Université Claude Bernard Lyon 1

Intergiciels et Services
Microservices

Master 2 Traitement de l'Information et Web

Lionel Médini

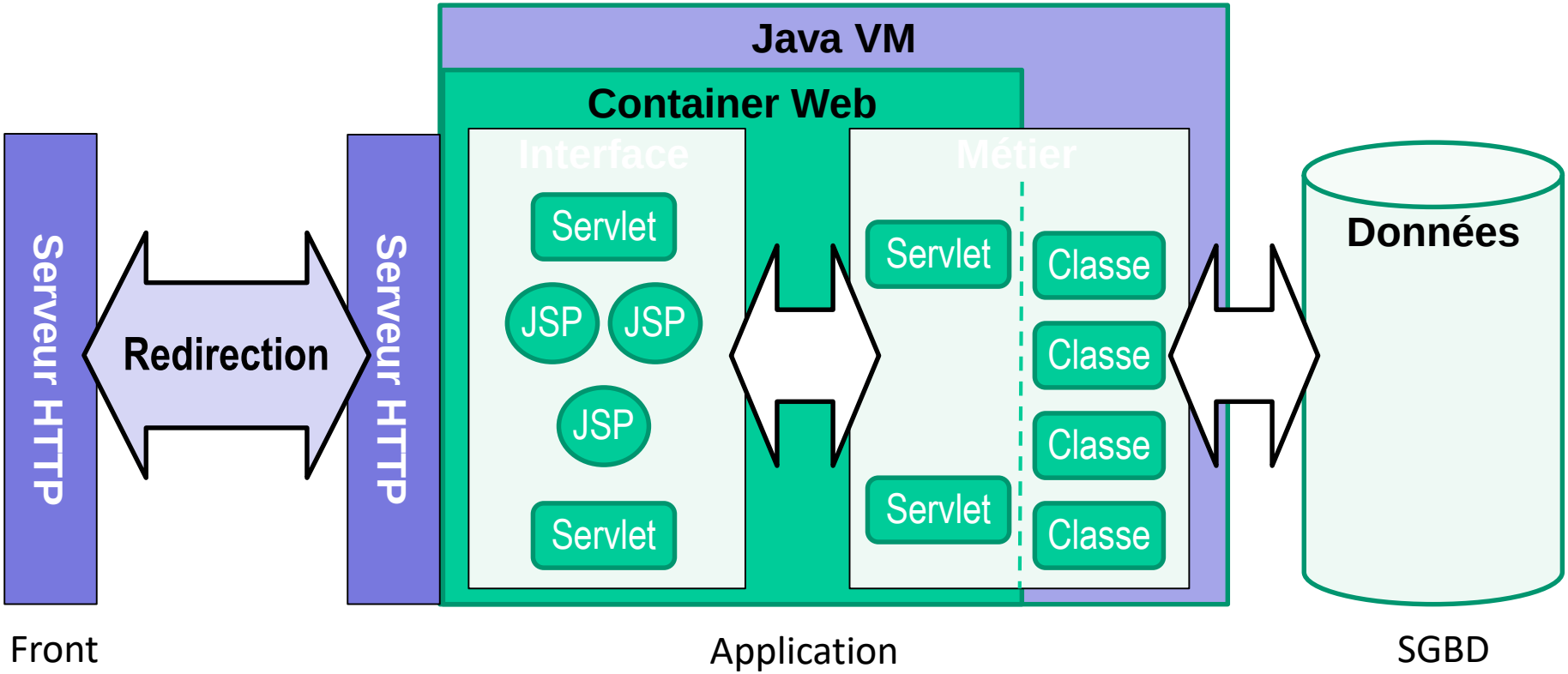
Janvier 2022

Objectifs de ce cours

- Un point de vue génie logiciel sur les architectures microservices
  - Caractéristiques des architectures microservices
  - Patterns liés aux architectures
  - Mise en place d'applications fondées sur ces architectures
- Complémentaire à l'approche TIW7

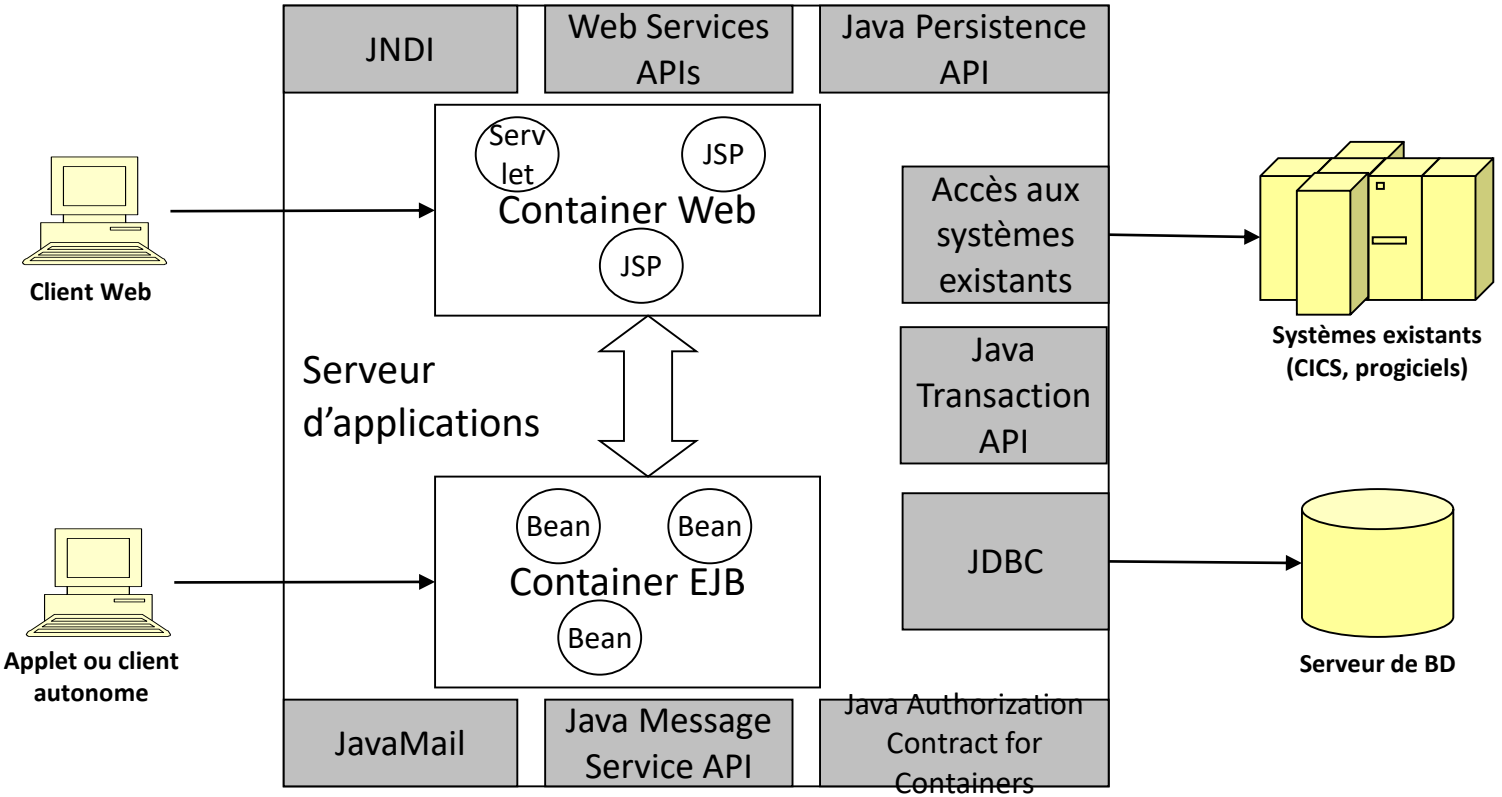
Position du problème

- L'architecture « monolithique »
  - Exemple d'architecture (Web)



Position du problème

- L'architecture « monolithique »
  - Exemple d'architecture (serveur d'applis)



Position du problème

- L’architecture « monolithique »
  - Avantages
    - Technologies homogènes
      - Types d’objets / composants adaptés aux besoins architecturaux
      - Stack « mainstream » -> facilement intégrable
    - Framework
      - Puissance, généricité, services annexes…
    - Déploiement
      - Simple, outils intégrés dans la stack
    - Cohésion de l’équipe de développement

Position du problème

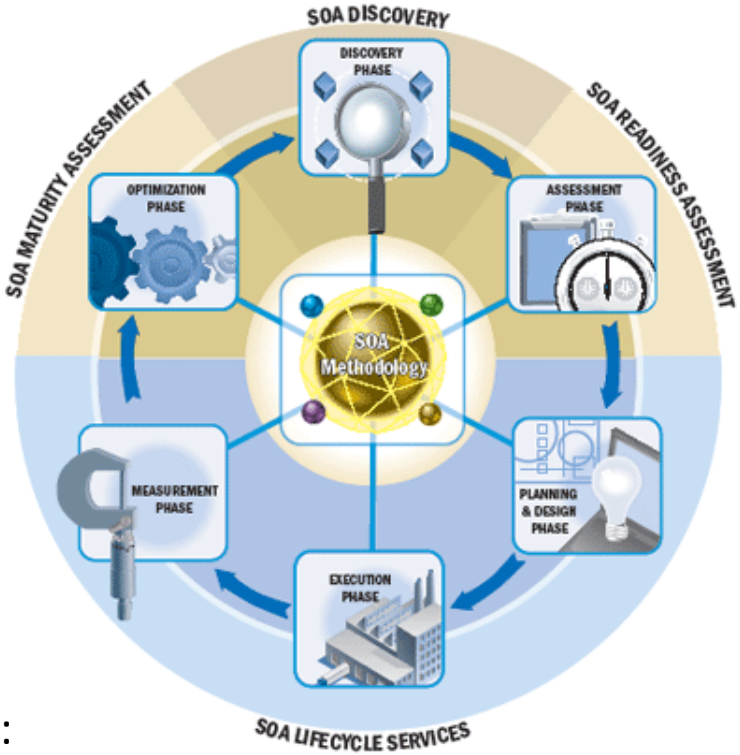
- L'architecture « monolithique »
  - Inconvénients
    - Technologies homogènes
      - Types d'objets / composants pas toujours adaptés aux besoins métier
      - Intégration de composants dans d'autres technos difficile
      - Choix technologiques à long terme
    - Framework
      - Peut devenir chargé quand la complexité de l'application augmente
    - Déploiement
      - Nécessite l'arrêt de toute l'application
      - Peut prendre du temps quand l'application grossit
    - Couplage fort entre les équipes de développement

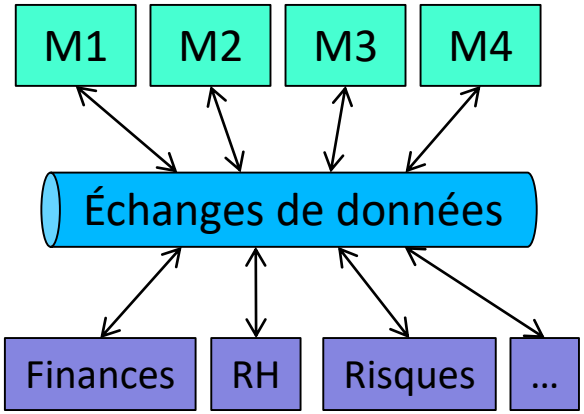
Position du problème

- L'architecture « monolithique »
  - Inconvénients
    - **Scalabilité**
      - Verticale
        - » L'application a la responsabilité de s'« auto-scaler »
        - » Limitée aux ressources du serveur
      - Horizontale
        - » Nécessite de réinstancier l'OS et le framework
        - » Gestion des ressources partagées à l'extérieur de l'application

Position du problème

- Les architectures orientées-services





Source : http://planforsoa.blogspot.fr/2012/02/soa-service-oriented-architecture.html

Position du problème

- Les architectures orientées-services
  - Avantages
    - Modularité, couplage faible
      - Séparation des préoccupations (et des développements)
      - Possibilité de redéployer composant par composant
    - Simplification du processus de développement
      - Liberté des choix technologiques
      - Indépendance des équipes de dév

Position du problème

- Les architectures orientées-services
  - Avantages
    - Interfaces de communication réseau
      - Pérennité des standards
      - Distribution possible
    - Plus adaptées
      - Aux environnements distribués
      - Aux échanges de services entre plusieurs applications
    - « Scalability by design »

Position du problème

- Les architectures orientées-services
  - Inconvénients
    - Complexité
      - Des mécanismes de communications
      - Des architectures applicatives
    - Overhead
      - Nécessite plus de ressources au final (VM, OS, plateforme d'exécution…)
      - Nécessite des services de monitoring/coordination
    - Testabilité plus délicate

Microservices : définition

- Martin Fowler :

« While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability,
automated deployment,
intelligence in the endpoints, and
decentralized control of languages and data. »

Source : https://martinfowler.com/articles/microservices.html

Microservices : définition

- En (plus) clair :
  - Une évolution des SOA
  - Une approche permettant de developer une application sous la forme de modules atomiques faiblement couplés
  - Un moyen de déployer et de maintenir les services indépendamment les uns des autres

Microservice : définition

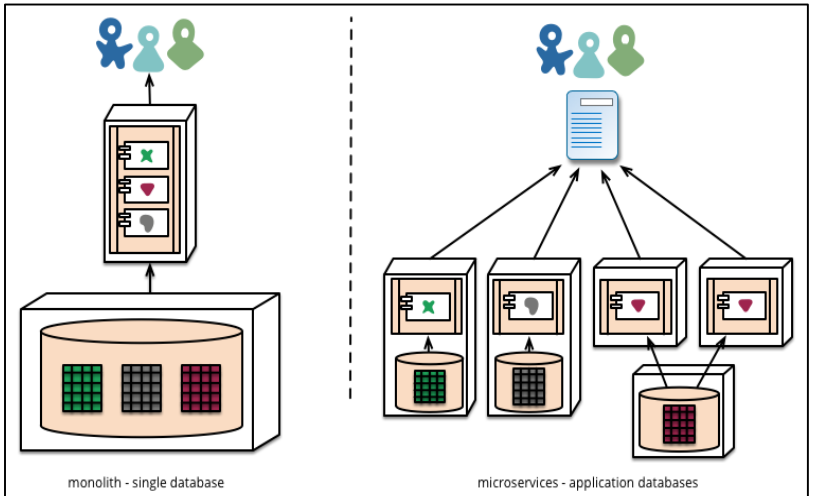
- Caractéristiques d'un microservice
  - Minimal mais complet / autonome
    - fournit une fonctionalité de l'application (**forte cohésion**)
  - Composable
    - expose une API simple aux autres microservices de l'application (ex : **REST**)
  - Élastique
    - (auto-)scalable
  - Résilient
    - ne bloque pas l'application en cas de panne (**couplage faible**)

Concepts

- Application
  - Une application = un assemblage de « petits » services indépendants
  - Chaque microservice réalise un processus métier ou une préoccupation transverse
    - « capability », « unité fonctionnelle »
    - Ex : vente, CRM, comptabilité, front-end, GUI…
  - Techniquement, les services sont
    - Programmés dans des langages hétérogènes
    - Exécutés dans des processus séparés
    - Liés à leurs propres supports de persistance
    - Développés et déployés dans des projets distincts

Concepts

- Bounded context
  - Pattern issu du Domain-Driven Design
  - Découpage d'un projet en groupements fonctionnels
  - Isolation de ces groupements entre eux
  - Chaque groupement est un mini-projet indépendant
    - Développement
    - Déploiement
    - Services de support et de pilotage



Source : https://martinfowler.com/articles/microservices.html

➔ Un microservice est un « running bounded context »

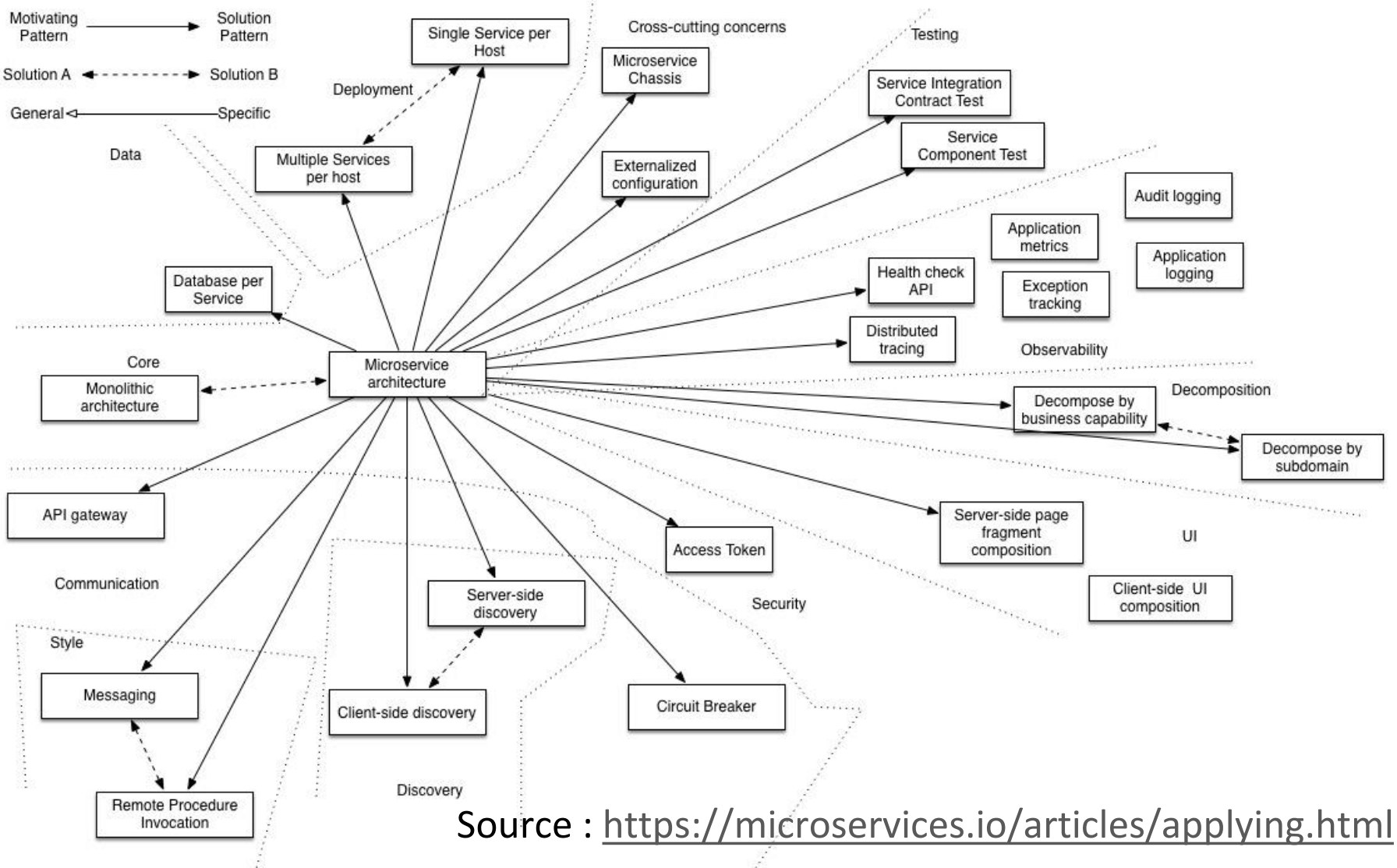
Concepts

- « Smart endpoint, dumb pipe »
  - La gestion centralisée de l'application est réduite au minimum
    - Chaque service embarque un morceau de l' « intelligence » de l'application
    - Les services peuvent utiliser des mécanismes de stockage différents
    - La logique de communication est répartie dans les services et non dans un médium de communication centralisé (ESB)
    - Communication par mécanismes « légers »

Méthodologie

- Comment concevoir et déployer une application à base de microservices ?
  - Maintenable
  - Scalable
- Hypothèse
  - Application complexe : framework, nombreux use cases…
  - Utilisation de patterns d'isolation : couches, adapter, façade…

Patterns liés



Source : https://microservices.io/articles/applying.html

Décomposition de l'application

- Objectif : séparer en plusieurs conteneurs
  - Architecture stable
  - Modules cohésifs et faiblement couplés
  - Cycles de vie des composants des modules similaires
  - Modules testables
  - Équipes resserrées ≤ 10 et autonomes

Décomposition de l'application

- Problème : comment découper ?
- 2 possibilités
  - Par « business capability » : processus producteurs de valeur
    - Liés à l'activité économique de l'entreprise (business architecture modeling)
  - Par sous-domaines d'activité : typologie des processus « classique » (cf. CM urbanisation)
    - Opérationnels, de support, de pilotage
- Critères de choix : granularité & aspects humains…

Aspects humains

- Loi de Conway : l'organisation entre les équipes doit refléter l'architecture du produit
  - Des équipes plus petites
    → agilité, autonomie, efficacité
  - Chaque équipe est responsable d'un service
    - Choix technologiques, conception, déploiement, maintenance
  - Avantages pour le produit
    - Cycles de développement courts
    - Déploiements indépendants
    - Testabilité
    - Isolation des / tolérance aux bugs

Gestion des données

- Problème : comment gérer les accès aux données une fois les services séparés dans des conteneurs différents ?
- Plusieurs solutions :
  - 1 BD par service
    - Avantage : plus simple à réaliser (si pas besoin de synchro)
    - Inconvénients : performances, synchronisation
    - Patterns liés : Saga, CQRS

Gestion des données

- Problème : comment gérer les accès aux données une fois les services séparés dans des conteneurs différents ?
- Plusieurs solutions :
  - 1 BD partagée entre plusieurs services
    - Avantage : plus simple à réaliser (si accès concurrents)
    - Inconvénients : goulot d'étranglement
  - Remarque : des solutions intermédiaires existent aussi

Communication entre conteneurs

- Problème : conserver des mécanismes de communication « légers »
- Solution : « Dumb pipe, smart endpoint »
  - Opérateur | (pipe)
  - REST (HTTP)
    - Adapté à l' « éclatement » d'une application monolithique en micro-services
    - Correspond à l'utilisation de patterns GRASP (lesquels ?)
    - Scalable à l'aide de mécanismes du Web (duplication, cache/proxy, load-balancing)

    ➔ Penser « ressource » dès la conception
  - Bus de messages (JMS, AMQP…)

Communication entre conteneurs

- Problème : conserver des mécanismes de communication « légers »
- Solution : « Dumb pipe, smart endpoint »
  - Opérateur | (pipe)
  - REST (HTTP)
  - Bus de messages (JMS, AMQP…)
    - Adapté à l' « éclatement » d'une application orientée-services en micro-services
    - Le bus reste le plus simple possible (ne contient pas de métier)
    - À proscrire : mécanismes de routage, orchestration, chorégraphie de service, BPEL…

    ➔ Penser « asynchrone » dès la conception

Communication entre conteneurs

- Problème : permettre aux services d'être informés des changements d'états des autres services
- Solution : pattern « Event sourcing »
  - Persister dans une BD partagée les états des conteneurs
  - Déclencher un événement à chaque insertion
  - Permettre de s'abonner à ces événements
- Remarques :
  - Permet de re-jouer le déroulement de l'application
  - Attention à ne pas « re-centraliser l'intelligence » (ESB)
  - Potentiel *Single Point of Failure*

Composition de l'interface

- Problème : au final, réaliser une interface applicative qui tire partie de tous les services
- 2 solutions :
  - Composer les vues côté serveur
    - Permet de générer différents types d'applications (Web, mobile, desktop, etc.)
  - Composer les vues côté client
    - SPA, AJAX…
- Remarque :
  - Dans les 2 cas, l'application est un mashup ☺

Autres problématiques

- Déploiement
  - Multiple service instances per host
  - Service instance per host
  - Service instance per VM
  - Service instance per Container
  - Serverless deployment
  - Service deployment platform
- Sécurisation
  - Access Token

Pièges

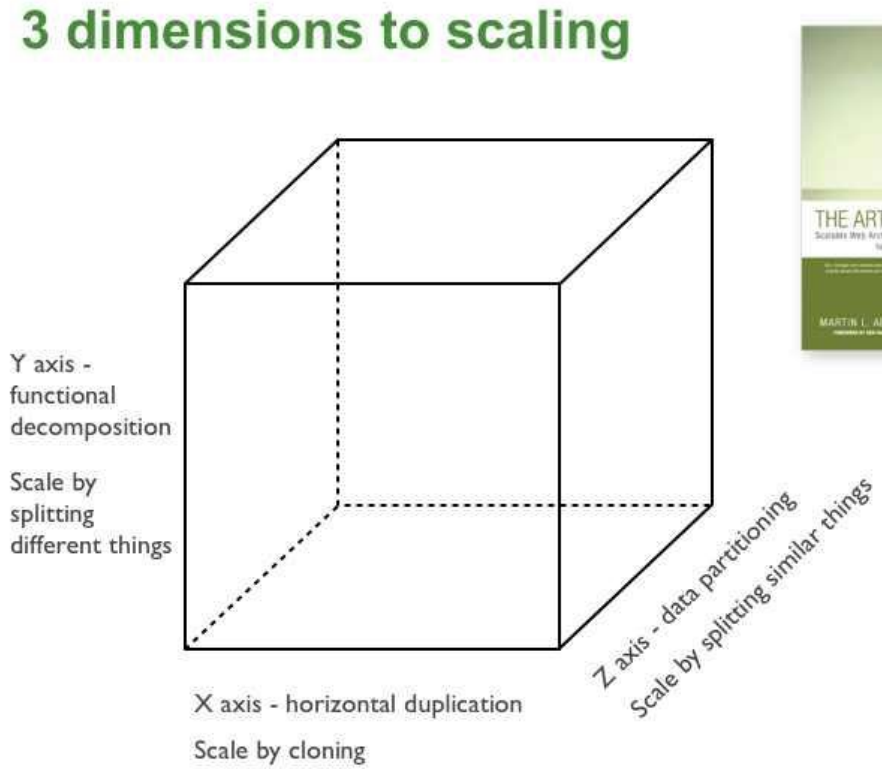
- Granularité
  - J'ai regroupé plusieurs services en un seul « macroservice »
  - Chaque méthode de mon application est devenue un microservice
- Architecture globale
  - Il est très difficile de comprendre ce qui se passe entre mes services
  - La recherche d'information dans les logs est trop longue
  - Impossible de relancer ou ajouter rapidement des instances de mes services

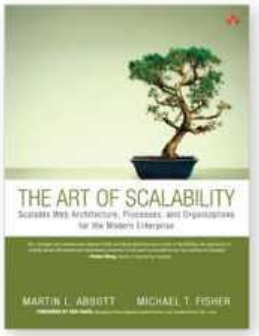
Source :

http://blog.xebia.fr/2015/03/16/microservices-des-pieges/

Scalabilité

- The Scale Cube

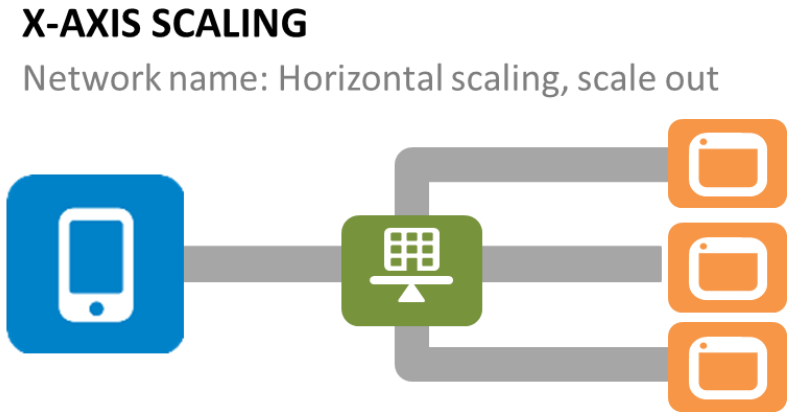




Source : http://microservices.io/articles/scalecube.html

Scalabilité

- Solution : The Scale Cube
  - Axe X : scalabilité horizontale
    - Dupliquer (cloner) les composants autant que nécessaire
    - Utiliser des techniques de load balancing pour les adresser



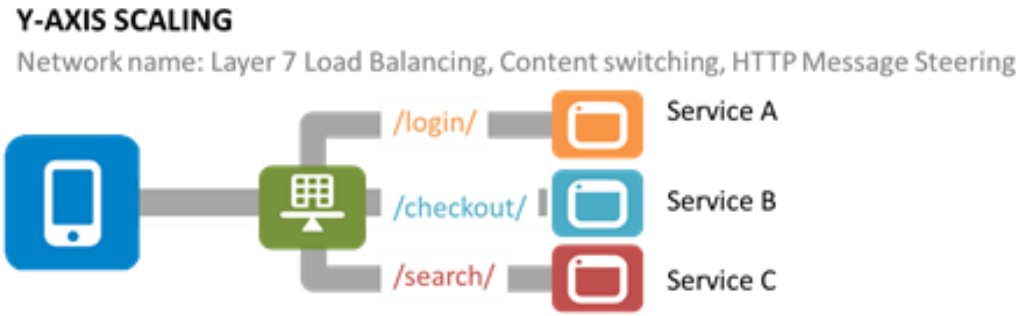
Source : https://devcentral.f5.com/articles/the-art-of-scale-microservices-the-scale-cube-and-load-balancing

Scalabilité

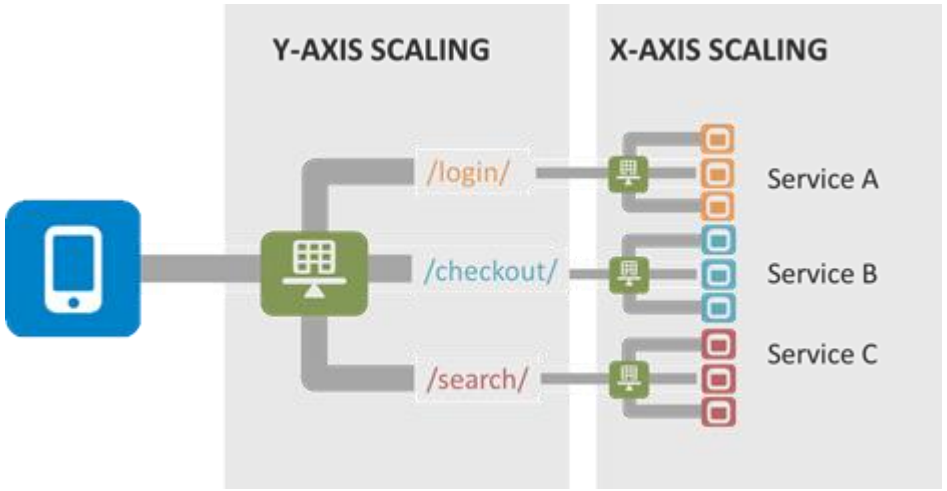
- Solution : The Scale Cube
  - Axe Y : scalabilité verticale (pattern couches)
    - Décomposer l'application en modules fonctionnels
      - En fonction des use cases à traiter
      - En fonction de leurs positions dans les patterns utilisés (couche, MVC…)
      - En fonction de leurs cycles de vie
      - En fonction de leurs caratéristiques techniques (langages…)
      - En fonction des équipes de développement

Scalabilité

- Solution : The Scale Cube
  - Axe Y :



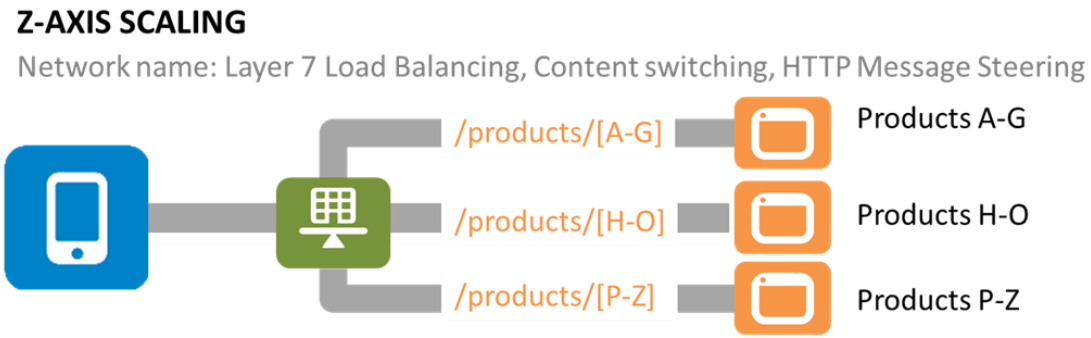
  - Axes X + Y :



Source : https://devcentral.f5.com/articles/the-art-of-scale-microservices-the-scale-cube-and-load-balancing

Scalabilité

- Solution : The Scale Cube
  - Axe Z : partitionnement des données
    - Faire en sorte que les données accédées par chaque microservice soient aussi indépendantes que possible
      - Dans les décompositions fonctionnelles
      - Dans la répartition horizontale



Source : https://devcentral.f5.com/articles/the-art-of-scale-microservices-the-scale-cube-and-load-balancing

Outils et plateformes

- Langage dédié
  - Jolie
- Plateformes d'exécution
  - Docker
  - Microsoft Service Fabric
  - MicroService4Net
  - NetKernel
- Plateformes de déploiement
  - VM dédiées
    - CoreOS, RancherOS , Ubuntu Snappy, Boot2Docker, docker-machine…

Outils et plateformes

- Outils Docker (rappels TIW7+)
  - « Mono-conteneur » : Dockerfile
  - « Mono-hôte » : Compose
  - Clusters : Machine
  - Monitoring / gestion de conteneurs : Portainer
  - Gestion de clusters : Swarm, Kubernetes
  - Repository : https://hub.docker.com/
  - Hébergement : AWS, MS Azure, Google Compute Engine, Heroku, Digital Ocean…

Conclusion

- Une forme d'architecture applicative
  - Dérivée des SOA
  - Avec des mécanismes de communication simples
  - Qui reprend des patterns connus

Conclusion

- Un outil prédominant
  - Qui facilite le déploiement
  - Dédié à la scalabilité et à la performance
  - Complet
  - Destiné à des hôtes / stacks homogènes

Avantages

- Agilité
  - Conception à l'échelle de la fonctionnalité
  - Isolation des fonctionnalités
- Légèreté
  - Permet de s'abstraire de la couche OS (VM)
- Scalabilité
  - Verticale : permet de ne répliquer que les services chargés
  - Horizontale : déport des services les plus chargés vers des nœuds différents

Inconvénients

- Overhead
  - Nécessite une communication orientée-message entre les services (*vs*. appel de méthodes)
  - Nécessite une « surveillance » du fonctionnement des services (monitoring, tolérance aux pannes, pilotage)
  - Accès aux ressources partagées
- Humain
  - Nécessite que les équipes soient effectivement structurées selon l'architecture du produit
- Vision / mise au point globale de l'application
  - Pas triviale, peut nécessiter plusieurs itérations

Problématiques

- Complexité des échanges de données
- Choix du mode de communication des services
- Sécurité des communications internes
- Monitoring / débogage de l'application
- Outils de développement, de packaging, de déploiement

➔ Sont-ce de nouvelles problématiques ?

Références

- http://microservices.io/
- http://martinfowler.com/articles/microservices.html
- https://martinfowler.com/bliki/BoundedContext.html
- http://alistair.cockburn.us/Patterns
- https://aws.amazon.com/fr/microservices/
- https://nirmata.com/2015/02/02/microservices-five-architectural-constraints/
- https://en.wikipedia.org/wiki/Scalability
- http://blog.xebia.fr/2015/03/09/microservices-des-architectures/
- http://blog.xebia.fr/2015/03/16/microservices-des-pieges/
- http://docs.docker.com/
- https://hub.docker.com/