IUT Lyon 1
l'excellence technologique

Architecture basée sur des Microservices

V. Deslandres,

LP DevOps

IUT Lyon1

Historique

- Le terme date de 2011
- Netflix en a popularisé l’utilisation
  - Suite à un bug de son architecture monolithique (dans la BD) qui a bloqué le S.I. pendant 4 jours (2008)
    - Scinder les fonctionnalités dans des ‘systèmes’ indépendants appelés ‘services’ pour réparer plus rapidement la panne
    - Tout le code a été mis en open source sur GitHub pour favoriser ce style d’architecture
    - Chaque service est amélioré, de nouveaux services sont créés qui étoffent le système

+ Google, Amazon, Spotify, eBay

- **Problématique** : comment gérer le partage des ressources toujours plus consommatrices, plus variées, plus changeantes
  - *n* serveurs, *n* postes clients
  - « How do you scale ? » (article de Robert C. Martin, oct 2014)

Evolution

- La tendance est :
  - A l'adoption généralisée du développement **dans le Cloud**
  - A une demande continuelle **d'actualisation** des applications
  - A la mise à disposition de plus en plus de **frameworks** pour les services Web

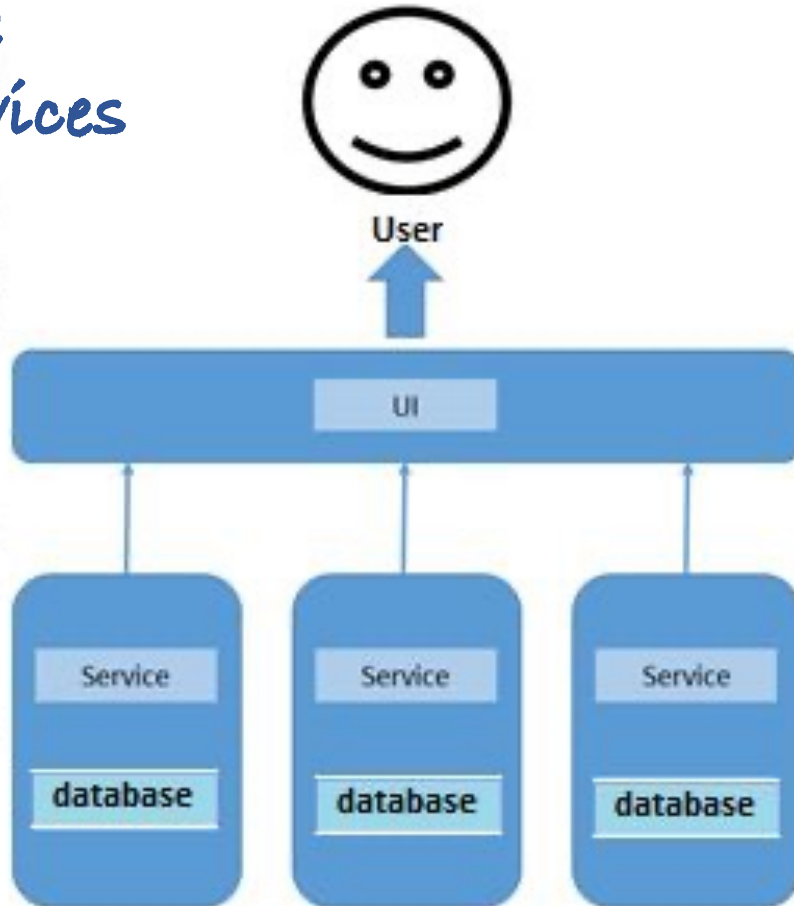
- IDC annonçait pour 2022 :
  - 90% des applications basées sur des architectures de microservices
  - 35% de ces applications seront **cloud-native,** càd. développées dans un environnement cloud en tant qu'IaaS
    - *Infrastructure as a Service :* applications dynamiques, càd. avec un nombre d'instances variables, distribuées sur des serveurs jetables, étendables
    - bref : modifiables ; avec un coût fonction de l'usage des ressources.

Description

- L'archi MS est une évolution de SOA mais avec des services plus indépendants. Dans une archi SOA, les services font partie d'un tout. Là ils sont autonomes mais utilisés conjointement par l'application.

- Services **petits**, remplissent une SEULE fonction, sont généralement asynchrones. Peuvent exploiter des langages / technologies différentes.
  - Petit : pouvant être développé par une équipe de 6 à 10 individus (« 2-pizza team »)
- Organisation du projet = équipes autonomes, qui développent et déploient sans avoir besoin des autres équipes. Mécanismes d'**automatisation,** de **déploiement incrémental** et de **tests**.
- Théoriquement chaque service doit être *élastique* (modifiable), *résilient, composable, minimal* (applications multi-tenant, on héberge tous nos clients sur une seule plateforme)*,* et *complet*.

Style Microservices



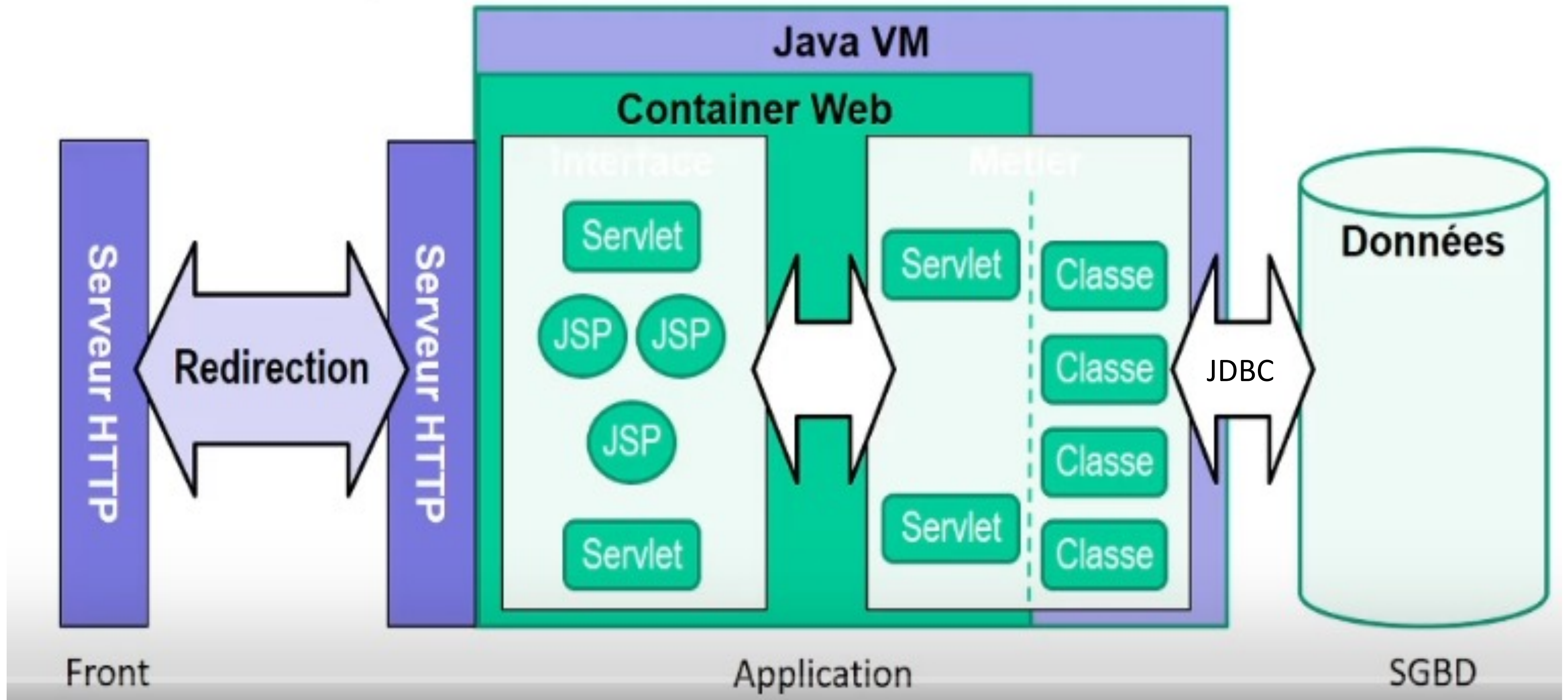
Les microservices, séparés et indépendants, fonctionnent en synergie pour accomplir les tâches

Style Monolythique

Les composants forment une entité indissociable

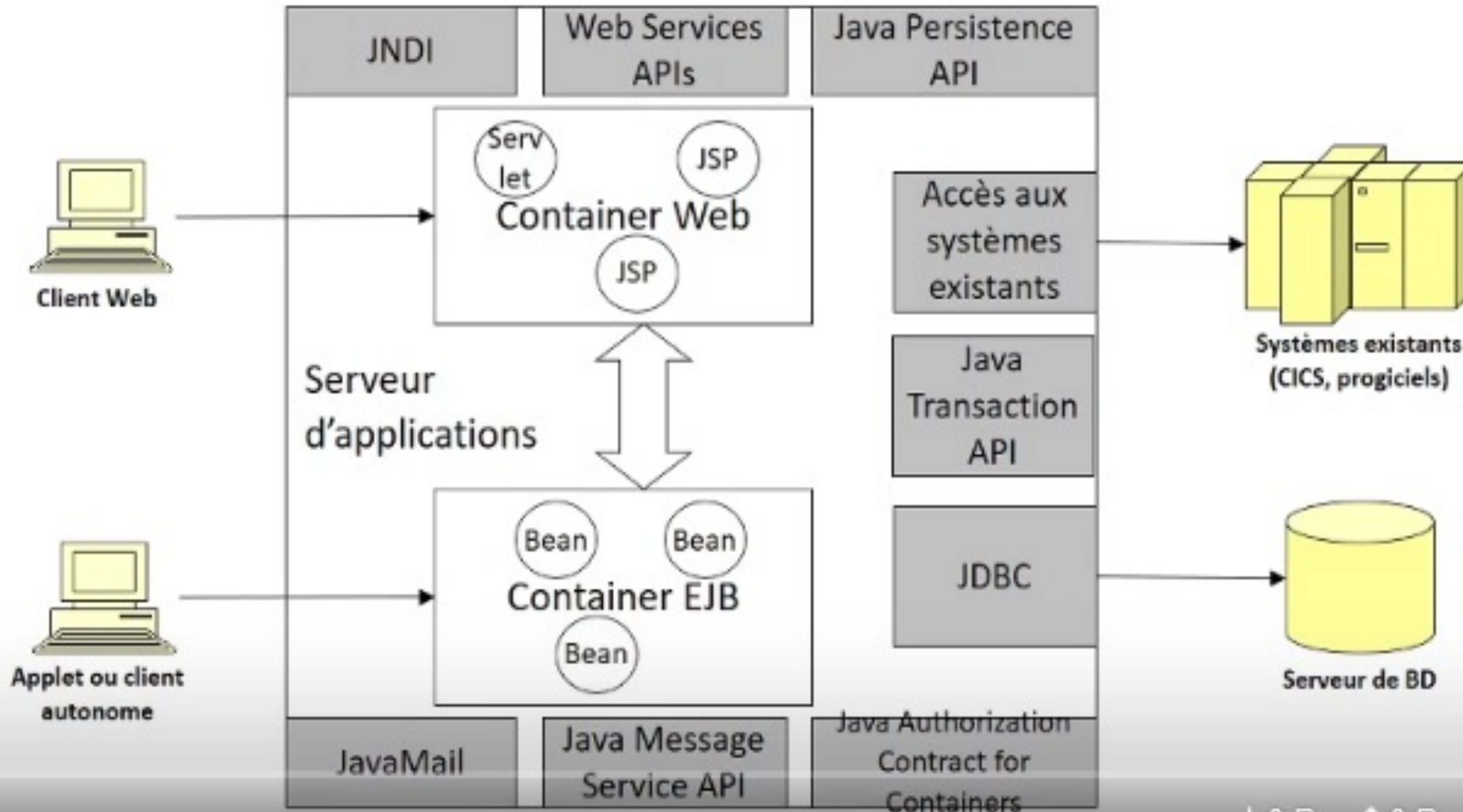
Source : https://www.ionos.fr/digitalguide/sites-internet/developpement-web/architecture-microservices/

Exemple d'application monolythique centralisée



Application web en Java SE (Standard Edition)

Exemple d'application monolythique distribuée



Java EE est une liste de **spécifications** de :

- services
- APIs
- protocoles

Application web en Java EE (Enterprise Edition)

Fonctionnement

- Bien que les microservices soient tous différents dans leur structure, ils doivent finalement avoir des moyens de communiquer communs.
- Ceux-ci doivent être aussi simples que possible.
- La plupart des développeurs de microservices font confiance aux API **REST***.
  - Grâce aux méthodes HTTP comme *GET* ou *POST*, les différents microservices peuvent facilement communiquer entre eux et échanger des informations.
  - Echanges synchrones avec HTTP, asynchrones avec AMQP - *Advanced Messaging Queuing Protocol*

** Diapo suivante*

Paradigme REST

Paradigme de programmation REST *REpresentational State Transfer (*Roy Fielding, PhD 2000) : un Client demande des données à un Serveur ressource

- Ne vise pas d'un protocole particulier même si HTTP est le plus souvent utilisé : le client d'une API REST navigue **uniquement avec des URLs** mises à disposition par le serveur ;
- **Serveur 'sans état'** = la requête du client doit transmettre toutes les informations requises pour fonctionner en l'état sur le serveur, sans dépendance à un contexte spécifique ; l'état des objets résultats n'est pas conservé entre 2 appels.
  - Toutefois les ressources sont sauvegardées entre 2 requêtes, ce qui permet quand même des transactions plus complexes que la simple récupération de données
- *Transfert des données sous différentes représentations :*
  - selon les exigences Client, formats HTML, JSON ou XML.

Implantation des microservices

- Deux modes de **virtualisation** sont possibles :
- Par conteneurs
  - Chaque conteneur contient un MS
  - Il partage le noyau du système d’exploitation avec les autres.
  - Dans les conteneurs, les microservices fonctionnent de manière totalement **autonome** : tout ce dont ils ont besoin est inclus dans le conteneur (technologies, langages, BD).
- Par machines virtuelles
  - On crée une machine virtuelle séparée pour chaque microservice, donc là aussi sont parfaitement **autonomes**
  - Chaque machine virtuelle nécessite son propre système d'exploitation et consomme donc plus de ressources.

Pro/cons Architecture de microservices

- Pro
  - Scalabilité **verticale** : si les ressources des serveurs deviennent trop limitées, on en ajoute un nouveau.
  - Scalabilité **horizontale** : la même application peut être instanciée rapidement, inutile de tout ré initialiser comme pour une application monolythique (versions des OS et des frameworks)
  - **Evolution facile** : renforcer ou modifier le service qui en a besoin, passer à une nouvelle version. De même, il est moins coûteux d'intégrer un service complètement nouveau dans le système.

Pro/cons Architecture de microservices

- Pro (suite)
  - Système global **plus robuste** : quand un MS tombe en panne, seule sa fonctionnalité manque, et la recherche de panne est facilitée.
  - L'architecture MS est adaptée quand l'organisation **n'a pas** de système de contrôle **centralisé** des développements
  - Quand l'application monolithique a pris une **certaine ampleur**, basculer vers une archi MS est une bonne chose. Mieux vaut quand même toujours démarrer en style monolithique.
  - **Indépendance des équipes** individuelles de développement.

Pro/cons Architecture de microservices

- Cons
  - Demande plus d’effort dans sa mise en place, surtout pour les petits systèmes qui ont peu de tâches à effectuer
    - Loi de **Conway** : une petite équipe qui n’est pas divisée **effectue plus de choses** que la même équipe, scindée en sous-groupes.
  - Maintenance, post-développement et surveillance plus complexes
    - Les résultats de chaque service sont en effet faciles à analyser et mesurer ; avec un grand nombre de services, la tâche de surveillance devient importante.
  - Pour les habitués des protocoles d’échanges structurés tels que SOAP, la transition vers REST est plus difficile. Mais SOAP demande **plus de ressources** (VM, OS, plateformes d’exécution), des tâches **d’orchestration** évoluées et les **tests** sont souvent délicats.

Comment décomposer une apps en services ?

- **Par processus d'entreprise** ('business capabilities', ex. développement de produits, gestion marketing, gestion des commandes, des clients) organisé en plusieurs niveaux
  - Par ex. pour le dév. de Produit : études de marché, définition d'une stratégie Produit, élaboration des produits, lancement des produits
  - Chaque processus sera géré par un service

- Par les **sous-domaines** de l'application
  - Analyser l'organigramme et les différents domaines d'expertises
  - Chaque BusinessUnit pouvant exercer différentes expertises : une expertise par service

Ex.: trouver les MS d'une application de vente en ligne

Par Business Capabilities

- Analyser les processus d'entreprise et leurs liens

- Gestion du **Catalogue Produit**
- Gestion des **devis et des commandes**
- Factures/paiements → Gestion des **paiements/relances**

- Organisation des **Livraisons**
- Produit en stock ? → Gestion des **inventaires**

Ex.: trouver les MS d’une application de vente en ligne

Par analyse des sous-domaines

- Analyser l’organigramme et les différents domaines d’expertises

- **Commerciaux** : gestion du Catalogue Produit, gestion des commandes, organisation des Livraisons (sous-traitance)
  - 3 services indépendants

- **Achat/approvisionnement** : gestion des inventaires, des commandes fournisseurs
  - 2 services
- **Comptabilité** : gestion des paiements/relances

Critères fondamentaux à considérer pour constituer les équipes de microservices

- Les aspects humains
  - Petites équipes de développement : plus agiles, efficaces, autonomes
  - Capables d’automatiser, tester, déployer
    - Les former si besoin

- La granularité
  - Equipes indépendantes
  - Autonomes

Les problématiques à gérer en AMS

AMS : Architecture MicroService

- Comment gérer la persistance
  - Une BD par service, une BD partagée ?

- Comment gérer la communication entre Mircro Service
  - doit être minimale pour assurer un faible couplage entre les services
  - Technologie HTTP/REST
  - Bus de messages avec AMQP (Advanced Message Queuing Protocol) et un *message broker*