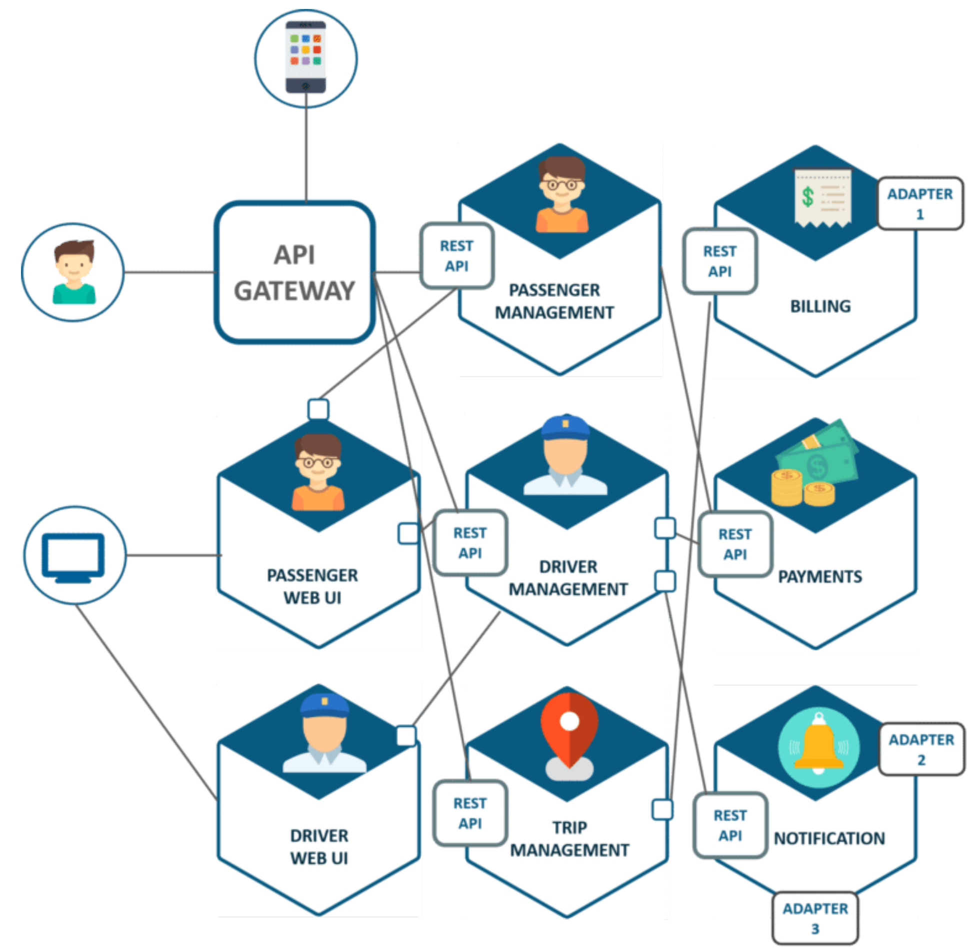
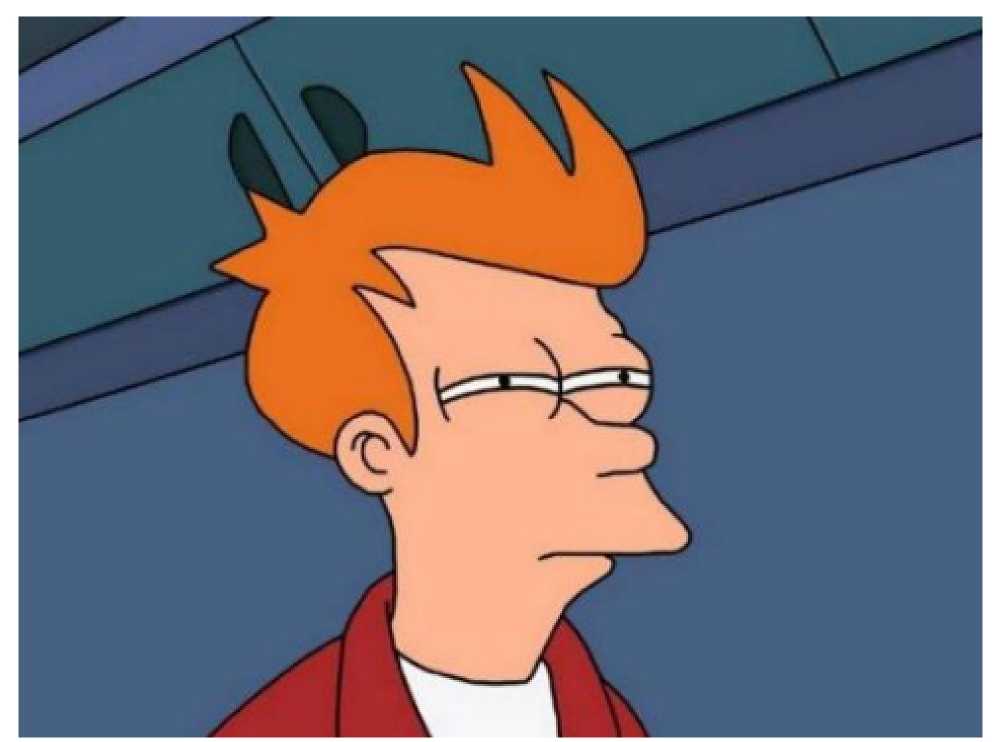
ALOM

UBER



PROBLÉMATIQUE

- Comment communiquer avec les autres micro-services?
- Comment communiquer avec les partenaires?



COMMENT FAIRE COMMUNIQUER DES PROCESSUS ?

Sur une même machine : IPC - Inter Process Communication

- mémoire partagée
- message queue
- sémaphores

Sur des machines séparées : Réseau

- sockets

EN JAVA ☕ - RMI (REMOTE METHOD INVOCATION)

Communication entre 2 JVM

`java.rmi.*`

Définition d’une interface qui `extends` `java.rmi.Remote`.

Paramètres sérialisés en binaire, interface `java.io.Serializable` à implémenter.

CONTRAINTES ⛓️

Toutes les applications ne sont pas écrites dans le même langage (Java, .Net, NodeJS, PHP, Ruby, Python...)

Les partenaires n'ont pas forcément les mêmes environnements (réseaux, firewall)

SOLUTION

Définition d'une norme de communication, basée sur des standards.

WEB SERVICES & WEB-SOCKETS 🌍

- Protocole HTTP(S)
  - Facile à implémenter (texte)
  - Passe les firewalls (port 80/443)
  - Sécurisation avec SSL/TLS
- Formats de données
  - SOAP : XML
  - REST : JSON
- Contrat de service
  - SOAP : WSDL
  - REST : Swagger...

WEB SERVICES SOAP

WEB SERVICES SOAP

Simple

Object

Access

Protocol

WEB SERVICES SOAP

PRINCIPES ARCHITECTURAUX

- Protocole de type RPC (Remote Procedure Call)
- Représentation XML (Enveloppe, Header, Body)
- S'appuie sur HTTP, ou tout autre protocole (SMTP, JMS...)

WEB SERVICES SOAP

EXEMPLE DE WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><wsdl:de
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ele

    <xs:element name="getCountryRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getCountryResponse">
        <xs:complexType>
            <xs:sequence>
```

EXEMPLE DE REQUÊTE

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/so
                  xmlns:gs="http://spring.io/guides/gs-produci
   <soapenv:Header/>
   <soapenv:Body>
      <gs:getCountryRequest>
         <gs:name>Spain</gs:name>
      </gs:getCountryRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

EXEMPLE DE RÉPONSE

```xml
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:getCountryResponse xmlns:ns2="http://spring.io/guides
      <ns2:country>
        <ns2:name>Spain</ns2:name>
        <ns2:population>46704314</ns2:population>
        <ns2:capital>Madrid</ns2:capital>
        <ns2:currency>EUR</ns2:currency>
      </ns2:country>
    </ns2:getCountryResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

EN JAVA ☕

2 POSSIBLITÉS D'IMPLÉMENTATION

- Contract-first (top-down) ⬇️
- Code-first (bottom-up) ⬆️

CONTRACT-FIRST

Approche historique

Rédaction d'un contrat WSDL (ouch!)

Récupération d'un contrat d'un partenaire

Génération d'un **stub**(client) et d'un **squeleton**(server) Java

CONTRACT-FIRST

Génération du code avec un plugin maven:

```xml
<build>
  <plugins>
    <plugin>
      <groupId>com.sun.xml.ws</groupId>
      <artifactId>jaxws-maven-plugin</artifactId>
      <version>3.0.2</version>
    </plugin>
  </plugins>
</build>
```

CODE-FIRST

Approche plus simple 😎

Écriture des classes/interfaces Java

Génération du WSDL à partir des classes (annotations/méthodes etc...)

CODE-FIRST

```xml
<dependencies>
  <dependency>
    <groupId>jakarta.xml.ws</groupId>
    <artifactId>jakarta.xml.ws-api</artifactId>
    <version>3.0.1</version>
  </dependency>
</dependencies>

<dependencies>
  <dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>jaxws-rt</artifactId>
    <version>3.0.1</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

WEB SERVICES REST

WEB SERVICES REST

REpresentational

State

Transfert

WEB SERVICES REST

PRINCIPES ARCHITECTURAUX

- Architecture découplée client/serveur
- Sans état (pas de session)
- Accès à des ressources:
    - Identifiées de manière unique
    - Manipulées via des représentations (JSON, XML, HTML...)
    - Compatible avec une mise en cache
    - Données Hypermédia

WEB SERVICES REST

Utilisation des codes HTTP (Http Status Dogs)

don’t

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 15

{"error":"Content Not Found"}
```

do

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json
Content-Length: 15

{"error":"Pokemon with id {152} does not exists"}
```

MIAGE Lille

NÉGOCIATION DE CONTENU

Principes au coeur du web

Le client indique au serveur ses attentes via des headers HTTP

```
1 Accept: text/plain
2 Accept: application/xml
3 Accept: application/json
4 Accept: application/json,text/plain
5 Accept: image/png
```

carbon
carbon.now.sh

NÉGOCIATION DE CONTENU

- Format des données : header `Accept`
- Traduction : header `Accept-Language`
- Retour avec les headers `Content-Type` et `Content-Language`

La RFC 4229 liste les headers possibles.

```
GET /pokemons-types/25
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5

HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
{...}
```

WEB SERVICES REST

HATEOAS : HYPERMEDIA AS THE ENGINE OF APPLICATION STATE

Le message contient les informations permettant de manipuler l'application

WEB SERVICES REST : EXEMPLE XML

```
GET /account/12345 HTTP/1.1
Host: somebank.org
Accept: application/xml
..

HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="http://somebank.org/account/12345/deposit" />
    <link rel="withdraw" href="http://somebank.org/account/12345/withdraw" />
    <link rel="transfer" href="http://somebank.org/account/12345/transfer" />
    <link rel="close" href="http://somebank.org/account/12345/close" />
</account>
```

carbon
carbon.now.sh

MIAGE Lille

WEB SERVICES REST : EXEMPLE JSON

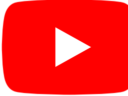
```
GET /account/12345 HTTP/1.1
Host: somebank.org
Accept: application/json
...

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
    "account_number" : "12345"
    "balance" : 100.00,
    "currency" : "usd",
    "links" : [
        { "rel": "deposit", "href" : "http://somebank.org/account/12345/deposit" },
        { "rel": "withdraw", "href" : "http://somebank.org/account/12345/withdraw" },
        { "rel": "transfert", "href" : "http://somebank.org/account/12345/transfert" },
        { "rel": "close", "href" : "http://somebank.org/account/12345/close" }
    }
}
```

carbon
carbon.now.sh

HATEOAS

Excellent talk de Julien Topçu

REST next level : Ecrire des APIs web orientées métier (Juli…



CONTRAT DE SERVICE REST

OPENAPI (EX SWAGGER)

OpenAPI est la spec, *Swagger* une implémentation

Description des API au format JSON ou YAML : *OpenApi*

Rendu Web + "Try It" : *Swagger*

Pokemon API

SWAGGER

Affichage de la documentation sous forme de page web

{ } SWAGGERhub
SMARTBEAR

Juwit

IFI Pokemon API

1.0.0

[Base URL: ifi-pokemon-api.herokuapp.com]

This is a simple Pokemon API implementation.

Schemes

HTTPS

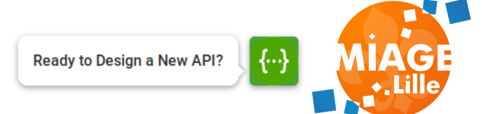
pokemons Everything about your Pokemons

GET /pokemons Get all the pokemons

GET /pokemons/{pokemonId} Get a pokemon by ID

Models

Pokemon >



SWAGGER

Génération de squelettes clients/serveur

Export

IFI Pokemon API

1.0.0

[Base URL: ifi-pokemon-api.herokuapp.com]

This is a simple Pokemon API implementation.

Schemes

HTTPS

pokemons Everything about your Poke

GET /pokemons Get all the pokemons

akka-scala
android
apex
clojure
cpprest
csharp
csharp-dotnet2
cwiki
dart
dynamic-html
flash
go
groovy

Codegen Options
‹ Client SDK
‹ Server Stub
‹ Download API

SWAGGER

Génération de squelettes clients/serveur

Export

Codegen Options

‹ Client SDK

‹ Server Stub

‹ Download API

akka-scala
android
apex
clojure
cpprest
csharp
csharp-dotnet2
cwiki
dart
dynamic-html
flash
go
groovy

IFI Pokemon API

1.0.0

[Base URL: ifi-pokemon-api.herokuapp.com]

This is a simple Pokemon API implementation.

Schemes

HTTPS

pokemons Everything about your Poke

GET /pokemons Get all the pokemons

SWAGGER ET *OPENAPI* EN SPRING BOOT

Pas d'implémentation de la part de Spring

1 projet Open Source

springdoc-openapi (doc)

Exposition d'un service REST Spring

```java
/**
 * A convenience annotation that is itself annotated with
 * {@link Controller @Controller} and {@link ResponseBody @ResponseBody}.
 * <p>
 * Types that carry this annotation are treated as controllers where
 * {@link RequestMapping @RequestMapping} methods assume
 * {@link ResponseBody @ResponseBody} semantics by default.
  */
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {

    /**
     * The value may indicate a suggestion for a logical component name,
     * to be turned into a Spring bean in case of an autodetected component.
     * @return the suggested component name, if any (or empty String otherwise)
     */
    @AliasFor(annotation = Controller.class)
    String value() default "";

}
```

EN SPRING

/api/pokemon-types/{id}

/api/pokemon-types?orderBy=name

/api/pokemon-types?type=poison

- `@RequestMapping` : écouter une URI
- `@PathVariable` : récupérer les variables d'URI entre '{}'
- `@RequestParam` : récupérer les paramètres de requête (query-strings '?a=b&c=d')
- `@RequestBody` : récupérer le corps de la requête

@REQUESTMAPPING

`/api/pokemon-types`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping("/pokemon-types")
    public Iterable<Pokemon> getAllPokemons() {
        ...
    }
}
```

carbon
carbon.now.sh

@PATHVARIABLE

`/api/pokemon-types/{id}`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping("/pokemon-types/{id}")
    public Pokemon getPokemon(@PathVariable String id) {
        ...
    }
}
```

carbon
carbon.now.sh

MIAGE Lille

@REQUESTPARAM

`/api/pokemon-types?orderBy=name`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping("/pokemon-types")
    public Iterable<Pokemon> getAllPokemons(@RequestParam String orderBy) {
        ...
    }
}
```

carbon
carbon.now.sh

@REQUESTPARAM

`/api/pokemon-types?type=poison`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping(path = "/pokemon-types", params = {"orderBy"})
    public Iterable<Pokemon> getAllPokemons(@RequestParam String orderBy) {
        ...
    }

    @GetMapping(path = "/pokemon-types", params = {"type"})
    public Iterable<Pokemon> getAllPokemons(@RequestParam String type) {
        ...
    }
}
```

MIAGE Lille

@REQUESTBODY

`POST /api/trainers`

```java
@RestController
@RequestMapping("/api")
public class TrainerController {

    @PostMapping("/trainers")
    public Trainer createTrainer(@RequestBody Trainer trainer) {
        ...
    }
}
```

carbon
carbon.now.sh

CONSOMMATION REST EN SPRING

`RestTemplate` (maintenance)

```java
@Service
class PokemonTypeServiceImpl implements PokemonTypeService{

    private RestTemplate restTemplate;

    PokemonTypeServiceImpl(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @Override
    public List<PokemonType> listPokemonsTypes() {
        var pokemonTypes = restTemplate
            .getForObject(pokemonServiceUrl+"/pokemon-types",
        return Arrays.asList(pokemonTypes);
    }
```

SPRING RESTTEMPLATE

Classe utilitaire pour effectuer des appels REST

- Exécute les requêtes HTTP : GET/POST/PUT/PATCH/OPTIONS/DELETE/HEAD
- Utilise `jackson-databind` pour convertir les objets Java en JSON !

HTTP INTERFACE

On définit une interface avec des méthodes annotées `@HttpExchange`.

Utilisation des annotations `@RequestParam`, `@PathVariable`, `@RequestBody`, `@RequestHeader` pour les paramètres.

On déclare en type de retour le type attendu, ou `ResponseEntity<T>`.

Spring génère un proxy dynamique qui implémente l’interface et exécute les appels HTTP (comme les Spring data repository).

MIAGE Lille

HTTP INTERFACE

On déclare une interface

```
@HttpExchange("/pokemon-types")
public interface PokemonTypeApiRepository {

    @GetExchange
    List<PokemonType> getAllPokemons();

    @GetExchange("/{id}")
    PokemonType getPokemonFromId(@PathVariable int id);

}
```

On a besoin de Spring WebFlux pour le client HTTP

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

On configure un client, et un proxy

```java
// configuration à mettre dans une classe annotée @Configurati
@Bean
PokemonTypeApiRepository configurePokemonTypeApiRepository(@Va
    var webclient = WebClient.create(pokemonTypeServiceUrl);
    var factory = HttpServiceProxyFactory.builder(WebClientAda
    return factory.createClient(PokemonTypeApiRepository.class
}
```

On reçoit le bean en injection de dépendance comme d'habitude.

WEBCLIENT

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

Client pour la programmation réactive & synchrone.

```java
// reactive
Flux<PokemonType> pokemonsFlux = client.get().uri("/pokemon-ty
    .retrieve()
    .bodyToFlux(PokemonType.class);

// synchrone
List<PokemonType> pokemonsList = pokemonsFlux
    .collectList()
    .block();
```

OUTILLAGE

ou

POSTMAN/INSOMNIA

LES AUTRES MOYENS DE COMMUNICATION RÉSEAU

- Reactive Streams - Support avec Spring WebFlux
- GraphQL - Support avec Spring GraphQL
- gRPC - pas d'implémentation officielle

TP

Interoperability