

**Proxy pattern** : Ajouter des fonctionnalités à une vieille API

**Facade pattern** : Simplifier l'appel à une API





# Graphs... Graphs everywhere!

« It's Graphs All the Way Down »

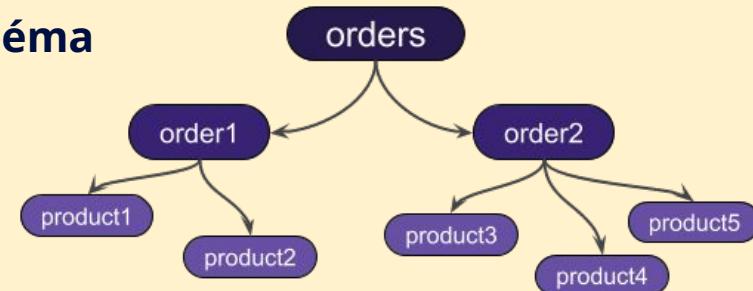
**Domaine métier = graphe** défini par un **schéma**

**Différents types** (de nœuds)

**Connexions** entre les nœuds

**Références de types à types**

<https://graphql.org/learn/thinking-in-graphs>





# Exemple d'appel

```
# Query
{
  orders {
    id
    products {
      product {
        id
        name
      }
      quantity
    }
    amount
  }
}
```



```
{
  "data": {
    "orders": [
      {
        "id": 0,
        "products": [
          {
            "product": {
              "id": 1,
              "name": "Vin de Touraine"
            },
            "quantity": 37
          }
        ],
        "amount": 100.00
      }
    ]
  }
}
```





# GraphQL

## Avantages

- Pas d'*over|underfetching*
- Composition d'appels
- Limitation du périmètre par client
- Documentation autogénérée
- Plupart des langages supportés
- *Operation registry*

## Inconvénients

- Gestion des erreurs
- *Up|download* de fichiers pas natif
- Une implémentation par plateforme
- Courbe d'apprentissage



	{ REST }	
Architecture	<i>Server driven</i>	<i>Client driven</i>
Endpoints	n	1
Schéma	Non mais...	Oui
Opérations	<i>Create, Read, Update, Delete</i>	<i>Query, Mutation, Subscription</i>
Formats	JSON   XML   PDF   HTML ...	GraphQL & JSON
Over underfetching	Oui	Non
Versioning	Oui	Non
Cache	GET et POST	Oui mais...
Sécurité	JWT / OAuth 2.0 / ...	Oui mais...



	gRPC	GraphQL
<b>Architecture</b>	<i>Server driven</i>	<i>Client driven</i>
<b>Endpoints</b>	n	1
<b>Schéma</b>	Oui	Oui
<b>Streaming/bidirection</b>	Oui (client, server, bidirection)	Oui (subscriptions)
<b>Formats</b>	<b>Binaire (Protocol Buffers)</b>	GraphQL & JSON
<b>Over underfetching</b>	<b>Oui</b>	<b>Non</b>
<b>Versioning</b>	Oui	<b>Non</b>
<b>Sécurité</b>	SSL/TLS / ALTS / JWT / OAuth	Oui mais...





# Pourquoi choisir GraphQL ?

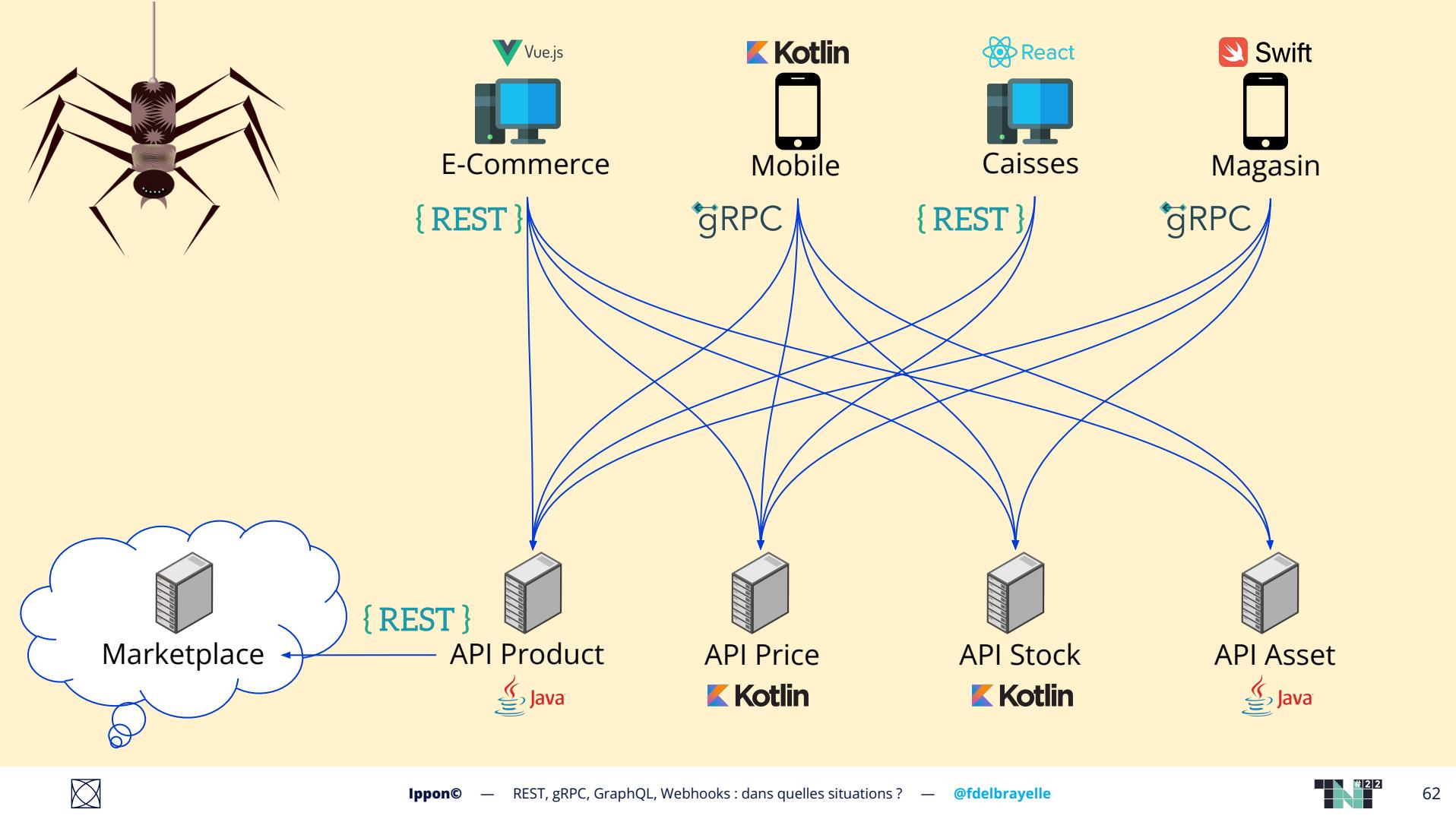
## Motivations :

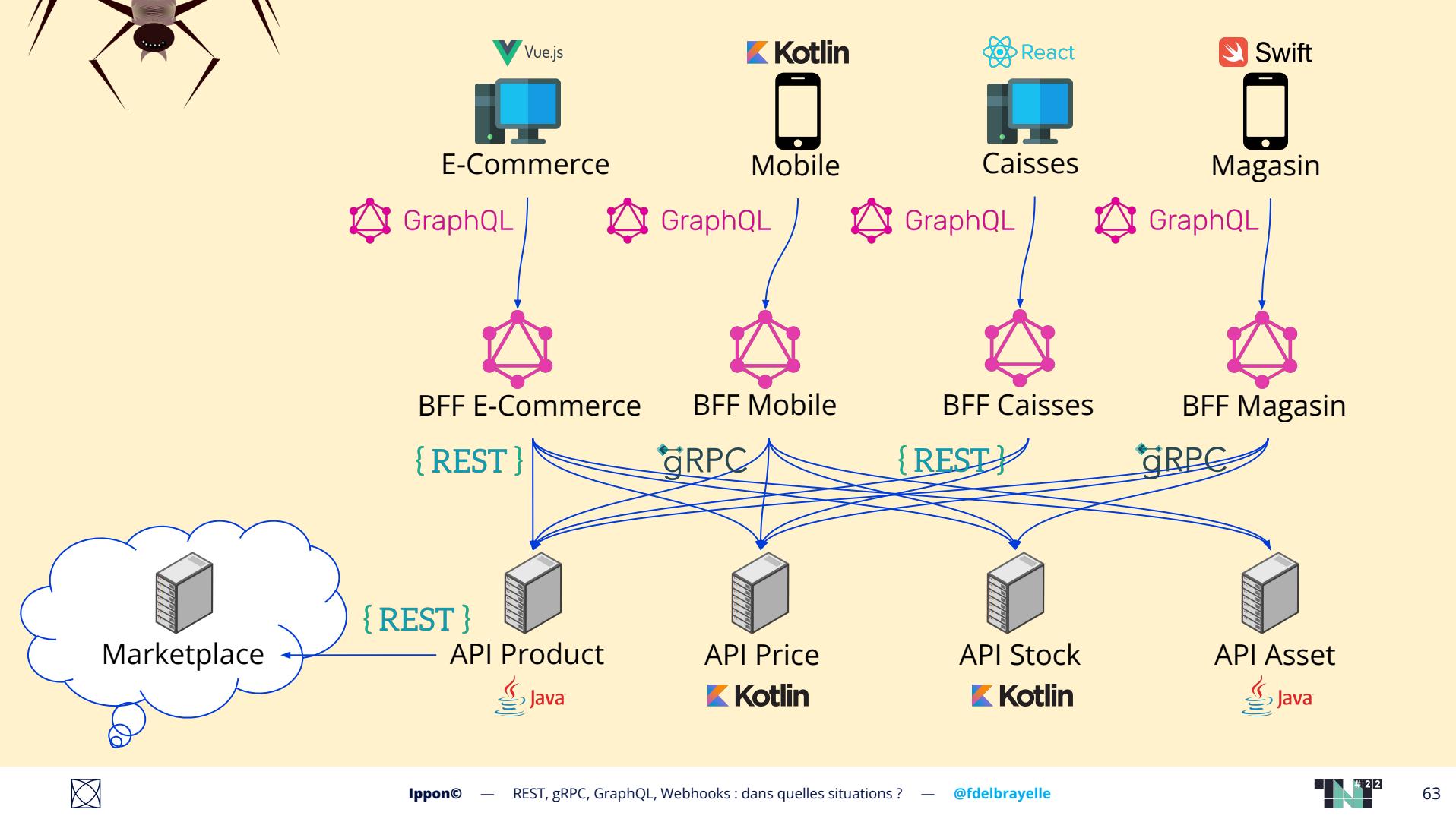
- Contrôle des données récupérées (*client-driven*)
- Utiliser plusieurs sources de données
- Bande passante limitée

## Cas d'utilisation :

- Grosses applications web (*e-commerce...*)
- Applications mobiles
- Applications *legacy* avec vieilles APIs difficiles à maintenir
- Exemples de cas d'utilisation en production (GitHub...)









# C'est quoi ?

Terme inventé en **2007** par **Jeff Lindsay**

**Pas un standard**

**Types d'événements** (*event-driven API*)

**(Dés)inscription**

**Envoi d'information via HTTP POST** (JSON ou XML)  
vers une ***callback URL*** en réponse à un **type d'événement**



# Ne pas confondre...

**Webhooks** : Envoi de message en réaction à un événement serveur à un client via une *callback URL* contenant un secret - Communication unidirectionnelle

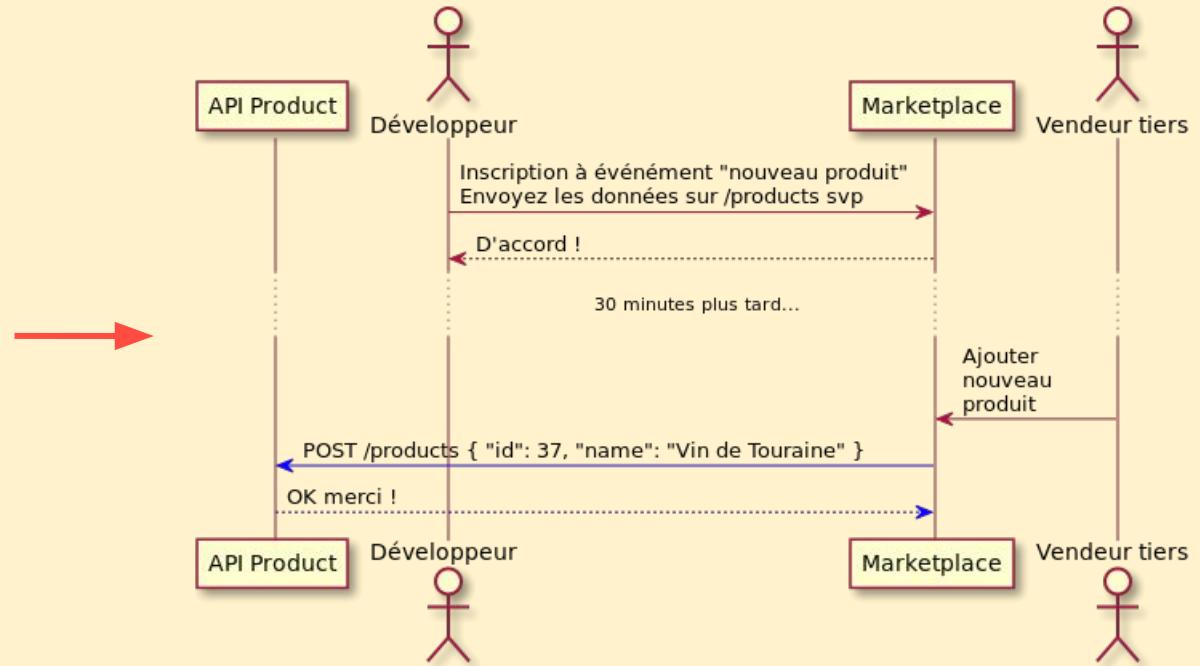
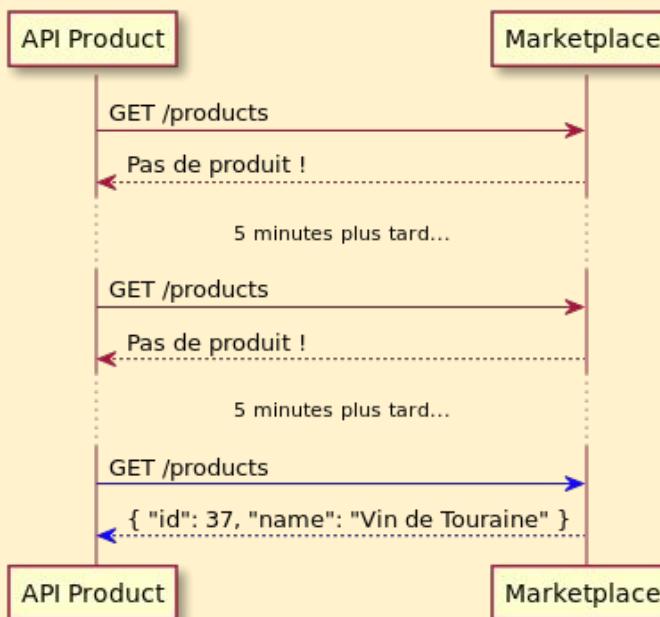
**Websockets** : Communication client / serveur bidirectionnelle à faible latence, client ouvre la connexion, challenges de scalabilité, réduit le nombre de requêtes HTTP

**Server-sent events (SSE)** : Transmission d'événements à un client web depuis le serveur - Communication unidirectionnelle

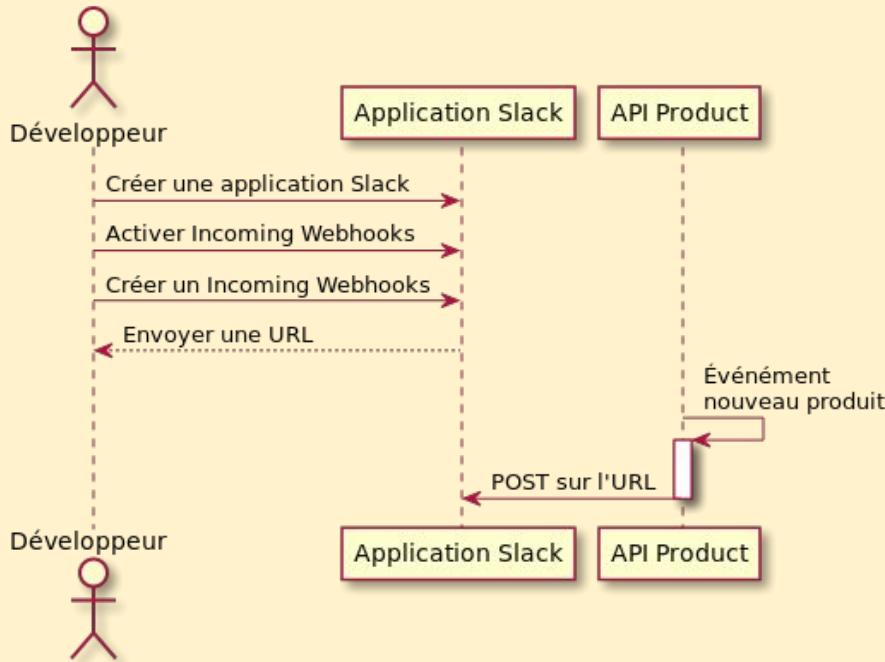
...



# Pull vs Push



# Exemple d'appel



POST

```
https://hooks.slack.com/services/T00000000/B000
0000/xxxxxxxxxxxxxxxxxxxxxxx
Content-type: application/json
{
  "text": "Nouveau produit ajouté !"
}
```



## Avantages

- *Push* plutôt que *pull*
- Envoi de données à un client externe

## Inconvénients

- Client externe doit être abonné
- 1 webhook = 1 événement (bruit)
- Serveur *down* = info perdue
- Moins de contrôle (flux de données)

	{ REST }	 webhooks
<b>Mécanisme</b>	<b><i>pull</i></b> des données ( <i>request based</i> )	<b><i>push</i></b> des données ( <i>event based</i> )
<b>Communication</b>	Bilatérale (⚠ pas streaming)	Unilatérale
<b>Temps réel</b>	<b>Non</b> mais...	<b>Oui</b>
<b>Verbes HTTP</b>	GET   POST   PUT   DELETE   PATCH ...	<b>POST</b>   GET (plus rare)
<b>Formats</b>	<b>JSON   XML   PDF   HTML ...</b>	JSON   XML
<b>Taille des messages</b>	Peut être importante	<b>En général petite</b>





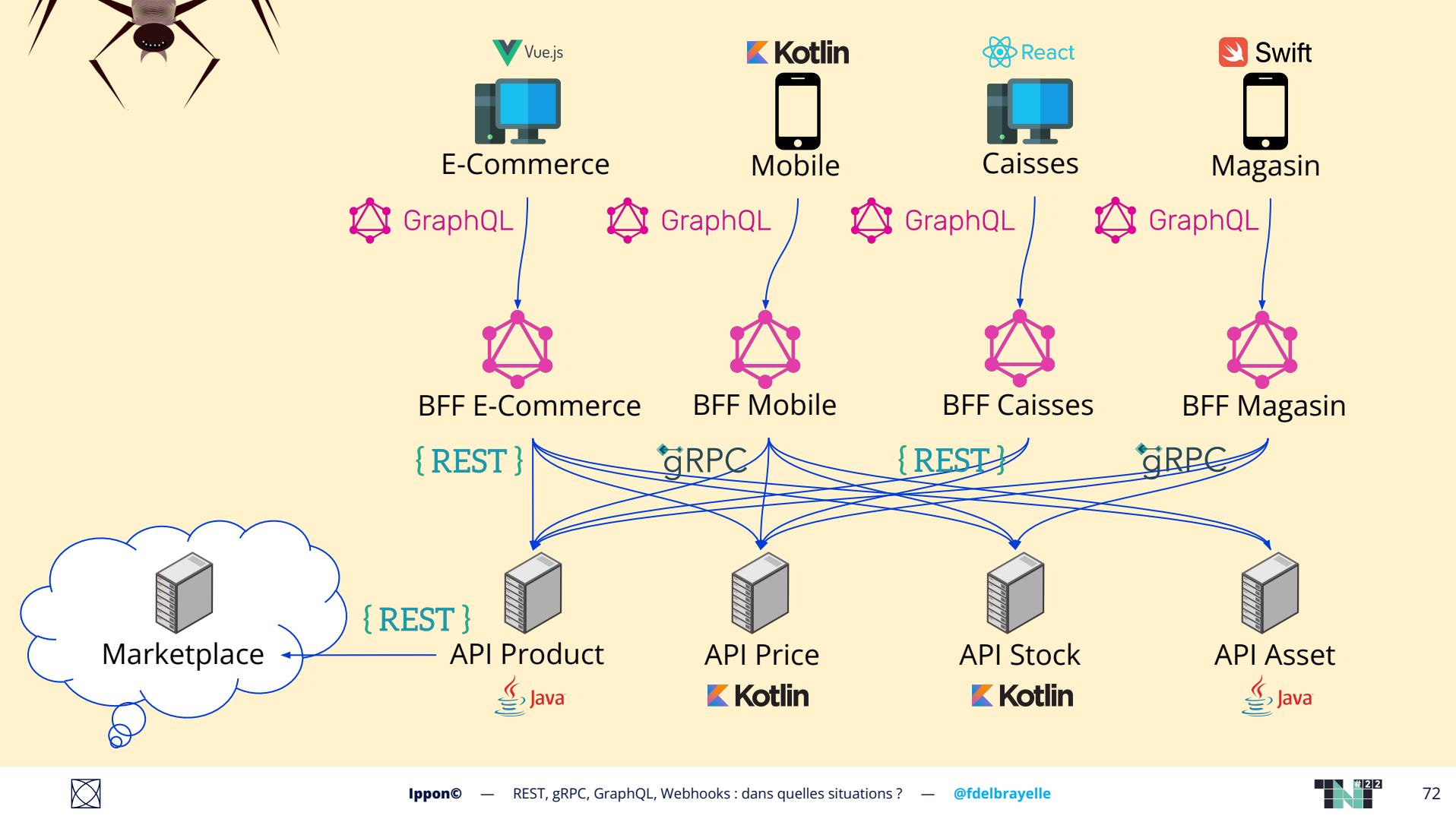
# Pourquoi choisir les webhooks ?

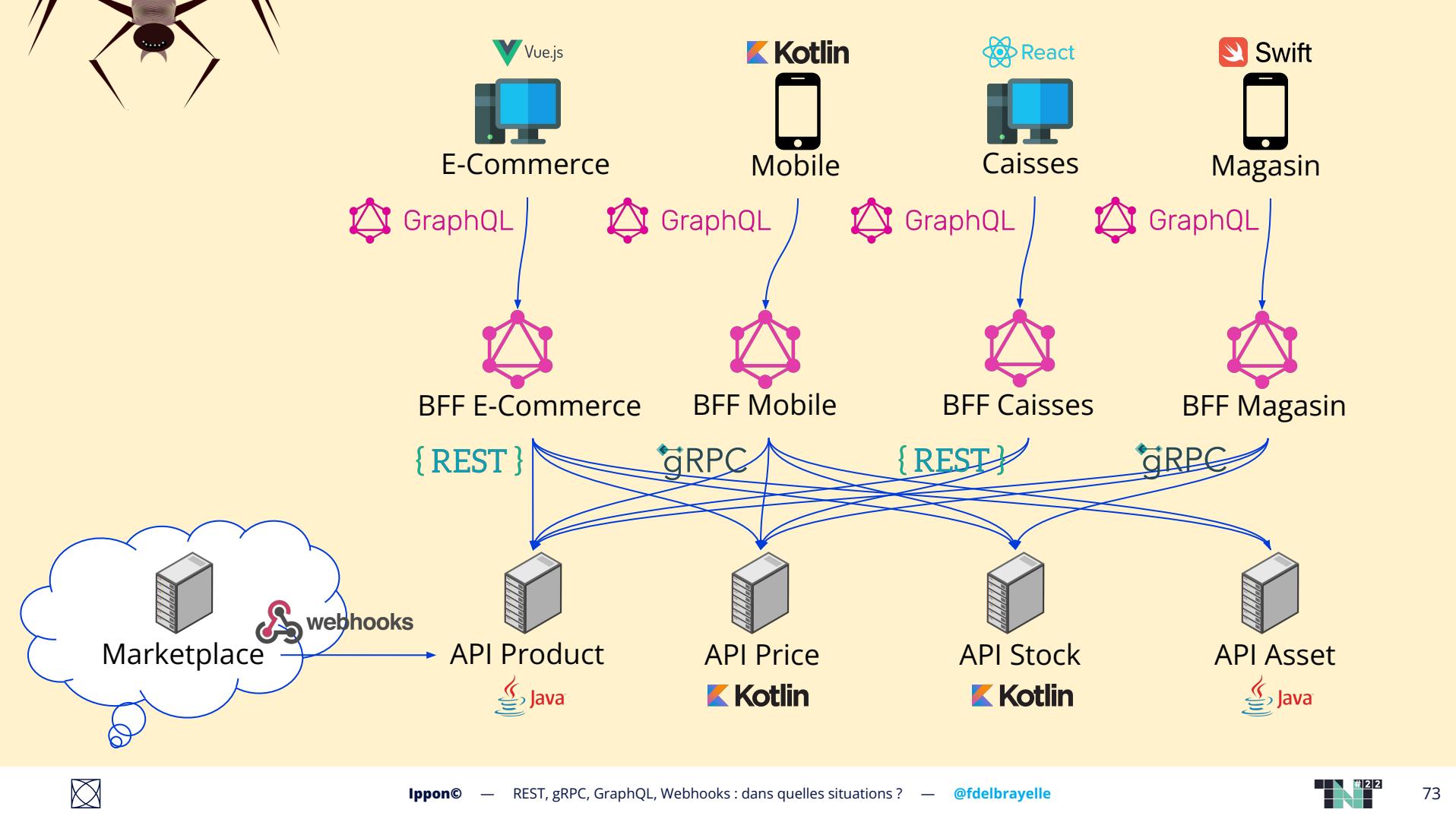
## Motivations :

- Besoin de *push* plutôt que de *pull*
- Déclencher des actions ponctuelles
- Ouvrir une application à des applications externes

## Cas d'utilisation :

- Synchronisation de systèmes en temps réel
- Notifications (alertes, informations...)





# C'est une bonne situation alors ?

🤔 Ça dépend !™ Il n'y a pas de bonne ou de mauvaise situation...





# Diversité des modes de communication

**REST** : Architecture stateless - Transfert de données métier - Standards (HTTP, URI, JSON) - Besoin d'itérations rapides - Web - CRUD - *Up | download* de fichiers

**gRPC** : Framework RPC + HTTP/2 + ProtoBuf - Besoin d'une quantité définie de données ou traitement routinier - Consommateur avec peu de puissance et de ressources (mobile, IoT) ou microservices

**GraphQL** : Langage de requêtes - Composer plusieurs sources de données - Éviter l'*over | underfetching* - *Client-driven*

**Webhooks** : HTTP POST - Mises à jour automatiques de données en réaction à des événements - *Push / pipe*





# Le mot de la fin : rester pragmatique



**Avec les bons outils**, tu travailleras

**Sur les communautés**, tu t'appuieras

**Plusieurs technos en même temps**, tu envisageras

**Que ton équipe maîtrise les technos choisies**, tu t'assureras

**Les besoins métiers avec l'approche appropriée**, tu aligneras

**Que les systèmes en place répondent avec les paramètres donnés**, tu vérifieras





Merci ! A hands clasped together emoji, colored blue and yellow, positioned next to the word "Merci".

Avez-vous des questions ?

**[www.ippontech.fr](http://www.ippontech.fr)**

contact@ippontech.fr — +33 1 46 12 48 48 — @ippontech

