

LES MICROSERVICES

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

SOMMAIRE

➤ INTRODUCTION

- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

PRENONS UN PEU DE HAUTEUR

OBJECTIFS DES SYSTÈMES D'INFORMATION

✓ ACCÈS À L'INFORMATION PERTINENTE

- RAPIDE
- INTÉGRÉ
- GÉNÉRALISÉ

✓ SYSTÈME OUVERT RÉDUISANT LES COÛTS

- INTEROPÉRABILITÉ INTERNE ET EXTERNE

✓ DÉVELOPPEMENT RAPIDE D'APPLICATIONS

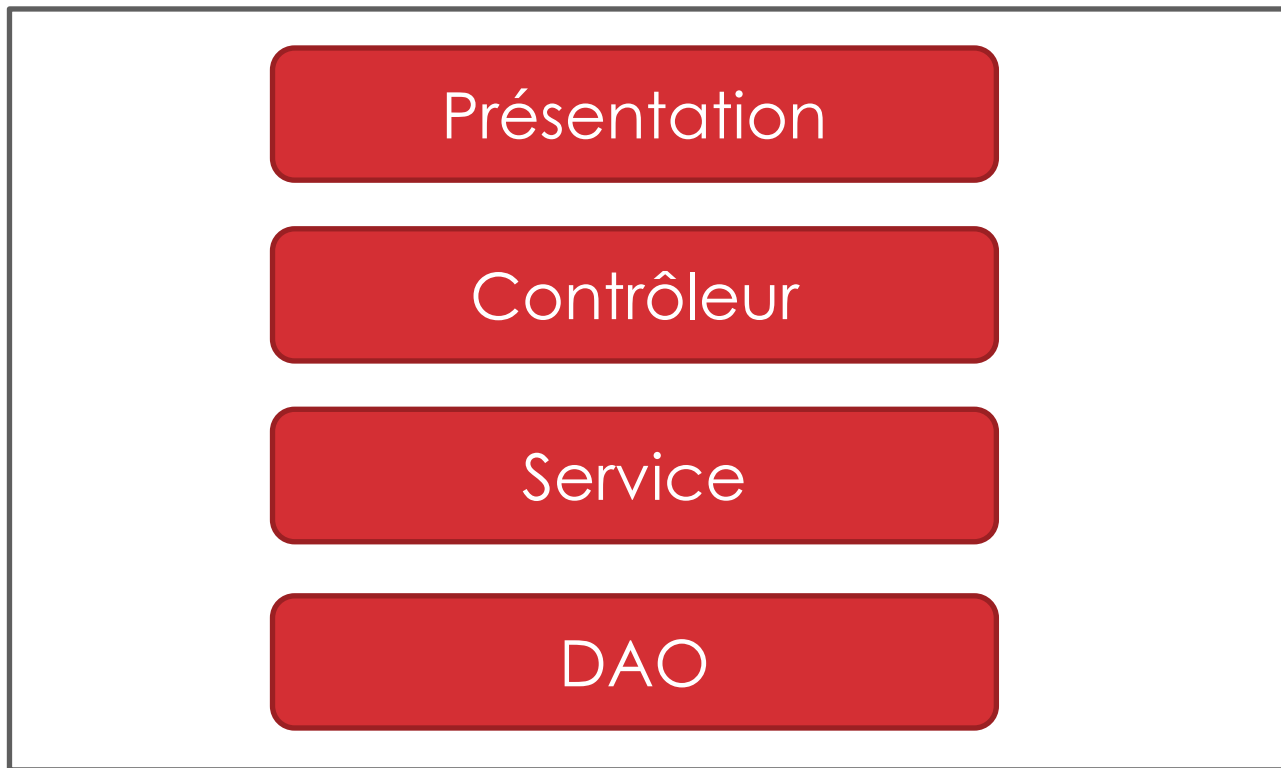
- RÉPONDRE AUX DEMANDES MÉTIER
- RÉUTILISATION DE COMPOSANTS
- RÉDUCTION DES COÛTS DE DÉVELOPPEMENT

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

MONOLITHES

ARCHITECTURE



MONOLITHES



APPLICATIONS MONOLITHIQUES

AVANTAGES

- ✓ **SIMPLICITÉ DE MISE EN ŒUVRE**
- ✓ **UNE SEULE CODEBASE**
- ✓ **EFFICIENCE À PETITE ÉCHELLE**
- ✓ **LATENCE APPLICATIVE**

BASE DE DONNÉES MONOLITHIQUES

AVANTAGES

- ✓ **SIMPLICITÉ DE MISE EN OEUVRE**
- ✓ **REQUÊTE ET JOINTURE FACILITÉS**
- ✓ **UN SEUL SCHÉMA, UN SEUL DÉPLOIEMENT**
- ✓ **EFFICIENCE À PETITE ÉCHELLE**

APPLICATIONS MONOLITHIQUES

INCONVÉNIENTS

- ✓ **MAUVAISE APPLICATION DE LA MODULARITÉ**
- ✓ **TEMPS DE BUILD LONG**
- ✓ **DÉPLOIEMENT DE TOUTE L'APPLICATION**
- ✓ **MAUVAISE MISE À L'ÉCHELLE**

BASE DE DONNÉES MONOLITHIQUES

INCONVÉNIENTS

- ✓ **COUPLAGE**
- ✓ **MAUVAIS SCALING ET REDONDANCE**
- ✓ **DIFFICULTÉ À TUNER CORRECTEMENT**
- ✓ **MANAGEMENT DU SCHÉMA**

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

LES WEB SERVICES

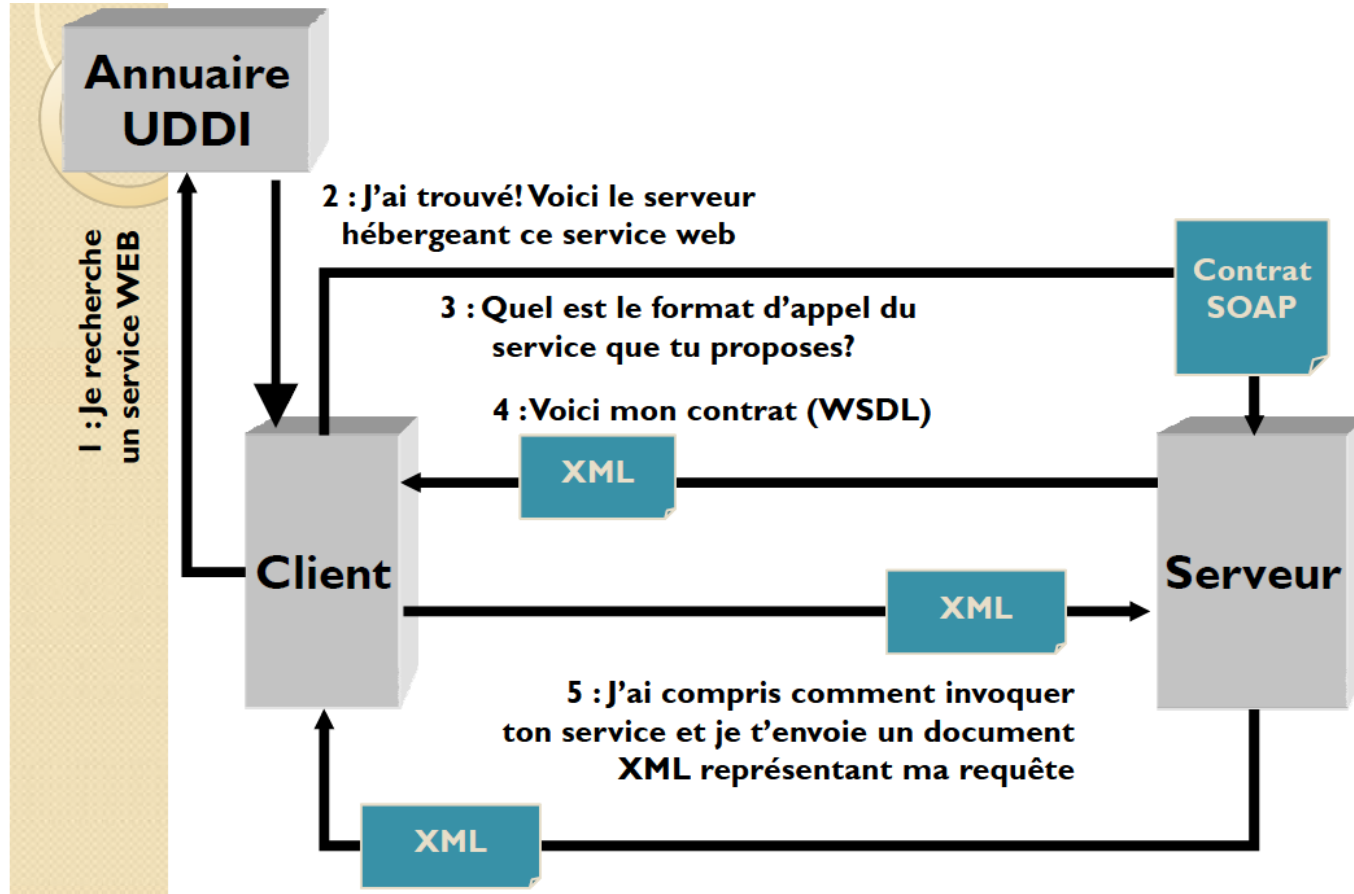
- ✓ **LES WEB SERVICES SONT DES COMPOSANTS WEB BASÉS SUR INTERNET (HTTP) EXÉCUTANT DES TÂCHES PRÉCISES ET RESPECTANT UN FORMAT SPÉCIFIQUE**
- ✓ **ILS FACILITENT L'ÉCHANGE DE DONNÉES INTER APPLICATIVE**
- ✓ **LES WEB SERVICES PERMETTENT AUX APPLICATIONS DE DIALOGUER À TRAVERS LE RÉSEAU INDÉPENDAMMENT DE LEUR PLATE-FORME D'EXÉCUTION ET DE LEUR LANGAGE D'IMPLÉMENTATION**
- ✓ **ILS S'INSCRIVENT DANS D'INITIATIVES TELLES QUE CORBA EN APPORTANT TOUTE FOIS UNE RÉPONSE PLUS SIMPLE, S'APPUYANT SUR DES TECHNOLOGIES ET STANDARDS RECONNUS ET MAINTENANT ACCEPTÉS DE TOUS**

CONCEPTS DES WEB SERVICES

✓ LE CONCEPT DES **WEB SERVICES** S'ARTICULE ACTUELLEMENT AUTOUR DES TROIS CONCEPTS SUIVANTS :

- SOAP (SIMPLE OBJECT ACCESS PROTOCOL)
 - PROTOCOLE D'ÉCHANGE INTER-APPLICATIONS INDÉPENDANT DE TOUTE PLATE-FORME BASÉ SUR LE LANGAGE XML
- WSDL (WEB SERVICES DESCRIPTION LANGUAGE)
 - DONNE LA DESCRIPTION DANS UN FORMAT XML DES WEB SERVICES EN PRÉCISANT LES MÉTHODES POUVANT ÊTRE INVOQUÉES
- UDDI (UNIVERSAL DESCRIPTION DISCOVERY AND INTEGRATION)
 - PERMET LA PUBLICATION, L'EXPLORATION ET LA DÉCOUVERTE DES WEB SERVICES
 - UDDI SE COMPORTE LUI-MÊME COMME UN WEB SERVICE DONT LES MÉTHODES SONT APPELÉES VIA LE PROTOCOLE SOAP.

CYCLE DE VIE D'UTILISATION



SOAP

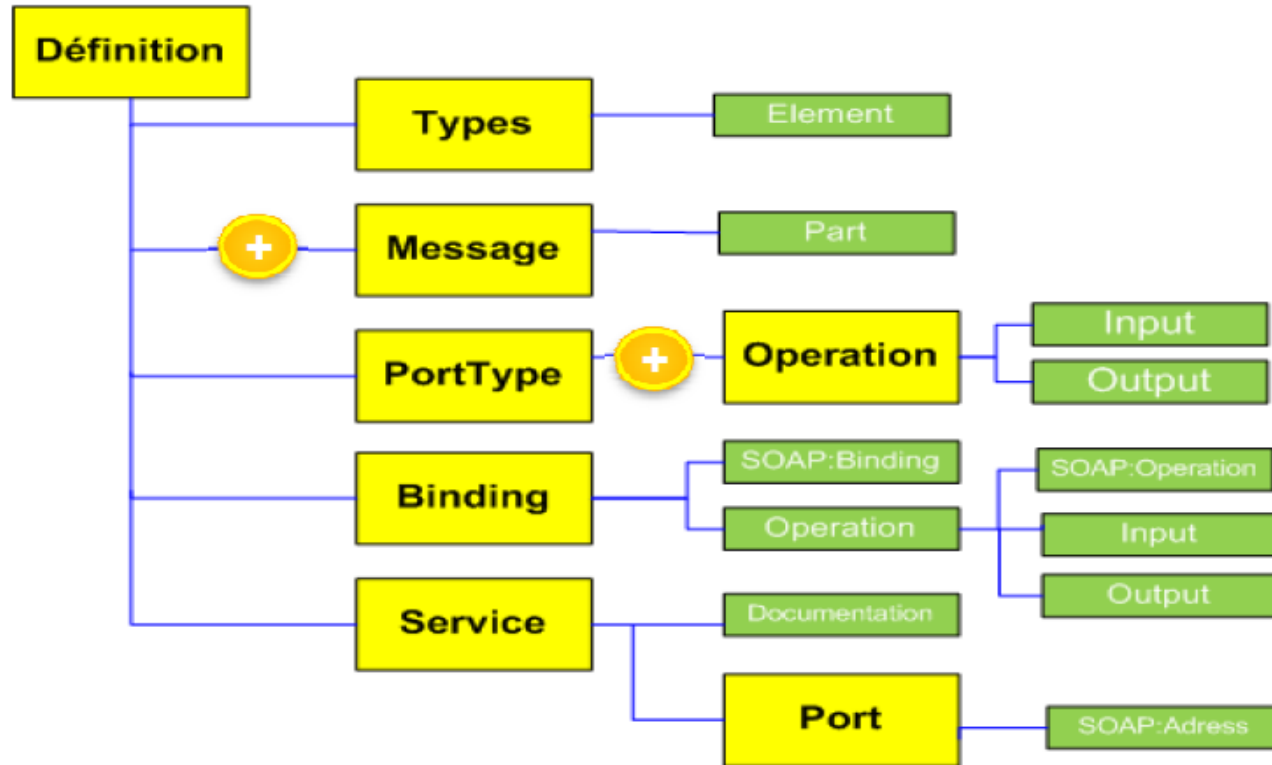
- ✓ **SOAP** EST UN PROTOCOLE D'INVOCATION DE MÉTHODES SUR DES SERVICES DISTANTS BASÉ SUR XML
- ✓ **SOAP** A POUR PRINCIPAL OBJECTIF D'ASSURER LA COMMUNICATION ENTRE MACHINES
- ✓ **STRUCTURE D'UN MESSAGE SOAP**
 - **SOAP ENVELOPPE**
 - **SOAP HEADER**
 - **SOAP BODY**
 - **SOAP FAULT**

WSDL

- ✓ **UN DOCUMENT WSDL SE COMPOSE D'UN ENSEMBLE D'ÉLÉMENTS DÉCRIVANT LES TYPES DE DONNÉES UTILISÉS PAR LE SERVICE, LES MESSAGES QUE LE SERVICE PEUT RECEVOIR, AINSI QUE LES LIAISONS SOAP ASSOCIÉES À CHAQUE MESSAGE.**

- ✓ TYPES: FOURNIT LA DÉFINITION DE TYPES DE DONNÉES UTILISÉES POUR DÉCRIRE LES MESSAGES ÉCHANGÉS
- ✓ MESSAGES: REPRÉSENTE UNE DÉFINITION DES DONNÉES EN COURS DE TRANSMISSION
- ✓ PORTTYPES: DÉCRIT UN ENSEMBLE D'OPÉRATIONS
- ✓ BINDING: SPÉCIFIE UNE LIASON ENTRE PORTTYPE ET UN PROTOCOLE CONCRET (SOAP, HTTP ...)
- ✓ SERVICE : INDIQUE LES ADRESSES DE PORT DE CHAQUE LIAISON
- ✓ PORT : REPRÉSENTE UN POINT D'ACCÈS DE SERVICES DÉFINI PAR UNE ADRESSE RÉSEAU ET UNE LIAISON.
- ✓ OPÉRATION : C'EST LA DESCRIPTION D'UNE ACTION EXPOSÉE DANS LE PORT.

WSDL



Structure d'un document WSDL

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- **LE SOA**
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

PRENONS UN PEU DE HAUTEUR

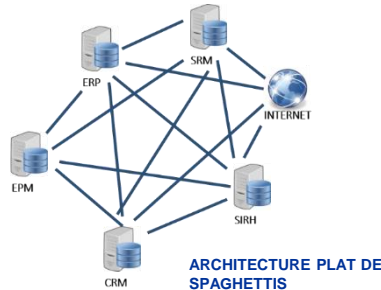
SYSTÈMES D'INFORMATION: PROBLÉMATIQUES

✓ CONSTRUCTION DES SI

- CHAQUE DOMAINE MÉTIER BÂTIT UN SOUS-SYSTÈME QUI LUI EST PROPRE
- UTILISATION DE TECHNOLOGIES HÉTÉROGÈNES ET RAREMENT INTEROPÉRABLES

✓ INTÉGRATION INTRA-ENTREPRISE

- DÉCOUPAGE DE L'ENTREPRISE EN DÉPARTEMENT FONCTIONNEL REPRIS PAR LE SYSTÈME D'INFORMATION
 - EXISTENCE DE SILO
- SI = PLAT DE SPAGHETTIS
 - COÛT DE MAINTENANCE ET D'ÉVOLUTION



✓ INTÉGRATION INTER-ENTREPRISE

- INTERACTION NÉCESSAIRE AVEC LES SI DES PARTENAIRES (FOURNISSEURS, CLIENTS,...)

SOA= SERVICE ORIENTED ARCHITECTURE (ARCHITECTURE ORIENTÉE SERVICE)

✓ **ARCHITECTURE**

- PARADIGME ABSTRAIT
- NI UNE IMPLÉMENTATION, NI UN LOGICIEL OU UNE TECHNOLOGIE EN PARTICULIER, MAIS UNE DESCRIPTION DE LA STRUCTURE DU SYSTÈME
- COMMENT LE SYSTÈME DOIT ÊTRE ORGANISÉ

✓ **SERVICE**

- ACTIVITÉ MÉTIER RÉUTILISABLE AVEC UNE INTERFACE D'UTILISATION (ENTRÉE/SORTIE) SPÉCIFIÉE
- VISION ABSTRAITE ORIENTÉE MÉTIER D'UNE FONCTIONNALITÉ

SOA – ARCHITECTURE ORIENTÉE SERVICE

DEFINITION (2/2)

✓ **LE SOA EST UN STYLE D'ARCHITECTURE UTILISÉ DANS LE DÉVELOPPEMENT D'APPLICATIONS, BASÉ**

SUR DES SERVICES « RÉUTILISABLES », « INTEROPÉRABLES » ET « EN COUPLAGES LÂCHES »

- LE SOA N'EST PAS LIÉ À UNE TECHNOLOGIE ... MAIS IL MET EN ŒUVRE DES STANDARDS INFORMATIQUES
- LE SOA SE BASE SUR LA DIVISION DES PROCESSUS MÉTIER EN SERVICES

POUR FONCTIONNER DANS UNE ARCHITECTURE SOA, UN SERVICE DOIT ÊTRE ...

RÉUTILISABLE



LES SERVICES SONT CONÇUS
AFIN D'ÊTRE UTILISABLES QUEL
QUE SOIT LE CONTEXTE

INTEROPÉRABLE



LES SERVICES SONT DÉFINIS À
PARTIR DE STANDARDS DU
MARCHÉ AFIN D'OFFRIR LE
MAXIMUM D'INTERACTION EN
INTERNE, COMME EN EXTERNE

À COUPLAGE LÂCHE



LE COUPLAGE LÂCHE DES
SERVICES OFFRE DES
POSSIBILITÉS D'AGENCEMENT
MULTIPLES ET MODULABLES



SOA – ARCHITECTURE ORIENTÉE SERVICE

POURQUOI PENSER SOA

DISPOSER D'UN SI AGILE ET OUVERT

✓ DÉSORMAIS SEUL LE CHANGEMENT EST PERMANANT

- LES SYSTÈMES D'INFORMATION SONT EN GÉNÉRAL CONSTITUÉS D'APPLICATIONS RIGIDES, MONOLITHIQUES, DÉVELOPPÉES AU COURS DU TEMPS POUR INSTRUMENTER DES PROCESSUS MÉTIER RÉPUTÉS STABLES
 - LES PROCESSUS DOIVENT ÉVOLUER EN PERMANENCE POUR S'ADAPTER AUX ÉVOLUTIONS DE PÉRIMÈTRE, DE RÉGLEMENTATION, D'ALLIANCES, DE POSITIONNEMENT CONCURRENTIEL, ...

➔ FAVORISER L'AGILITÉ ET ANTICIPER

SOA – ARCHITECTURE ORIENTÉE SERVICE

UNE EVOLUTION

AVANT

ORIENTÉE FONCTIONNALITÉS
CONÇU POUR DURER
DÉVELOPPEMENT LONG
SILOS APPLICATIFS
COUPLAGE FORT
ORIENTÉ OBJET

APRES

ORIENTÉE SERVICES
CONÇU POUR CHANGER
DÉVELOPPEMENT ET DÉPLOIEMENT
RAPIDE
APPLICATIONS ET DONNÉES
DÉSILOTÉES
COUPLAGE FAIBLE
ORIENTÉ MESSAGE

SOA – ARCHITECTURE ORIENTÉE SERVICE

LES PRINCIPALES IMPLÉMENTATIONS

- ✓ **ESB : ENTERPRISE SERVICE BUS** – BUS DE COMMUNICATION PERMETTANT DE METTRE EN RELATION DIFFÉRENTES APPLICATIONS QUI N'ONT PAS ÉTÉ CONÇUES POUR FONCTIONNER ENSEMBLE. C'EST UNE MÉTHODE DE MÉDIATION INTER-APPLICATIVE
- ✓ **BPM: BUSINESS PROCESS MANAGEMENT** – A POUR BUT DE MAÎTRISER L'ORCHESTRATION DES PROCESSUS MÉTIER.
- ✓ **ETL: EXTRACT TRANSFORM LOAD** – PERMET D'EXTRAIRE DES DONNÉES BRUTES DEPUIS UNE SOURCE DE DONNÉES, POUR ENSUITE LES RESTRUCTURER, ET ENFIN LES CHARGER DANS UNE AUTRE SOURCE DE DONNÉES
- ✓ **MDM: MASTER DATA MANAGEMENT** – UNE TECHNOLOGIE PERMETTANT LA GOUVERNANCE DES DONNÉES DE RÉFÉRENCES

ESB – ENTERPRISE SERVICE BUS

LES ENJEUX (1/2)

C'EST LE COMPOSANT PRINCIPAL D'UNE APPROCHE SOA

✓ IL GÈRE LES PROBLÉMATIQUES D'INTÉGRATION DE L'ENTREPRISE:

- INTÉGRATION DE DONNÉES (RÉFÉRENTIELS), DE SERVICES, D'APPLICATIONS EXISTANTES ET DE NOUVELLES APPLICATIONS, DE NOUVEAUX PRODIGES DANS UN ENVIRONNEMENT EXISTANT, DE NOUVEAUX PROCESSUS, DE PARTENAIRES ...
- GESTION DE LA CONNECTIVITÉ, LA TRANSFORMATION ET ROUTAGE DES DONNÉES, LA LOGIQUE D'ÉCHANGE...

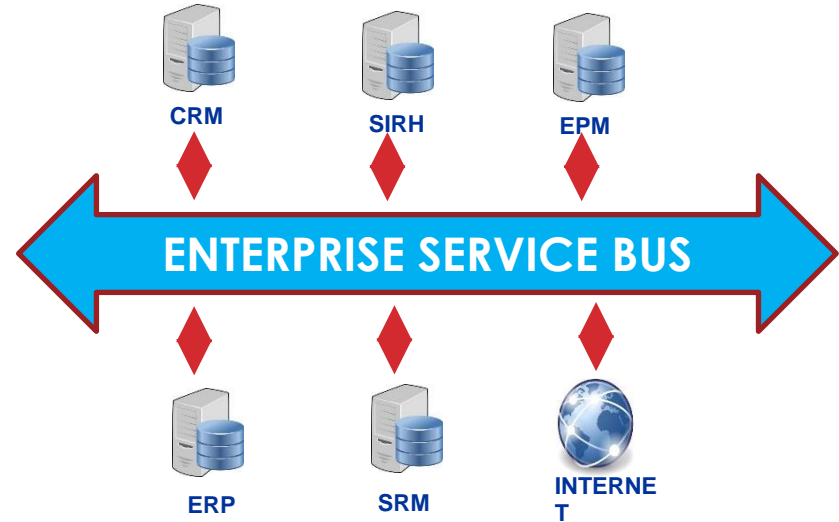
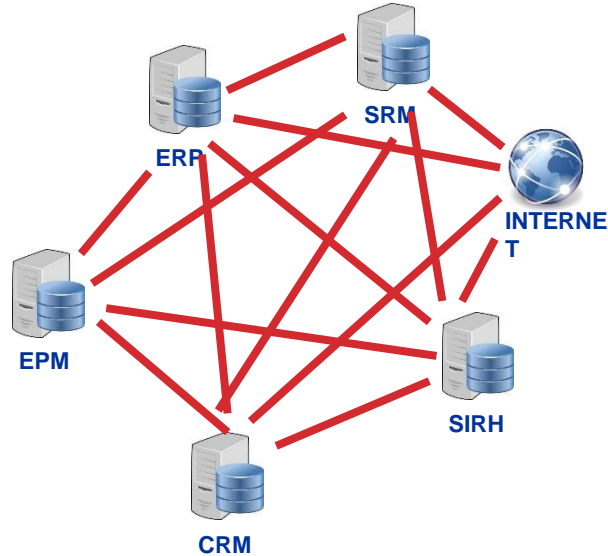
✓ PRINCIPES:

FÉDÉRER	Rationaliser	Maîtriser l'évolution du SI
<ul style="list-style-type: none">▪ UNE PLATE-FORME D'INTÉGRATION FÉDÉRATRICE▪ CONTRE-PIED DE L'APPROCHE POINT À POINT (SPAGHETTIWARE)▪ CENTRALISATION DES TECHNOLOGIES D'INTÉGRATION, DU DÉVELOPPEMENT DES COMPOSANTES D'INTÉGRATION, DE L'ADMINISTRATION ET DU SUIVI DES ÉCHANGES	<ul style="list-style-type: none">▪ UNE PLATEFORME D'INTÉGRATION BASÉE SUR LES STANDARDS (JMS, XML, JSON...)▪ RÉUTILISATION DES COMPOSANTES D'INTÉGRATION (CONNECTEURS, PROCESSUS)▪ DIMINUTION DES COÛTS D'INTERFAÇAGES, DE MAINTENANCE DES INTERFACES ET D'EXPLOITATION	<ul style="list-style-type: none">▪ STANDARDISATION, CAPITALISATION, HOMOGENÉISATION▪ DÉCOUPLAGE DES APPLICATIONS▪ AMÉLIORATION DE LA FLEXIBILITÉ DU SI▪ AMÉLIORATION DE LA COHÉRENCE DU SI : VISION GLOBALE

ESB – ENTERPRISE SERVICE BUS

LES ENJEUX (2/2)

PASSER D'UN SI MULTI-APPLICATIFS CONNECTÉS EN MODE POINT-À-POINT À UN SI URBANISÉ, AUX ÉCHANGES MUTUALISÉS ET NORMALISÉS



ESB – ENTERPRISE SERVICE BUS

QUELQUES ÉLÉMENTS DE VOCABULAIRE

- ✓ **API - INTERFACE DE PROGRAMMATION APPLICATIVE:** TECHNOLOGIE PERMETTANT À UN DÉVELOPPEUR DE POUVOIR UTILISER UN PROGRAMME INDÉPENDAMMENT DE SA TECHNOLOGIE DE PROGRAMMATION.
- ✓ **INTERFACE :** C'EST UN POINT D'ENTRÉE UNIQUE POUR UNE FONCTION DONNÉE. ELLE PEUT PRENDRE LA FORME D'UNE API, D'UNE TABLE D'ÉCHANGE, D'UN FICHIER D'ÉCHANGE, D'UNE FILE D'ATTENTE ...
- ✓ **CONTRAT D'INTERFACE:** C'EST UN ACCORD ENTRE LE PRODUCTEUR DE L'INTERFACE ET LE CONSOMMATEUR DE L'INTERFACE. IL FORMALISE L'ENGAGEMENT DU PRODUCTEUR QUANT AU SERVICE RENDU ET LES CONDITIONS D'UTILISATION À RESPECTER PAR L'UTILISATEUR.
- ✓ **BROKER DE MESSAGES:** C'EST UNE TECHNOLOGIE QUI PERMET DE GÉRER UNE FILE D'ATTENTE DE MESSAGES SELON LE PRINCIPE PRODUCTEUR DE MESSAGES-FILE D'ATTENTE-CONSOMMATEUR DE MESSAGES
- ✓ **FORMAT D'ÉCHANGE OU FORMAT DES DONNÉES:** C'EST LA FAÇON DONT SONT REPRÉSENTÉS LES MESSAGES EN ENTRÉE ET EN SORTIE DE L'ESB (JSON, XML, COPYBOOK, CSV ...)
- ✓ **PATTERN:** UN PATTERN EST UNE ABSTRACTION, UNE NORMALISATION D'UN CAS D'UTILISATION DES SERVICES DE L'ESB

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

Vous avez dit micro-services ?

Késako ?



**« PETITS SERVICES
AUTONOMES QUI
FONCTIONNENT
ENSEMBLE »**

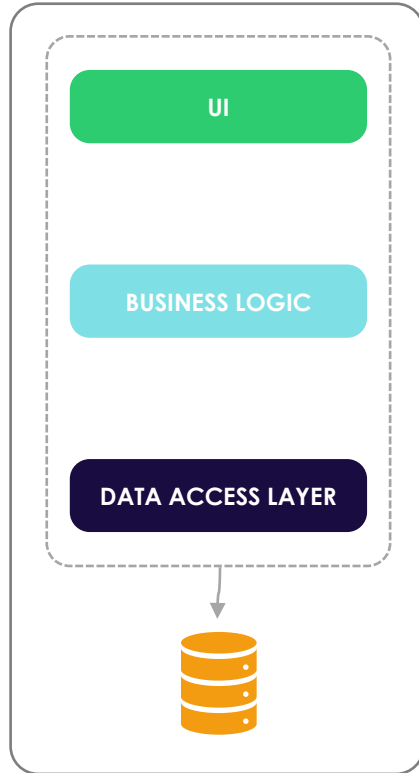


**« ARCHITECTURE ORIENTÉE
SERVICES, FAIBLEMENT
COUPLÉE ET AVEC DES
CONTEXTES DÉLIMITÉS »**

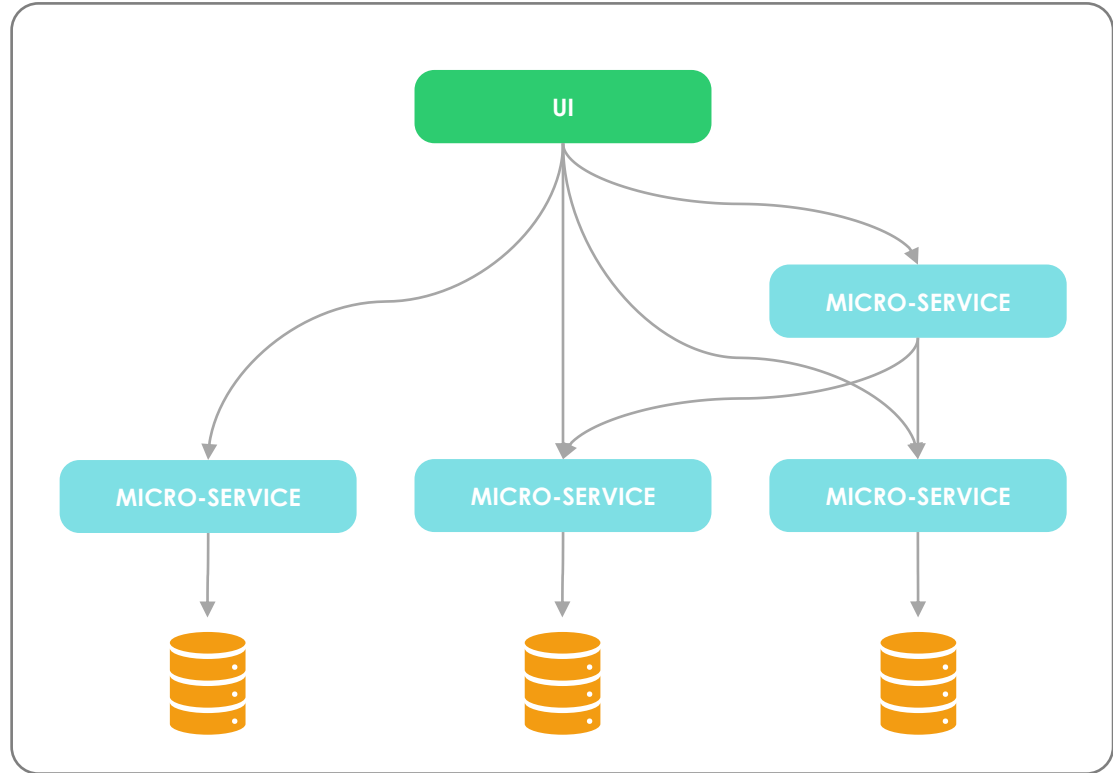


Diviser pour mieux régner

Architecture monolithique vs. Micro-services



VS.

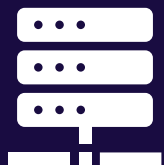


Du monolithique aux micro-services

Exemple : application de livraison de nourriture

ARCHITECTURE MONOLITHIQUE

Une grosse application



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

Clients

ARCHITECTURE ORIENTÉE SERVICE

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

Connecteur
envoi SMS

Connecteur
envoi email

Connecteur
paiement



Interfaces
web

Base de
données

API
REST/SOAP



Clients

ARCHITECTURE MICRO-SERVICE

Une application découpée par des services fonctionnels

Bus d'événements



Service
paiement

Service
facturation

Service
notifications



Service
commande client

Service
restaurant

Interface web
commande client



Clients

Caractéristiques des micro-services

Exemple : application de commande et livraison de nourriture à domicile

Fonctionnalité unique

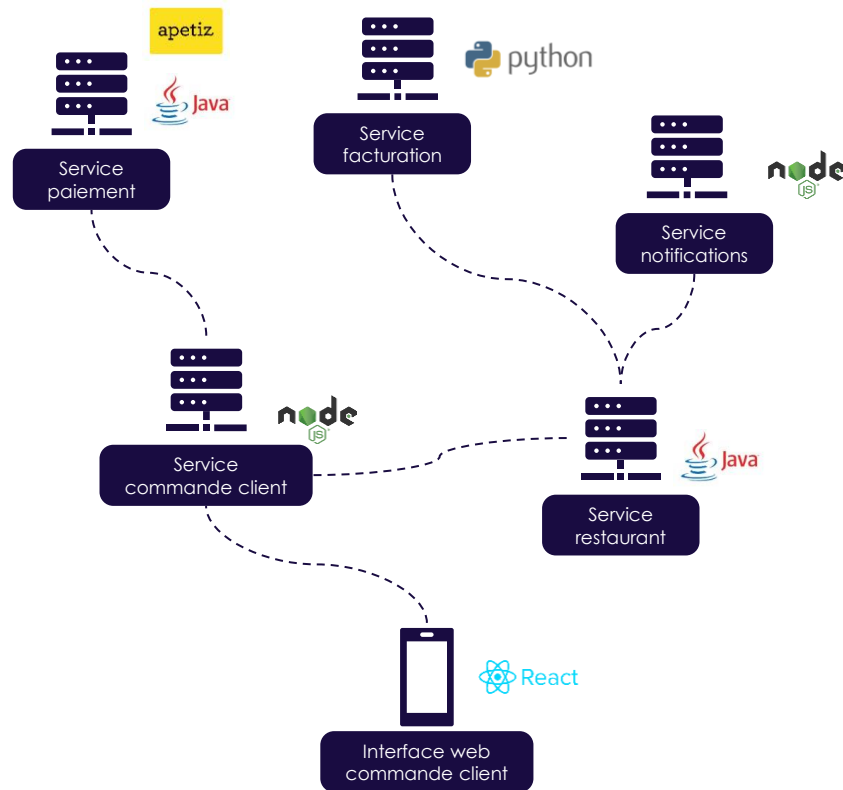
Flexibilité technologique

Capacité de montée en charge

Équipe réduite et pluridisciplinaire

Déploiement ciblé

Tests facilités



Avantages des architectures micro-services

A la recherche du R.O.I.

RÉDUCTION DU TTM

Màj fréquentes
Cycle de dev. Courts

RÉSILIENCE

OUVERTURE

API
Technologique



HAUTE DISPONIBILITÉ DES SERVICES

FACILITÉ DE DÉPLOIEMENT

ADAPTÉS AUX MÉTHODES AGILES

Les défis d'une architecture micro-services

Trucs et astuces

**BIEN
CONCEVOIR
ET DÉLIMITER
SES SERVICES**

**BIEN GÉRER LA
COMMUNICATION
ENTRE SERVICES**

API
Versionning
Tests
Mécanismes de
communication

**MAÎTRISER
L'INFRASTRUCTURE
DE DÉPLOIEMENT**



**MAÎTRISER
LES MÉCANISMES
DE DÉPLOIEMENT**

**BIEN GÉRER
L'ÉCOSYSTÈME
DES SERVICES**

Vision globale, accès
Monitoring, traçabilité et
gestion des pannes
Centralisation des logs
Authentification

**S'APPUYER
SUR LES
TECHNOLOGIES
ADAPTÉES**

Non mais allo quoi !

Bien gérer la communication entre les services



API

REST
Contrats de services



VERSIONING

Compatibilité ascendante
Avoir plusieurs versions d'un
même micro-service en
parallèle



TESTS

S'appuyer sur les contrats de
service
Réaliser des tests d'intégration



LE MÉCANISME DE COMMUNICATION LE PLUS ADAPTÉ

Appel direct : couplage,
latence
Asynchronisme :
indépendance, complexité

Appliquer la bonne communication et le bon standard

En fonction de vos besoins et de votre architecture



API REST.

- Se servir des micro-services existants pour créer un nouveau service



GRAPHQL

- Créer des requêtes pour accéder à la bonne information



WebDAV



GeoJSON



HL7



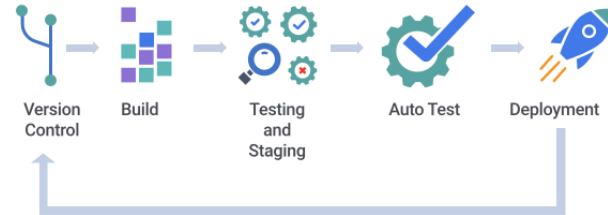
GTFS

Maîtriser le déploiement

Infrastructure et mécanismes



CI/CD Pipeline



Jenkins



GitLab

Bien gérer l'écosystème des services

Supervision des services



**VISION GLOBALE,
ACCÈS AUX SERVICES**

Mise en place d'outils de type API
Gateway et API Management
Contrôle de l'état et de la mise à
disposition des services
Monétisation des services

AUTHENTIFICATION



SERVICES

**MONITORING, TRAÇABILITÉ
ET GESTION DES PANNES**



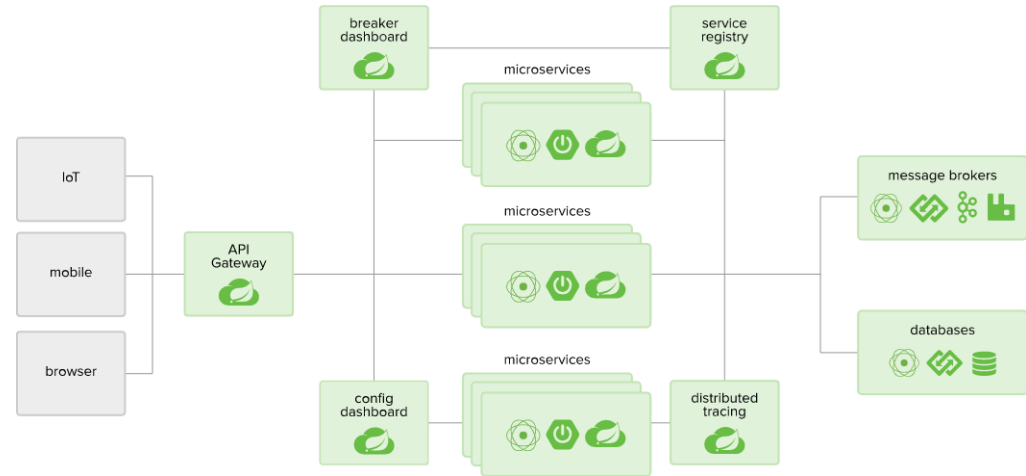
**CENTRALISATION
DE LOGS**

S'appuyer sur des technologies adaptées

MICRO FRAMEWORKS



FRAMEWORKS ET OUTILS



Des services 3D

Bien concevoir et délimiter ses services

LES MICRO-SERVICES OFFRENT DE NOMBREUX AVANTAGES...

... MAIS REQUIÈRENT UN CERTAIN NIVEAU DE MATURITÉ

(ORGANISATIONNEL, MÉTIER, TECHNIQUE, ...)

IL EST CONSEILLÉ D'Y ALLER PROGRESSIVEMENT

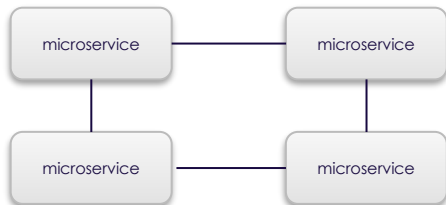
SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

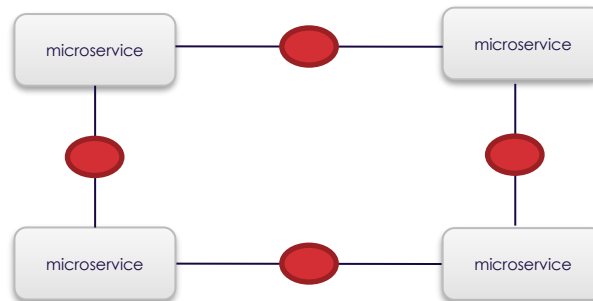
AVANTAGES

- ✓ **ARCHITECTURE ADAPTÉE AUX BESOINS MÉTIER**
- ✓ **RÉSILIENCE**
- ✓ **SCALABILITÉ**
- ✓ **FACILITÉ DE DÉPLOIEMENT**
- ✓ **PETITES ÉQUIPES CONCENTRÉES SUR LEURS SERVICES**
- ✓ **EVOLUTION CAR FACILITÉ DE REMPLACEMENT**

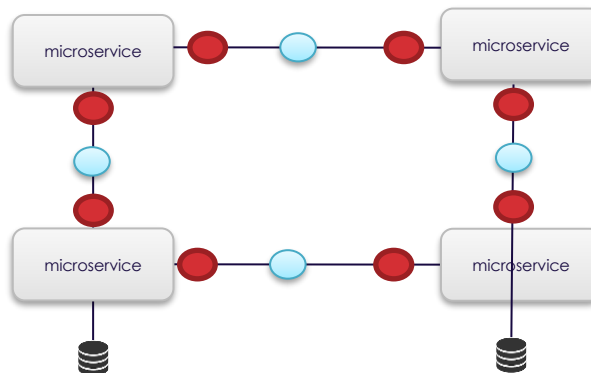
INCONVENIENTS



Communication
réseau



Load balancer
et Base de
données



INCONVENIENTS

- ✓ **LE NOMBRE D'ÉLÉMENTS À SURVEILLER AUGMENTE**
- ✓ **AUGMENTATION DU TRAFFIC RÉSEAU**
- ✓ **LE COÛT DE LA LATENCE AUGMENTE**
- ✓ **LE TRAÇAGE DES REQUÊTES**

RECOMMANDATIONS

✓ AU MINIMUM

- UN DÉPLOIEMENT ET UN PROVISIONNING RAPIDE DES APPLICATIONS
- EFFECTUER UN MONITORING
- CULTURE DEVOPS
- LOG EFFICACE DES TRANSACTIONS MÉTIERS
- « DÉVELOPPEURS MULTI ENVIRONNEMENT »

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

ETAPE PAR ETAPE

✓ DÉFINIR LES RAISONS QUI VOUS POUSSENT À DÉCOUPER VOTRE APPLICATION

- UNE PARTIE DE VOTRE APPLICATION VA OPÉRER DE GROS CHANGEMENTS
- LA STRUCTURE DE VOTRE ÉQUIPE (MULTI OFFICE)
- LA SÉCURITÉ
- LA TECHNOLOGIE

✓ DÉCOUPER UN MONOLITHE

- PENSER À FAIRE UN DÉCOUPAGE PAR ITÉRATION
- ÉVITER UN DÉCOUPAGE TROP FIN, TROP ORIENTÉ SUR LA TECHNIQUE

DÉCOUPER UN MONOLITHE

- ✓ **PENSER À FAIRE UN DÉCOUPAGE PAR ITÉRATION**
- ✓ **ÉVITER UN DÉCOUPAGE TROP FIN , TROP ORIENTÉ SUR LA TECHNIQUE**

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

Définition d'une API

1 API =

- 1 objet métier
- N opérations
- N objets satellites
- N règles de sécurité

1 API = 1 puits de données

1 API = 1 Swagger

1 Swagger = 1 documentation (portail)

1 Swagger = 1 collection de mocks

Nommage de l'API

Le nommage doit prendre en compte les informations suivantes :

- nom de l'api
- version majeure de l'api
- nom de la ressource
- nom des ressources parentes si nécessaires

Les données issues des données de prélèvements du module des adhérents doit pouvoir être exposées comme il suit

- [https://api-\[EnvCouloir\].\[Sous domaine\].krj.gie/\[nom de l'api\]/\[Version\]/\[URI ressource\]](https://api-[EnvCouloir].[Sous domaine].krj.gie/[nom de l'api]/[Version]/[URI ressource])
- Ex : <https://api-d01.dcaas.krj.gie/partenaires/v1/conventions/999999>

L'introduction du terme 'api' offre une plus grande souplesse quant aux données exposées sur un serveur.

Il pourra a terme y avoir des points d'entrées orientés documentation, monitoring, audit, configuration, ...

Les Verbes HTTP - GET

Le verbe GET est utilisé pour les interrogations

Toutes les informations sont dans l'URL. Le corps du message est vide.

Les retours se font sous la forme d'un flux JSON.

Exemples d'utilisations :

- GET /adherents
 - retourne la liste de tous les adhérents
- GET /adherents?departement=045&nom=CORNU
 - retourne la liste des adhérents dont le nom est CORNU dans le département 45.
- GET /adherents/12345
 - retourne l'adhérent avec l'identifiant '12345'
- GET /adherents/12345/telephones
 - retourne la liste des toutes les numéros de téléphone de l'adhérent '12345'

Les Verbes HTTP - POST

Le verbe POST est utilisé pour les créations

Les informations nécessaires au traitement sont passées dans le corps du message au format JSON.

Renvoie d'un flux JSON contenant l'ID (ou toutes les informations) de l'entité créée

Exemples d'utilisations :

- **POST /adherents**

```
{ "nom": "Dupont", "prenom": "Jean" }
```

- Créé Mr Dupont dans la base des adhérents

- **POST /adherents/12345/telephones**

```
{ "numero": "0102030405", "nom": "fixe1" }
```

- Rajoute un numéro de téléphone à l'adhérents '12345'

Les Verbes HTTP - PUT

Le verbe PUT est utilisé pour mettre à jour une entité **de façon complète**, et la créer si celle-ci n'existe pas.

Les informations nécessaires au traitement sont passées dans le corps du message au format JSON.

Dans le cadre des méthodes PUT l'appel sur la même ressource avec les mêmes paramètres doit retourner la même chose.

Exemples d'utilisations :

- PUT /adherents/12345

```
{"nom": "Dupont", "prenom": "Albert"}
```

- Modifie les toutes les informations de l'adhérents '12345'

- PUT /adherents/12345/telephones/fixe1

```
{"numero": "0102030499"}
```

- Modifie le numéro de téléphone 'fixe' de l'adhérents '12345'

Les Verbes HTTP - PATCH

- Le verbe PATCH est utilisé pour mettre à jour une entité **de façon partielle**.
- Les informations nécessaires au traitement sont passées dans le corps du message au format JSON.
- Exemples d'utilisations :
 - `PATCH /adherents/12345`
`{"prenom": "Albert"}`
 - Modifie uniquement le prénom de l'adhérent '12345'

Les Verbes HTTP - DELETE

Le verbe DELETE est utilisé pour supprimer des informations.

L'appel d'un DELETE sur une entité déjà supprimée doit retourner un statut OK (204).

Exemples d'utilisations :

- DELETE /adherents
 - Supprime tous adhérents
- DELETE /adherents/12345
 - Supprime l'adhérent '12345'
- DELETE /adherents/12345/telephones/fixe1
 - Supprime le téléphone 'fixe1' de l'adhérent '12345'

Les Verbes HTTP - DELETE

Le verbe DELETE est utilisé pour supprimer des informations.

L'appel d'un DELETE sur une entité déjà supprimée doit retourner un statut OK (204).

Exemples d'utilisations :

- DELETE /adherents
 - Supprime tous adhérents
- DELETE /adherents/12345
 - Supprime l'adhérent '12345'
- DELETE /adherents/12345/telephones/fixe1
 - Supprime le téléphone 'fixe1' de l'adhérent '12345'

Les codes HTTP

Les codes HTTP sont normalisées et découpés en grandes familles. Elles même découpées avec des sous codes permettant d'expliquer le traitement de la requête.

Famille	Description
2XX	Le serveur a bien traité la requête, aucune erreur n'est survenue, le détail du code de retour donne une indication à l'utilisateur de l'API de comment le serveur a traité sa requête
3XX	Le serveur a déplacé une ressource et propose une redirection, ces codes sont principalement utilisé dans le cadre de traitement long pour indiquer à l'utilisateur de l'API que la ressource n'est pas encore disponible mais qu'un traitement est en cours
4XX	Le serveur n'a pas pu traiter la requête car des informations fournies par l'utilisateurs sont manquantes, incomplètes ou erronées. Il s'agit d'erreurs fonctionnelles sur lesquelles l'utilisateur de l'API a la main pour corriger son erreur
5XX	Le serveur n'a pas pu traiter la requête car un élément de sa structure est en défaut, il peut s'agit d'un bug de code, d'un traitement trop long, d'un service non disponible

2XX – Status OK

Les codes 2XX expriment que le serveur a bien traité la requête.

Code	Status	Description
200	OK	Requête traitée avec succès
201	CREATED	Création d'une ressource, <i>dans le niveau de maturité 3 des API Rest, la réponse doit retourner l'url d'accès à cette ressource nouvellement créée</i>
202	ACCEPTED	Traitement pris en compte avec un traitement asynchrone à la clé
204	NO CONTENT	Requête traitée avec succès ne nécessitant pas de réponses (Exemple : Suppression d'une ressource)
206	PARTIAL CONTENT	La réponse retournée est un contenu partiel (Gestion de pagination), la réponse doit porter les urls pour accéder aux éléments suivants
207	MULTI STATUS	Réponse apportée lorsqu'une requête permet de créer / mettre à jour, supprimer plusieurs ressources. Le contenu de la réponse donne le statut d'exécution pour chaque ressource.

4XX – Erreurs fonctionnelles

Les données fournies ne permettent pas d'exécuter la requête.

Code	Status	Description
400	BAD REQUEST	La syntaxe de la requête est erronée, des champs nécessaires ne sont pas fournis, le format des données est incorrect
401	UNAUTHORIZED	L'accès à la ressource nécessite une authentification non fournie ou non valide
402	PAYMENT REQUIRED	L'accès à la ressource nécessite des crédits / quotas / licence
403	FORBIDDEN	La ressource n'est pas accessible pour l'utilisateur authentifié
404	NOT FOUND	La ressource demandée n'existe pas
405	METHOD NOT ALLOWED	Le verbe utilisé sur cette ressource n'est pas autorisé.
406	NOT ACCEPTABLE	Le format du message n'est pas cohérent avec les propositions de l'API (attendu json / fourni xml)
408	REQUEST TIME-OUT	La requête risque de prendre trop de temps ou de retourner trop de lignes. Les critères de recherche sont à affiner.

5XX – Erreurs techniques

Les codes commençant par 5XX représentent les erreurs techniques, sur lesquelles l'utilisateur de l'API n'a pas la main.

Il peut s'agit d'un service indisponible, ou d'une erreur qui n'aurait pas du se produire sur le serveur (Exemple base de données non accessible)

Les erreurs techniques pouvant être de multiple cause, il est important de ne pas remonter d'information trop détaillée pour des raisons de sécurité

Code	Status	Description
500	INTERNAL ERROR	Erreur interne inattendue est survenue, cela peut provenir d'une base de données ou d'un service tiers inaccessible, d'une accès disque impossible, d'un bug, ...)
501	NOT IMPLEMENTED	Le service est prévu mais non implémenté à l'heure actuelle
503	SERVICE UNAVAILABLE	Le service ne répond pas, est en maintenance, ou est actuellement trop sollicité
504	GATEWAY TIME-OUT	Le temps d'attente d'exécution de l'API est écoulé

SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

Approche développement d'un swagger

Deux approches possibles (*) :

- Cliquez / Bouton via un outil comme Apicur.io
 - Pas de serveur apicur.io actuellement
 - Ne contient pas l'ensemble de la norme OAS 3
 - Ne permet pas de faire référence à certains éléments / projets apicur.io externes
- As Code - Possibilité d'utiliser l'ensemble de la norme OAS 3
 - Nécessite de maîtriser le yaml
 - Template d'exemples complet
 - Répartition du swagger sous des fichiers yaml unitaires
 - Éléments de swagger réutilisables
 - Objets métiers
 - Type de réponses (400, 401, 404, 500, 503)
 - Paramètres
 - Serveurs / Méthodes

* Attention le passage de l'un à l'autre n'est pas toujours possible et garanti



Attention à l'effet cliquodrome et la re-saisie d'information

Peut nécessiter du développement pour contourner les effets chronophage du design de swaggers via un outil graphique



Démarrage rapide des développements

Nécessite une bonne connaissance de OAS 3

Possibilité de réfléchir à du tooling de génération de blocs avec déjà les éléments communs (erreurs / ...)

Ecriture d'une API

La description d'API doit présenter un ensemble d'informations, la norme pour ce faire est l'Open Api Specification.

L'OAS3 définit les points d'entrées suivants :


Champs	Description
info	Informations de l'API (titre, description, termes d'utilisation, contact, licence et version)
tags	Liste des tags permettant de regrouper plusieurs opérations
servers	Liste des serveurs disponibles (mock, essai, ...)
components	Liste des objets métiers qui sont référencés par les apis
paths	Liste des ressources de l'API et les verbes associés
security	Liste des mécanismes d'authentification à l'API

Informations d'un swagger : « info »

- Donne des informations générales sur l'API

~ INFO

Version


1.0.0 

Description


API de gestion des partenaires

~ CONTACT


Name

Fanny Pascal 


Email

fpascal@mgen.fr 

URL

No URL provided. 

~ LICENSE

[MGEN](#) 

You have configured a license that we are not familiar with. For information about the license, click the link above! Or else click the button below to choose a different license..

Change License

```
# =====  
# Information sur l'API  
# =====  
info:  
  title: Partenaires  
  version: 1.0.0  
  description: API de gestion des partenaires  
  license:  
    name: MGEN  
    url: https://api.mgen.fr/license  
  contact:  
    name: Fanny Pascal  
    email: fpascal@mgen.fr
```

Définition des tags

Permet de regrouper les opérations pour une meilleur lisibilité

▼ TAG DEFINITIONS (6)



- > Contact Elément en lien avec les contacts d'un partenaire
- > Employeur Elément en lien avec la gestion d'un employeur
- > Etablissement Elément en lien avec la gestion d'un établissement
- > Partenaire Elément en lien avec la gestion d'un ou plusieurs partenaires
- > ProfessionnelDeSante Elément en lien avec la gestion d'un professionnel de santé
- > Tiers Elément en lien avec la gestion d'un tiers

```
# =====  
# Liste des tags pour organiser nos APIs dans Swagger-ui  
# =====  
tags:  
  - name: Partenaire  
    description: Elément en lien avec la gestion d'un ou plusieurs partenaires  
  - name: Contact  
    description: Elément en lien avec les contacts d'un partenaire  
  - name: ProfessionnelDeSante  
    description: Elément en lien avec la gestion d'un professionnel de santé  
  - name: Etablissement  
    description: Elément en lien avec la gestion d'un établissement  
  - name: Tiers  
    description: Elément en lien avec la gestion d'un tiers  
  - name: Employeur  
    description: Elément en lien avec la gestion d'un employeur
```

Partenaire Elément en lien avec la gestion d'un ou plusieurs partenaires ▼

- GET /partenaires Récupère une collection de partenaires
- GET /partenaires/{id-part} Détail d'un partenaire
- POST /partenaires/{id-part}/evenements Ajoute un événement à un partenaire
- PUT /partenaires/{id-part}/evenements/{idx-evt} Modifie un des événements associé à un partenaire
- DELETE /partenaires/{id-part}/evenements/{idx-evt} Supprime un des événements associé à un partenaire






Contact Elément en lien avec les contacts d'un partenaire ▼

- GET /partenaires/{id-part}/contacts Informations de contact d'un partenaire
- POST /partenaires/{id-part}/contacts Ajoute un contact à un partenaire

Définition des servers

Identifie un ensemble d'environnements, à disposition sur le réseau (local ou internet).
Le portail généré à partir du swagger permet d'appeler les APIs sur n'importe lequel de ces environnements (y compris la prod*).

▼ SERVERS (6)

>  http://localhost:8080	Mocks locaux
>  https://dev.mgen.fr/api/partenaire/v1	Environnement de dev
>  https://int.mgen.fr/api/partenaire/v1	Environnement d'intégration
>  https://mocks.mgen.fr/api/partenaire/v1	Mocks officiels
>  https://partenaire.mgen.fr/v1	Environnement de prod
>  https://pprod.mgen.fr/api/partenaire/v1	Environnement de pré-production

```
# =====  
# Les serveurs d'APIs à disposition  
# =====  
servers:  
- url: 'http://localhost:8080'  
  description: Mocks locaux  
- url: 'https://mocks.mgen.fr/api/partenaire/v1'  
  description: Mocks officiels  
- url: 'https://dev.mgen.fr/api/partenaire/v1'  
  description: Environnement de dev  
- url: 'https://int.mgen.fr/api/partenaire/v1'  
  description: Environnement d'intégration  
- url: 'https://pprod.mgen.fr/api/partenaire/v1'  
  description: Environnement de pré-production  
- url: 'https://partenaire.mgen.fr/v1'  
  description: Environnement de prod
```

* Un appel nécessite un jeton qui ne peut être obtenu qu'avec le login/mot de passe/validation smartphone/etc... de l'utilisateur lui-même. Il n'y a donc pas de problème de sécurité

Définition des composants

Définition d'un ensemble d'éléments ré-utilisables :

- Des objets : components/schemas
- Des réponses standard : components/responses
- Des paramètres : components/parameters
- Et d'autres éléments

On peut ensuite faire référence à ces éléments via un « \$ref » local (commençant par un « # »)



Un outil comme Apicurio ne sait gérer que des composants de type « schema » !

```
components:
  schemas:
    IBAN:
      description: Coordonnées bancaires de type IBAN
      type: object
      properties:
        titulaire:
          description: Le nom du titulaire du compte
          type: string
        iban:
          description: L'IBAN identifiant le compte
          type: string
  parameters:
    id-part:
      name: id-part
      description: Identifiant du partenaire
      in: path
      required: true
      schema:
        type: string

paths:
  '/partenaires/{id-part}/iban':
    get:
      parameters:
        - $ref: '#/components/parameters/id-part'
      operationId: get-partenaire-iban
      description: Cette opération permet de récupérer l'IBAN d'un partenaire
      tags:
        - Partenaire
      responses:
        '200':
          description: Le partenaire existe, et voici son IBAN.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/IBAN'
```

Définition des paths

Restful :

- On manipule des « collections » et des « ressources »
- Chacune d'elles est identifiée par une URI (= un chemin = un path)
- On les manipule ensuite avec un verbe HTTP

```
paths:
  '/partenaires/{id-part}/iban':
    get:
      parameters:
        - $ref: '#/components/parameters/id-part'
      operationId: get-partenaire-iban
      description: Cette opération permet de récupérer l'IBAN d'un partenaire
      tags:
        - Partenaire
      responses:
        '200':
          description: Le partenaire existe, et voici son IBAN.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/IBAN'
```

/partenaires/{id-part}/iban

Design Source

~ INFO

Summary
No Summary ✎

Description
No description, ✎

> SERVERS

~ PATH PARAMETERS (1)

id-part > No description, > No Type [+ Create](#)

> QUERY PARAMETERS

> HEADER PARAMETERS

~ OPERATIONS (1)

Get Put Post Delete Options Head Patch Trace

~ INFO

Summary
No Summary ✎

Operation ID
get-partenaire-iban ✎

Description
Cette opération permet de récupérer l'IBAN d'un partenaire

Tags
Partenaire ✎

> SERVERS

~ PATH PARAMETERS (1)

id-part > No description, > No Type [+ Create](#)

> QUERY PARAMETERS

> HEADER PARAMETERS

> REQUEST BODY

~ RESPONSES (1)

200 OK

Description
Le partenaire existe, et voici son IBAN.

Response Body

☐ application/json > IBAN > No examples defined.
[Add a media type](#)

~ SECURITY REQUIREMENTS

No security requirements have been configured. [Add security requirement](#)

Définition des security

Le swagger permet de modéliser :

- La liste des droits (scopes) nécessaires pour accéder à l'API
- Une liste supplémentaire pour accéder à un chemin donné
- Et une liste supplémentaire pour accéder à chaque opération

Une API =

- Des ressources (objets métiers, satellites)
- La manière d'y accéder = Les chemins (paths)
- Les droits nécessaires = Les « security »

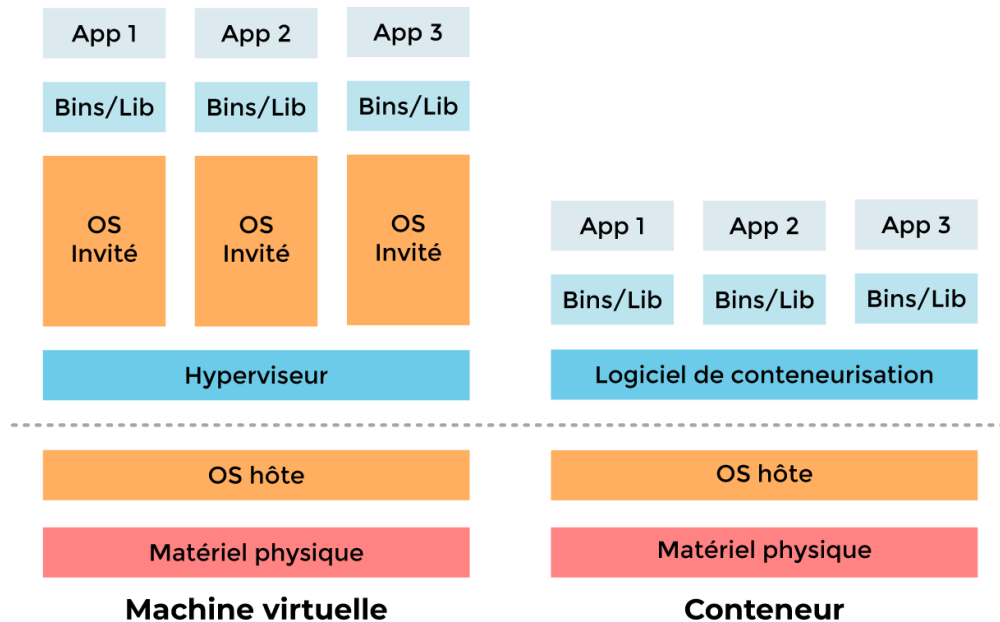
SOMMAIRE

- INTRODUCTION
- ARCHITECTURE MONOLITHIQUE
- LES WEB SERVICES
- LE SOA
- LES MICROSERVICES
- AVANTAGES ET INCONVÉNIENTS DES MICROSERVICES
- DÉCOUPER UNE APPLICATION MONOLITHIQUE
- LES API RESTFUL
- LES SWAGGERS
- DOCKER

QU'EST CE QU'UN CONTENEUR

✓ POURQUOI UTILISER LES CONTENEURS

- GESTION COUTEUSE ET DE NOMBREUX PROBLÈMES LOGISTIQUE AVEC LES SERVEURS PHYSIQUES
- LES MACHINES VIRTUELLES SONT LENTES À DÉMARRER ET RESERVE LES RESSOURCES DU SYSTÈME HÔTE



AVANTAGES DES CONTENEURS

- ✓ LES CONTENEURS N'UTILISENT QUE LES RESSOURCES NÉCESSAIRES
- ✓ LES CONTENEURS DÉMARRENT BEAUCOUP PLUS RAPIDEMENT QU'UNE VM OU UN SERVEUR PHYSIQUE
- ✓ DONNE BEAUCOUP PLUS D'AUTONOMIE AUX DÉVELOPPEURS

Les conteneurs permettent de réduire les coûts et d'augmenter la densité de l'infrastructure

DOCKER

- ✓ **DOCKER EST UN SYSTÈME D'EXPLOITATION POUR CONTENEUR. C'EST UNE PLATEFORME LOGICIELLE QUI PERMET DE CONCEVOIR, TESTER ET DÉPLOYER DES APPLICATIONS RAPIDEMENT.**
- ✓ **DOCKER EST UTILISABLE SUR TOUS LES ENVIRONNEMENTS**
- ✓ **UN CONTENEUR DOCKER DOIT ÊTRE STATELESS ET IMMuable**
- ✓ **TROIS VERSIONS DISPONIBLE:**
 - DOCKER COMMUNITY EDITION
 - DOCKER DESKTOP
 - DOCKER ENTERPRISE

DOCKER: QUELQUES COMMANDES

- ✓ **DOCKER RUN: PERMET DE DÉMARRER UN CONTENEUR**
 - -D DÉTACHE LE CONTENEUR DU PROCESSUS PRINCIPAL
 - -P DÉFINI L'UTILISATION DE PORT SPÉCIFIQUE
- ✓ **DOCKER STOP ID_RETOURNÉ_LORS_DU_DOCKER_RUN : ARRÊTE UN CONTENEUR**
- ✓ **DOCKER RM ID_RETOURNÉ_LORS_DU_DOCKER_RUN : SUPPRIME UN CONTENEUR**
- ✓ **DOCKER PULL: RÉCUPÈRE DES IMAGES DU DOCKER HUB**
- ✓ **DOCKER PS: AFFICHE L'ENSEMBLE DES CONTENEURS EXISTANTS**
- ✓ **DOCKER SYSTEM PRUNE : NETTOYAGE DU SYSTÈME**
 - -D DÉTACHE LE CONTENEUR DU PROCESSUS PRINCIPAL
 - -P DÉFINI L'UTILISATION DE PORT SPÉCIFIQUE