

JEE 2

GOOGLE FORM

<https://forms.gle/FneevFP4BcWrTHAb8>

Chapitre 1

Mise en œuvre des architectures Micro-services avec Spring Boot et Spring Cloud

Evolution des architectures

2006

2011

Legacy Applications

Une grosse application



Connecteurs pour SMS, EMAIL et paiement
Interfaces web
Base de données
API REST/SOAP

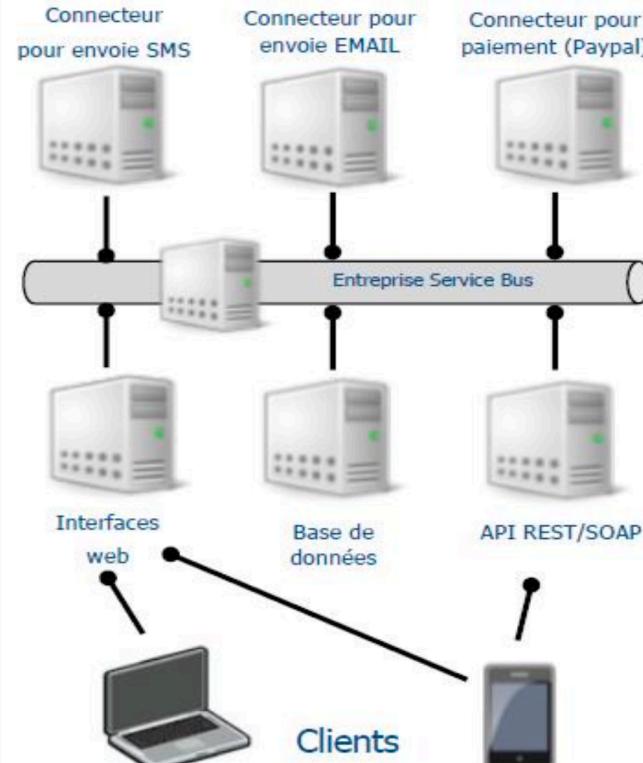
Architecture dite « Monolithique »



Clients

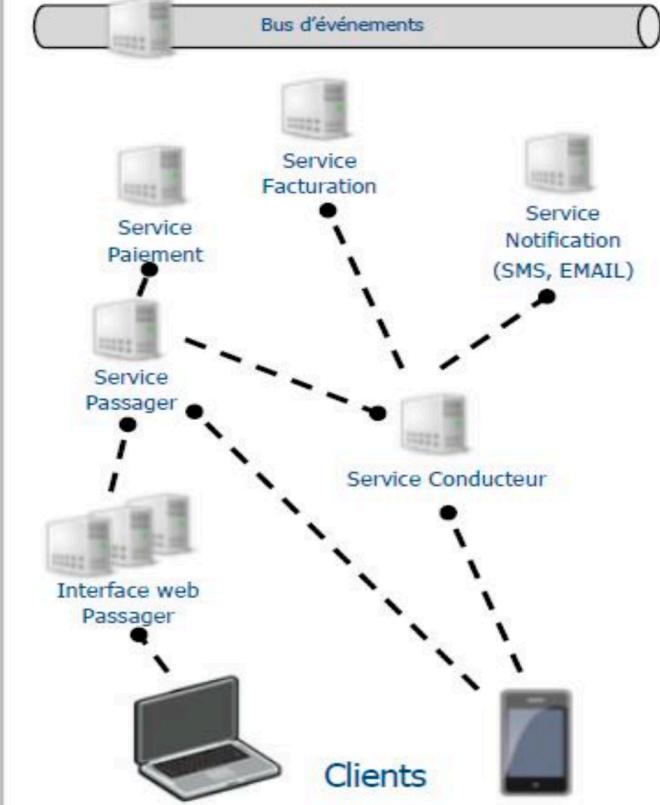


Une Application découpée par des services techniques reliés par un bus d'intégration



Architecture Orientée Service

Une Application découpée par des services fonctionnels



Architecture Microservice

Architecture Monolithique

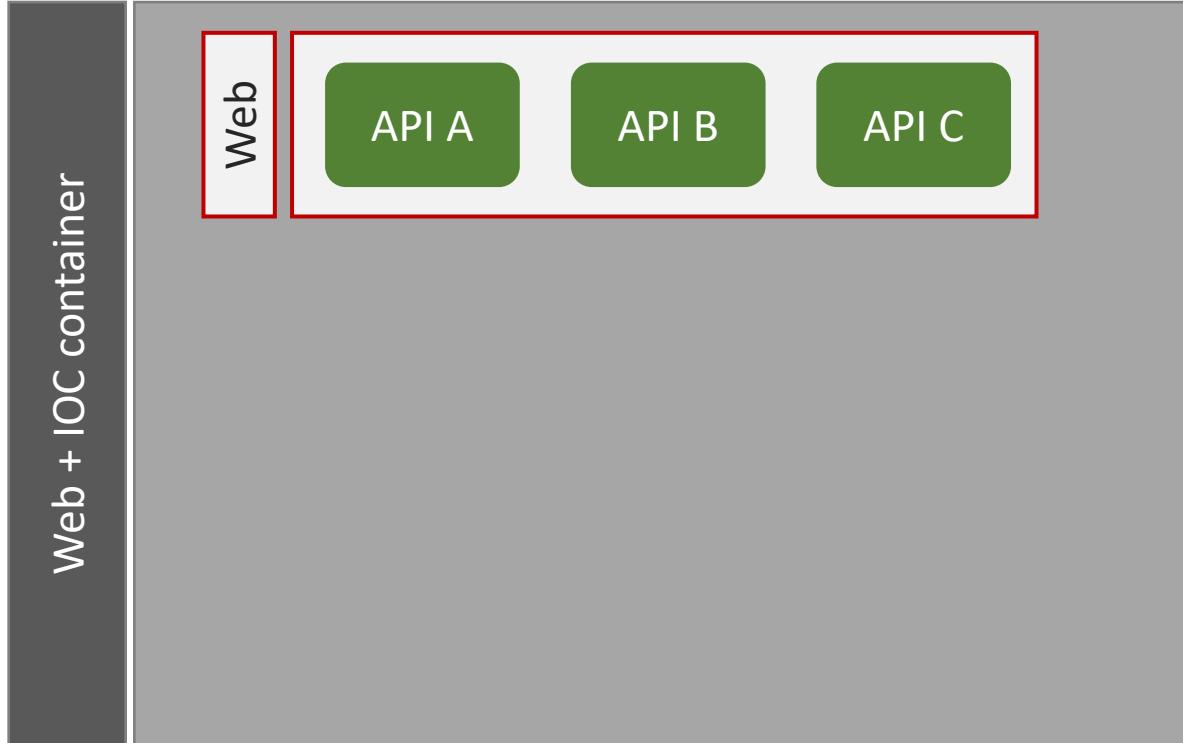
- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architecture Monolithique

Web + IOC container

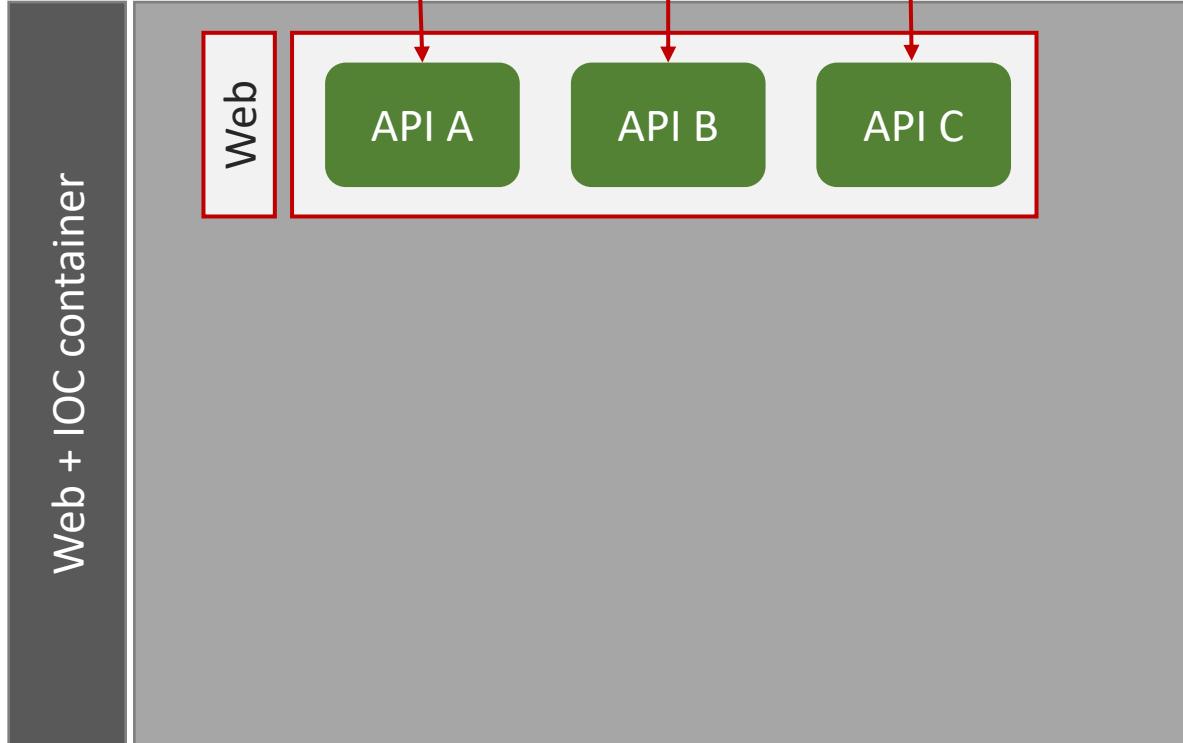
- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architecture Monolithique



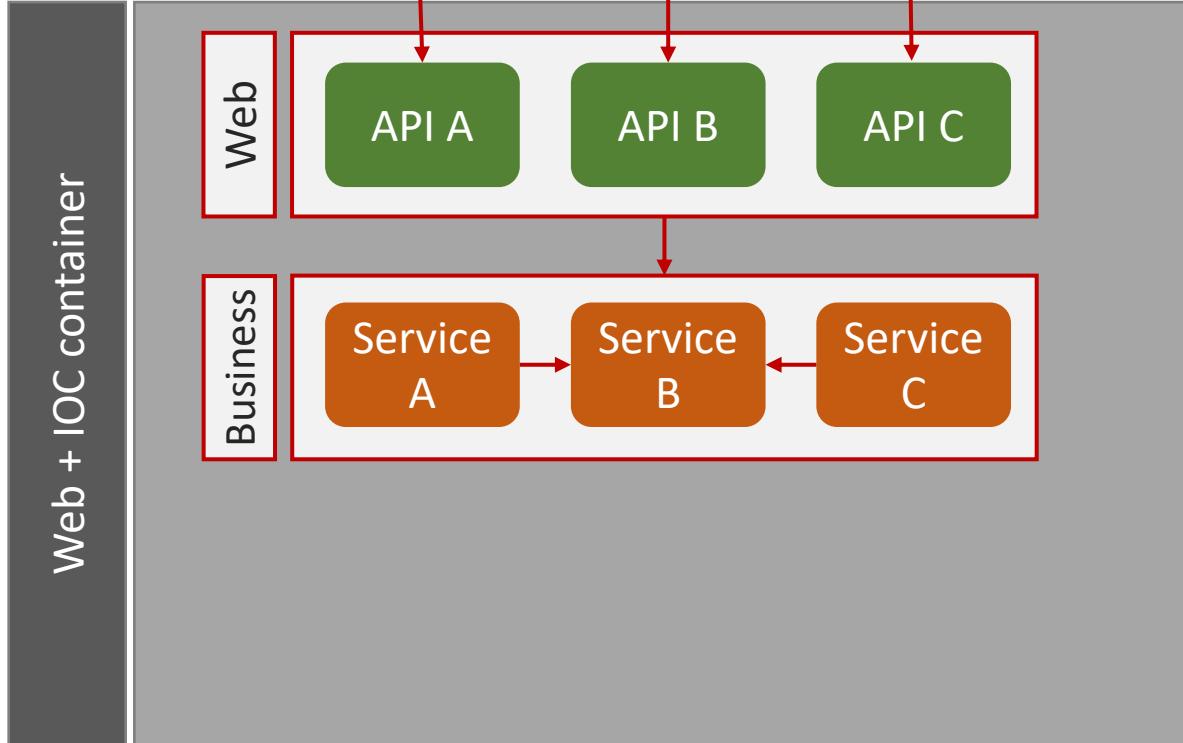
- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architecture Monolithique



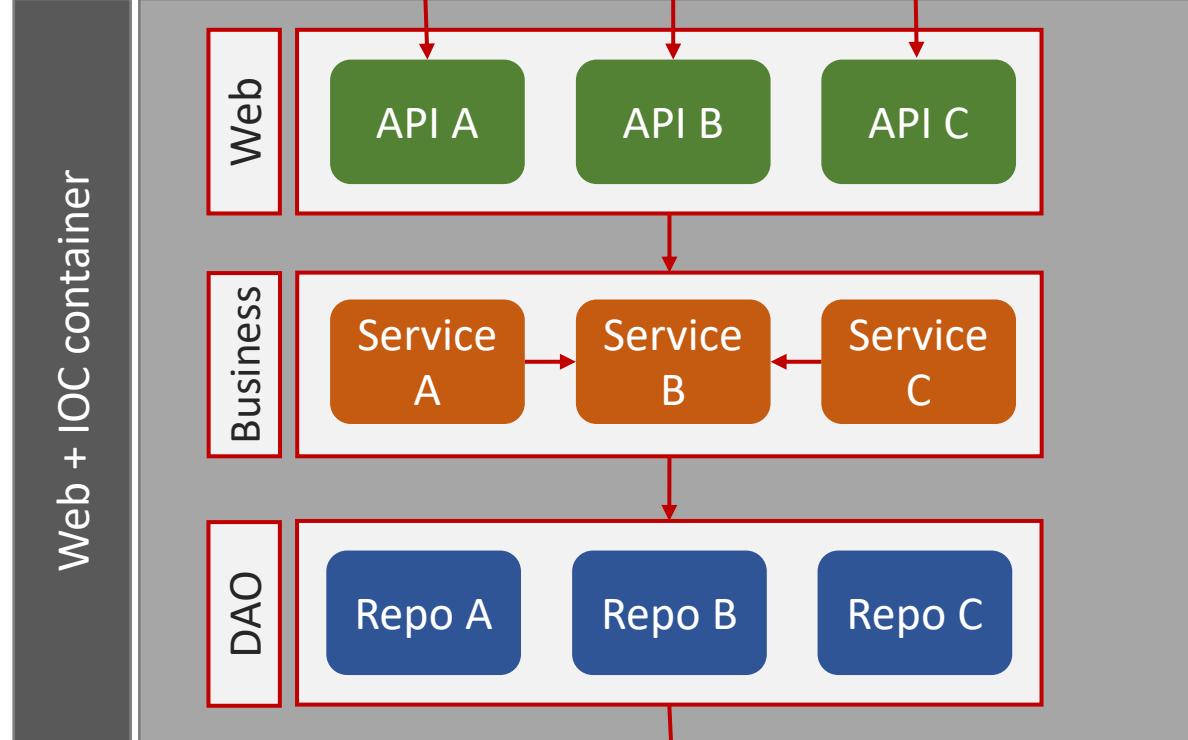
- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architecture Monolithique



- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

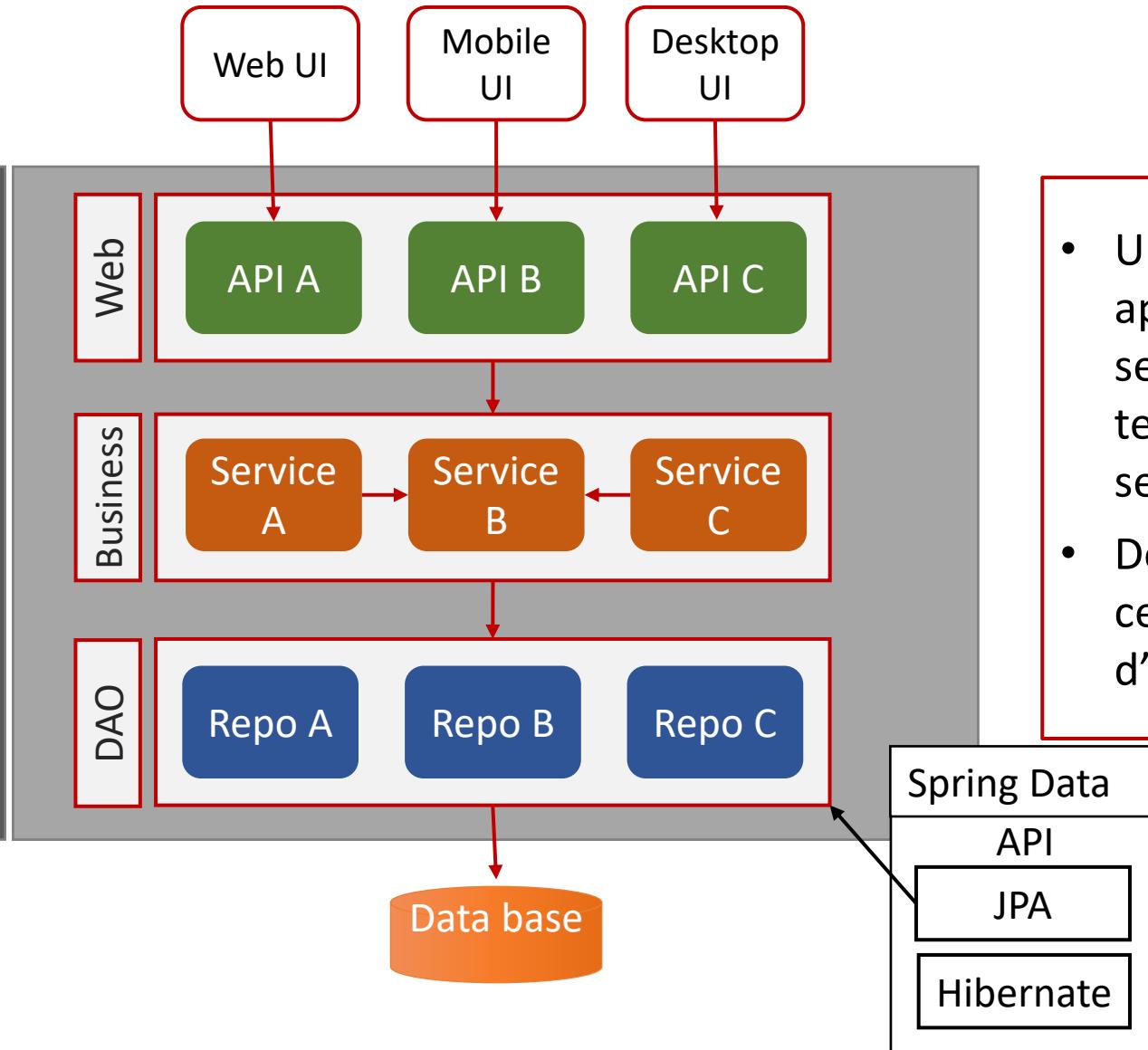
Architecture Monolithique



- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

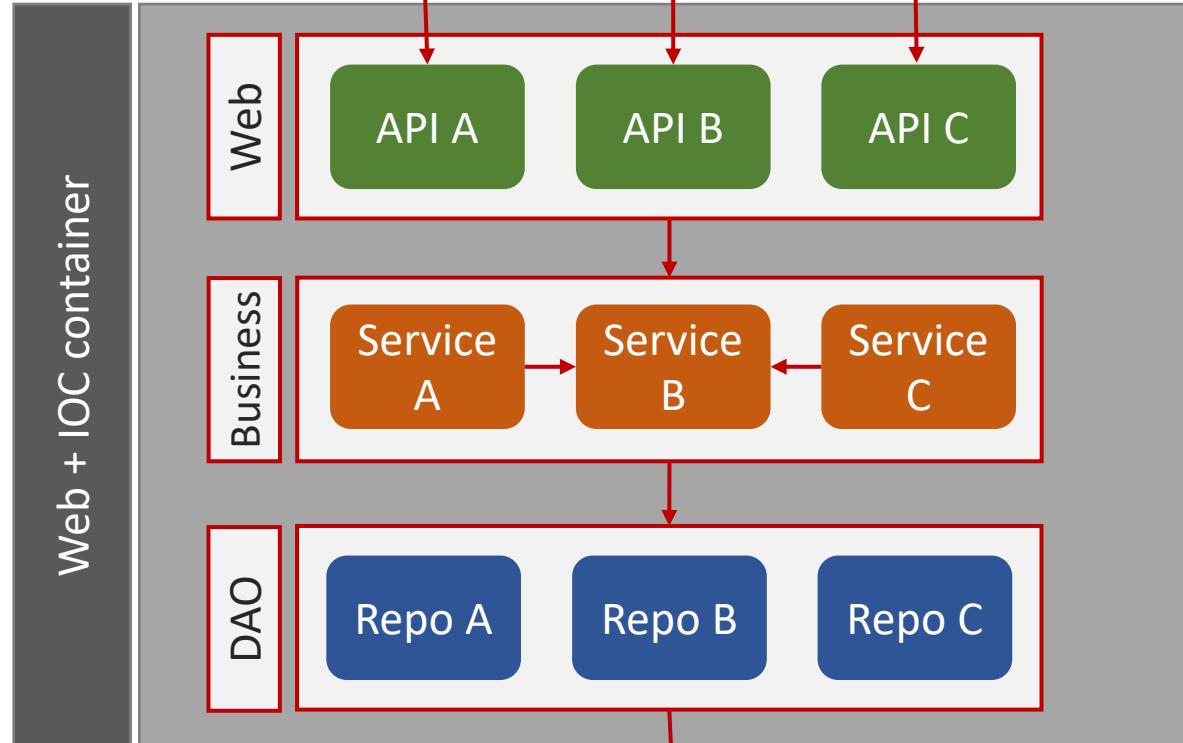
Architecture Monolithique

Web + IOC container



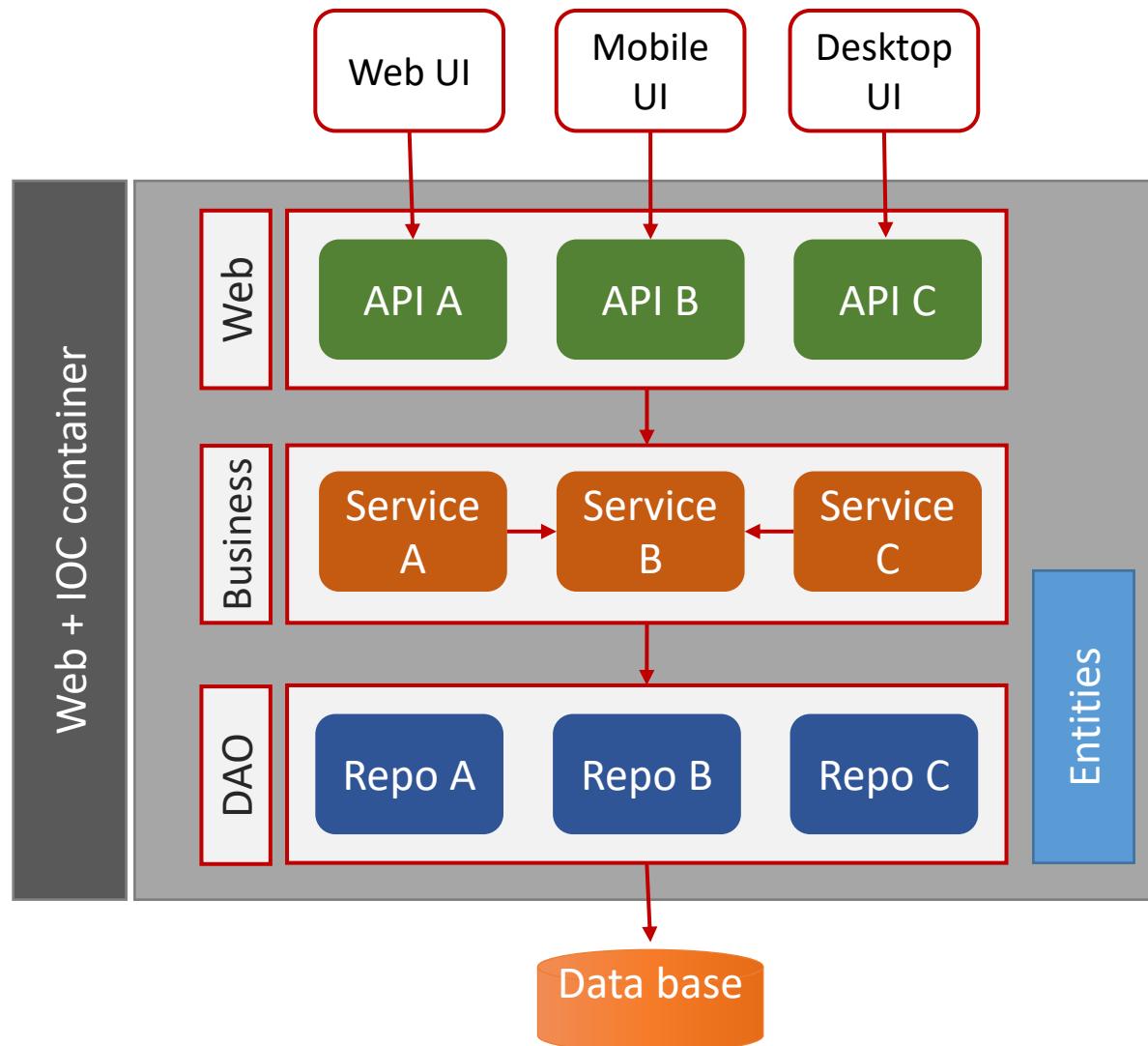
- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architecture Monolithique



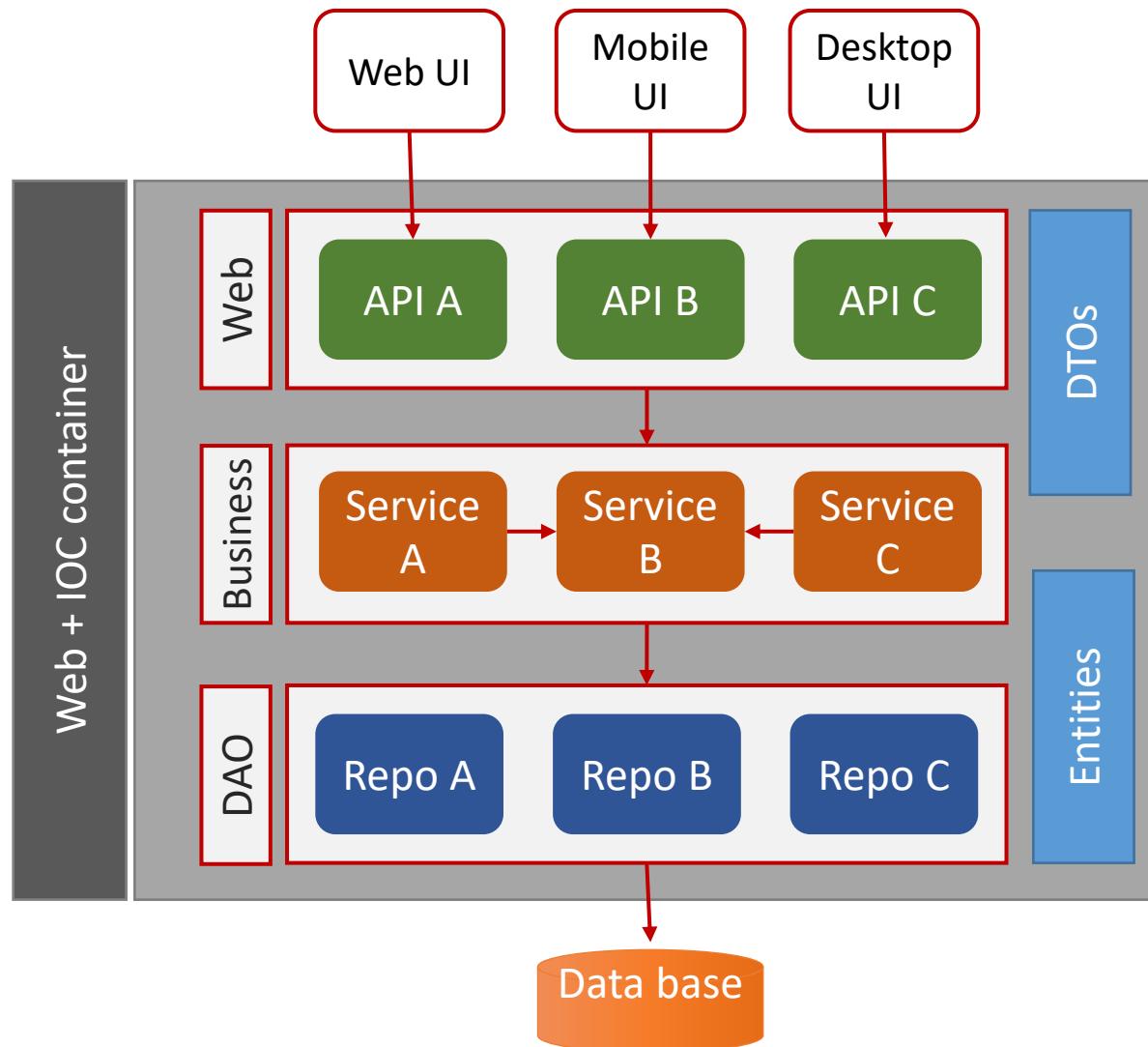
- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architecture Monolithique



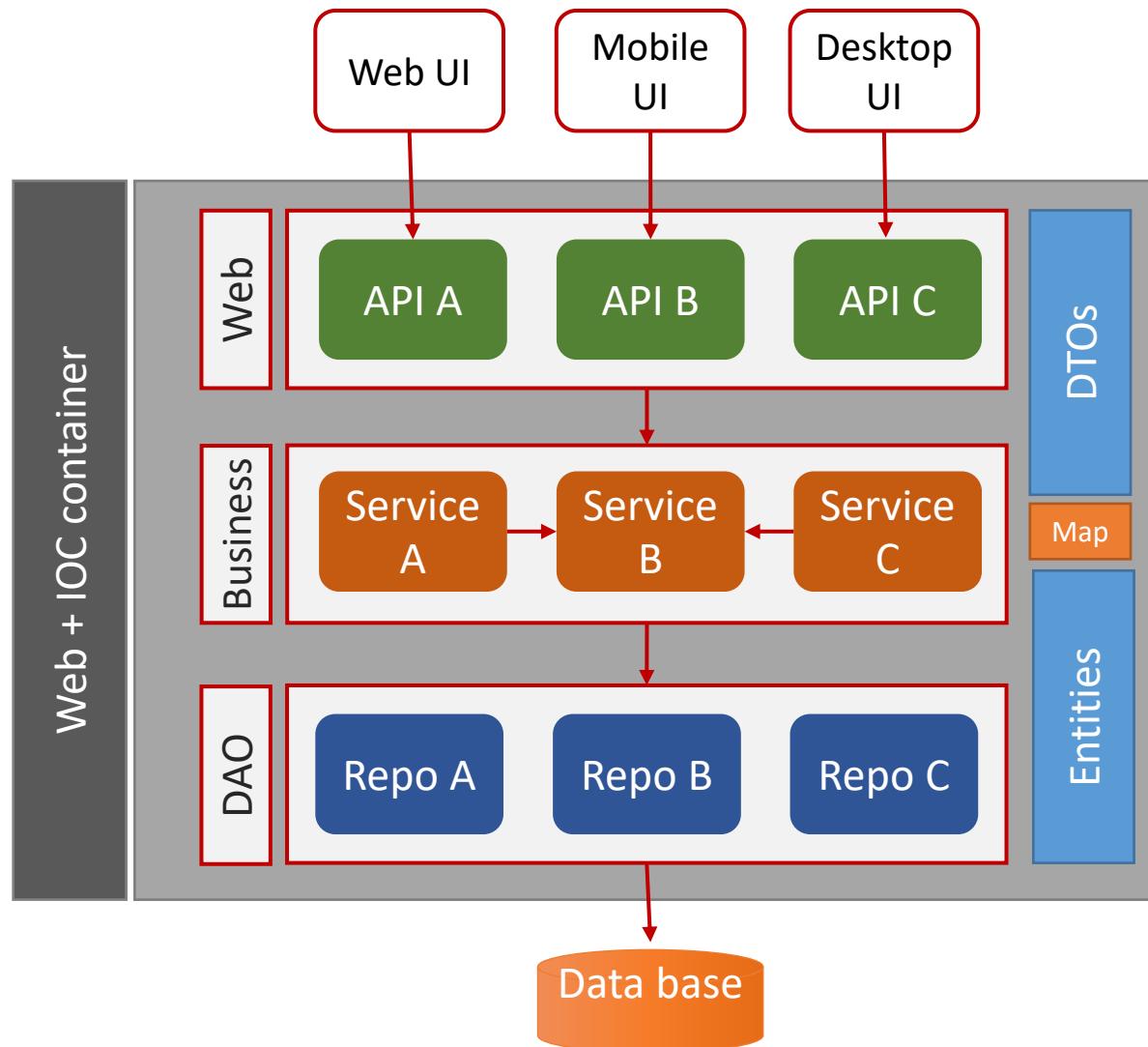
- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architecture Monolithique



- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

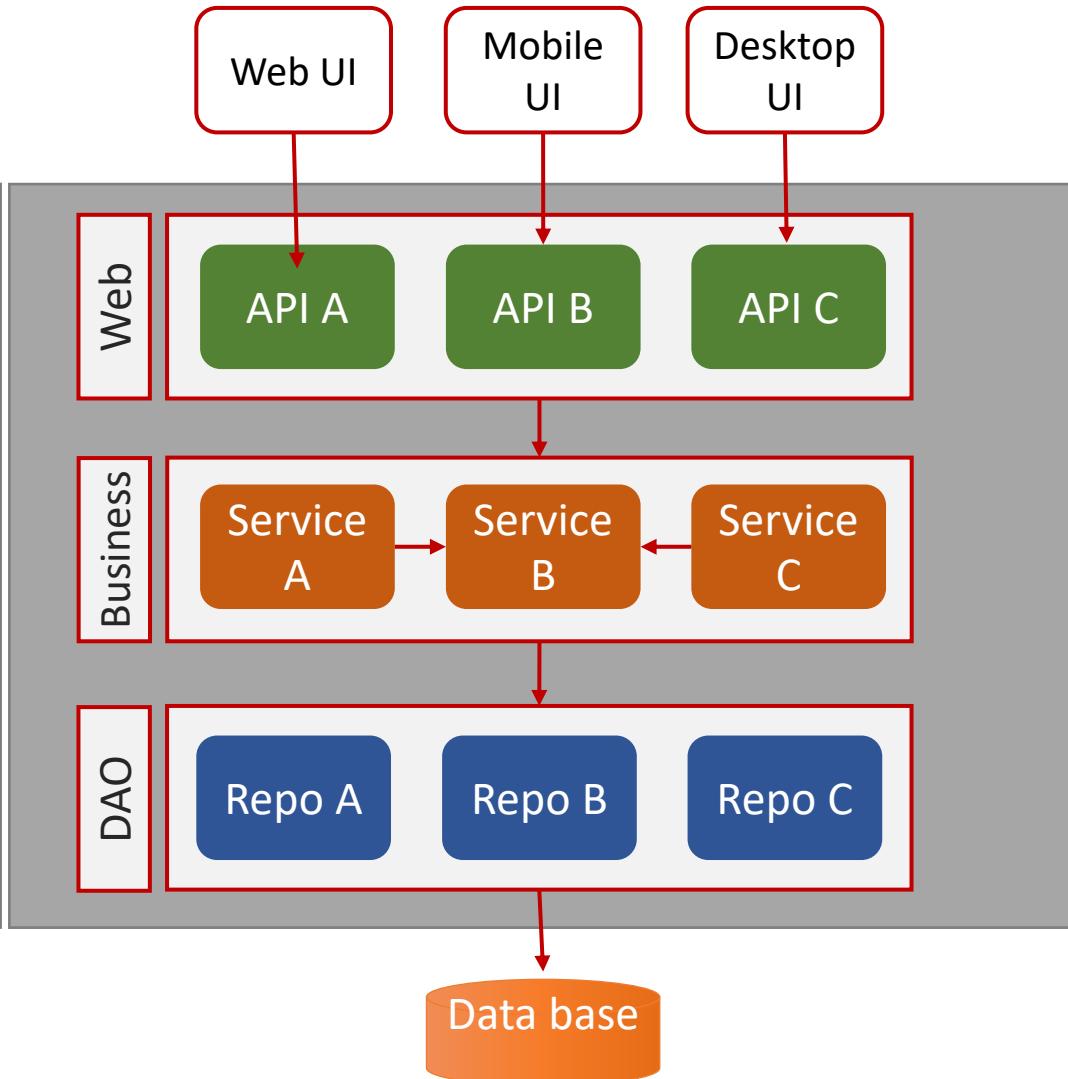
Architecture Monolithique



- Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.
- Développer une seul application qui centralise toutes les fonctionnalités d'un grand problème donné.

Architectures Monolithiques

Web + IOC container

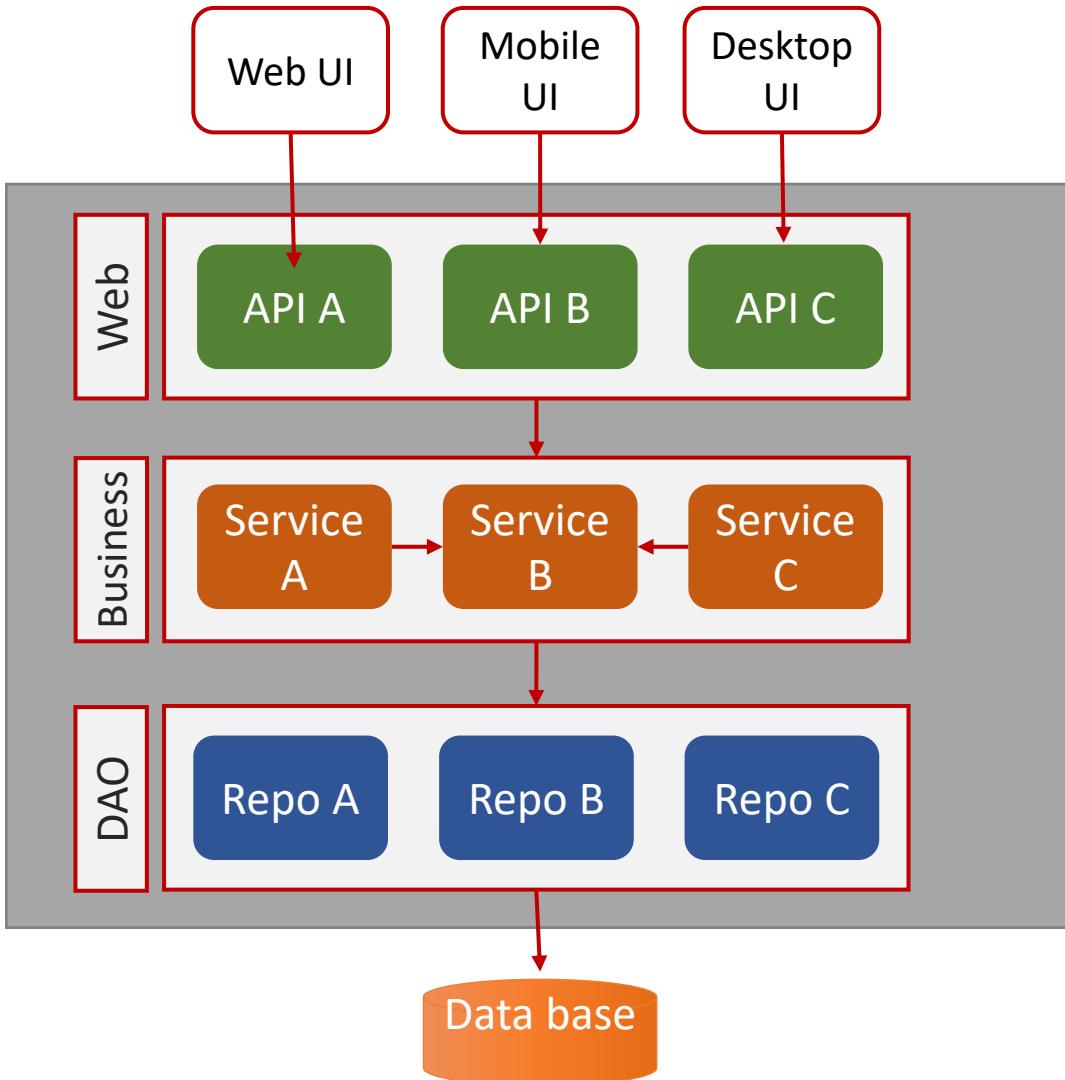


Avantage:

- Simples à développer puisque nos IDE et autres outils sont taxés sur la création d'une seule application.
- Simples à tester: mise en œuvre des tests de bout en bout en lançant simplement l'application et en testant l'interface utilisateur avec Selenium.
- Simples à déployer: Il vous suffit de copier l'application packagée sur un serveur.
- Vous pouvez également faire évoluer l'application en exécutant plusieurs copies derrière un équilibrEUR de charge. Au début du projet, cela fonctionne bien.

Architectures Monolithiques

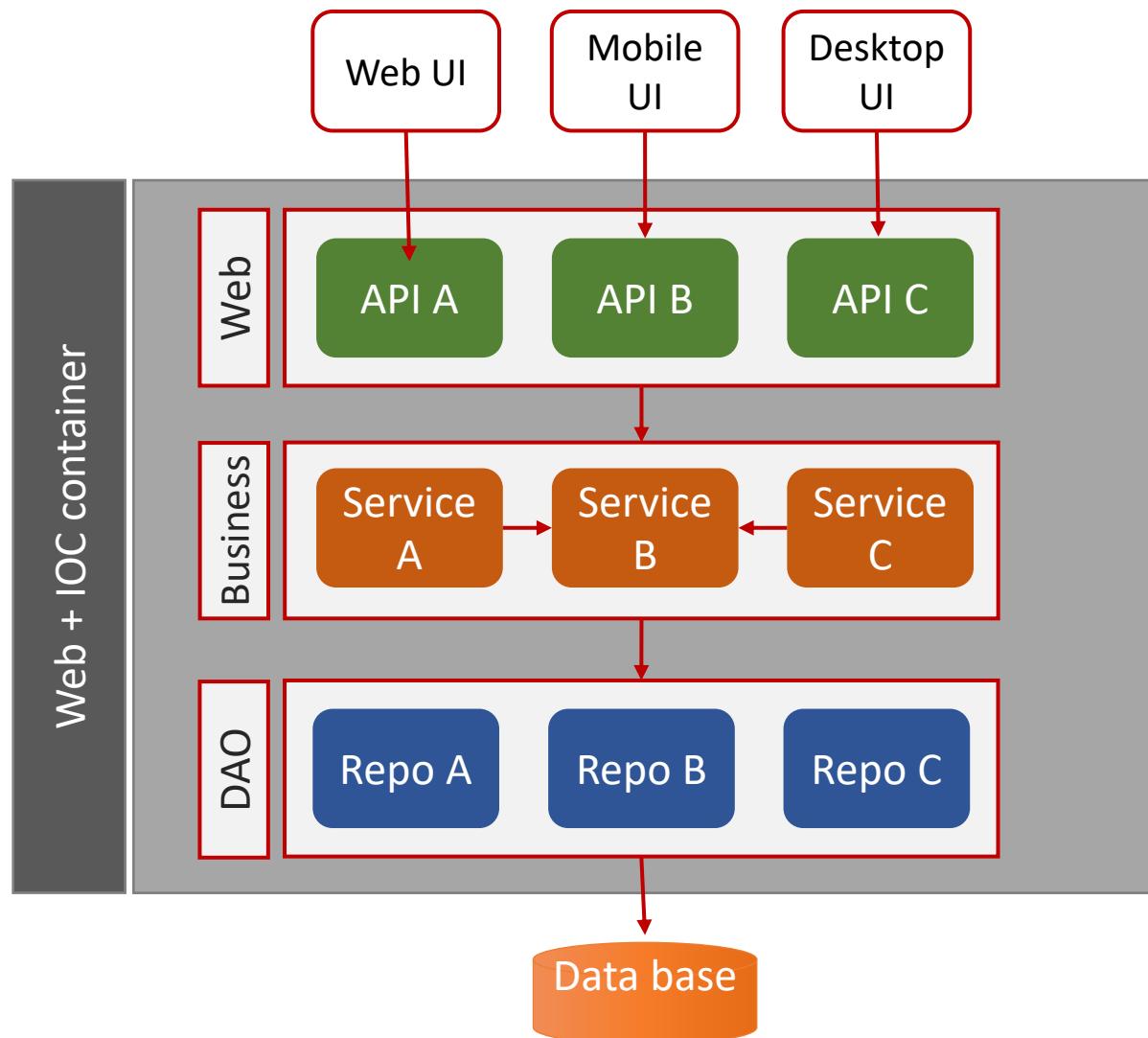
Web + IOC container



Inconvénients:

- Un gros code contenant toutes les fonctionnalités et les différentes couches logicielles
- Une seule grosse compilation et un seul livrable (un gros fichier WAR/JAR/EAR)
- Une seule pile logicielle (Linux, JVM, Tomcat et bibliothèques tierces)
- Approche non alignée avec les méthodes agiles

Contrainte principales des Architectures Monolithiques



- Un gros code contenant toutes les fonctionnalités et les différentes couches logicielles
- Une seule grosse compilation et un seul livrable (un gros fichier WAR/JAR/EAR) . Elles sont réalisées dans une seule technologie.
- Chaque modification nécessite de:
 - Tester les régressions.
 - Redéployer toute l'application.
 - Difficile à faire évoluer au niveau fonctionnel.
- Livraison en bloc (Le client attend beaucoup de temps pour commencer à voir les premières versions).
- Mise en production prend beaucoup de temps.
- Difficile à tester.
- **Performance (scalabiliter)**
- Etc ...

Pourquoi une nouvelle Architecture ? C'est quoi la problématique ?

- Les clients de plus en plus exigeants : phénomène international
- Minimiser le TTM : Time To Market
- Les solutions deviennent de plus en plus complexes en termes d'architecture et de fonctionnalités
- Fort couplage → Fiabilité de l'application lors d'un Bug dans un module fonctionnel mais qui fait cracher toute l'application
- Complexité des déploiements et des mises à jour, vue la dépendance forte.
- Actuellement : l'état de l'art pour les applications SaaS consiste à « pusher » les modifications en production plusieurs fois par jour
- Difficultés techniques et nécessité de beaucoup de ressources pour les tests, notamment les tests de non-régression.
- Problème de montée en charge et de performance « Scalabilité »
- Hétérogénéités techniques : Plateformes et langages de développement

Architectures Micro-Service

Généralités et Définition:

- Premières discussions autour du terme microservice en 2011 lors d'un workshop sur les architectures logicielles
- Les architectures microservices sont une évolution, une spécialisation des architectures orientées services (SOA)
- Les microservices constituent une approche moderne du logiciel dans laquelle le code d'application est fourni en petits éléments gérables, indépendants des autres.
- Les architectures de microservices sont la « nouvelle norme ». La création de petites applications autonomes et prêtes à être exécutées peut apporter une grande flexibilité et une résilience accrue au code.

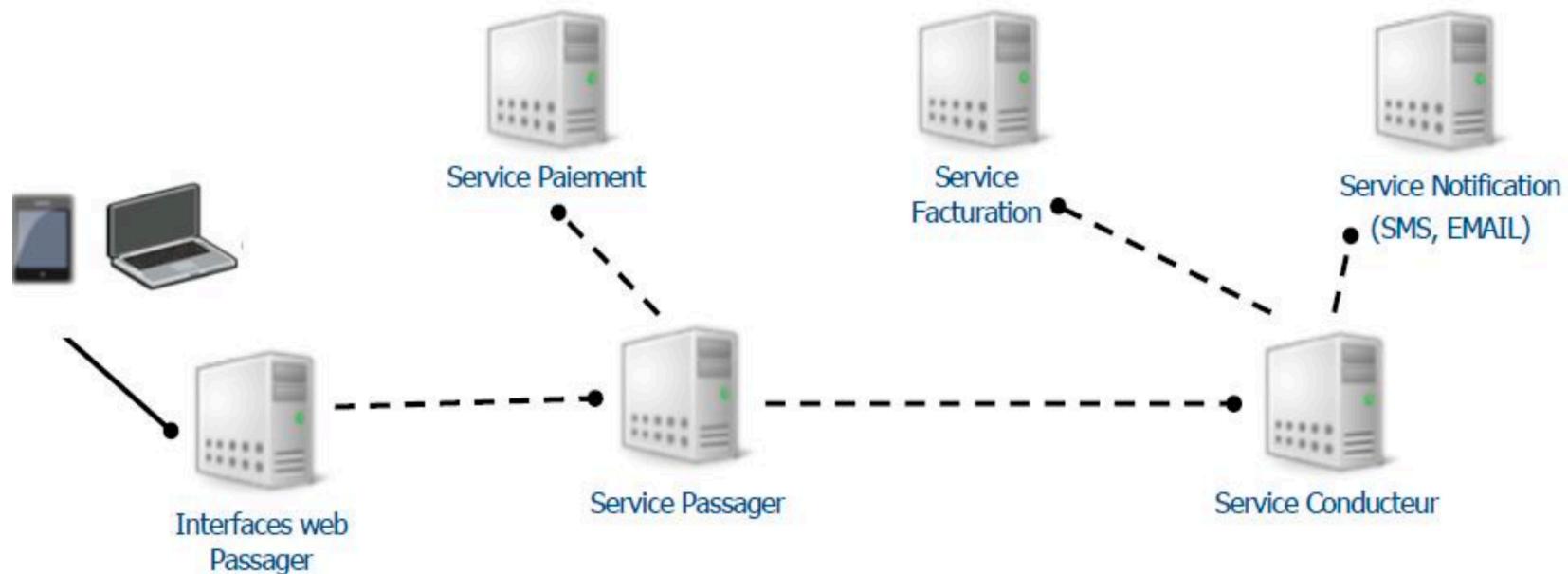
Aspects caractérisant un microservice:

- | | | |
|---------------------------|-------------------------------------|-------------------|
| 1. Fonctionnalité unique; | 2. Flexibilité technologie | 3. Equipe réduite |
| 4. Déploiement ciblé | 5. Montée en charge « scalabilité » | 6. Tests facilité |

Architectures Micro-Service

1 FONCTIONNALITÉ UNIQUE

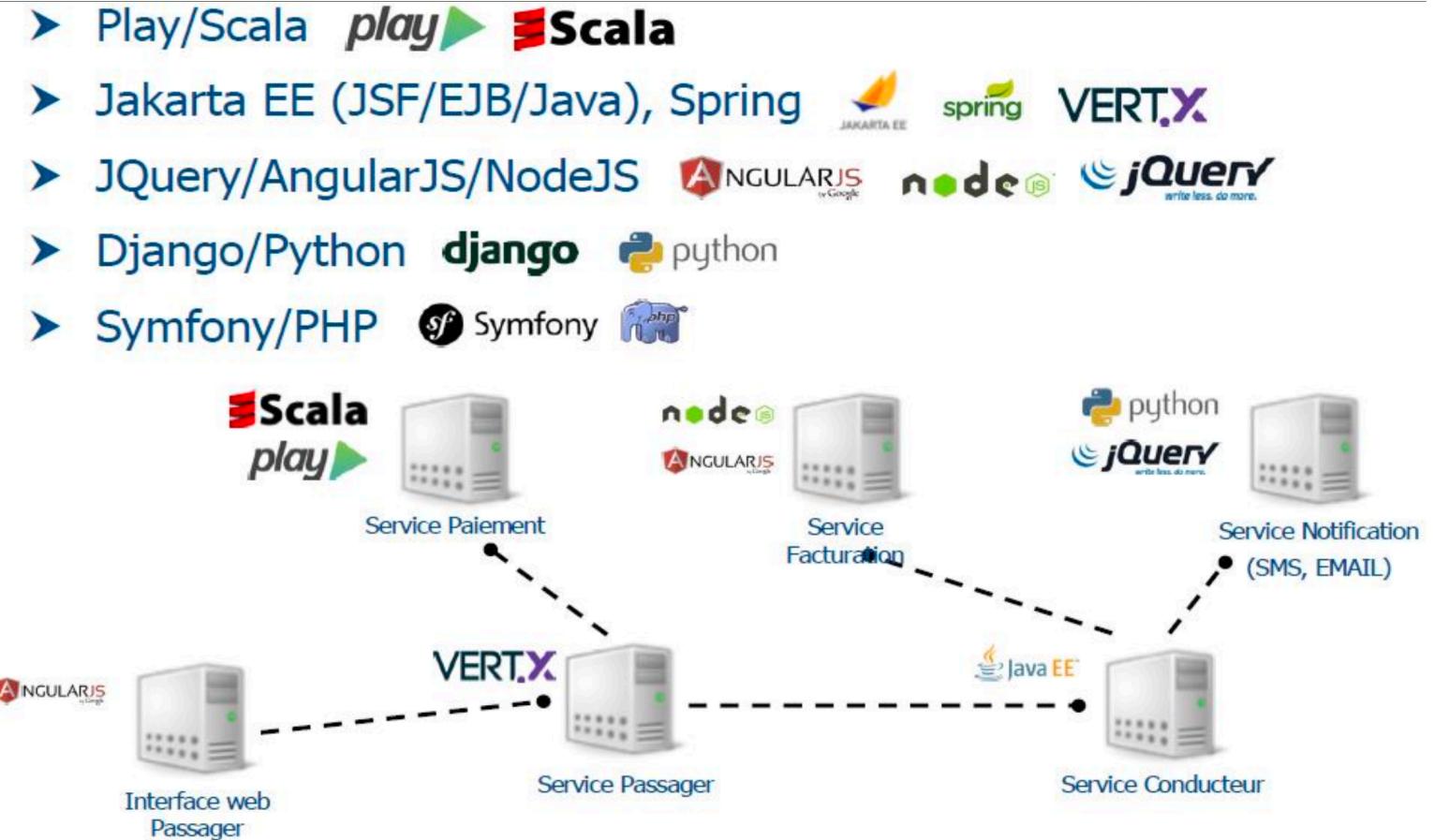
- Un microservice doit réaliser une seule fonctionnalité de l'application globale
- Un microservice peut contenir toutes les couches logicielles (IHM, middleware et base de données)
- Un microservice possède un contexte d'exécution séparé des autres (exemple : machine virtuelle ou conteneur)



Architectures Micro-Service

2 FLEXIBILITÉ TECHNOLOGIQUE

- Utiliser les bons langages et frameworks selon la fonctionnalité à réaliser

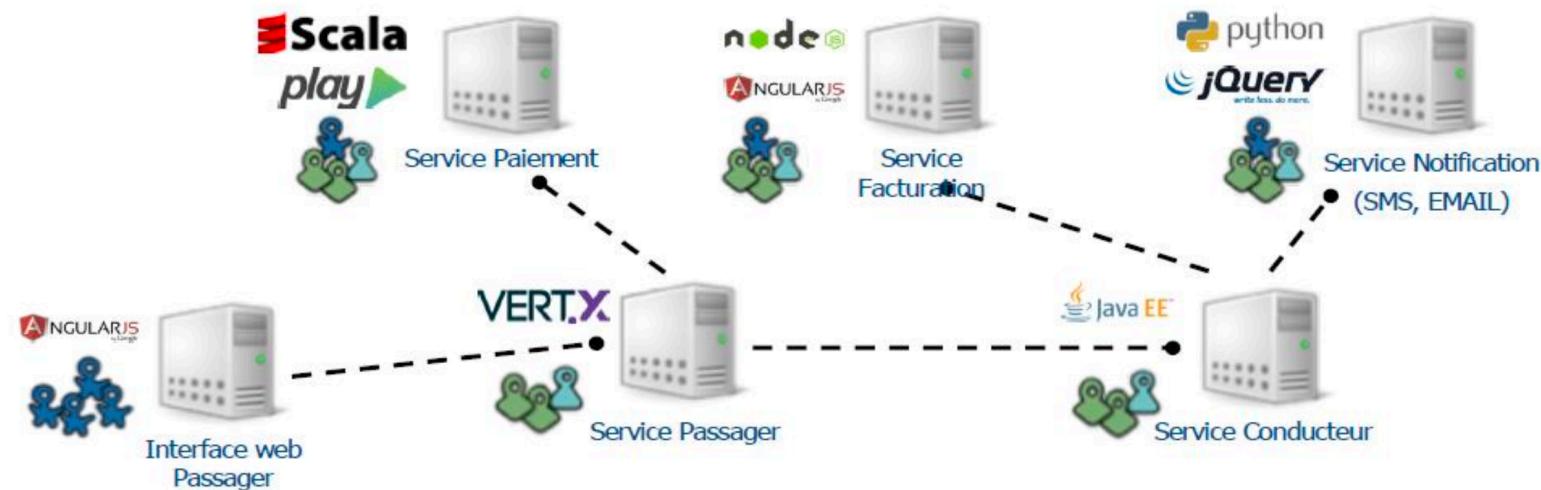


Architectures Micro-Service

3 EQUIPE RÉDUITE

- Chaque microservice à sa propre équipe de développement
- L'équipe de développement orientée fonctionnalité est réduite et pluridisciplinaire (Full Stack)

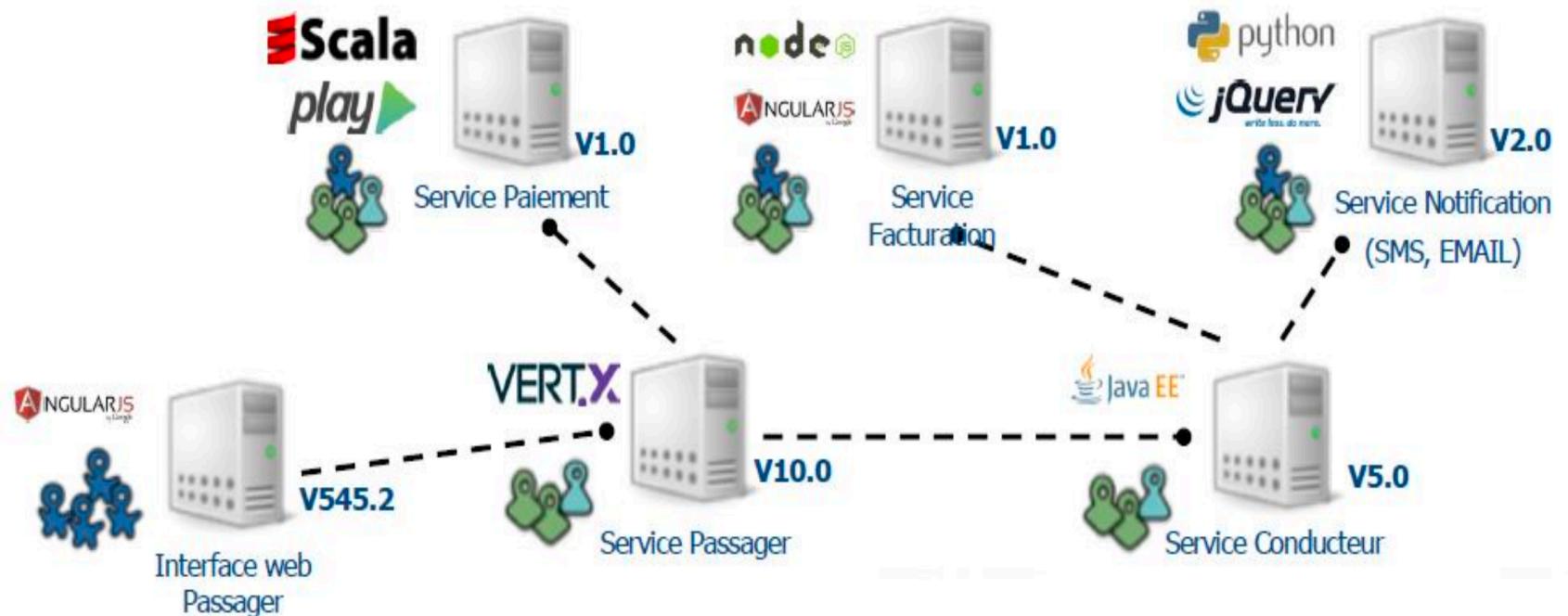
-  Développeurs backend
-  Développeurs web
-  Administrateurs bases de données



Architectures Micro-Service

4 DÉPLOIEMENT CIBLÉ

- Evolution d'une certaine partie sans tout redéployer
- Tests fonctionnels et techniques ciblés
- Tests de non-régression allégés
- Un seul livrable à partir d'un seul code source
- Moins de risques
- Moins d'impacts
- Plus rapide

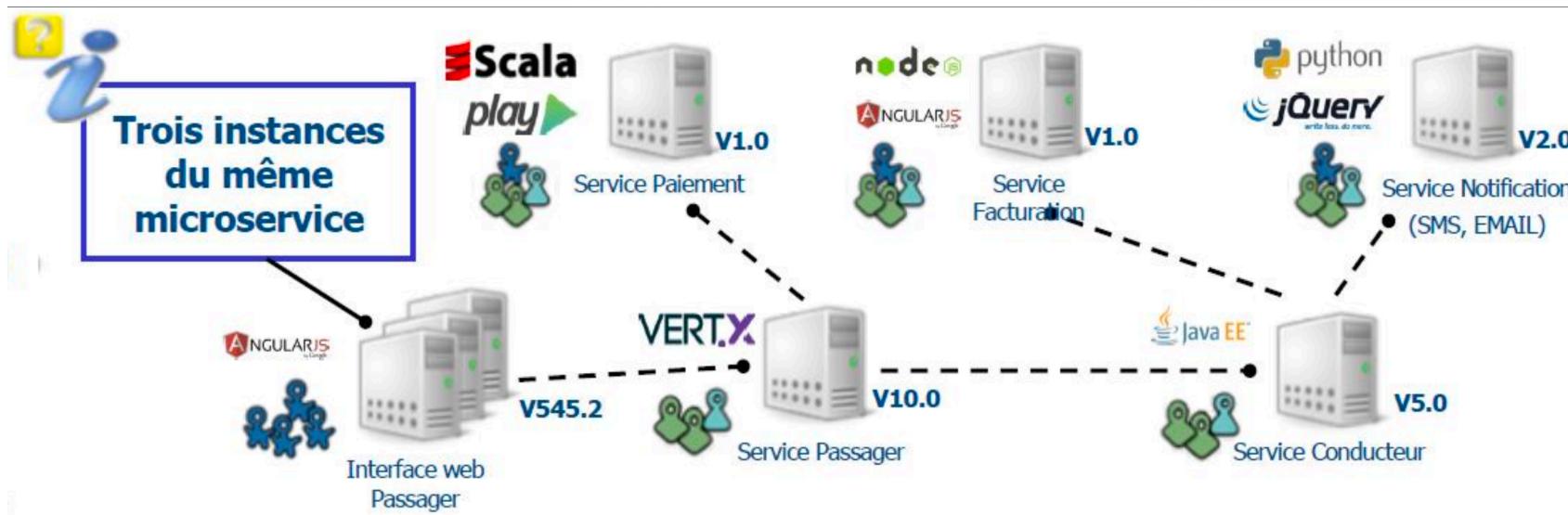


Architectures Micro-Service

5 MONTÉE EN CHARGE / SCALABILITÉ

Forte sollicitation sur un microservice ?

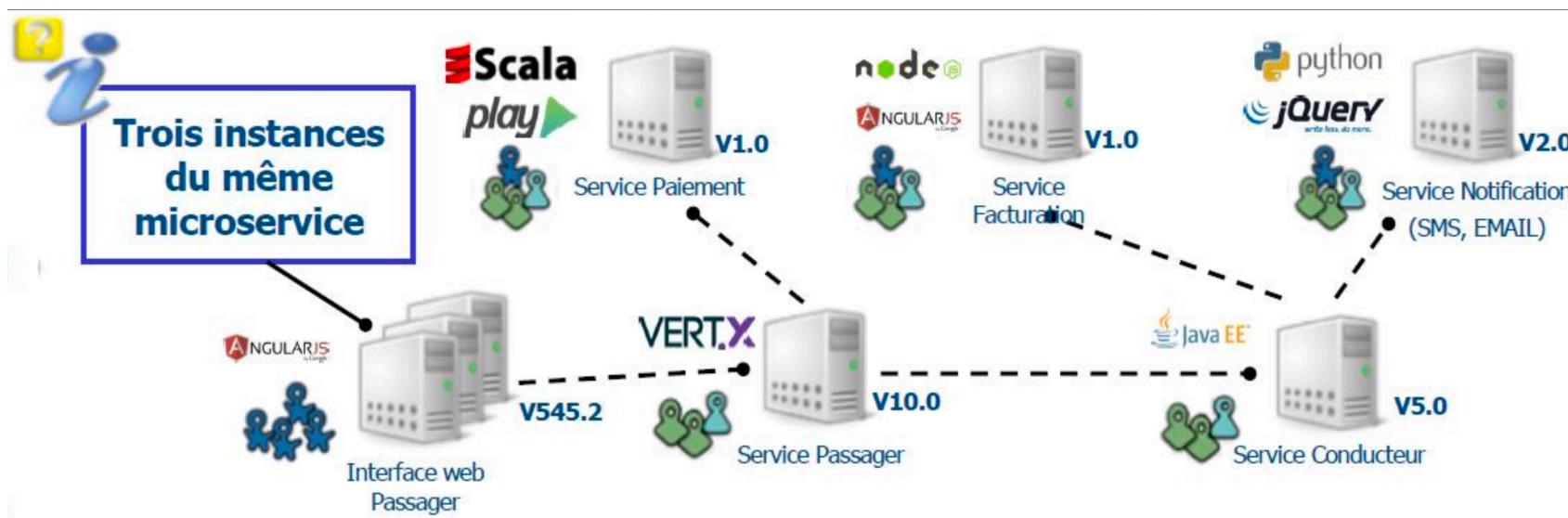
- Site web : Amazon en période de « Black Friday »
- Applications bancaires lors des périodes des fêtes pour le retrait d'argent
- Batch : compression de vidéos
- Puisque chaque fonctionnalité est isolée, il y'a la possibilité de multiplier le nombre d'instances d'un microservice



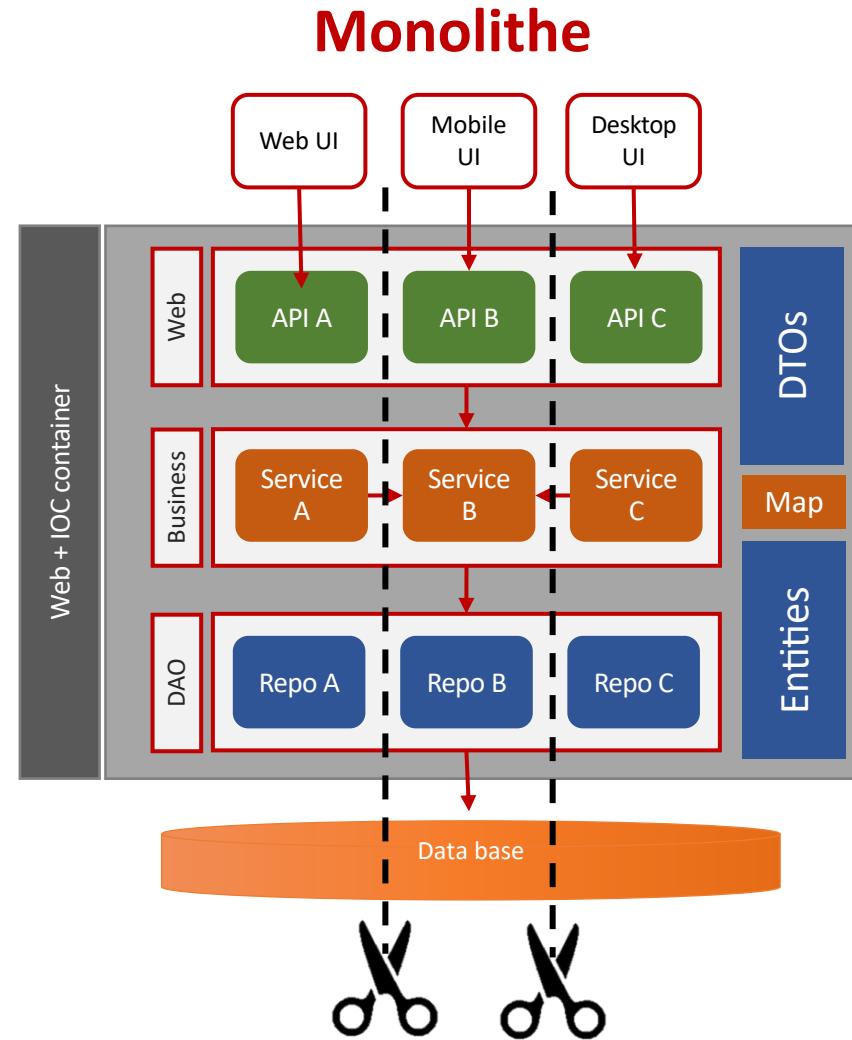
Architectures Micro-Service

6 MONTÉE EN CHARGE / SCALABILITÉ : CUBE DE SCALABILITÉ

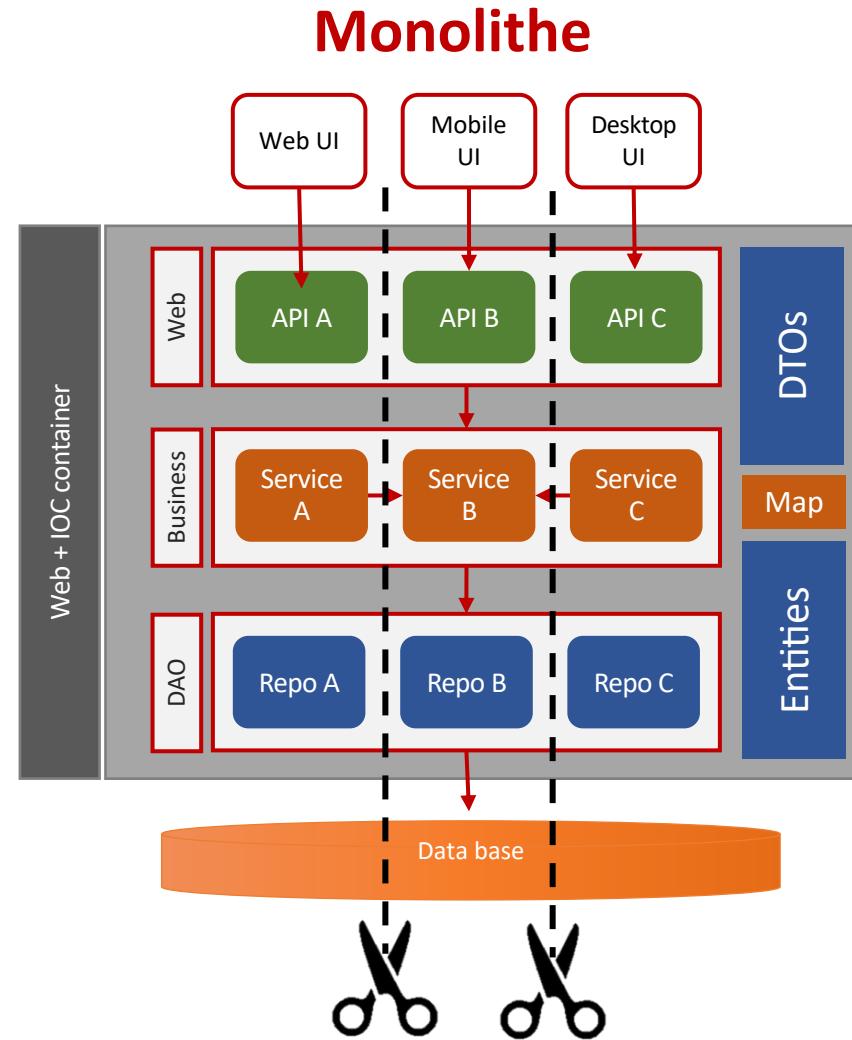
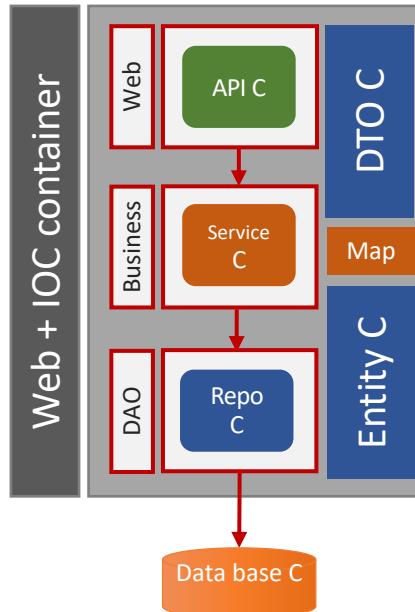
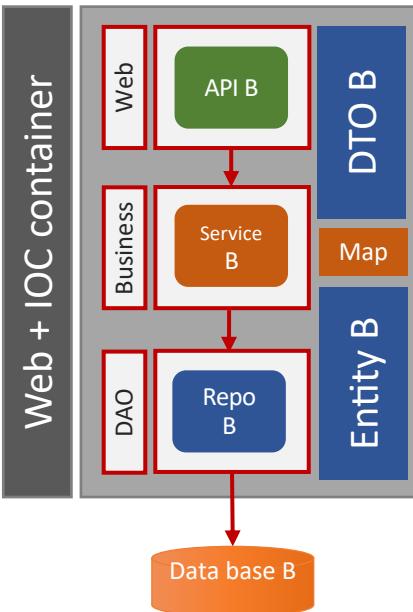
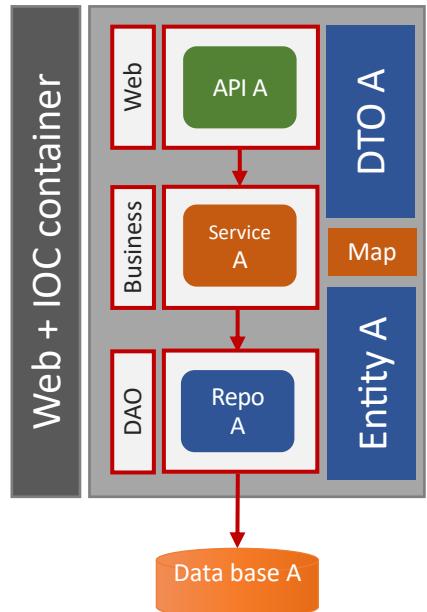
Cloner les micro-services qui ont une forte sollicitation (nécessite un Loadbalancer)



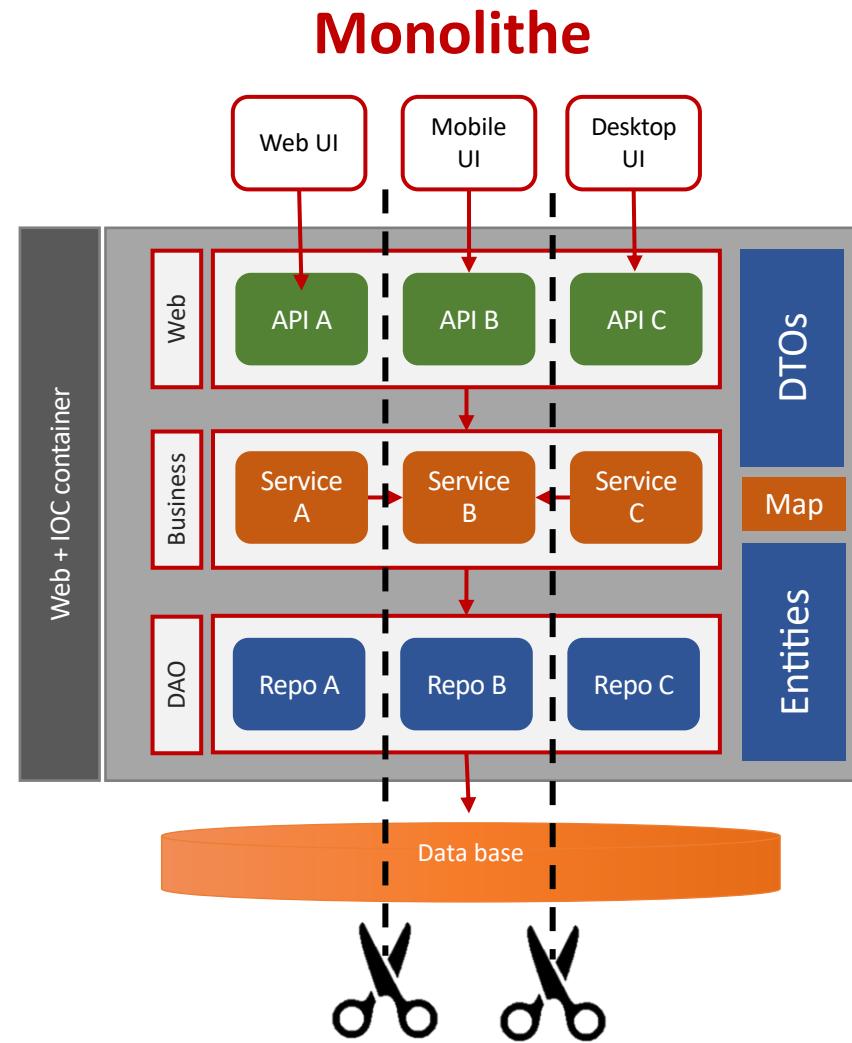
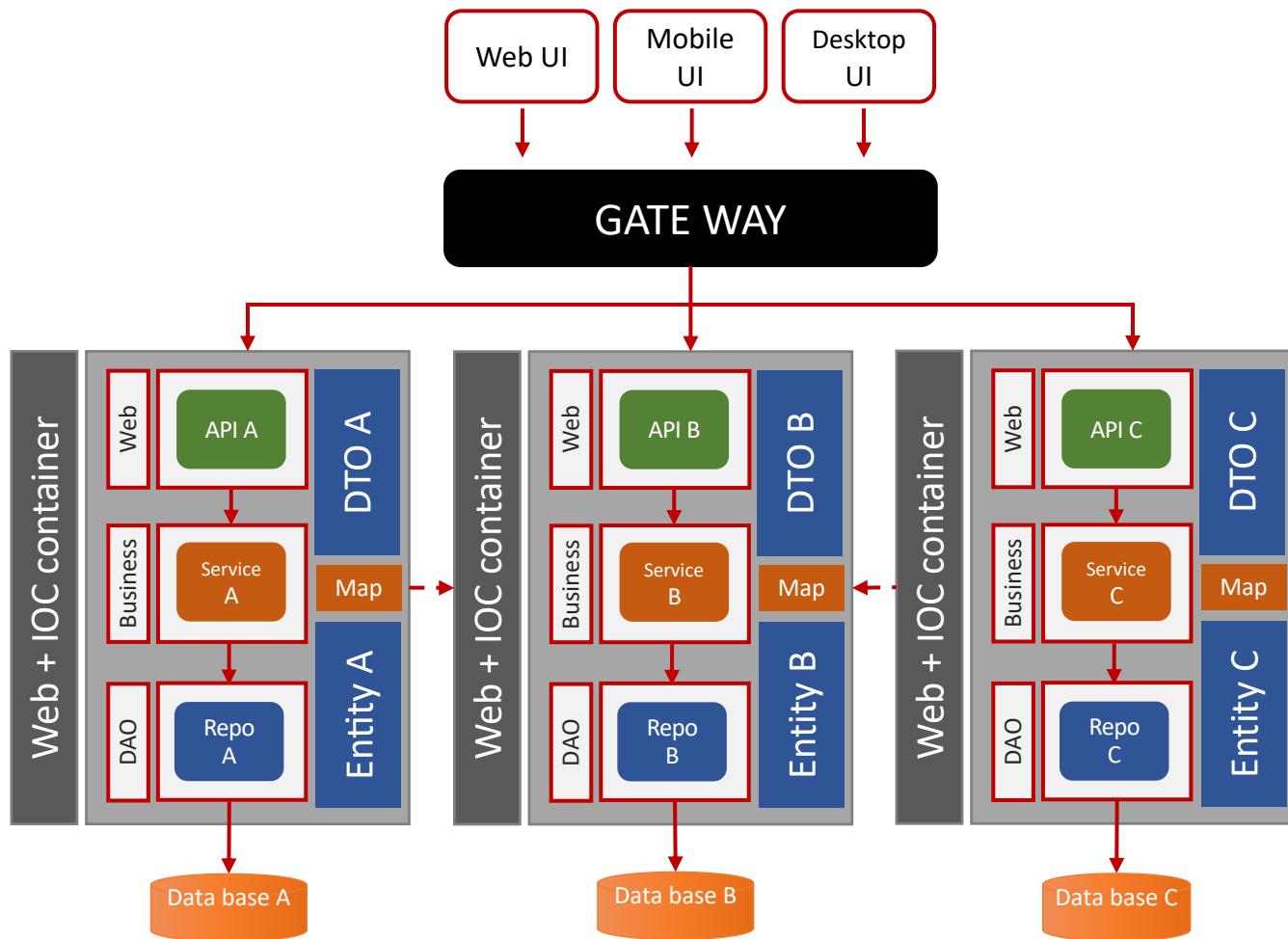
Architectures Micro-Service



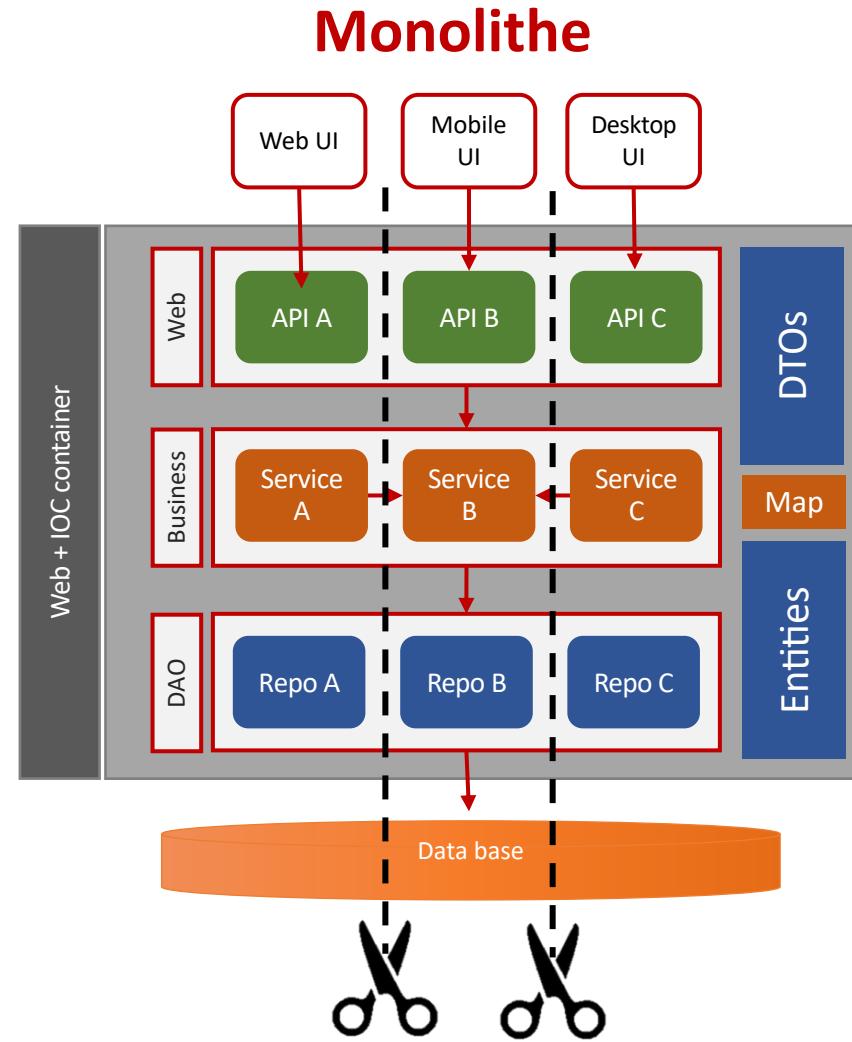
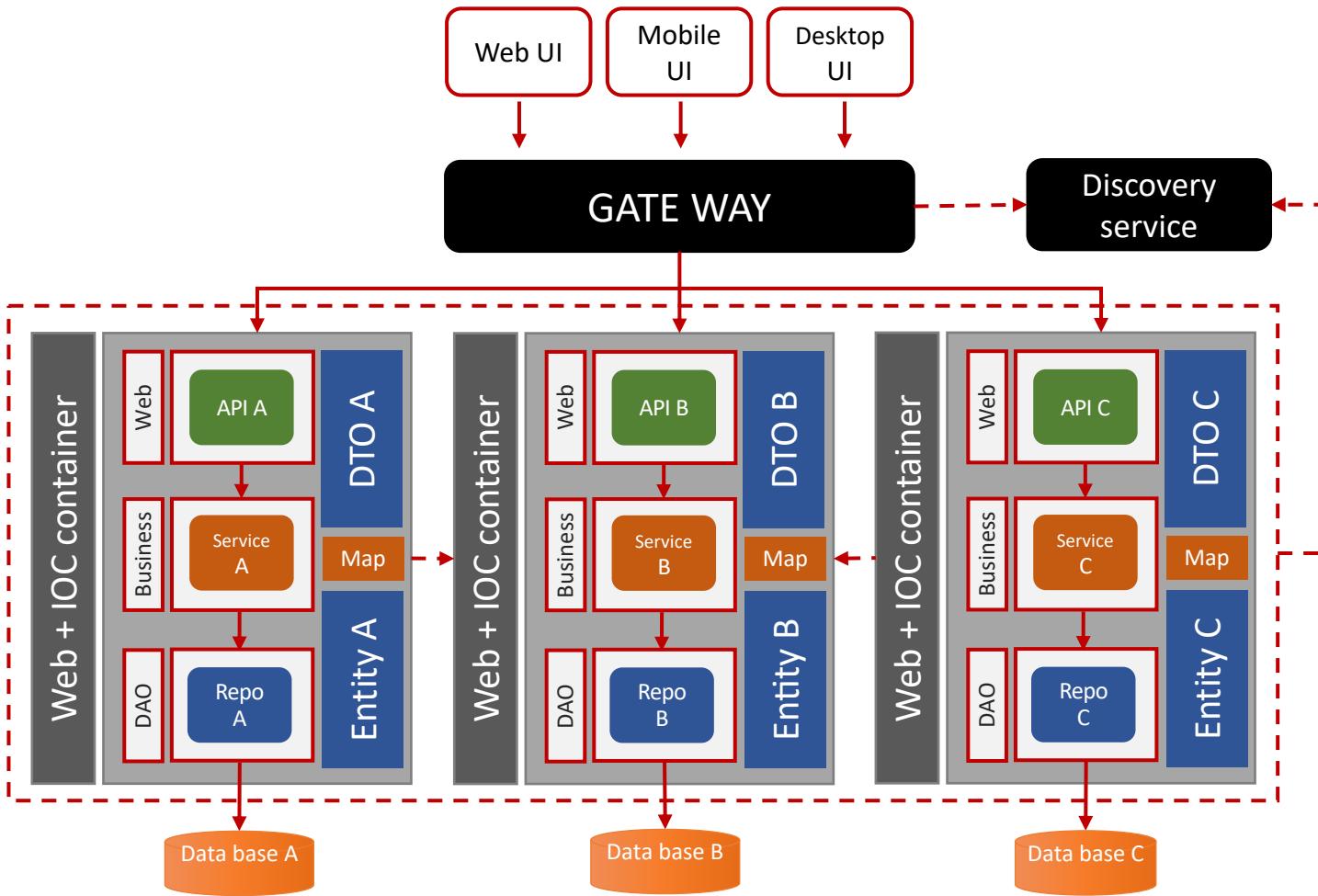
Architectures Micro-Service



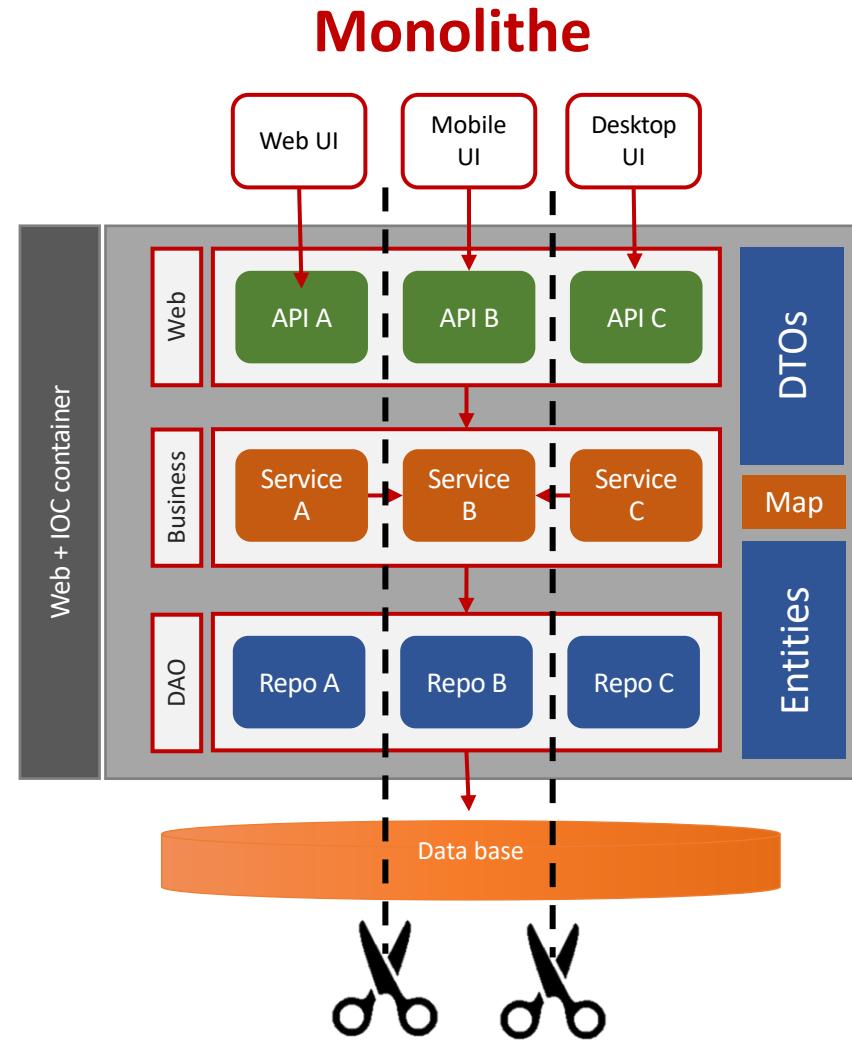
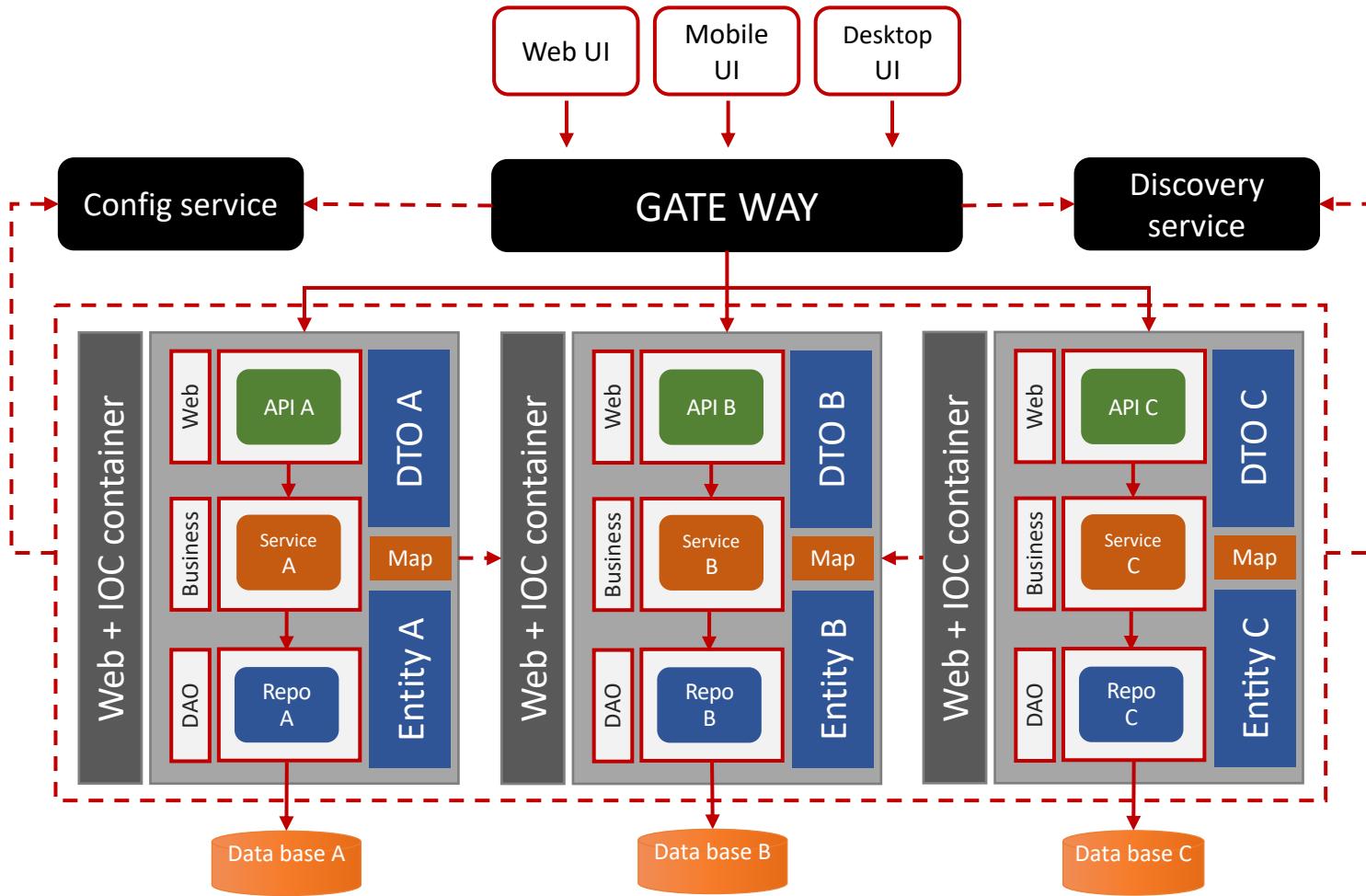
Architectures Micro-Service



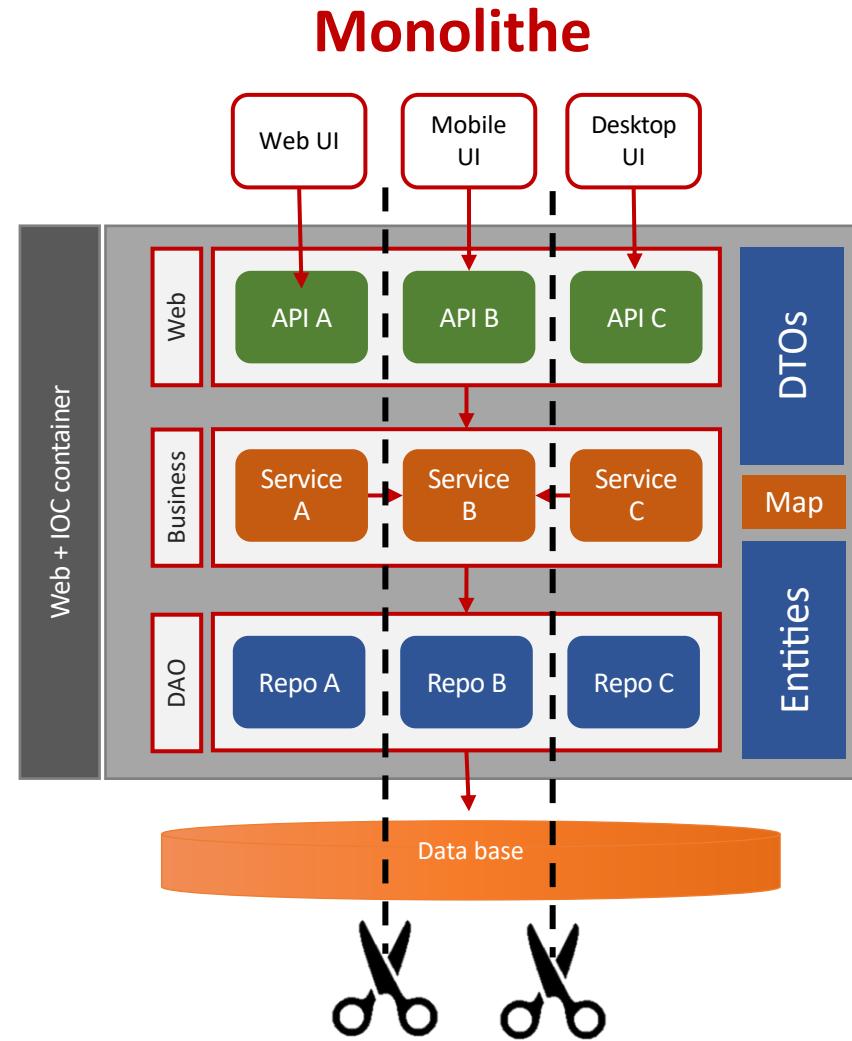
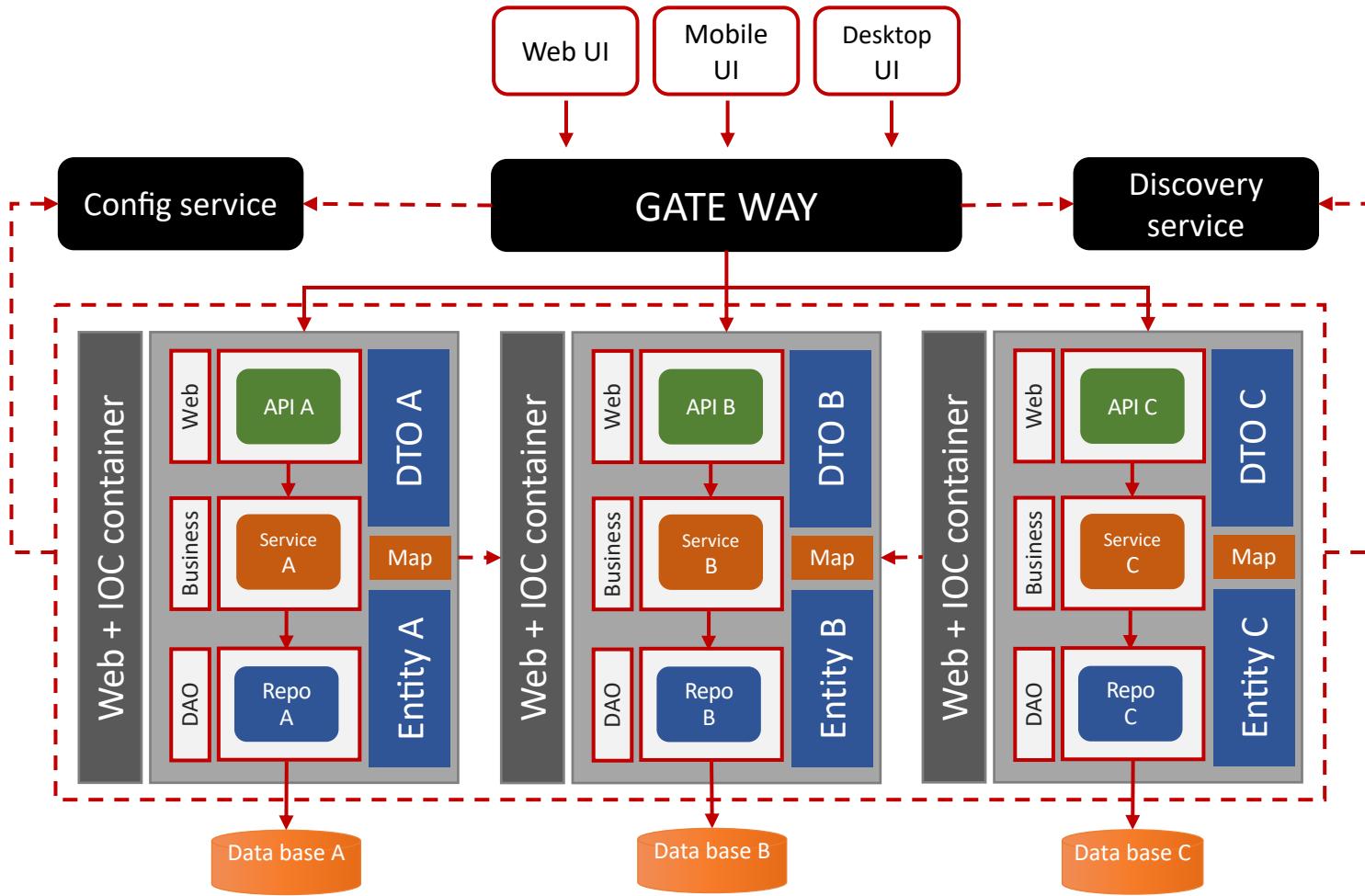
Architectures Micro-Service



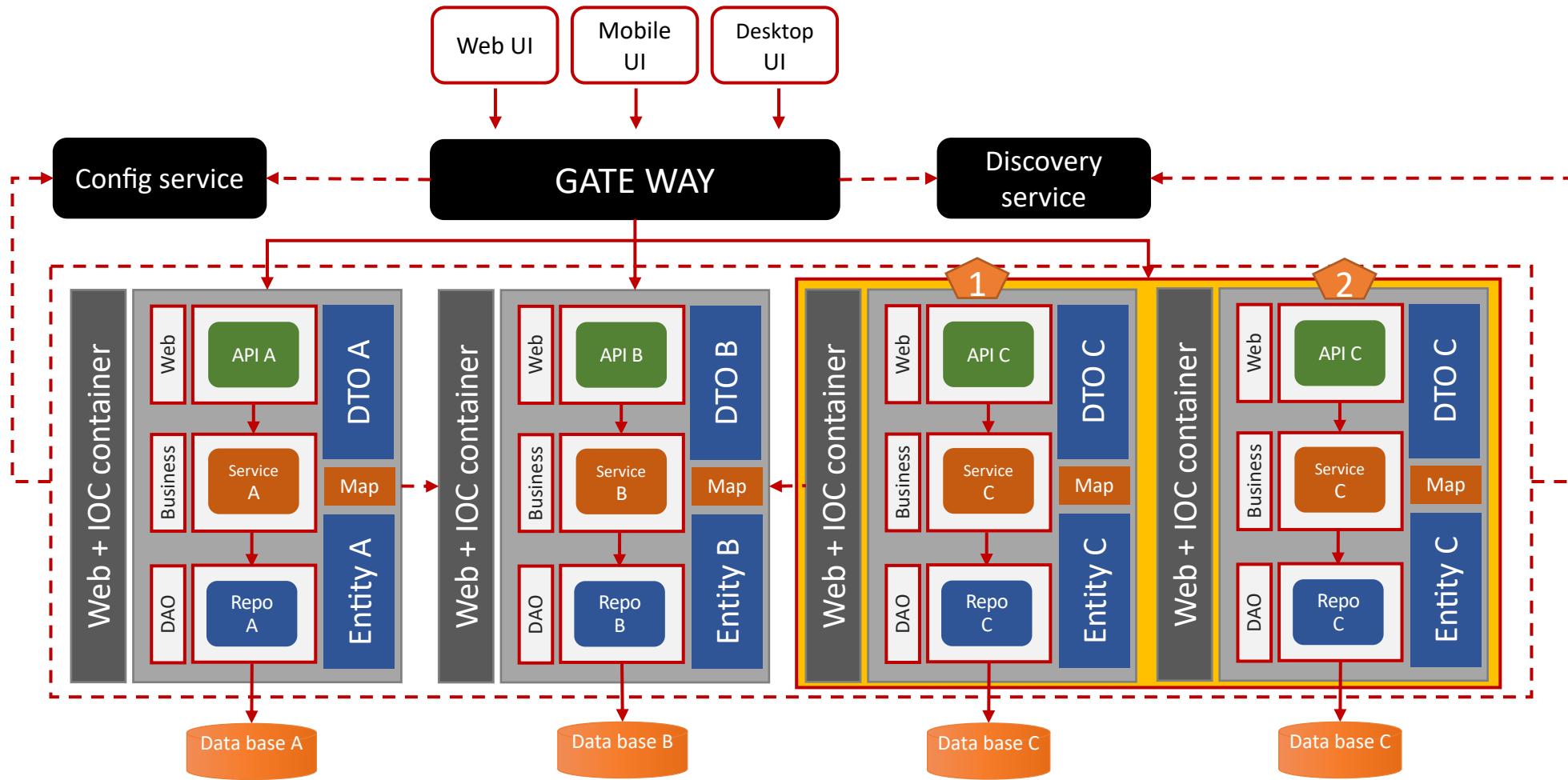
Architectures Micro-Service



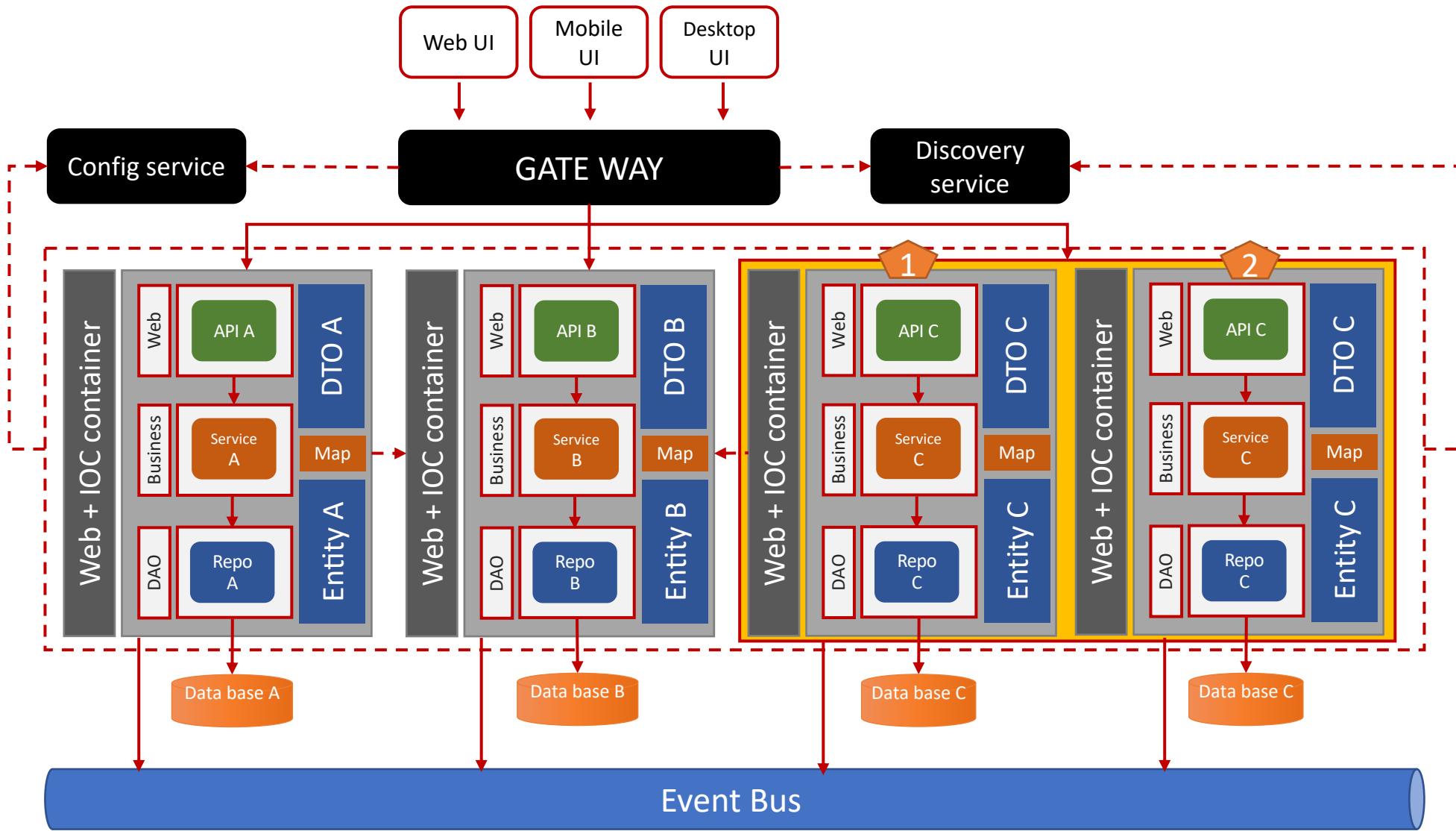
Architectures Micro-Service



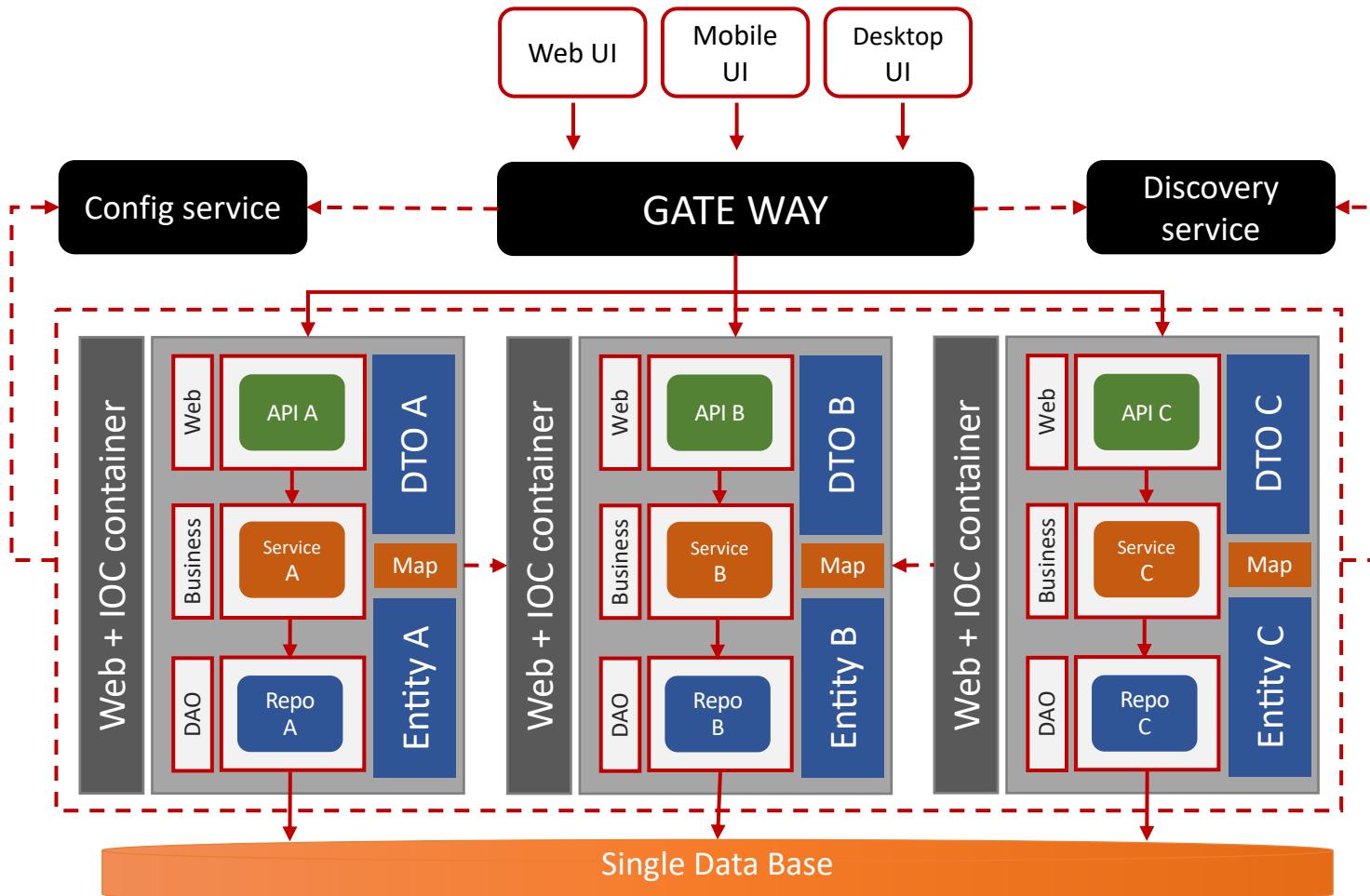
Architectures Micro-Service



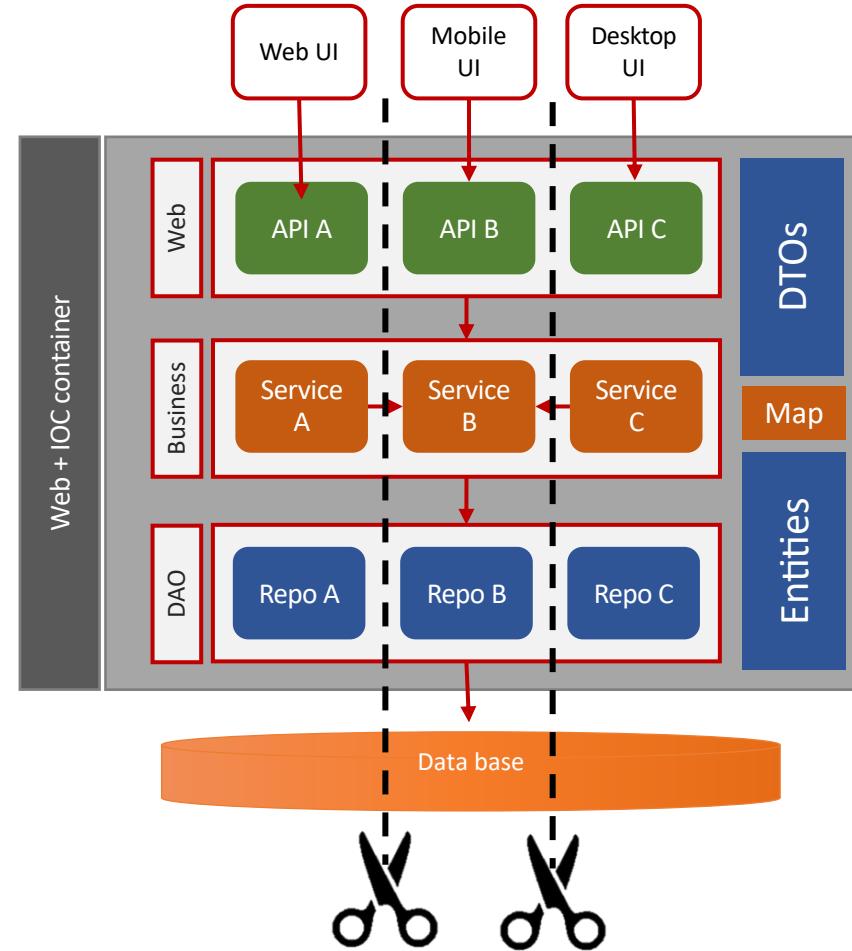
Architectures Micro-Service



Architectures Micro-Service Single Data Base (Monolithe Distribué)

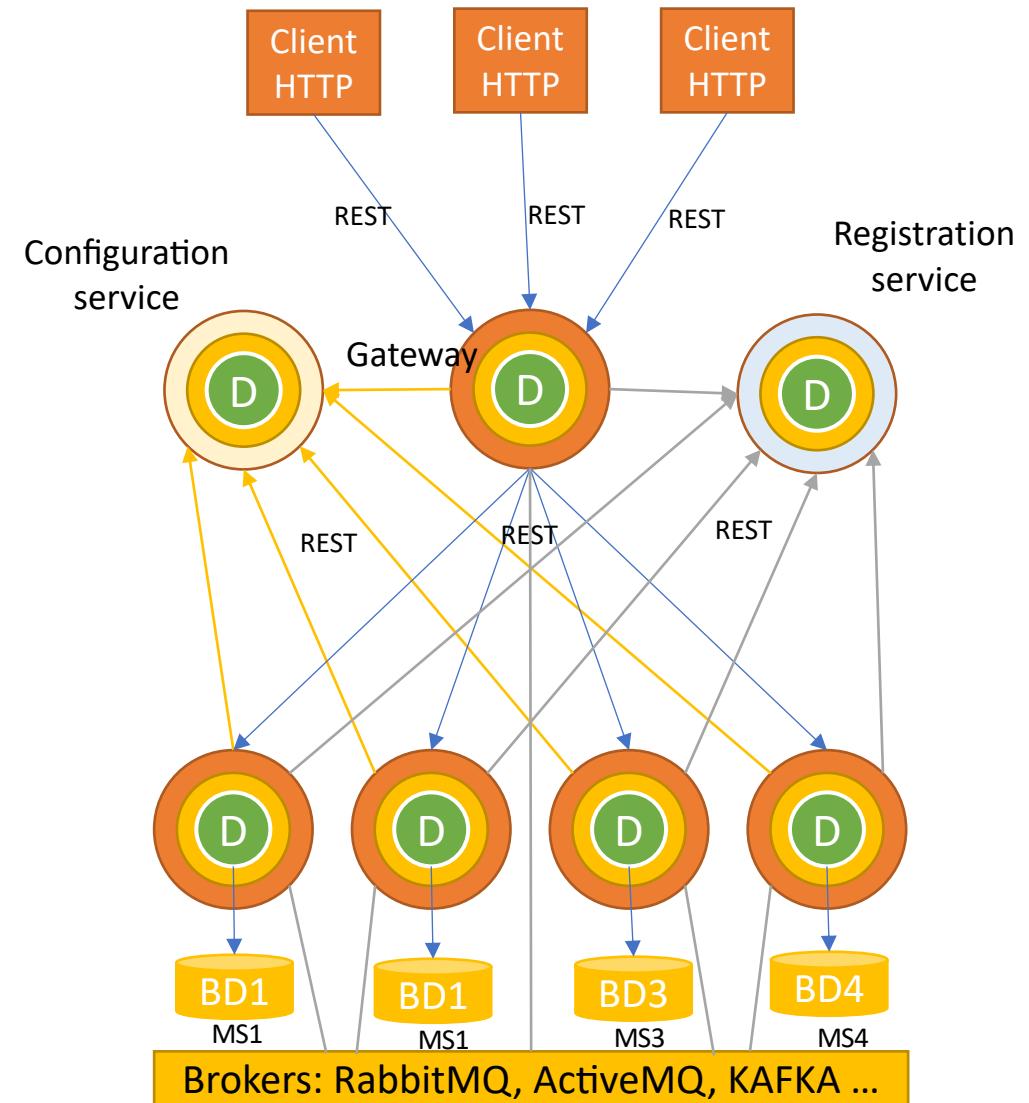


Monolithe



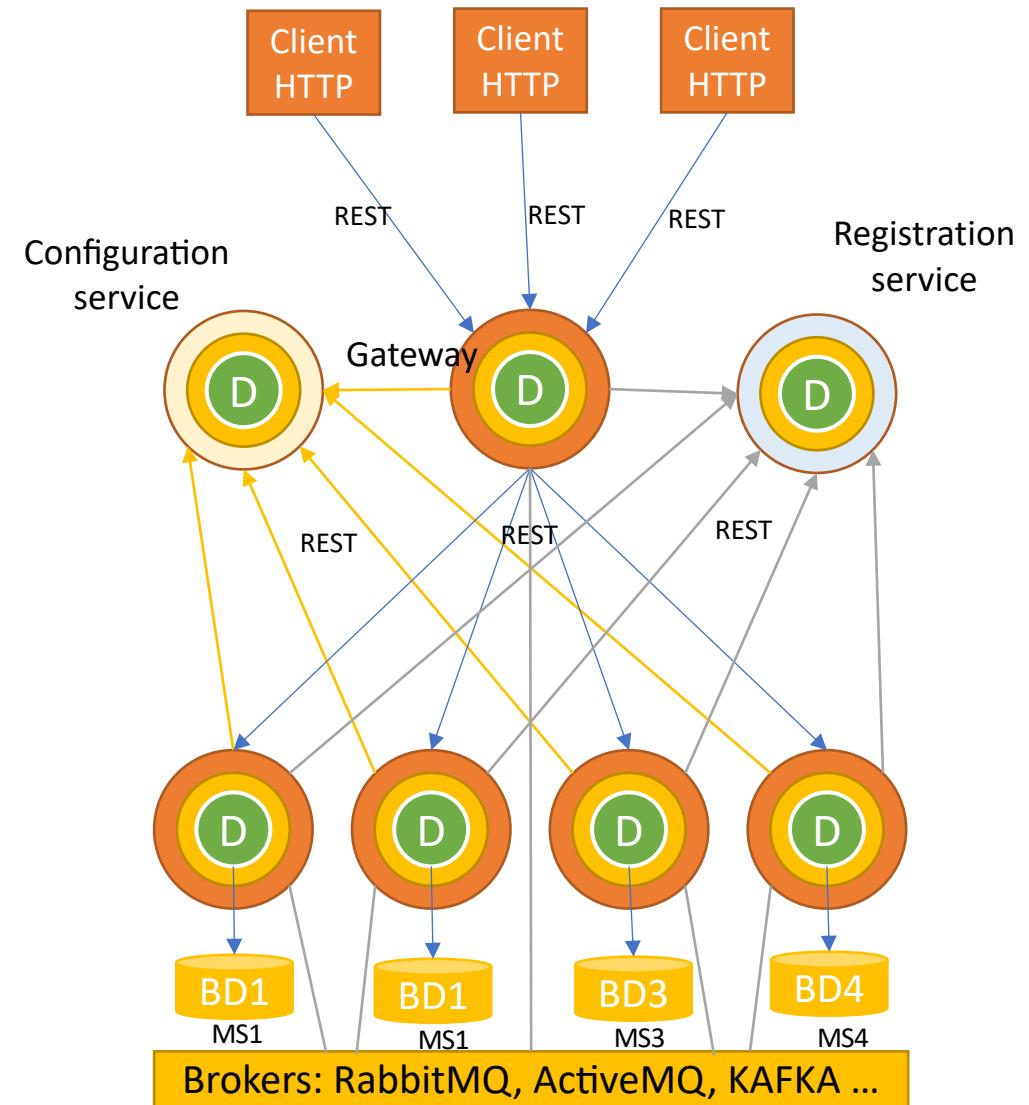
Approche Micro services

- Les micro-services sont une approche d'architecture et de développement d'une **application composées de petits services**.
- L'idée étant de découper un grand problème en petites unités implémentées sous forme de micro-services.
- Chaque service est **responsable d'une fonctionnalité**.
- Chaque micro-service est **développé, testé, et déployé** séparément des autres.
- Chaque micro service est **déployé en utilisant une technologie qui peut être différente des autres** (Java, C++, C#, PHP, NodeJS, Phyton,...)
- Chaque service **tourne dans un processus séparé**.
- Utilisant des mécanisme de communication légers (REST).
- La seule relation entre les différent micro-service est l'échange de donne effectue a travers les différentes APIs qu'ils exposent (**SOAP, REST, RMI, CORBA, JMS, MQP, ...**)
- Lorsqu'on **les combinent** les micro-services peuvent **réaliser des opérations très complexes**.



Approche Micro services

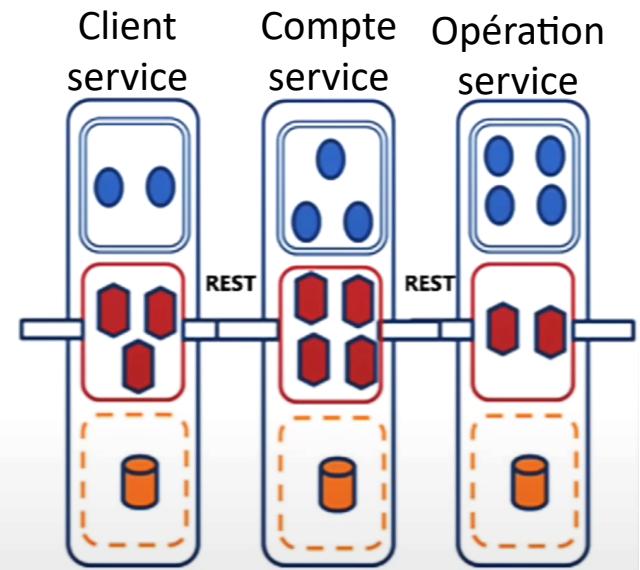
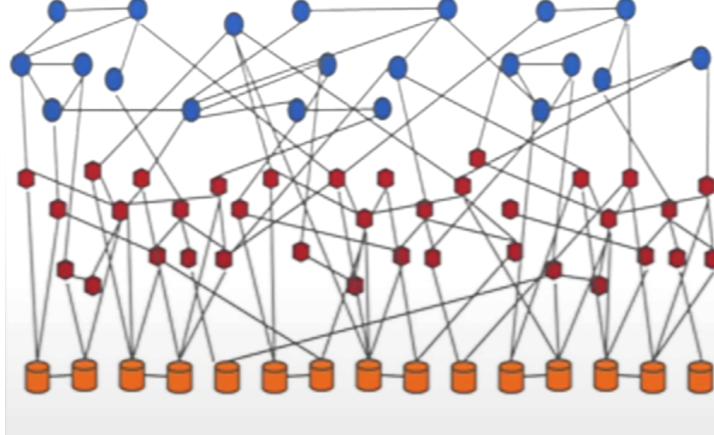
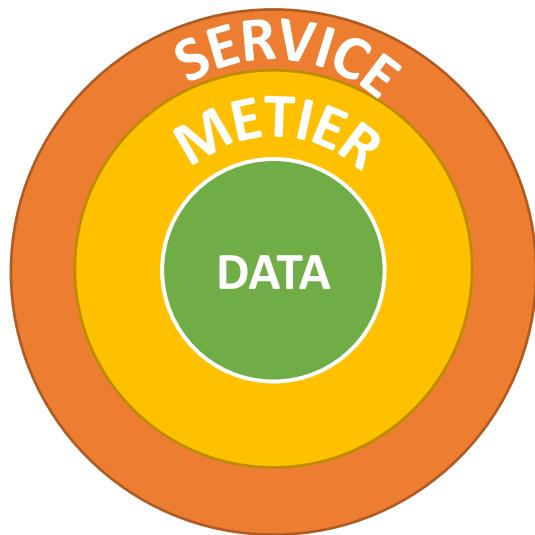
- Ils sont **faiblement couplés** puisque chaque micro-service est physiquement séparé des autres.
- **Indépendante relative entre les différentes équipes** qui développent les différents micro-services.
- **Facilité du teste est du déploiement.**
- **Livraison continue.**
- S'apprête bien au processus du GL: **TDD** (Test Driven Developpement) et les **méthodes agiles**.



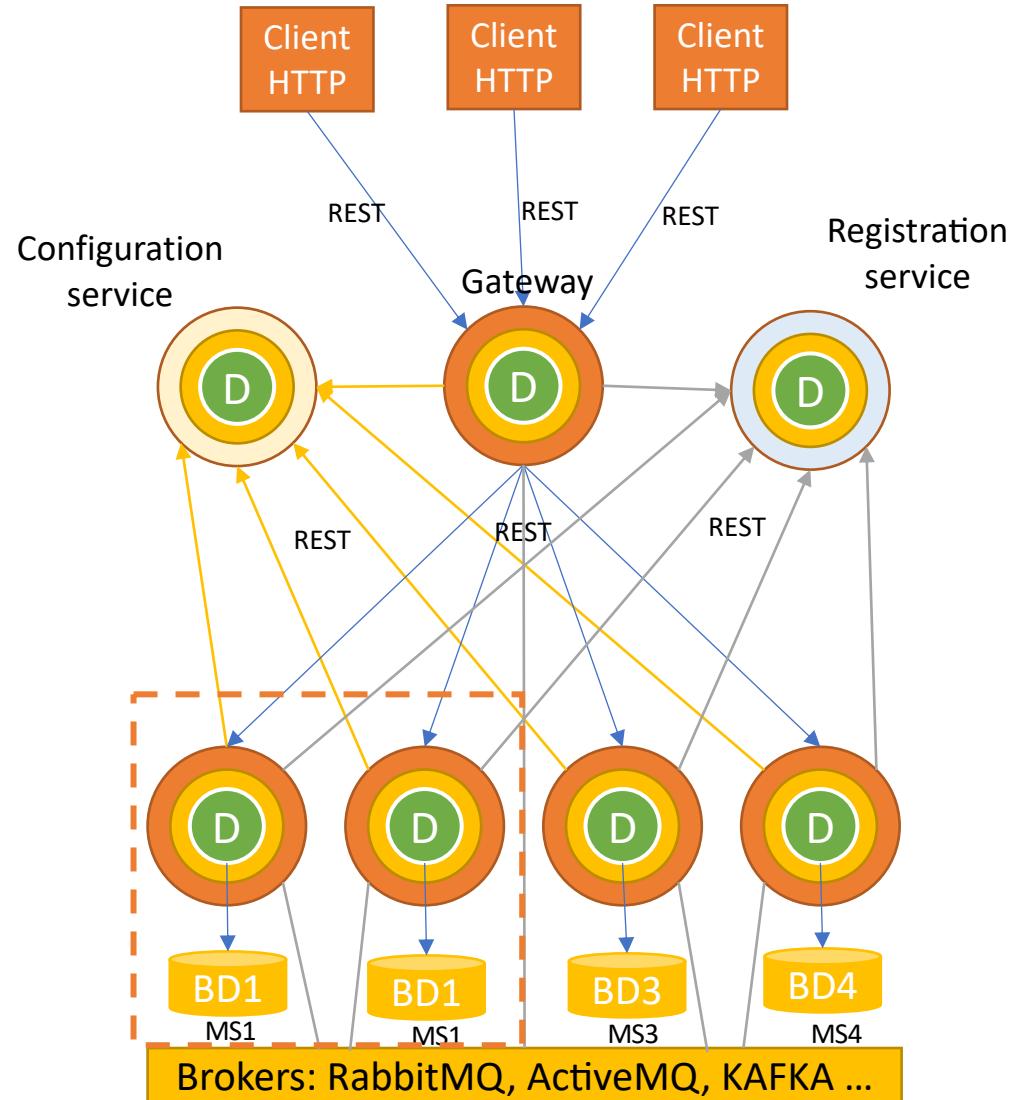
Approche Micro services

Comme pour le cas d'une application monolithique, un micro-service peut être composé de plusieurs couches (petite couche):

- Couche DAO,
- Couche métier,
- Couche techniques (REST, SOAP, RMI, JMS, AMQP, Sécurité, etc...)



Approche Micro services

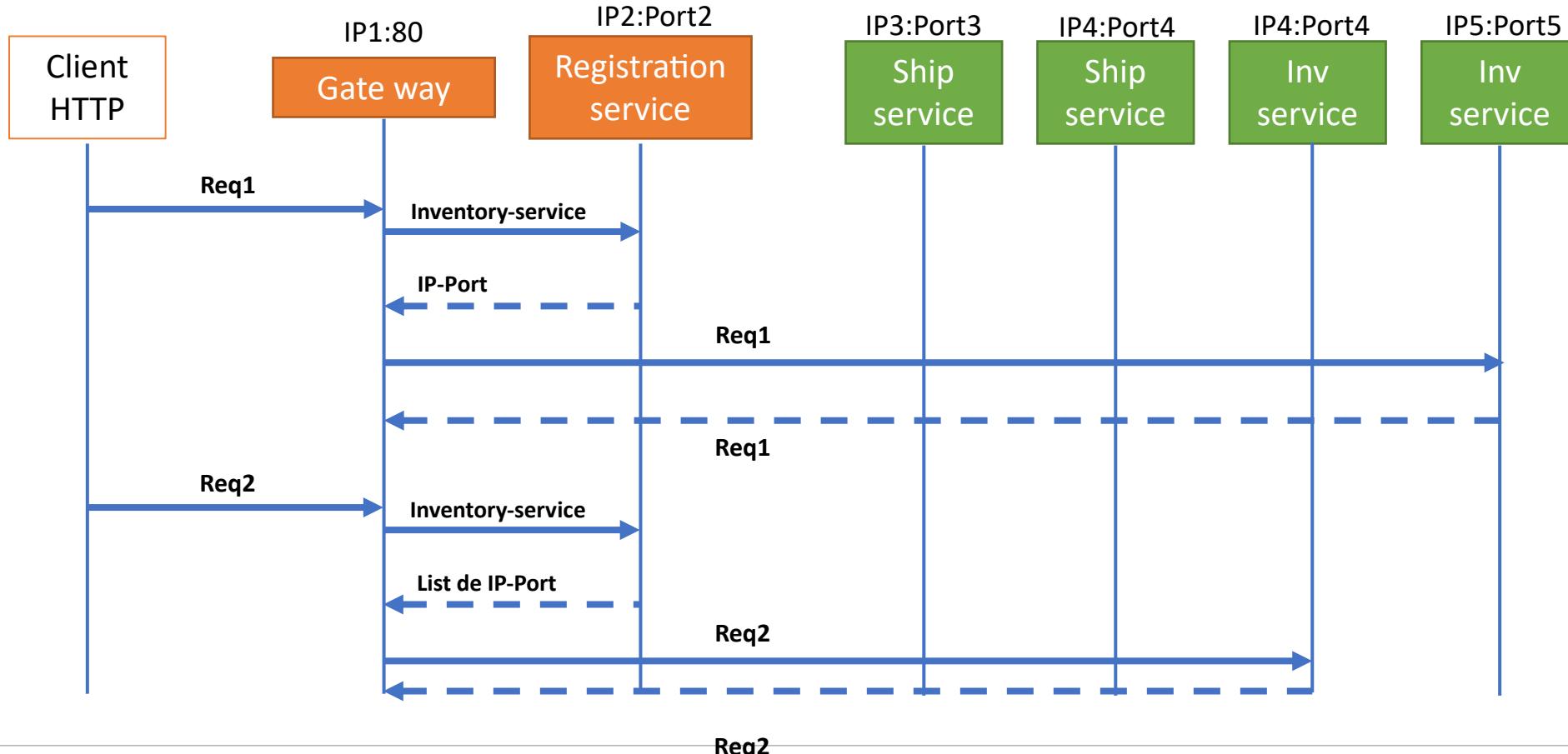


Approche Micro services

Consulter le service via le service proxy

Req 1: GET <http://gateway/inventory-service/products>

Req 2: GET <http://gateway/inventory-service/products>



Approche Micro services

QUI UTILISE LES MICROSERVICES

Ci-après quelques références d'entreprises ayant adoptées l'architecture microservice :

- Uber : <https://eng.uber.com/soa/>
- Netflix : <http://techblog.netflix.com/>
- Amazon : <http://fr.slideshare.net/apigee/i-love-apis-2015-microservices-at-amazon>
- Sound Cloud : <https://developers.soundcloud.com/blog/buildingproducts-at-soundcloud-part-2-breaking-the-monolith>
- Le prochain Eclipse Che IDE : <https://eclipse.dev/che/docs/stable/overview/introduction-to-eclipse-che/>

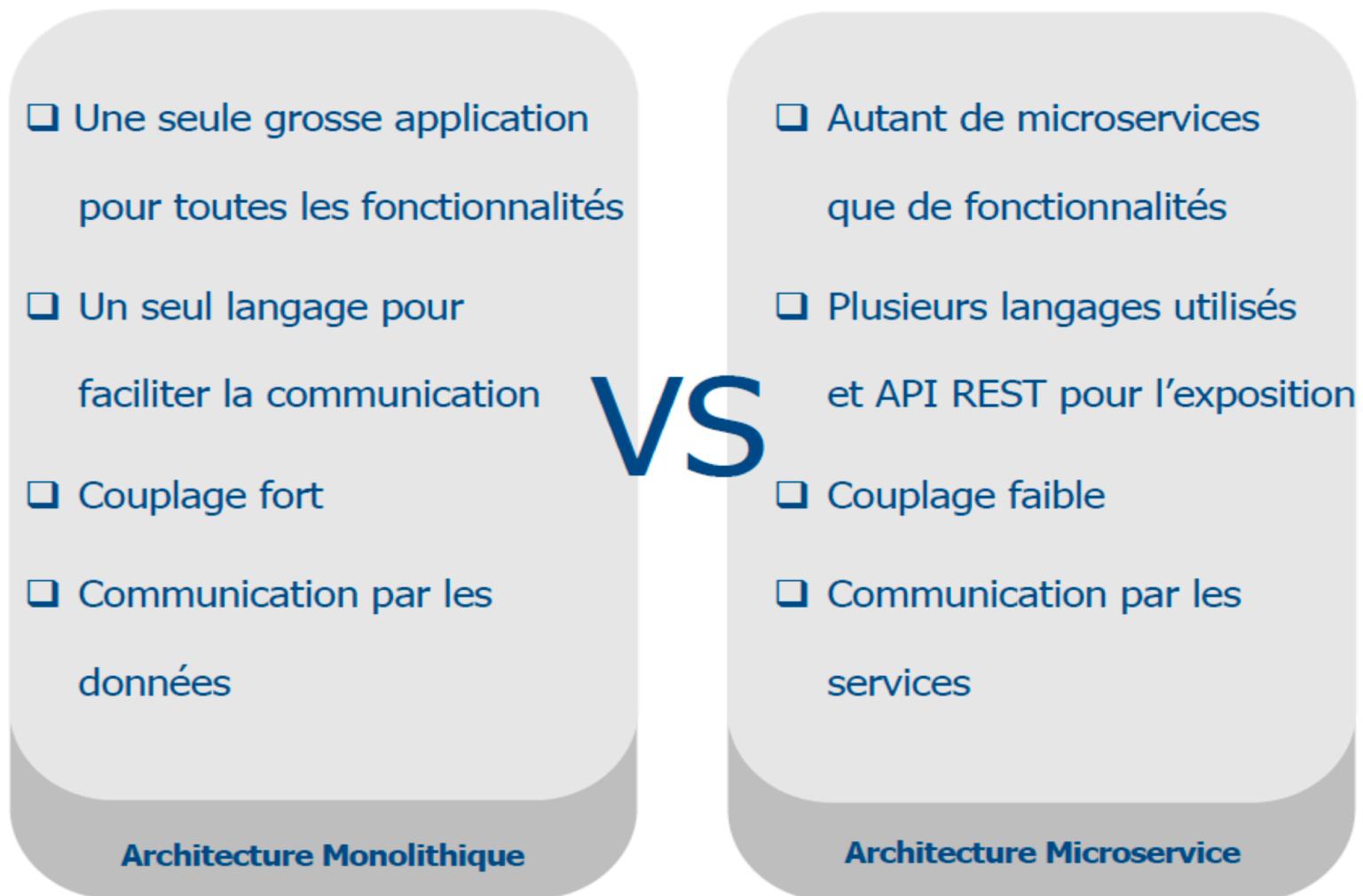
Approche Micro services

INCONVÉNIENTS

- Une architecture microservice est donc un ensemble de microservices autonomes
- Plus de complication lors de l'intégration pour vérifier si une API est cassée ou pas
- Multiplication des SGBD utilisés (relationnel ou NoSQL) et donc compétences nouvelles
- Ne s'applique pas à toutes les entreprises:
 - Si déploiement fréquent → Micorservice
 - Si déploiement une fois par an → Monolithique

Approche Micro services

APPLICATIONS MONOLITHIQUE VERSUS MICROSERVICE



Approche Micro services

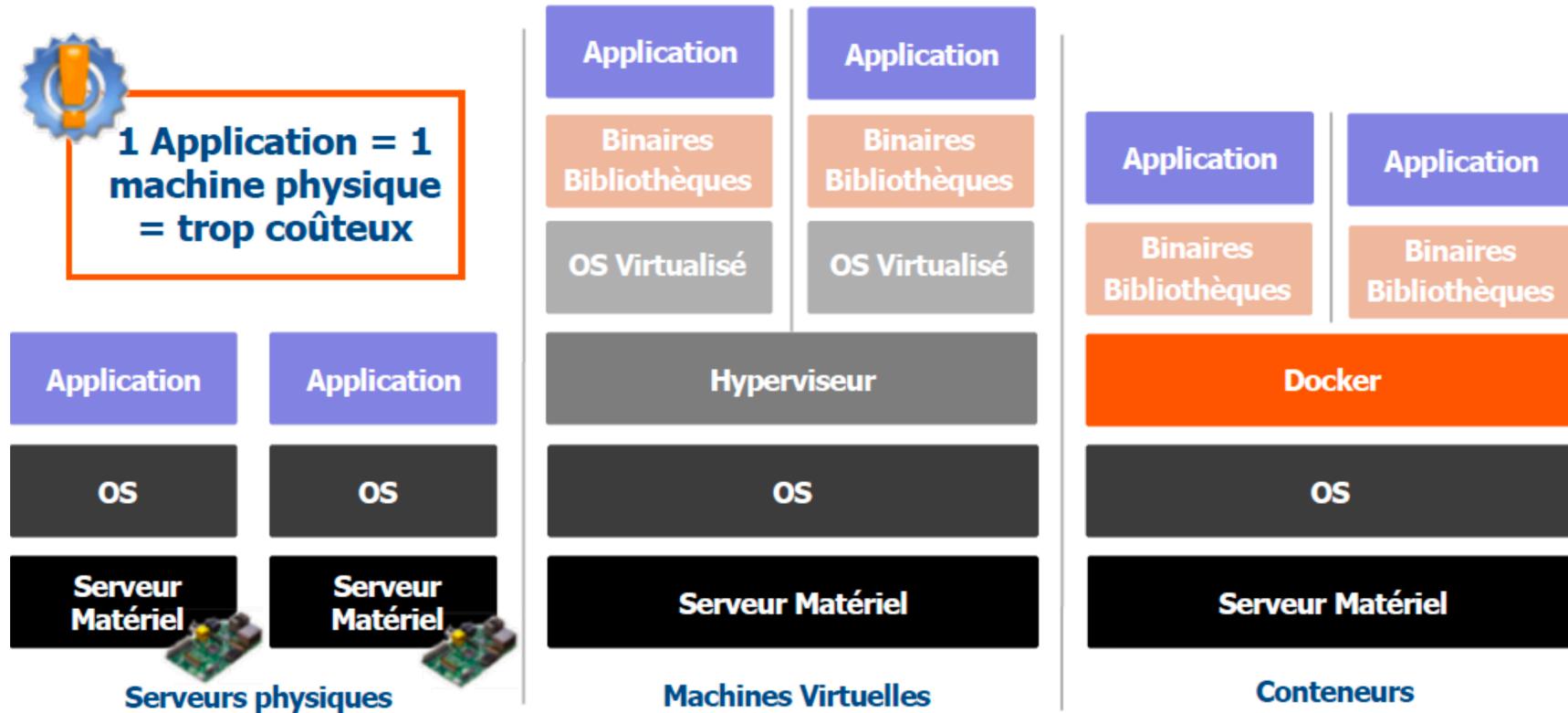
DÉVELOPPER DES ARCHITECTURES MICROSERVICE ?

- Savoir **Isoler** un microservice
- Savoir **Coder** le contenu du microservice
- Savoir faire **Communiquer** des microservices
- Savoir **Composer** les microservices
- Savoir **Répartir** les charges

Approche Micro services

ISOLER : MACHINE VIRTUELLE OU CONTENEUR ?

- Pour isoler un microservice : utilisation de techniques de virtualisation
- Trois grandes techniques s'affrontent



Approche Micro services

ISOLER : LES OUTILS EXISTANTS

Les outils pour construire des machines virtuelles

- Hyperviseur type 1 : Xen, VMware vSphere, Microsoft HyperV, KVM
- Hyperviseur type 2 : Oracle VirtualBox, VMware Player, QEMU

Les outils pour construire des conteneurs

- Conteneur : Docker, CoreOS rkt, LXC

Docker (<https://www.docker.com>) → le plus tendance.

- Basé sur LXC mais plus simple à mettre en œuvre
- Dispose d'un dépôt central fournissant des images prêtes à l'emploi
- Logiciel libre

Approche Micro services

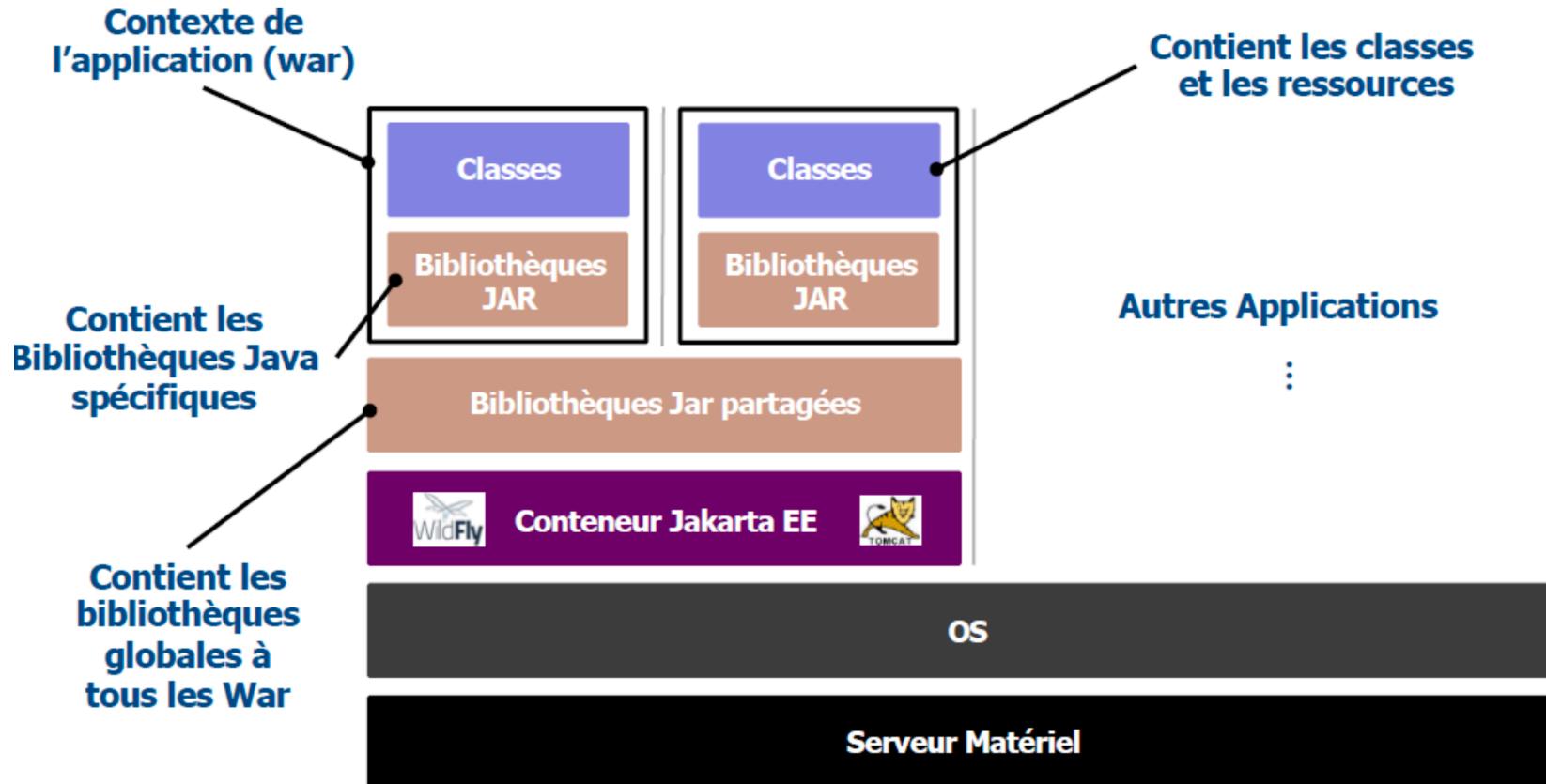
CODER :

- Utiliser les plateformes logicielles (Java, Python, JavaScript) que vous avez l'habitude d'utiliser
- Utiliser votre environnement de développement préféré (Eclipse, IntelliJ, Netbeans, Atom...)
- Utiliser les bibliothèques qui permettent d'exposer des services web (REST ou SOAP)
- Automatiser le déploiement de votre application afin de minimiser le temps

Approche Micro services

CODER : LE CAS DE JAVA (SANS MICROSERVICE)

- Actuellement une application web (war) développée en Java est déployée dans un serveur d'application Java

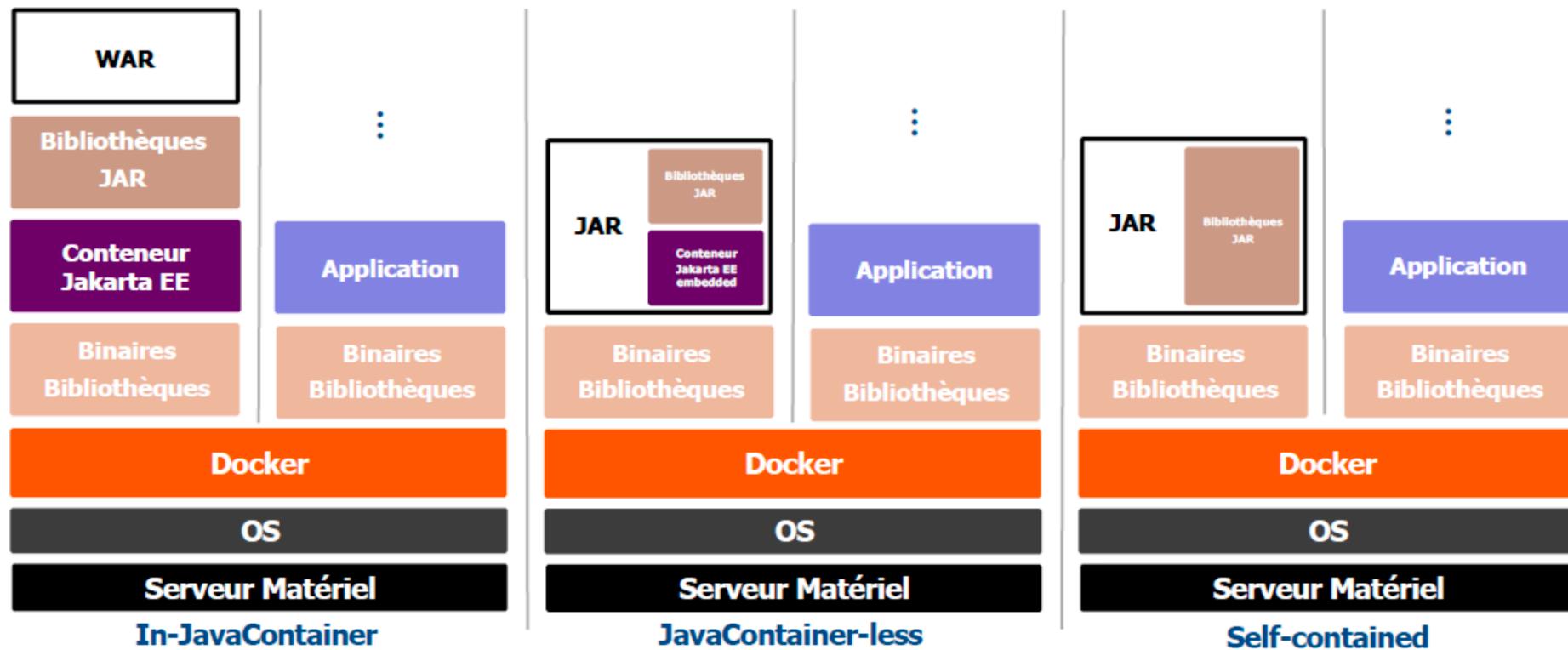


Approche Micro services

CODER : LE CAS DE JAVA (AVEC MICROSERVICE)

Trois stratégies pour développer des microservices en Java

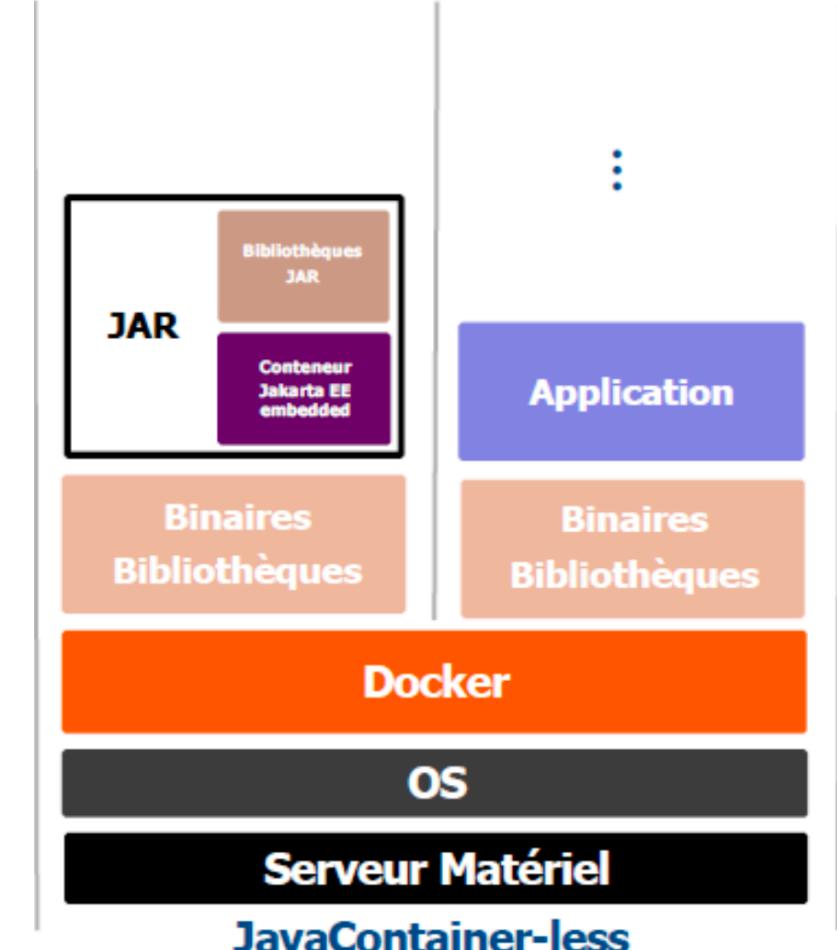
- in-JavaContainer : conteneur Jakarta EE et un War (comme avant)
- JavaContainer-less : un gros Jar contenant toutes les bibliothèques
- self-contained : un gros Jar en utilisant un micro-framework



Approche Micro services

CODER : LE CAS DE JAVA (JAVACONTAINER-LESS)

- Le serveur d'application est embarqué dans l'application
- Le code source est le même que pour une application War
- Un gros Jar est distribué et prêt à l'exécution:
`$ java -jar my-big-app-container-less.jar`
- Solutions du marché
 - Apache TomEE (anciennement OpenEJB)
 - Wildfly Swarm (anciennement JBoss)
 - Payara (à base de Glassfish)



Approche Micro services

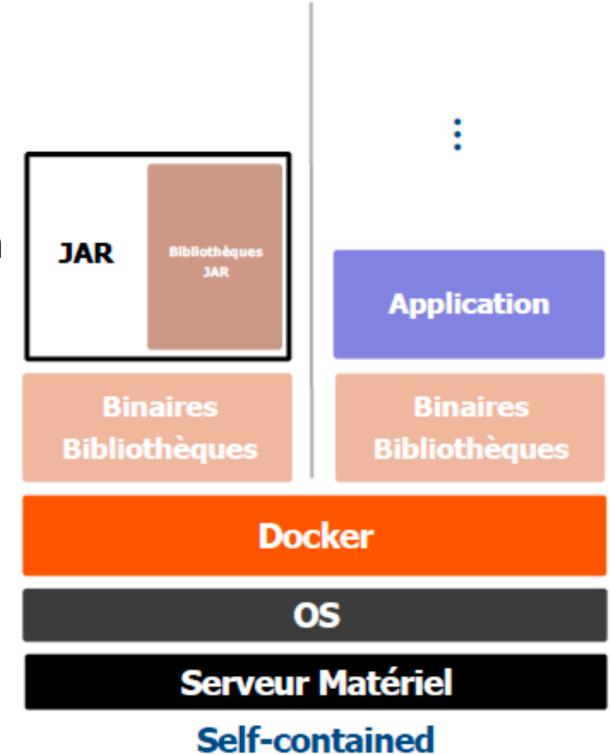
CODER : LE CAS DE JAVA (SELF-CONTAINED) : SPRINGBOOT

- Utilisation d'un micro-framework
- Le code source est le même que pour une application War .
- Un gros Jar est distribué et prêt à l'exécution :

```
$ java -jar my-big-app-self-contained.jar
```

- C'est quoi un micro-framework ?
 - Framework orienté service web minimaliste
 - Opposé au framework qui font tout full-stack
 - Des dépendances Maven « à la carte »
 - Des modules pour simplifier le développement et le monitoring d'exécution
- **Spring Boot** Basé sur **Spring** : pour faciliter la création de projet
- Disponibiliter de Starters pour gérer les dépendances ;
- Auto-configuration (moins d'XML)
- Métriques accessibles via services web (Actuator) :
 - Métrique de l'application (CPU, mémoire);
 - Beans (liste des Beans)
 - Trace (requêtes HTTP envoyées) ;
 - HEALTH (état de santé de l'application)

Req2



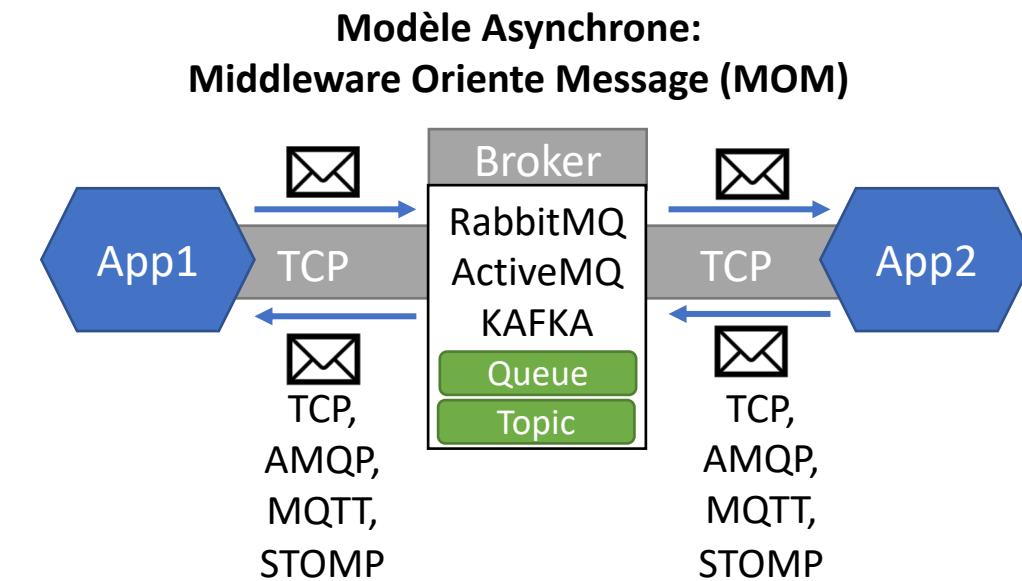
Architectures Micro-Service

COMMUNIQUER

- Deux mode de communication réseaux:
 - Mode déconnecté: par paquet ⇒ (DataGram) Protocole **UDP**
 - Mode connecté: par flux ⇒(Streams) Protocole **TCP (sockets)**
- Pour développer un système distribué il faut utiliser des middlewares:
- Couche logiciel intermédiaire qui permet aux applications distribuées de communiquer de manière transparentes.
- Deux modèles de middlewares:
 - Synchrone:
 - RMI (Remote Method Invocation)
 - CORBA (Common Object Request Broker Architecture)
 - Web service:
 - SOAP (http1.1+XML)
 - REST (http1.1, JSON, XML, ...)
 - GraphQL (http1.1, JSON)
 - GRPC (http2, ProtoBuf)
 - Asynchrone (Middleware Oriente Message (MOM))
 - Faire communiquer les systèmes distribués par l'intermédiaire d'un Broker (RabbitMQ, KAFKA, etc...)



Modèle Synchrone:
RMI, CORBA
Web services: SOAP, REST, GraphQL, GRPC



Modèle Asynchrone:
Middleware Oriente Message (MOM)

Approche Micro services

COMPOSER

- La composition va permettre de résoudre le problème d'orchestration des conteneurs
- Exemple simple pour comprendre cette problématique
 1. A = un conteneur pour la base de données (MySQL)
 2. B = un conteneur applicatif
 3. A doit être démarré avant B et B a besoin d'accéder à A (ouverture de port)
- Solutions du marché:
 - **Docker Compose** (Docker inc.) → le plus simple
 - Docker Swarm (Docker inc.)
 - Kubernetes (Linux Foundation Project, ex projet Google)
 - Mesos (Apache)
 - Fleet (CoreOS)
 - Crane (github.com/michaelsauter/crane)

Approche Micro services

RÉPARTIR LA CHARGE (REVERSE PROXY)

- Problématique du pourquoi utiliser un reverse proxy
 - Plusieurs microservices vont être créés
 - Plusieurs instances d'un même microservice peuvent être créées
- Reverse Proxy ou « Proxy inversé » :
 - Donne accès depuis un réseau externe aux conteneurs d'un réseau interne
 - Distribuer en fonction de la charge, les requêtes web aux conteneurs les moins occupées (d'une même image par exemple)
- Solutions libres du marché
 - Nginx(www.nginx.com)
 - Haproxy(www.haproxy.org)
 - Træfik (traefik.io)
 - Vulcand(vulcand.readthedocs.io)
 - HTTPD

Approche Micro services: SYNTHESE

- L'architecture micro-services n'est pas la solution à tous les problèmes de performances d'un applicatif
- Problèmes courants
 - Modélisation
 - Persistance des données
 - Transaction
 - Front-end non adapté
- Les micro-services adressent le problème d'architecture et sont là pour faciliter la montée en charge
- Toutefois : il n'a jamais été précisé que les architectures monolithiques étaient mortes

Premier Micro services – Spring Boot

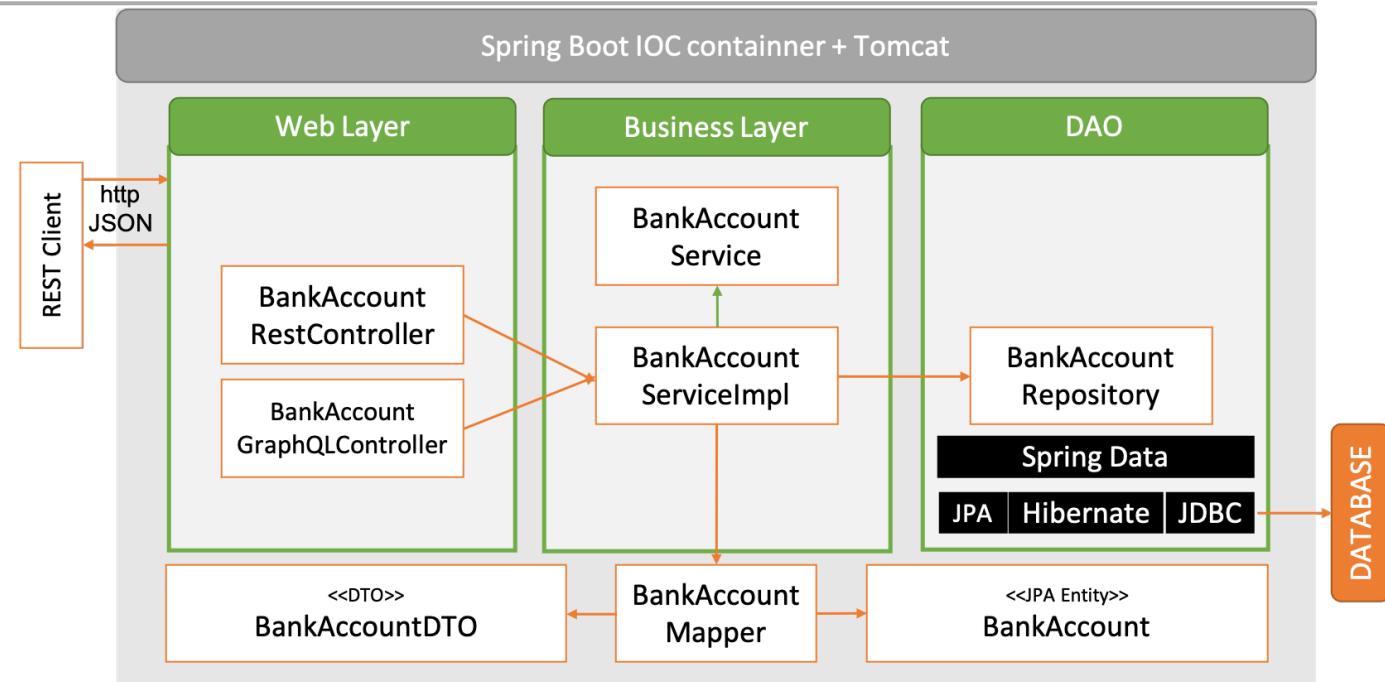
Cree un micro-service qui permet de gérer les comtes bancaire.

Chaque compte bancaire est défini par:

- id de type String
- CreatedAt de type Date
- Balance de type Double
- Currency de type String
- Type de type AccountType

L'application va permettre de:

- Lister les comtes bancaires
- Rechercher un compte bancaire sachant son id
- Ajouter un compte bancaire
- Mettre à jour un compte bancaire
- Supprimer un compte bancaire



Premier Micro services – Spring Boot

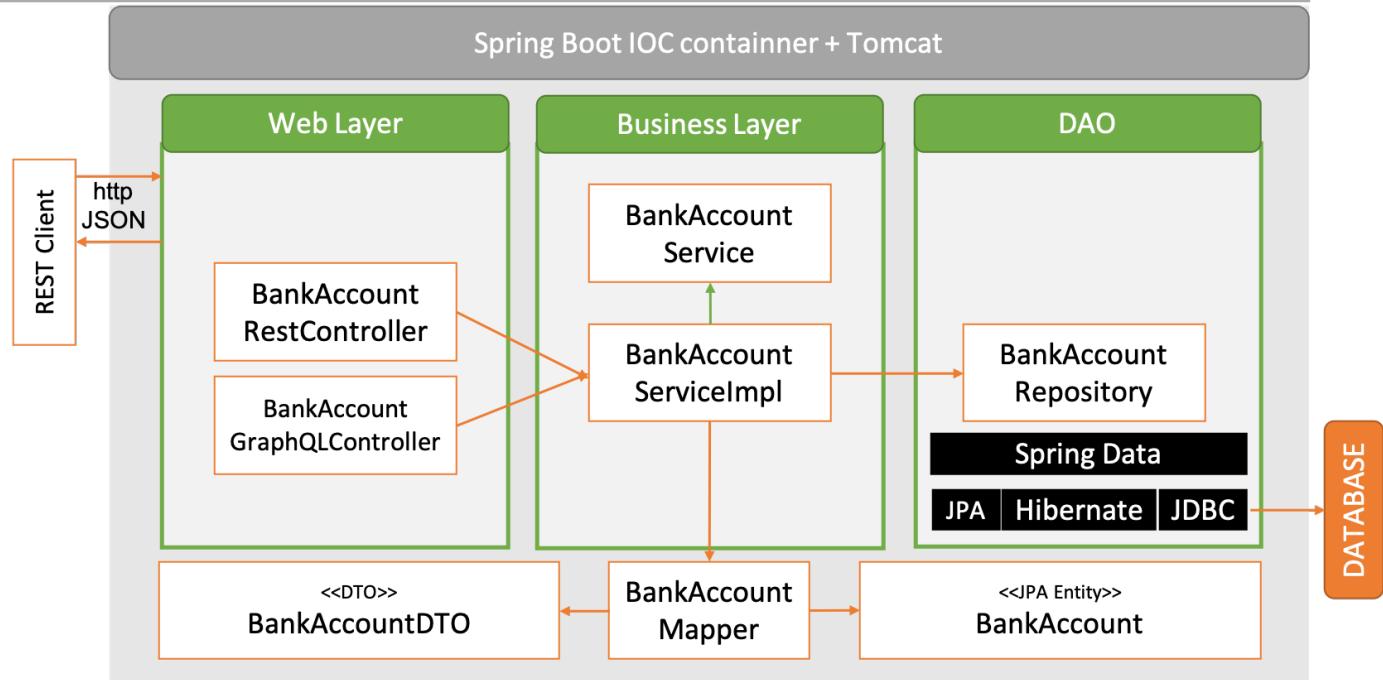
Cree un micro-service qui permet de gérer les comtes bancaire.

Chaque compte bancaire est défini par:

- id de type String
- Created At de type Date
- Balance de type Double
- Currency de type String
- Type de type AccountType

L'application va permettre de:

- Lister les comptes bancaires
- Rechercher un compte bancaire
- Ajouter un compte bancaire
- Mettre à jour un compte bancaire
- Supprimer un compte bancaire



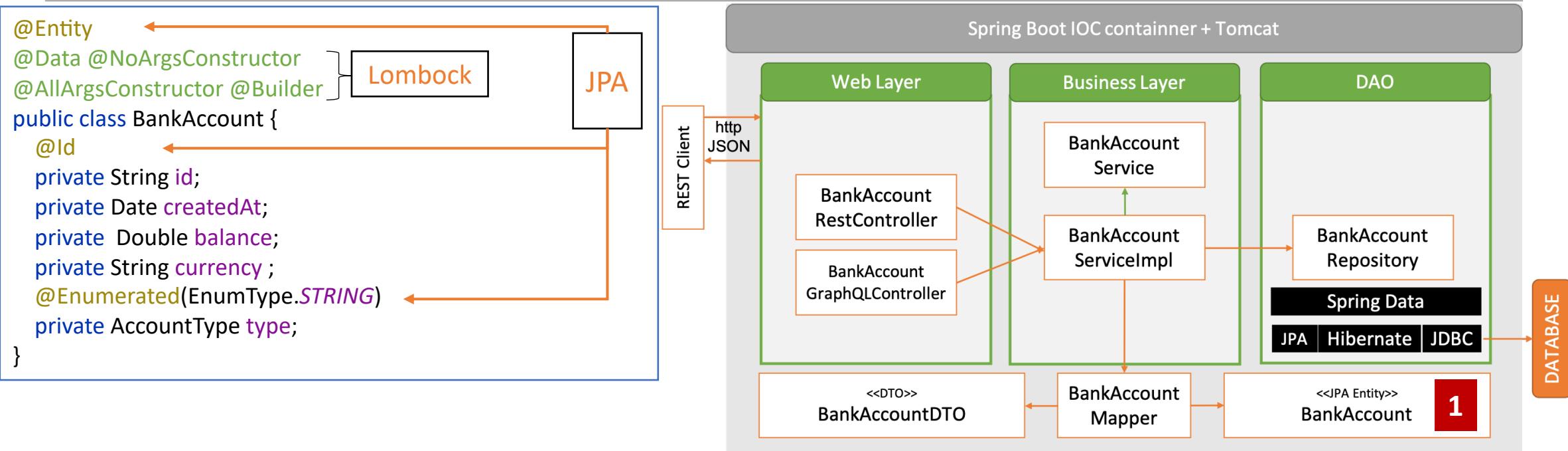
```
@SpringBootApplication  
public class BankAccountServiceApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(BankAccountServiceApplication.class, args);  
    }  
}
```

Spring Boot Application

```
spring.datasource.url=jdbc:h2:mem:account-db  
spring.h2.console.enabled=true  
server.port:8081
```

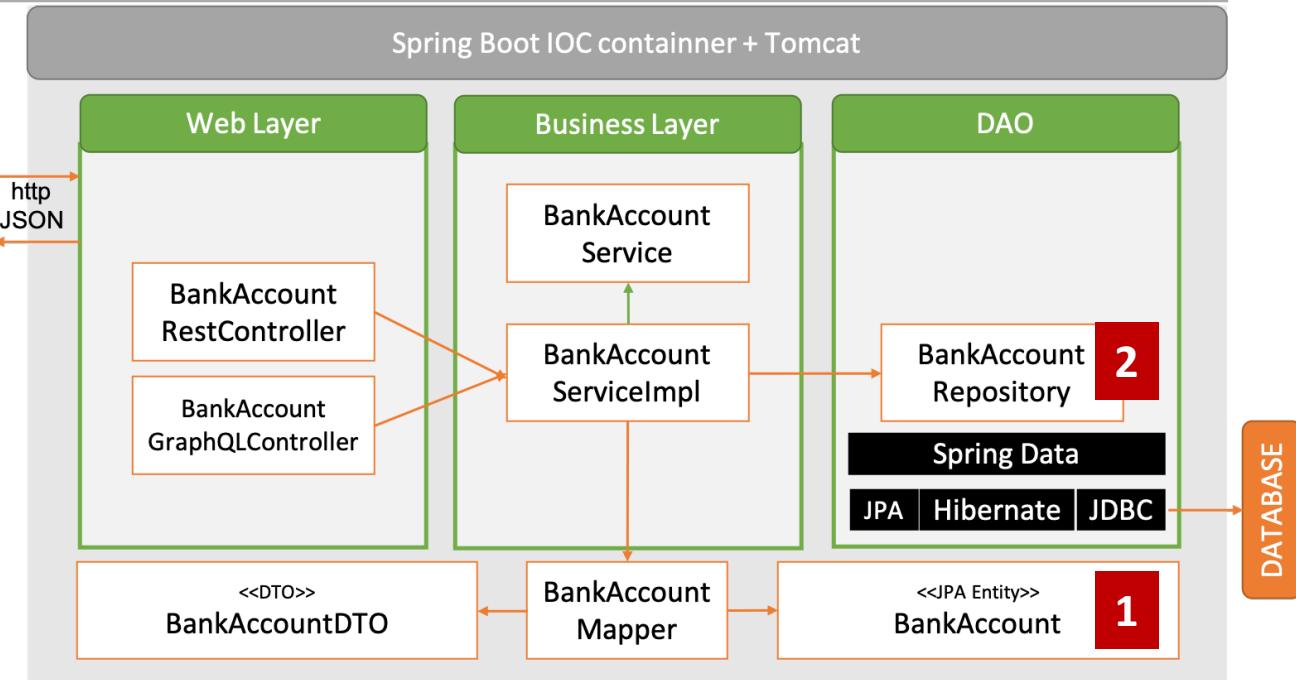
Spring Boot Application

Premier Micro services – Spring Boot



Premier Micro services – Spring Boot

```
@Entity  
@Data @NoArgsConstructor  
@AllArgsConstructor @Builder  
public class BankAccount {  
    @Id  
    private String id;  
    private Date createdAt;  
    private Double balance;  
    private String currency;  
    @Enumerated(EnumType.STRING)  
    private AccountType type;  
}
```



```
public interface BankAccountRepository extends JpaRepository<BankAccount, String> {  
    @RestResource(path="/byType")  
    List<BankAccount> findByType(@Param("t") AccountType type);  
}
```

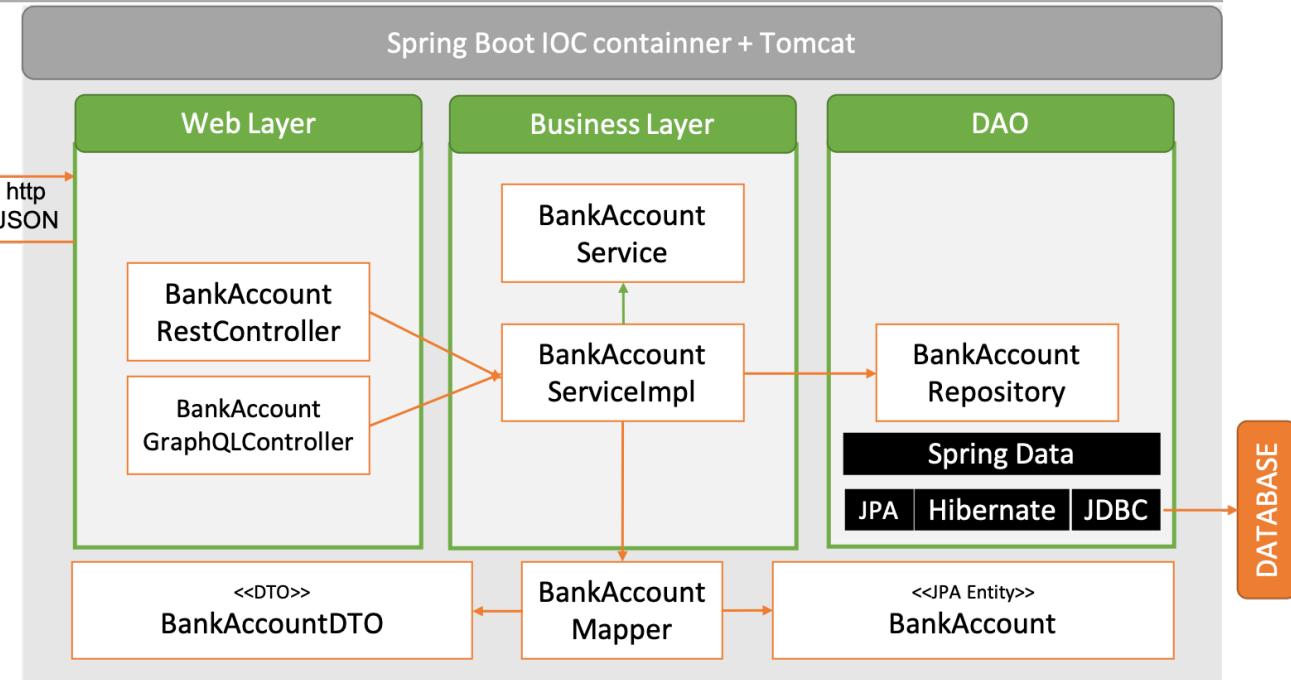
Spring Data JPA

Premier Micro services – Spring Boot

```

@Entity
@Data @NoArgsConstructor
@AllArgsConstructor @Builder
public class BankAccount {
    @Id
    private String id;
    private Date createdAt;
    private Double balance;
    private String currency;
    @Enumerated(EnumType.STRING)
    private AccountType type;
}

```

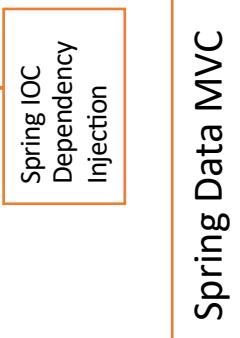


```

@RestController
public class AccountRestController {
    @Autowired
    private BankAccountRepository bankAccountRepository;

    @GetMapping("/bankAccounts")
    public List<BankAccount> bankAccounts(){
        return bankAccountRepository.findAll();
    }
}

```



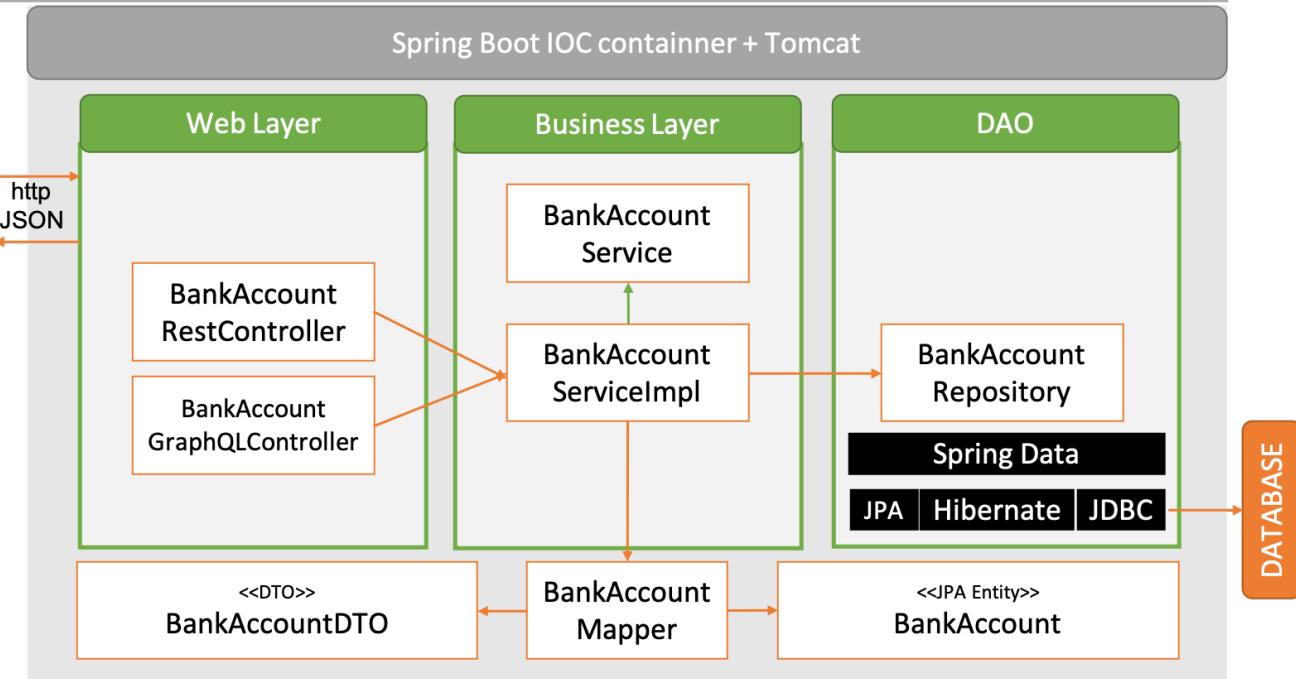
Premier Micro services – Spring Boot

```

@Entity
@Data @NoArgsConstructor
@AllArgsConstructor @Builder
public class BankAccount {
    @Id
    private String id;
    private Date createdAt;
    private Double balance;
    private String currency;
    @Enumerated(EnumType.STRING)
    private AccountType type;
}

```

The diagram illustrates the mapping between Lombok annotations and JPA annotations. It shows a code snippet for a `BankAccount` entity with various fields and annotations. Orange arrows point from the Lombok annotations to their corresponding JPA annotations: `@Entity` to `@Id`, and `@Data` to `@Id` and `@Builder`. A bracket groups the Lombok annotations under the JPA annotations.



```

@RestController
@RequestMapping("/api")
public class AccountRestController {
    @Autowired
    private BankAccountRepository bankAccountRepository;

    @GetMapping("/bankAccounts")
    public List<BankAccount> bankAccounts(){
        return bankAccountRepository.findAll();
    }
}

```

A large red X is drawn over the code snippet, indicating that it is no longer used. A red arrow points from the code to a box labeled "Spring Data JPA". Another red arrow points from the code to a box labeled "Spring IOC Dependency Injection".

```

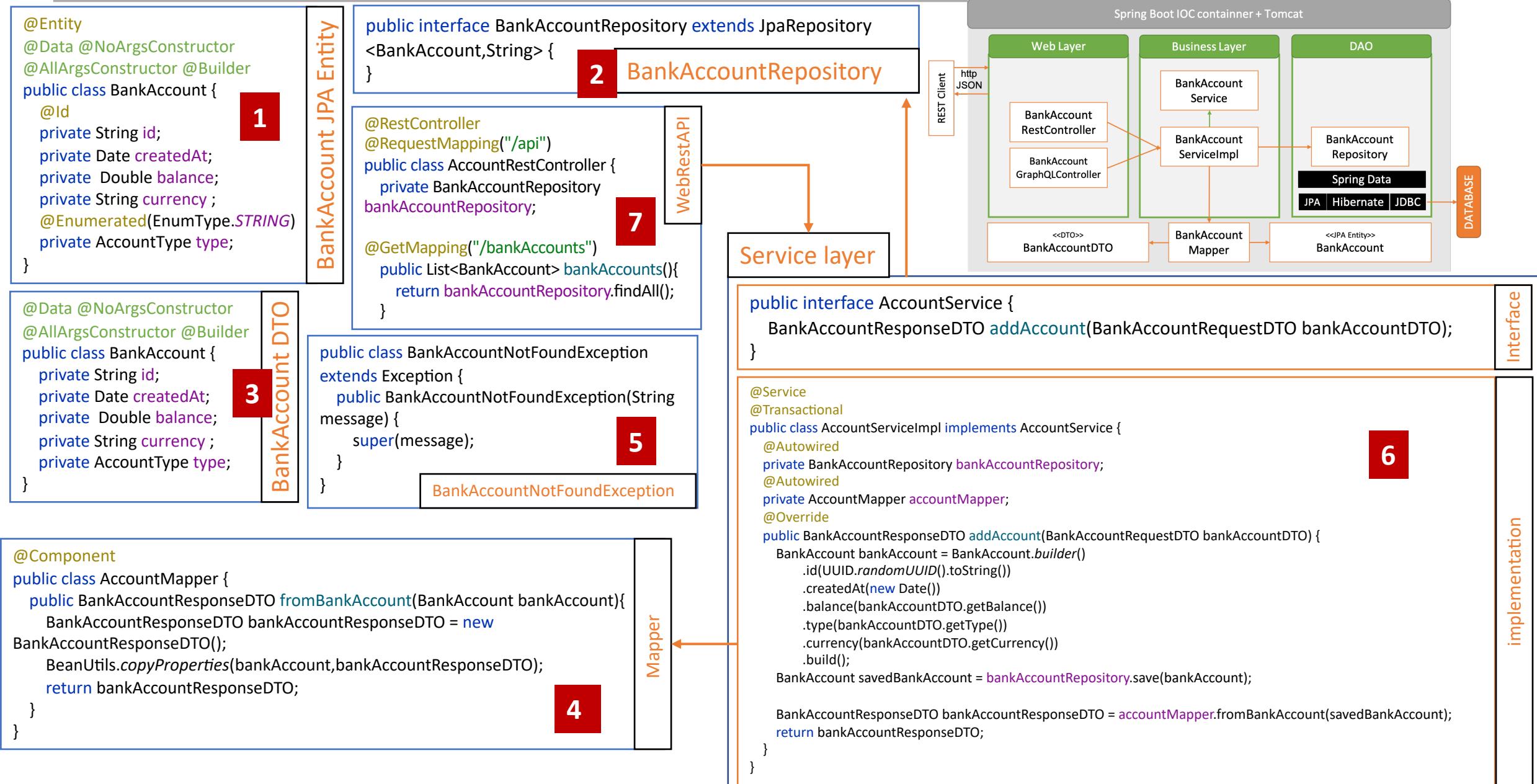
@RepositoryRestResource
public interface BankAccountRepository extends JpaRepository<BankAccount, String> {
    @RestResource(path="/byType")
    List<BankAccount> findByType(@Param("t") AccountType type);
}

```

A red arrow points from the code to a box labeled "Spring Data Rest".

Spring Data JPA

Premier Micro services – Spring Boot



Premier Micro services - TP



<https://start.spring.io/>



Project

- Gradle - Groovy Gradle - Kotlin
 Maven

Language

- Java Kotlin Groovy

Spring Boot

- 3.2.0 (SNAPSHOT) 3.2.0 (M3) 3.1.5 (SNAPSHOT) 3.1.4
 3.0.12 (SNAPSHOT) 3.0.11 2.7.17 (SNAPSHOT) 2.7.16

Project Metadata

Group	org.sid
Artifact	bank-account-service
Name	bank-account-service
Description	Demo project for Spring Boot
Package name	org.sid.bank-account-service
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 21 <input checked="" type="radio"/> 17 <input type="radio"/> 11 <input type="radio"/> 8

Dependencies

[ADD DEPENDENCIES... ⌘ + B](#)

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Spring for GraphQL WEB

Build GraphQL applications with Spring for GraphQL and GraphQL Java.



twitter.com/springboot

[GENERATE ⌘ + ↵](#)

[EXPLORE CTRL + SPACE](#)

[SHARE...](#)

Premier Micro services - TP

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** On the left, the project tree shows a single module named "bank-account-service" located at "/Downloads/bank-account-service". It contains ".idea", ".mvn", "src", and "main" directories. "main" has "java", "org.sid.bankaccountservice", "entities", "enums", "repositories", and "web" sub-directories. "BankAccountServiceApplication" is also listed under "main".
- Code Editor:** The main window displays the content of "BankAccount.java". The code uses Lombok annotations (@Entity, @Data, @Builder, @NoArgsConstructor, @AllArgsConstructor) and imports javax.persistence.Entity, javax.persistence.EnumType, javax.persistence.Id, and java.util.Date.
- Toolbars and Status Bar:** At the top, there are tabs for "BankAccount.java", "BankAccountRepository.java", "BankAccountServiceApplication.java", and "AccountR...". A status bar at the bottom shows "25" and a green checkmark icon.

```
import lombok.NoArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NonNullArgsConstructor;
import org.sid.bankaccountservice.enums.AccountType;

import javax.persistence.Entity;
import javax.persistence.Enumerated;
import javax.persistence.Id;
import java.util.Date;

@Entity
@NoArgsConstructor
@Builder
@Data
public class BankAccount {
    @Id
    private String id;
    private Date createdAt;
    private Double balance;
    private String currency ;
    @Enumerated(EnumType.STRING)
    private AccountType type;
}
```

1. Cree un package entities
2. Cree la class BankAccount.java

Premier Micro services - TP

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** On the left, the project tree shows `bank-account-service` at `~/Downloads/bank-account-service`. It contains `.idea`, `.mvn`, `src` (which has `main` and `java`), and `org.sid.bankaccountservice` (which contains `entities`, `enums`, `repositories`, `web`, and `BankAccountServiceApplication`). The `entities` folder is currently selected.
- Code Editor:** On the right, the code editor displays the `AccountType.java` file. The code is as follows:

```
1 package org.sid.bankaccountservice.enums;
2
3     5 usages
4     public enum AccountType {
5         1 usage
6             CURRENT_ACCOUNT, SAVING_ACCOUNT
7     }
```

3. Cree un package enums
4. Cree un enumerateur
AccountType.java

Premier Micro services - TP

The screenshot shows a Java code editor with the following details:

- Project Structure:** On the left, the project tree shows a single module named "bank-account-service" located at "/Downloads/bank-account-service". It contains ".idea", ".mvn", "src", and "main" directories. "main" has "java", "org.sid.bankaccountservice", "entities", "enums", "repositories", "web", and "BankAccountServiceApplication". The "entities" folder is currently selected.
- Code Editor:** The main window displays the content of "BankAccountRepository.java".

```
1 package org.sid.bankaccountservice.repositories;
2
3 import org.sid.bankaccountservice.entities.BankAccount;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface BankAccountRepository extends JpaRepository<BankAccount, String> {
7 }
```
- Toolbars and Status:** At the top, there are tabs for "BankAccount.java", "BankAccountRepository.java" (which is active), "BankAccountServiceApplication.java", and "AccountR". A green checkmark icon is visible on the right side of the toolbar.

5. Cree un package repositories

6.Cree une interface

BankAccount.java

Premier Micro services - TP

```
bank-account-service ~/Downloads/bank-account-service
> .idea
> .mvn
< src
  < main
    < java
      < org.sid.bankaccountservice
        entities
```

7. Dans BankAccountServiceApplication.java

The screenshot shows a Java code editor with the following details:

- File:** BankAccountServiceApplication.java
- Content:**

```
package org.sid.bankaccountservice;
import ...
@SpringBootApplication
public class BankAccountServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(BankAccountServiceApplication.class, args);
    }

    no usages
    @Bean
    CommandLineRunner start(BankAccountRepository bankAccountRepository){
        return args -> {
            for (int i = 0; i < 10; i++) {
                BankAccount bankAccount=BankAccount.builder()
                    .id(UUID.randomUUID().toString())
                    .type(Math.random()>0.5? AccountType.CURRENT_ACCOUNT:AccountType.SAVING_ACCT)
                    .balance(10000+Math.random()*90000)
                    .createdAt(new Date())
                    .currency("MAD")
                    .build();
                bankAccountRepository.save(bankAccount);
            }
        };
    }
}
```
- Toolbars and Status:** Shows tabs for other files like BankAccount.java, BankAccountRepository.java, and AccountR. A warning icon with '1' is visible.

Premier Micro services - TP

The screenshot shows the IntelliJ IDEA interface with the project tree on the left and the code editor on the right. The project tree for 'bank-account-service' includes '.idea', '.mvn', 'src' (with 'main' and 'java' subfolders), 'resources' (with 'graphql', 'static', and 'templates' subfolders), and 'application.properties'. The code editor displays the 'application.properties' file with the following content:

```
1 spring.datasource.url=jdbc:h2:mem:account-db
2 spring.h2.console.enabled=true
3 server.port:8081
4
```

A tooltip message at the top of the editor area states: "Spring Boot configuration files are supported by IntelliJ IDEA Ultimate". There are three warning icons (yellow exclamation marks) in the status bar at the bottom right of the editor.

8. Dans application.properties

Premier Micro services - TP

```
bank-account-service ~/Downloads/bank-account-service
> .idea
> .mvn
> src
> main
> java
> org.sid.bankaccountservice
> entities
> enums
> repositories
> web
> BankAccountServiceApplication
```

1. Cree un package entities
- 2.Cree la class BankAccount.java

5. Cree un package repositories
6. Cree une interface
BankAccount.java

```
package org.sid.bankaccountservice.repositories;

import org.sid.bankaccountservice.entities.BankAccount;
import org.springframework.data.jpa.repository.JpaRepository;

5 usages
public interface BankAccountRepository extends JpaRepository<BankAccount, String> {
}
```

3. Cree un package enums
4. Cree un enumerateur AccountType.java

```
1 package org.sid.bankaccountservice.enums;
2
3     5 usages
4     public enum AccountType {
5         1 usage
6             CURRENT_ACCOUNT, SAVING_ACCOUNT
7     }
```

7. Dans BankAccountServiceApplication.java

```
Account.java    BankAccountRepository.java    BankAccountServiceApplication.java    AccountR ...
package org.sid.bankaccountservice;
import ...

@SpringBootApplication
public class BankAccountServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(BankAccountServiceApplication.class, args);
    }

    no usages
    @Bean
    CommandLineRunner start(BankAccountRepository bankAccountRepository){
        return args -> {
            for (int i = 0; i < 10; i++) {
                BankAccount bankAccount=BankAccount.builder()
                    .id(UUID.randomUUID().toString())
                    .type(Math.random()>0.5? AccountType.CURRENT_ACCOUNT:AccountType.SAVING_ACCT)
                    .balance(10000+Math.random()*90000)
                    .createdAt(new Date())
                    .currency("MAD")
                    .build();
                bankAccountRepository.save(bankAccount);
            }
        };
    }
}
```

8. Dans application.properties

```
spring.datasource.url=jdbc:h2:mem:account-db
spring.h2.console.enabled=true
server.port:8081
```

Premier Micro services - TP

← → ⌂ ⓘ localhost:8081/h2-console/login.jsp?jsessionid=795e7dcd50f49ca7985969c86d61f251

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:account-db

User Name: sa

Password:

jdbc:h2:mem:account-db | Auto commit | Max rows: 1000 | Run | Run Selected | Auto complete | Auto select | ?

+ jdbc:h2:mem:account-db
+ BANK_ACCOUNT
+ INFORMATION_SCHEMA
+ Users
① H2 2.1.214 (2022-06-13)

Run Run Selected Auto complete Clear SQL statement:
SELECT * FROM BANK_ACCOUNT;

SELECT * FROM BANK_ACCOUNT;

ID	BALANCE	CREATED_AT	CURRENCY	TYPE
9498f090-7ce2-4bc9-beb1-1ecc79948fa4	69614.70913173445	2023-10-14 13:53:08.246	MAD	CURRENT_ACCOUNT
65a52eb0-10bb-486a-a2e7-72554f5e1d48	56639.08394987335	2023-10-14 13:53:08.295	MAD	SAVING_ACCOUNT
6abe91e7-0f44-4763-833c-f5b1486f791b	58079.394117099364	2023-10-14 13:53:08.296	MAD	CURRENT_ACCOUNT
29cc4dce-5346-424a-bdc4-76987de0cc0b	70672.4004732124	2023-10-14 13:53:08.296	MAD	CURRENT_ACCOUNT
aeb54be7-56ce-41d4-b6b6-2b37d062c45e	87569.2962628175	2023-10-14 13:53:08.297	MAD	CURRENT_ACCOUNT
b64546a7-d84e-4cdf-918a-13f2a3aac9f6	57216.202458349355	2023-10-14 13:53:08.298	MAD	SAVING_ACCOUNT
b7d90398-4673-4f57-941d-4422db5d3581	61106.9712515361	2023-10-14 13:53:08.298	MAD	CURRENT_ACCOUNT
00cbd9e4-93d6-443f-a853-bc09e5e61844	78345.74946468501	2023-10-14 13:53:08.299	MAD	CURRENT_ACCOUNT
4e098f30-c4e0-4270-b8b6-37e93d17e91f	29486.6082161305	2023-10-14 13:53:08.299	MAD	CURRENT_ACCOUNT
7d1c6f7b-3993-4be1-83a4-a618ee8c376b	36655.45350772113	2023-10-14 13:53:08.3	MAD	SAVING_ACCOUNT

(10 rows, 12 ms)

Edit

Premier Micro services - TP

9. Cree un package web

10. Cree une class AccountRestController

```
11  @RestController
12  public class AccountRestController {
13      3 usages
14      private BankAccountRepository bankAccountRepository;
15      no usages
16      public AccountRestController(BankAccountRepository bankAccountRepository){
17          this.bankAccountRepository = bankAccountRepository;
18      }
19      no usages
20      @GetMapping("/bankAccounts")
21      public List<BankAccount> bankAccounts(){
22          return bankAccountRepository.findAll();
23      }
24      no usages
25      @GetMapping("/bankAccounts/{id}")
26      public BankAccount bankAccounts(@PathVariable String id){
27          return bankAccountRepository.findById(id).orElseThrow(()->new RuntimeException(String.format("Account %s not found",id)));
28      }
29 }
```

Vérification

← → C i localhost:8081/bankAccounts

```
[{"id": "456bdedb-8b59-4b1a-8829-251d3783d9b5", "createdAt": "2023-09-29T20:01:26.080+00:00", "balance": 50074.981073606876, "currency": "MAD", "type": "CURRENT_ACCOUNT"}, {"id": "b2865324-7cc2-4a22-9f50-c5f6f532a754", "createdAt": "2023-09-29T20:01:26.118+00:00", "balance": 16687.121022361574, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "6be7b9e3-2e2e-4225-b647-6528daldf8a", "createdAt": "2023-09-29T20:01:26.119+00:00", "balance": 86942.57383417788, "currency": "MAD", "type": "CURRENT_ACCOUNT"}, {"id": "1db265de-bee8-4b5c-81c9-9ec9a35ded37", "createdAt": "2023-09-29T20:01:26.120+00:00", "balance": 34248.93033510799, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "e9c94a5d-f89c-4d97-81ec-37fe39096aa1", "createdAt": "2023-09-29T20:01:26.121+00:00", "balance": 70465.60790547318, "currency": "MAD", "type": "CURRENT_ACCOUNT"}, {"id": "9ee4f7b6-1134-4bc8-99aa-95eaea57564d", "createdAt": "2023-09-29T20:01:26.121+00:00", "balance": 94867.08863560727, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "11c9dd02-0ddc-4b27-a811-b02ca17896bf", "createdAt": "2023-09-29T20:01:26.122+00:00", "balance": 48426.822542207155, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "58f229d1-9b51-473b-908d-36b67e5af945", "createdAt": "2023-09-29T20:01:26.123+00:00", "balance": 51149.062826782356, "currency": "MAD", "type": "CURRENT_ACCOUNT"}, {"id": "4d523bab-3da3-43e1-a72b-b620c433efdd", "createdAt": "2023-09-29T20:01:26.123+00:00", "balance": 68597.91223103378, "currency": "MAD", "type": "CURRENT_ACCOUNT"}, {"id": "df5b9c6e-2772-47ce-92eb-0a0e17b4bd0f", "createdAt": "2023-09-29T20:01:26.124+00:00", "balance": 43072.11024543038, "currency": "MAD", "type": "CURRENT_ACCOUNT"}]
```

← → C i localhost:8081/bankAccounts/456bdedb-8b59-4b1a-8829-251d3783d9b5

```
{"id": "456bdedb-8b59-4b1a-8829-251d3783d9b5", "createdAt": "2023-09-29T20:01:26.080+00:00", "balance": 50074.981073606876, "currency": "MAD", "type": "CURRENT_ACCOUNT"}
```