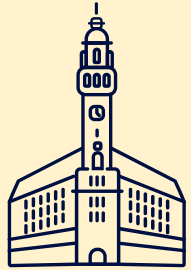

ippon
POSITIVE
TECHNOLOGY
#22
REST, gRPC,
GraphQL, Webhooks
Dans quelles situations ?

$ whoami

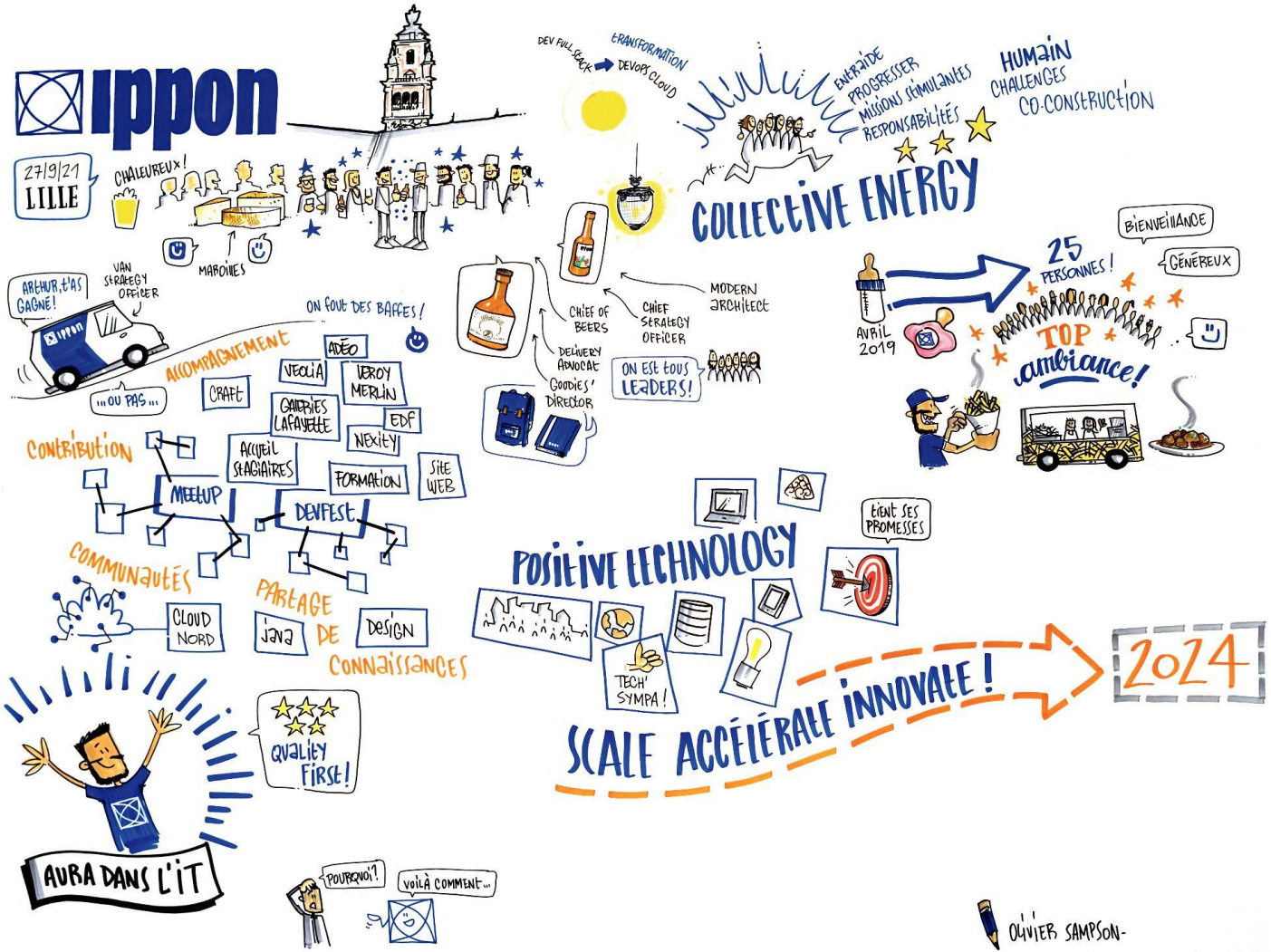


François Delbrayelle (@fdelbrayelle)

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

- **Développeur** *Full-Stack*
- ***Tech Lead***
- **Architecte** Solutions 👶
- **Formateur**
- Référent blog.ippon.fr


OMEGA


ippon
27/9/21
LILLE
CHALEUREUX !
MARQUES
DEV FULL STACK
TRANSFORMATION
DEVOPS CLOUD
ENERGIE
PROGRESSER
MISSIONS STIMULANTES
RESPONSABILITÉS
HUMAIN
CHALLENGES
CO-CONSTRUCTION
COLLECTIVE ENERGY
ARTHUR, T'AS GAGNÉ !
VAN STRATEGY OFFICER
...OU PAS...
ACCOMPAGNEMENT
ON FOUT DES BAFFES !
ADEO
VECLIA
VEROY MERLIN
CRAFT
GALERIES LAFAYETTE
EDF
NEXITY
CONTRIBUTION
MEETUP
ACCUEIL STAGIAIRES
FORMATION
SITE WEB
DEVFEST
COMMUNAUTÉS
CLOUD NORD
PARTAGE DE CONNAISSANCES
JAVA
DESIGN
QUALITY FIRST !
AURA DANS L'IT
CHIEF OF BEERS
CHIEF STRATEGY OFFICER
MODERN ARCHITECT
DELIVERY AVOCAT
GOODIES' DIRECTOR
ON EST TOUS LEADERS !
BIENVEILLANCE
GÉNÉREUX
25 PERSONNES !
AVRIL 2019
TOP ambiance !
POSITIVE TECHNOLOGY
TIENT SES PROMESSES
TECH' SYMPA !
SCALE ACCÉLÉRALE INNOVALE !
2024
POURQUOI ?
VOILÀ COMMENT...
OLIVIER SAMPSON-


The Fellowship of the Ring
J.R.R. TOLKIEN
THROUGH THE LOOKING-GLASS
Lewis Carroll
Illustrated by SIR JOHN TENNIEL
M
THE NEW FOREST
A.G. De Crespigny and Horace Hutchinson
LONDON JOHN MURRAY
ARTHUR RANSOME


I'M HEROIN
LAZARUS
COLOSSAL WAR
LOOSE SUTURES
Throwdown
OM
God is Good
DEVOTIONAL DOOM
mare cages
DOOM
SESSIONS VOL2
1782
ACID MAMMOTH
HEAVY PSYCH SOUNDS
RECORDS
BLACK SABBATH
MASTER OF REALITY
Throwdown
Jupiter Maker
BASTARD LORD
LOSER FOR
ROBOT GOD
BLACK SABBATH
THE END
Birmingham
TEMPLE of the FUZZ WITCH
Throwdown
BLACK SABBATH
PARANOID
ROBBERO BOBBERO
& THORSAKER
DEAD WAVE
Bobby
BLACK SABBATH
dazethrone
Pyro Weed Vol 3
sahara
the light
BLACK SABBATH
SABOTAGE
DOOMSMOKER
ZERAIN
Waiting at the Fuzzslides
acid king
ACID
SEPULTURA
Sabbath
EGYPT
PANTERA
COWBOYS FROM HELL

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**

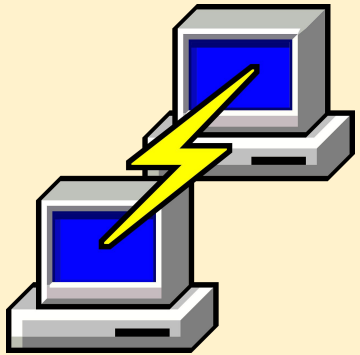
🧠 **Pas de biais cognitifs (au moins essayer)**

🔁 **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

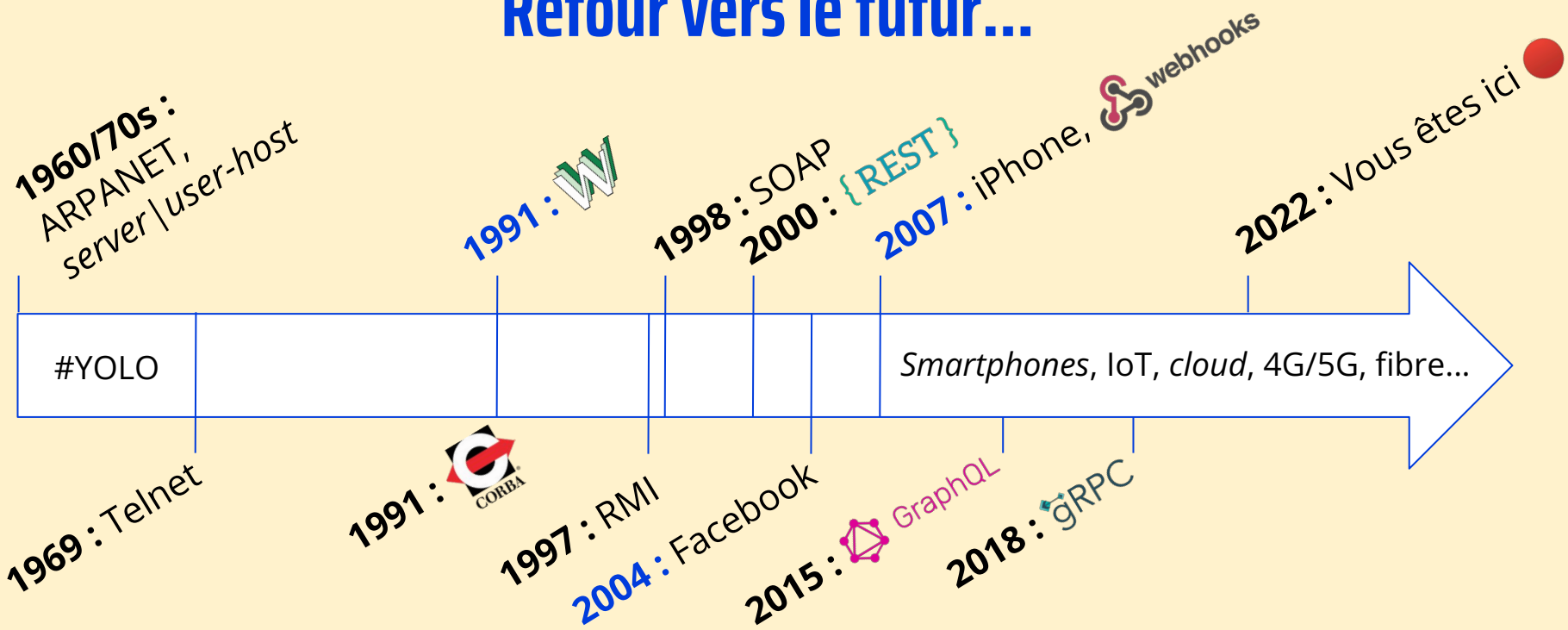
Sockets

TCP/IP : protocole de transport commun

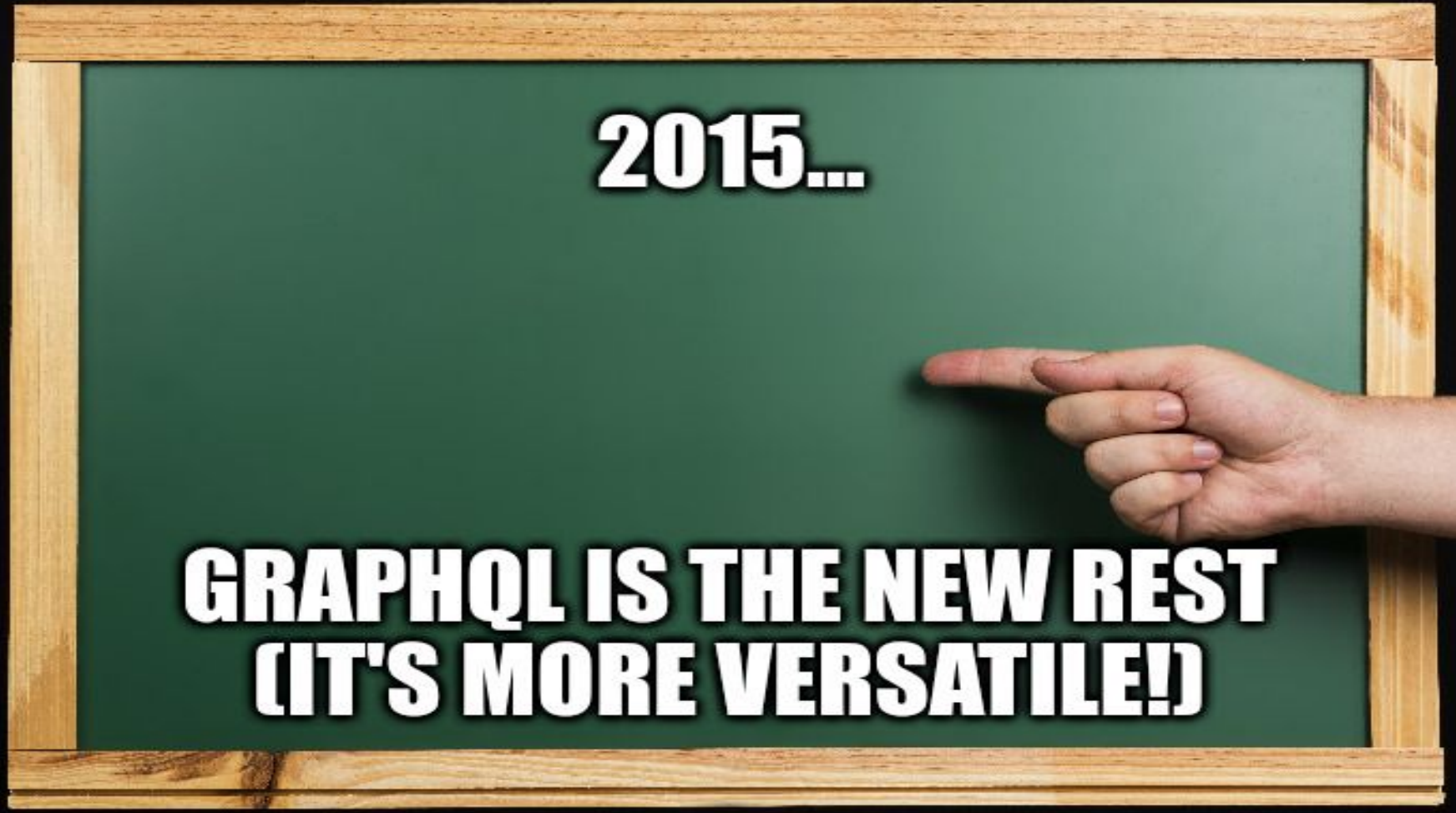
Communication inter-processus (IPC)

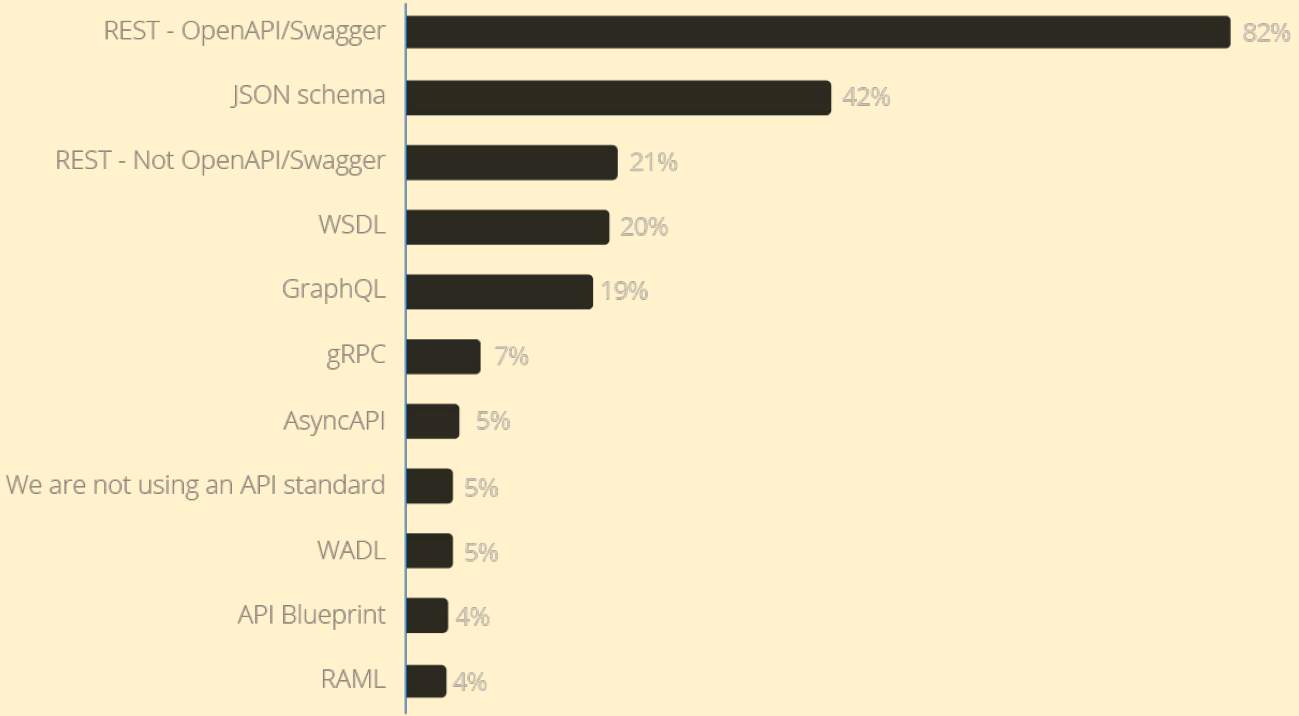
Besoins non fonctionnels

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>
```





Source : SmartBear's 2020 State of API Report


C'EST UNE BONNE SITUATION ÇA ?
imgflip.com

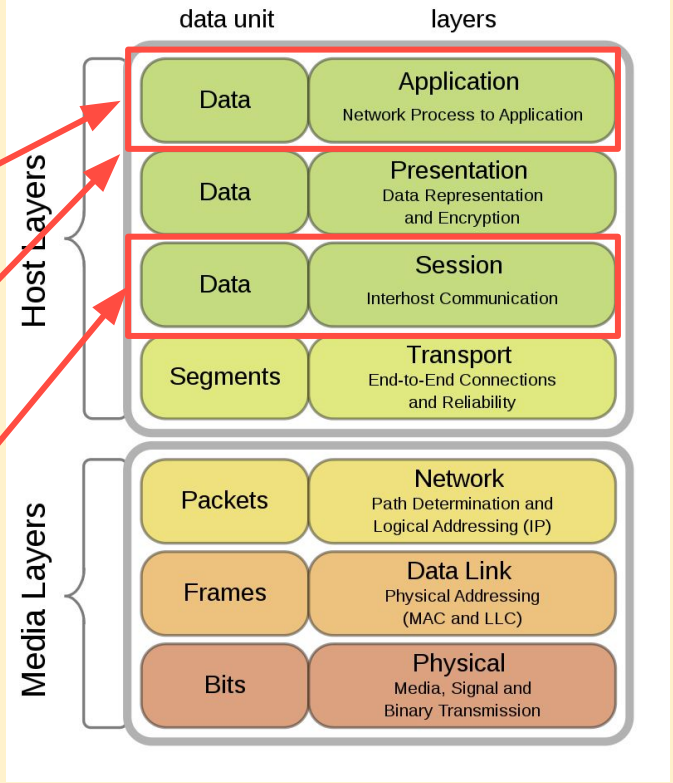
Différentes typologies

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

GraphQL : langage (**HTTP** POST)

Webhooks : HTTP POST

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



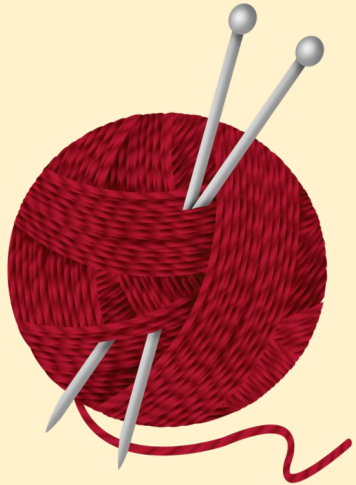
Source : Wikipédia

Produit fil rouge

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

Applications web / mobiles

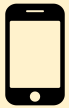
PIM *(Product Information Management)*



Vue.js

E-Commerce

Kotlin

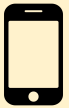


Mobile

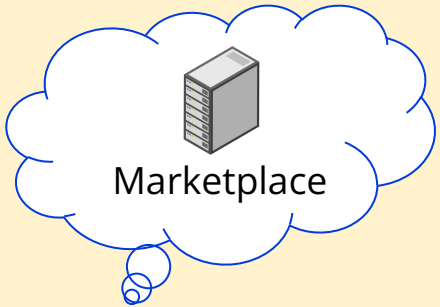
React

Caisses

Swift



Magasin



API Product

Java

API Price

Kotlin

API Stock

Kotlin

API Asset

Java

API - C'est quoi ?

***A**pplication **P**rogramming **I**nterface*

Client / Serveur

Échange d'informations

Utilise des standards

Cycle propre


{ REST }

C'est quoi ?

***RE**presentational **S**tate **T**ransfer*

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**

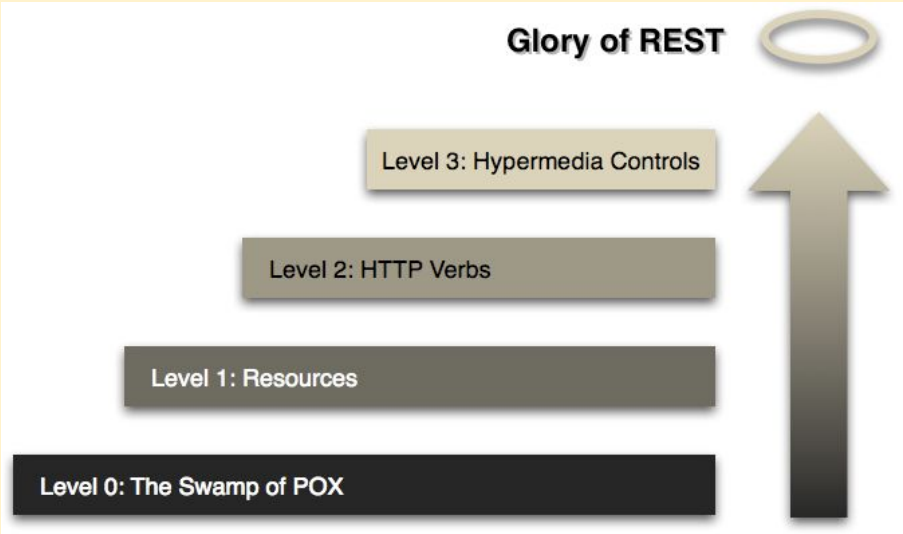
Approche **sans état** (*stateless*)

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)

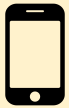
Cas d'utilisation :

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

Vue.js

E-Commerce

Kotlin

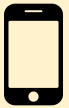


Mobile

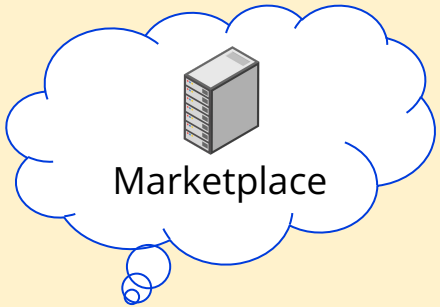
React

Caisses

Swift



Magasin



API Product

Java

API Price

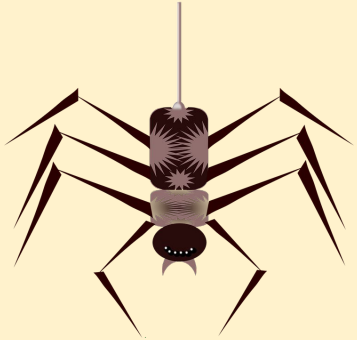
Kotlin

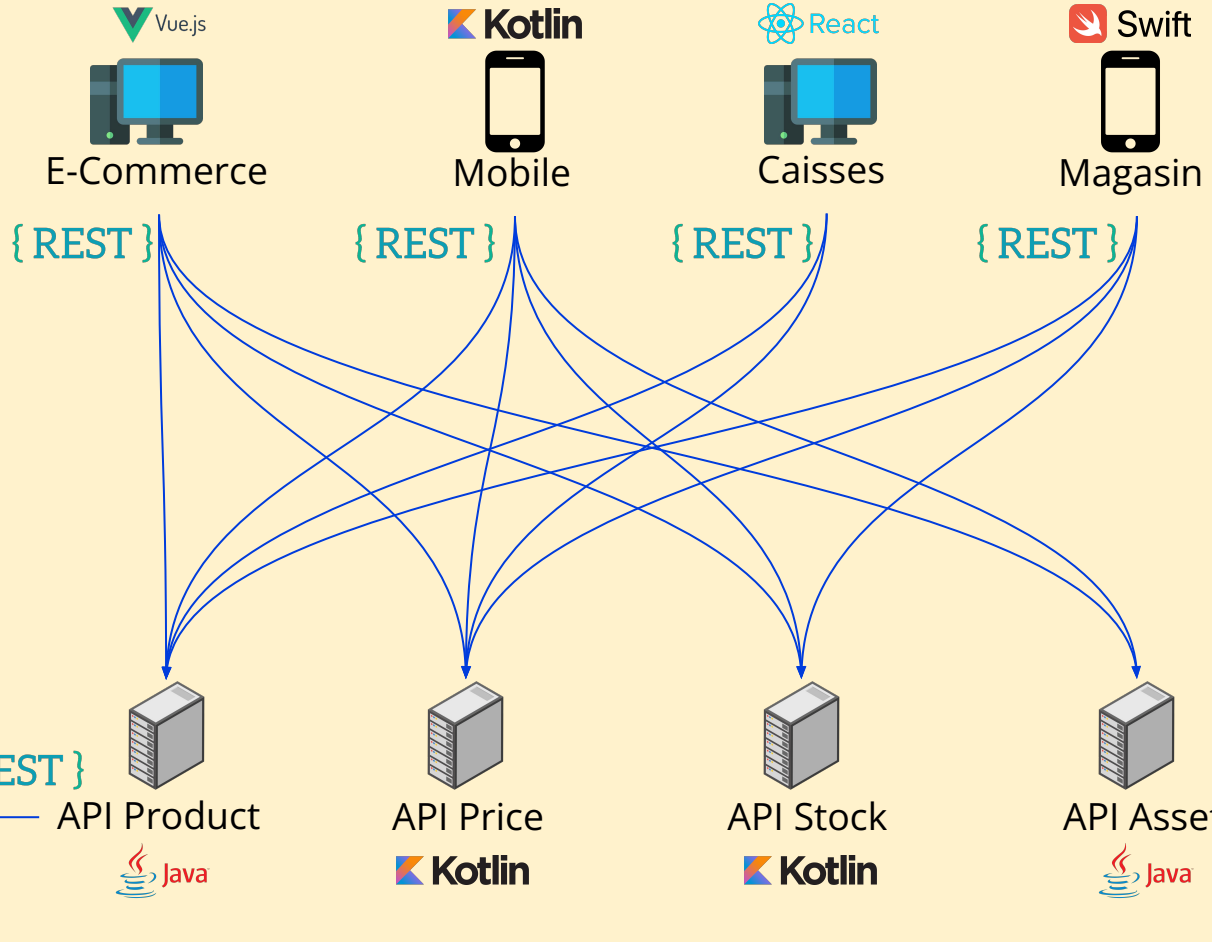
API Stock

Kotlin

API Asset

Java




Vue.js
Kotlin
React
Swift
E-Commerce
Mobile
Caisses
Magasin
{ REST }
{ REST }
{ REST }
{ REST }
Marketplace
{ REST }
API Product
API Price
API Stock
API Asset
Java
Kotlin
Kotlin
Java


gRPC

gRPC

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

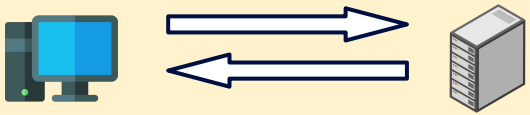
Streams formés de **paquets** (*frames* avec format spécifique et taille)

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

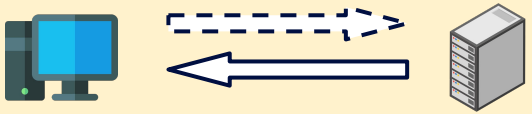
< 46,9 % des sites l'utilisent en janvier 2022
https://w3techs.com/technologies/details/ce-http2

💡 Modes de communication

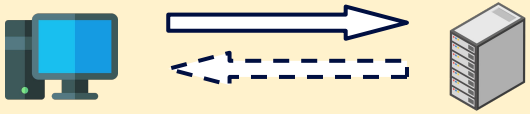
Unary



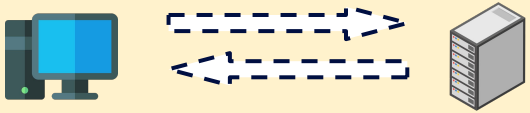
Client Streaming



Server Streaming



Bidirectional Streaming



💡 Protocol Buffers

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

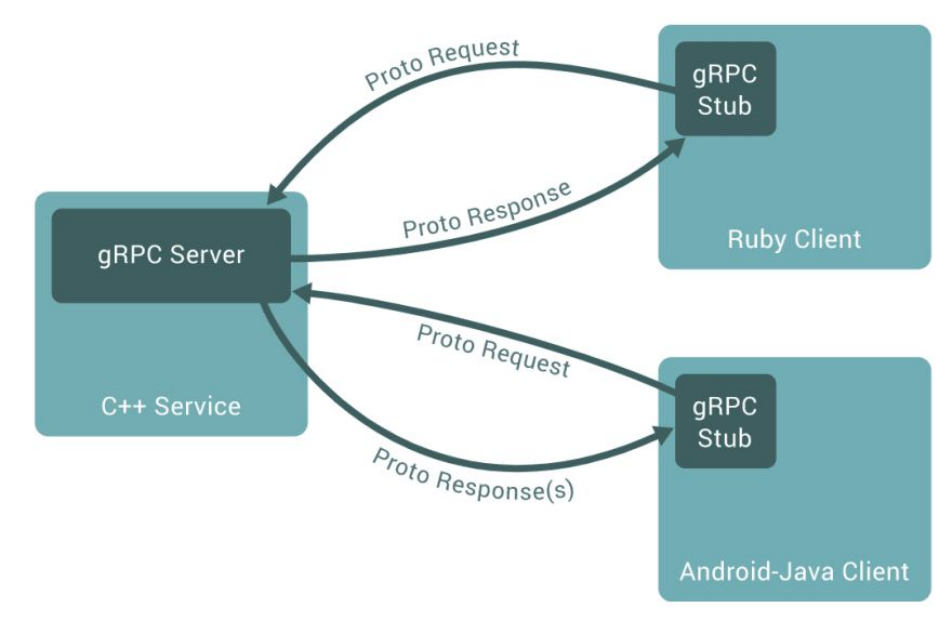
Agnostique

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/

gRPC

```
// 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;
}
```

gRPC

```
// 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
}
```

gRPC

```
// 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))
});
```

gRPC

💡 CRUD (REST) vs Actions (RPC)

```
POST /users/501/messages HTTP/1.1
Host: api.example.com
Content-Type: application/json

{"message": "Hello!"}
```

```
POST /SendUserMessage HTTP/1.1
Host: api.example.com
Content-Type: application/json

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

gRPC

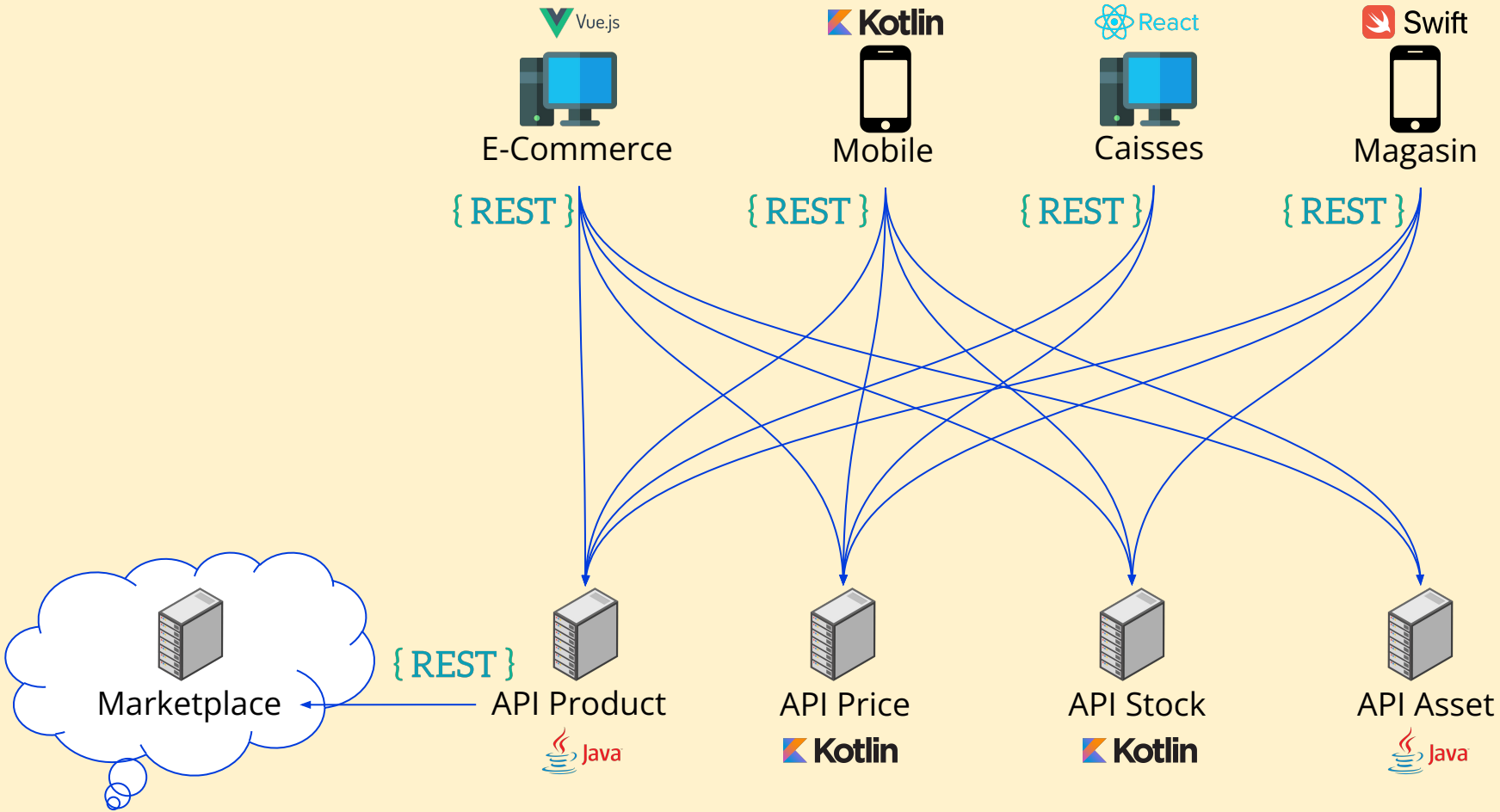
🤔 Pourquoi choisir gRPC ?

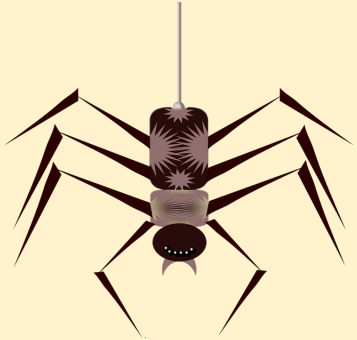
Motivations :

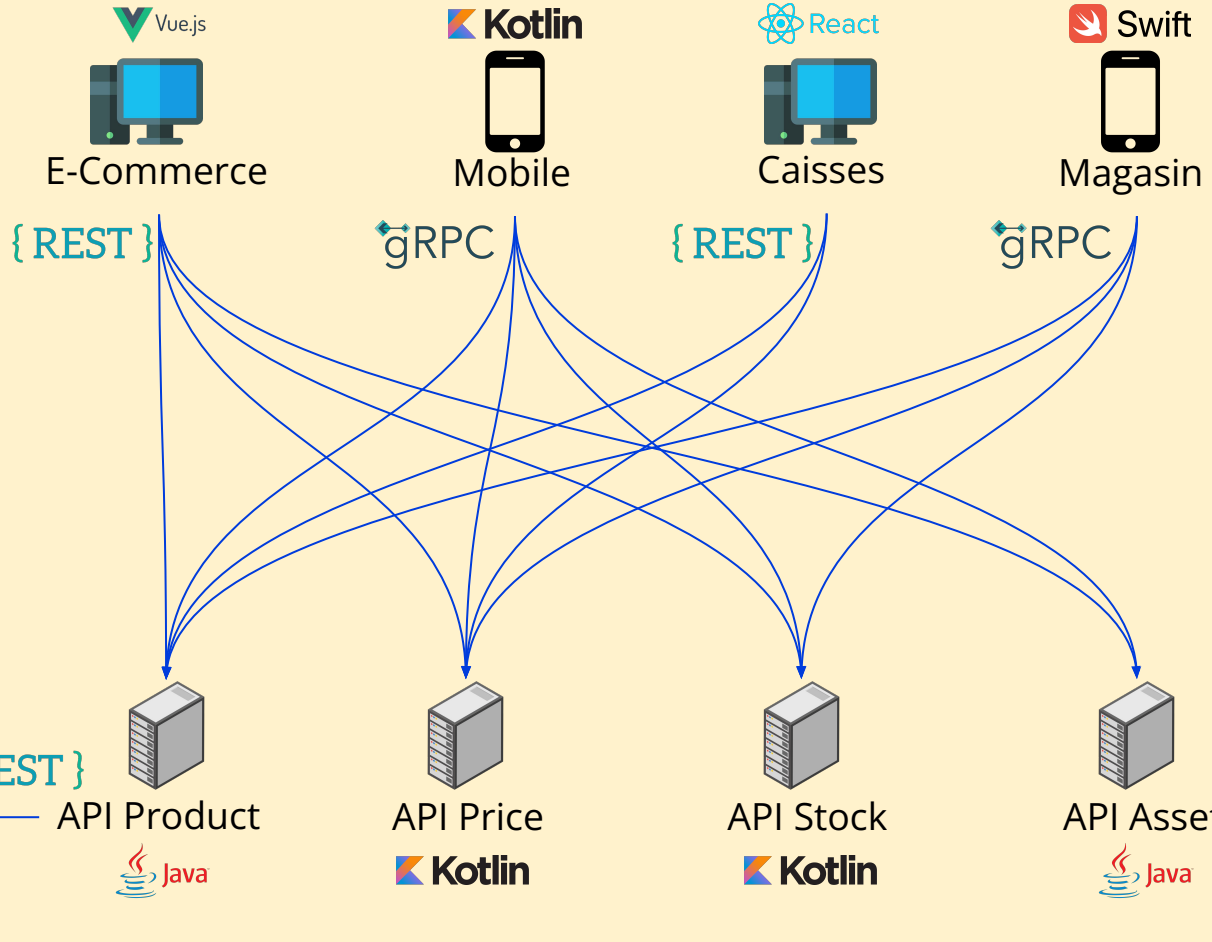
- Performances
- Légèreté des *payloads*
- Communication bilatérale et *streaming* en temps-réel
- Actions plus complexes

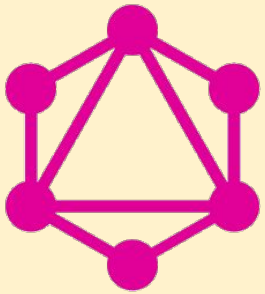
Cas d'utilisation :

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


Vue.js
Kotlin
React
Swift
E-Commerce
Mobile
Caisses
Magasin
{ REST }
{ REST }
{ REST }
{ REST }
Marketplace
{ REST }
API Product
API Price
API Stock
API Asset
Java
Kotlin
Kotlin
Java




Vue.js
Kotlin
React
Swift
E-Commerce
Mobile
Caisses
Magasin
{ REST }
gRPC
{ REST }
gRPC
Marketplace
{ REST }
API Product
API Price
API Stock
API Asset
Java
Kotlin
Kotlin
Java



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 pour clients 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 »

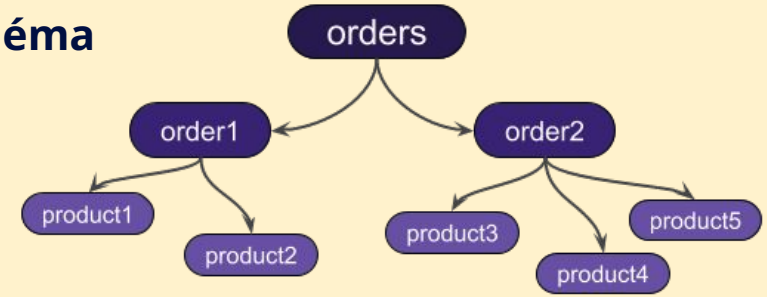
Domaine métier = **graphe** défini par un **schéma**

Différents types (de nœuds)

Connexions entre les nœuds

Références de types à types

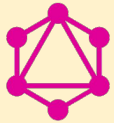
https://graphql.org/learn/thinking-in-graphs



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 }	GraphQL
Architecture	***Server driven***	***Client driven***
Endpoints	**n**	**1**
Schéma	Non mais…	**Oui**
Opérations	*Create, Read, Update, Delete*	*Query, Mutation, Subscription*
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
Architecture	***Server driven***	***Client driven***
Endpoints	**n**	**1**
Schéma	Oui	Oui
***Streaming*/bidirection**	Oui (client, server, bidirection)	Oui (subscriptions)
Formats	**Binaire (Protocol Buffers)**	GraphQL & JSON
Over|underfetching	**Oui**	**Non**
Versioning	Oui	**Non**
Sécurité	SSL/TLS / ALTS / JWT / OAuth	Oui mais…

🤔 Pourquoi choisir GraphQL ?

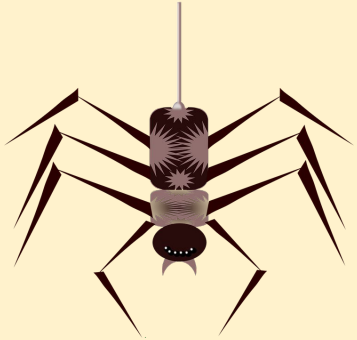
Motivations :

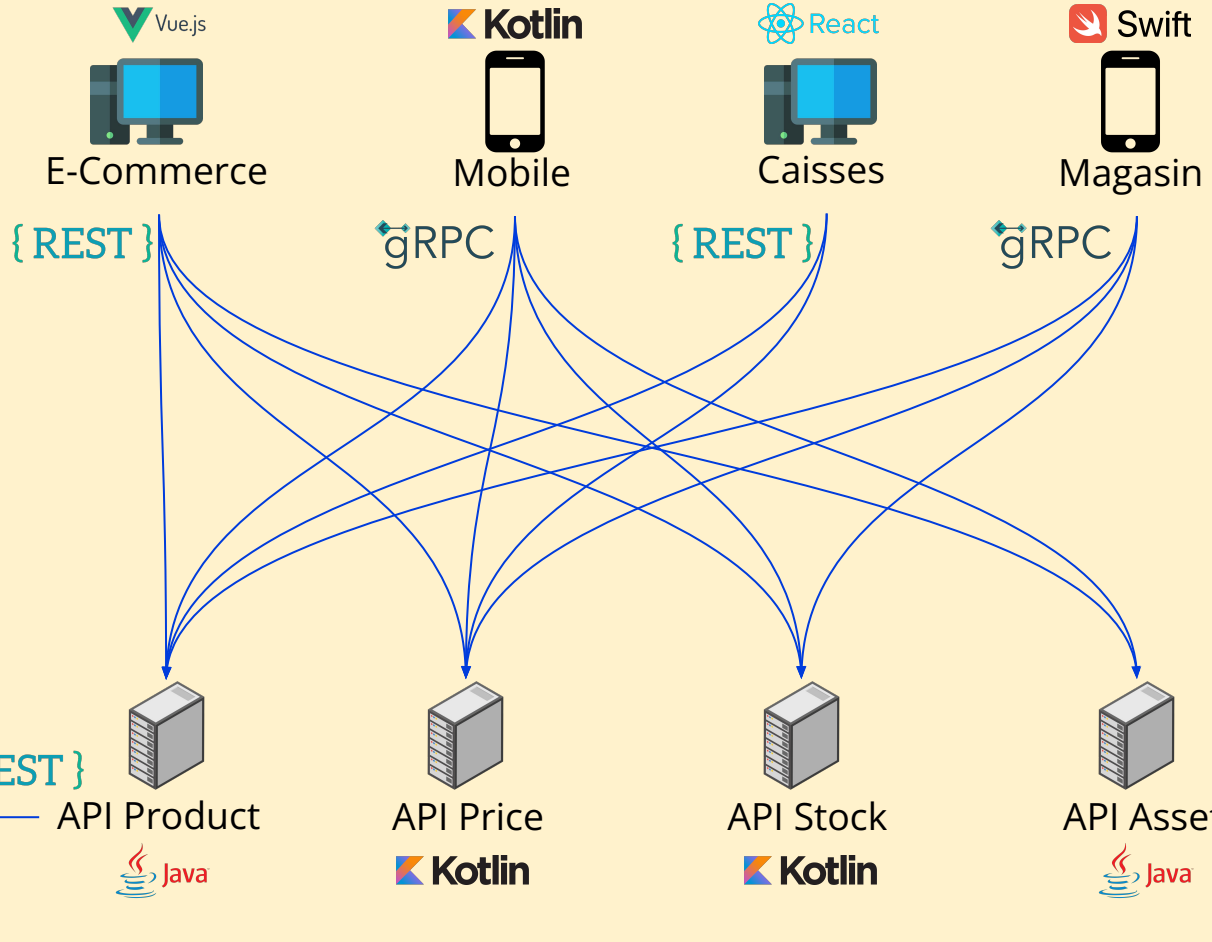
- Contrôle des données récupérées (*client-driven*)
- Utiliser plusieurs sources de données
- Bande passante limitée

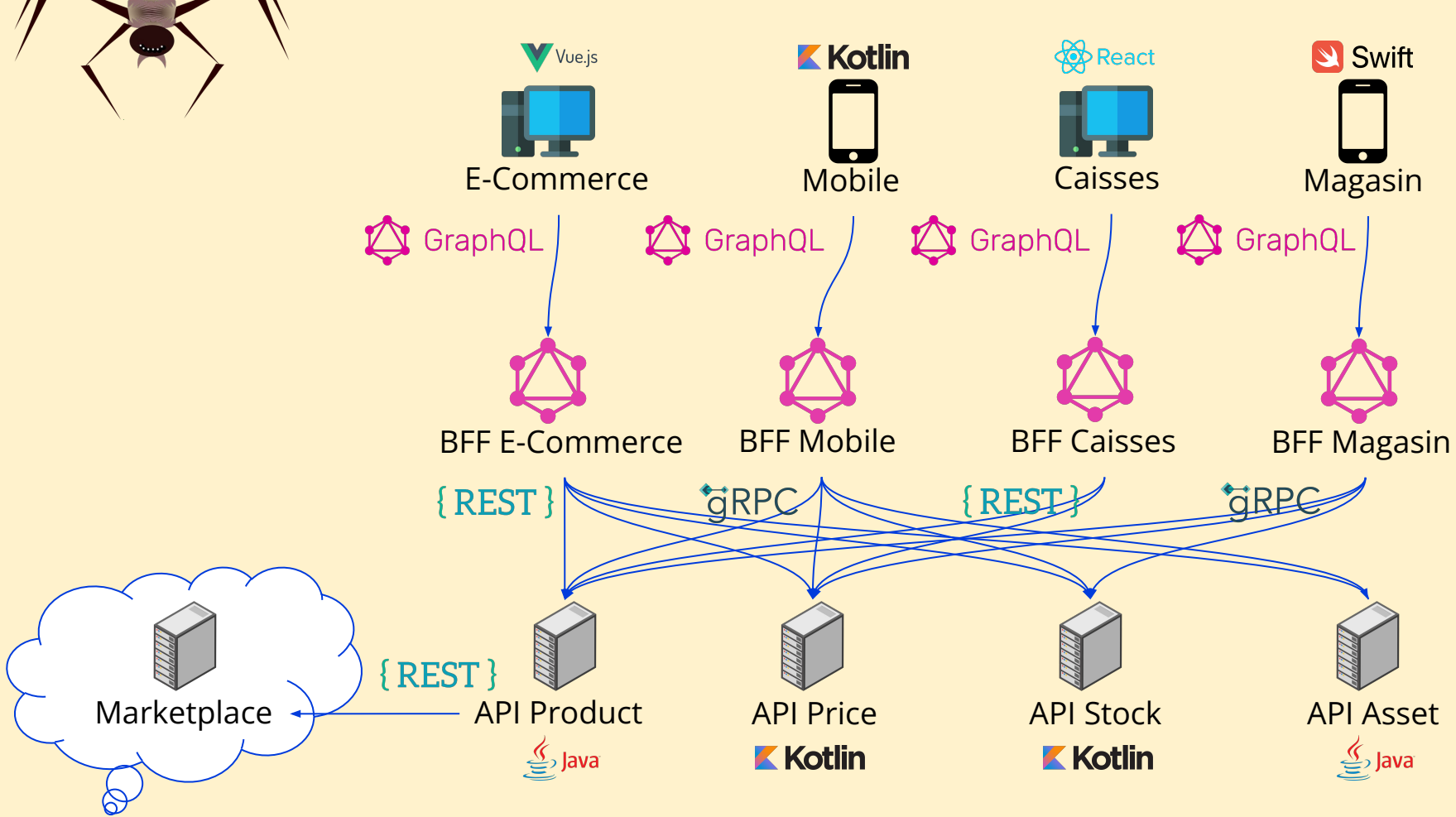
Cas d'utilisation :

- Grosses applications web (*e-commerce*…)
- Applications mobiles
- Applications *legacy* avec vieilles APIs difficiles à maintenir

- *Exemples de cas d'utilisation en production* (GitHub…)




Vue.js
Kotlin
React
Swift
E-Commerce
Mobile
Caisses
Magasin
{ REST }
gRPC
{ REST }
gRPC
Marketplace
{ REST }
API Product
API Price
API Stock
API Asset
Java
Kotlin
Kotlin
Java


Vue.js
Kotlin
React
Swift
E-Commerce
Mobile
Caisses
Magasin
GraphQL
GraphQL
GraphQL
GraphQL
BFF E-Commerce
BFF Mobile
BFF Caisses
BFF Magasin
{ REST }
gRPC
{ REST }
gRPC
Marketplace
{ REST }
API Product
API Price
API Stock
API Asset
Java
Kotlin
Kotlin
Java


webhooks

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...

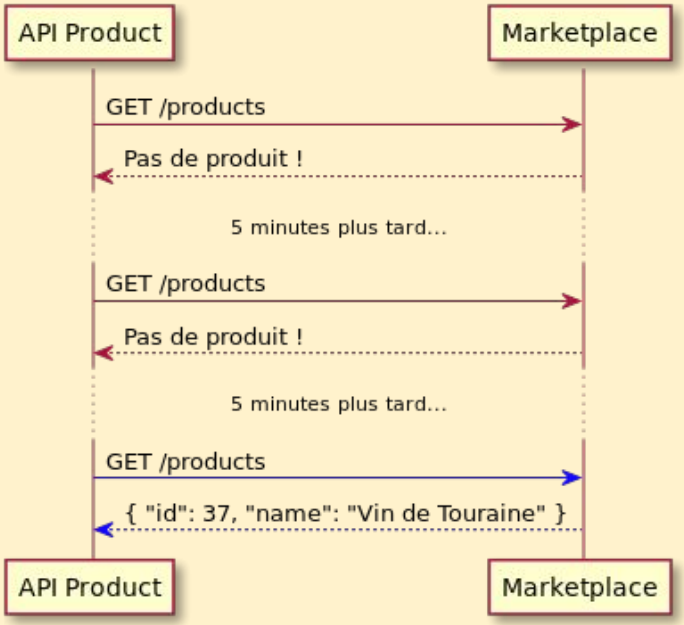
Webhooks : Envoi de message en réaction à un événement serveur à un client via une *callback URL* contenant un secret - Communication unidirectionnelle

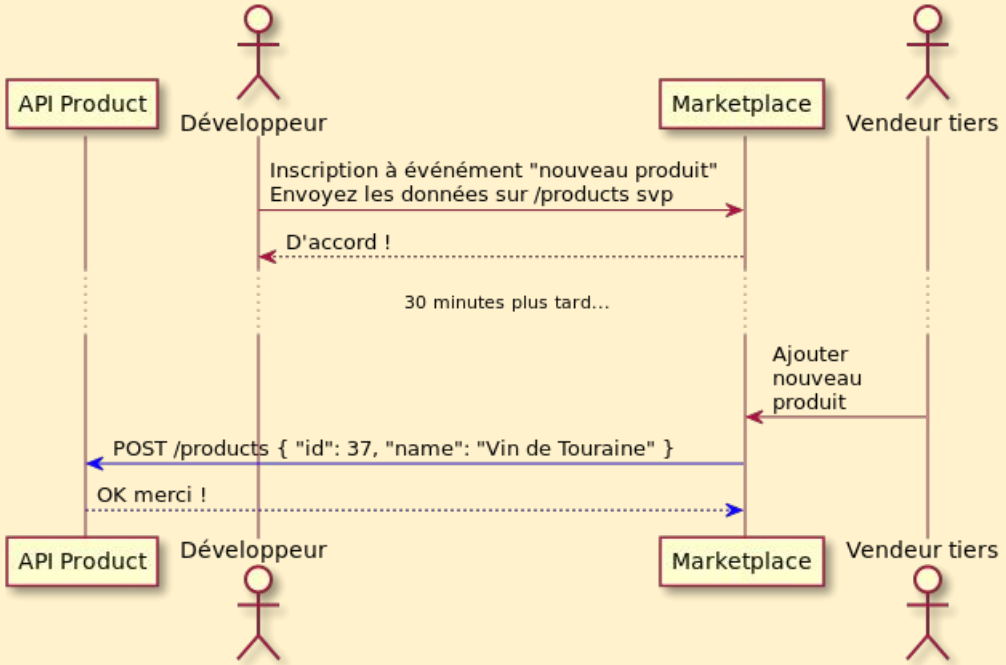
Websockets : Communication client / serveur bidirectionnelle à faible latence, client ouvre la connexion, challenges de scalabilité, réduit le nombre de requêtes HTTP

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

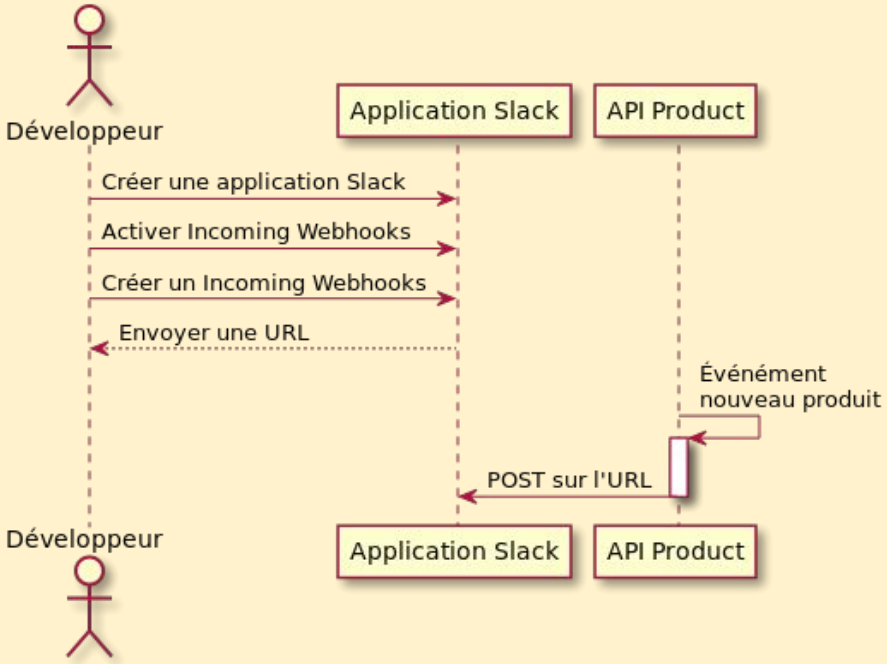
...

Pull vs Push


API Product
Marketplace
GET /products
Pas de produit !
5 minutes plus tard...
GET /products
Pas de produit !
5 minutes plus tard...
GET /products
{ "id": 37, "name": "Vin de Touraine" }
API Product
Marketplace


API Product
Développeur
Marketplace
Vendeur tiers
Inscription à événement "nouveau produit"
Envoyez les données sur /products svp
D'accord !
30 minutes plus tard...
Ajouter nouveau produit
POST /products { "id": 37, "name": "Vin de Touraine" }
OK merci !
API Product
Développeur
Marketplace
Vendeur tiers

Exemple d'appel




```
POST
https://hooks.slack.com/services/T00000000/B000
00000/XXXXXXXXXXXXXXXXXXXXXXXX
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
Mécanisme	***pull*** des données (*request based*)	***push*** des données (*event based*)
Communication	**Bilatérale** (⚠️ pas streaming)	Unilatérale
Temps réel	**Non** mais…	**Oui**
Verbes HTTP	GET | POST | PUT | DELETE | PATCH …	**POST** | GET (plus rare)
Formats	**JSON | XML | PDF | HTML …**	JSON | XML
Taille des messages	Peut être importante	**En général petite**

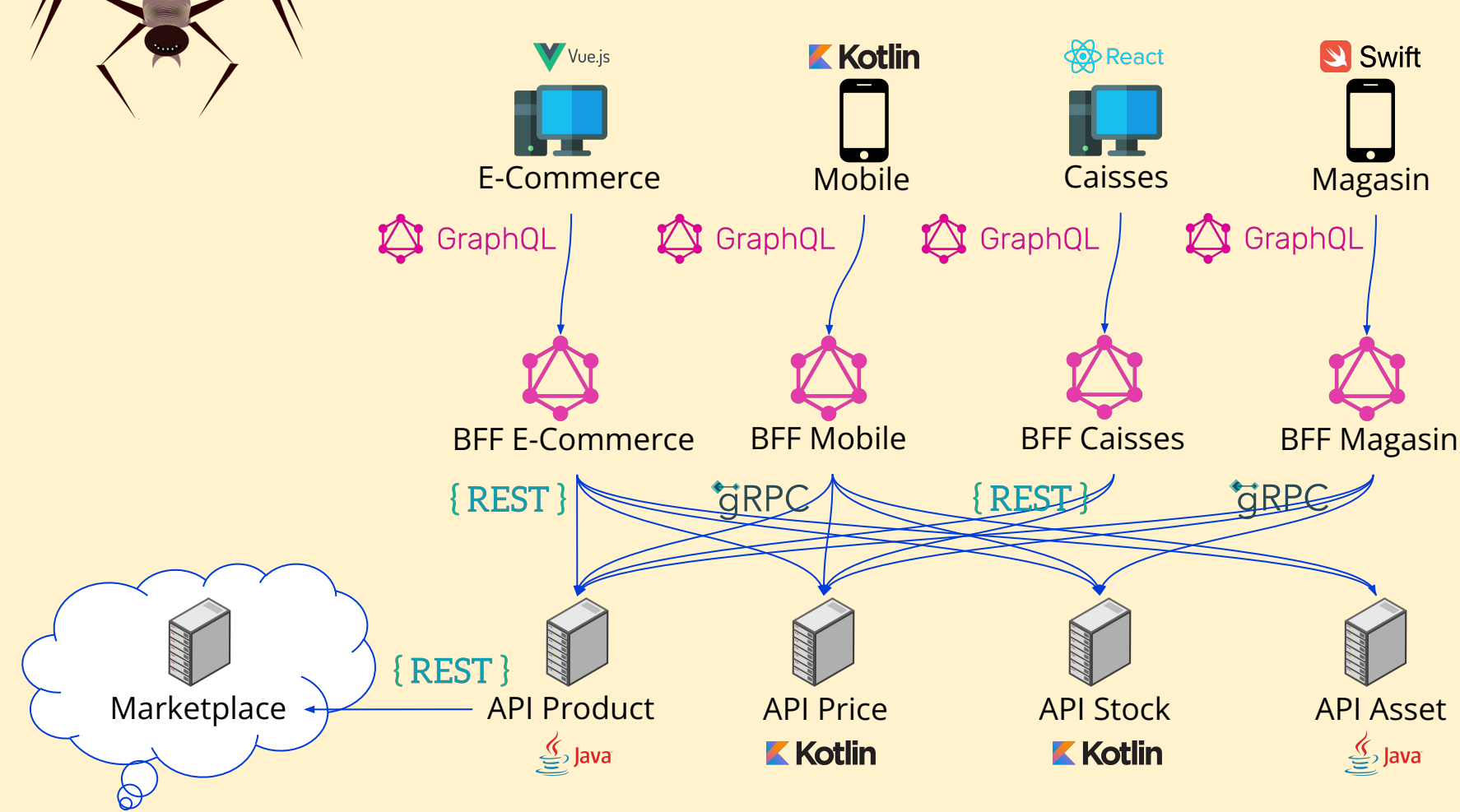
🤔 Pourquoi choisir les webhooks ?

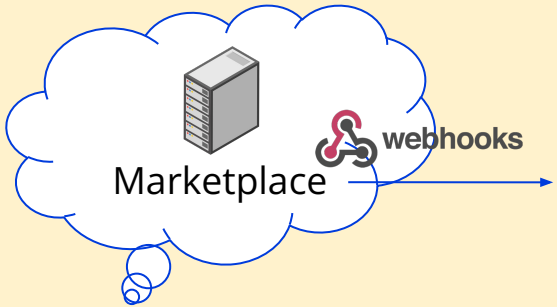
Motivations :

- 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…)


Vue.js
Kotlin
React
Swift
E-Commerce
Mobile
Caisses
Magasin
GraphQL
GraphQL
GraphQL
GraphQL
BFF E-Commerce
BFF Mobile
BFF Caisses
BFF Magasin
{ REST }
gRPC
{ REST }
gRPC
Marketplace
{ REST }
API Product
API Price
API Stock
API Asset
Java
Kotlin
Kotlin
Java



Vue.js
Kotlin
React
Swift
E-Commerce
Mobile
Caisses
Magasin
GraphQL
GraphQL
GraphQL
GraphQL
BFF E-Commerce
BFF Mobile
BFF Caisses
BFF Magasin
{ REST }
gRPC
{ REST }
gRPC
API Product
Java
API Price
Kotlin
API Stock
Kotlin
API Asset
Java

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

ippon

Merci ! 🙏

Avez-vous des questions ?

www.ippon.fr

contact@ippon.fr —— +33 1 46 12 48 48 —— @ipponTech