Master 2 Traitement
de l'Information

Université Claude Bernard Lyon 1

Microservices

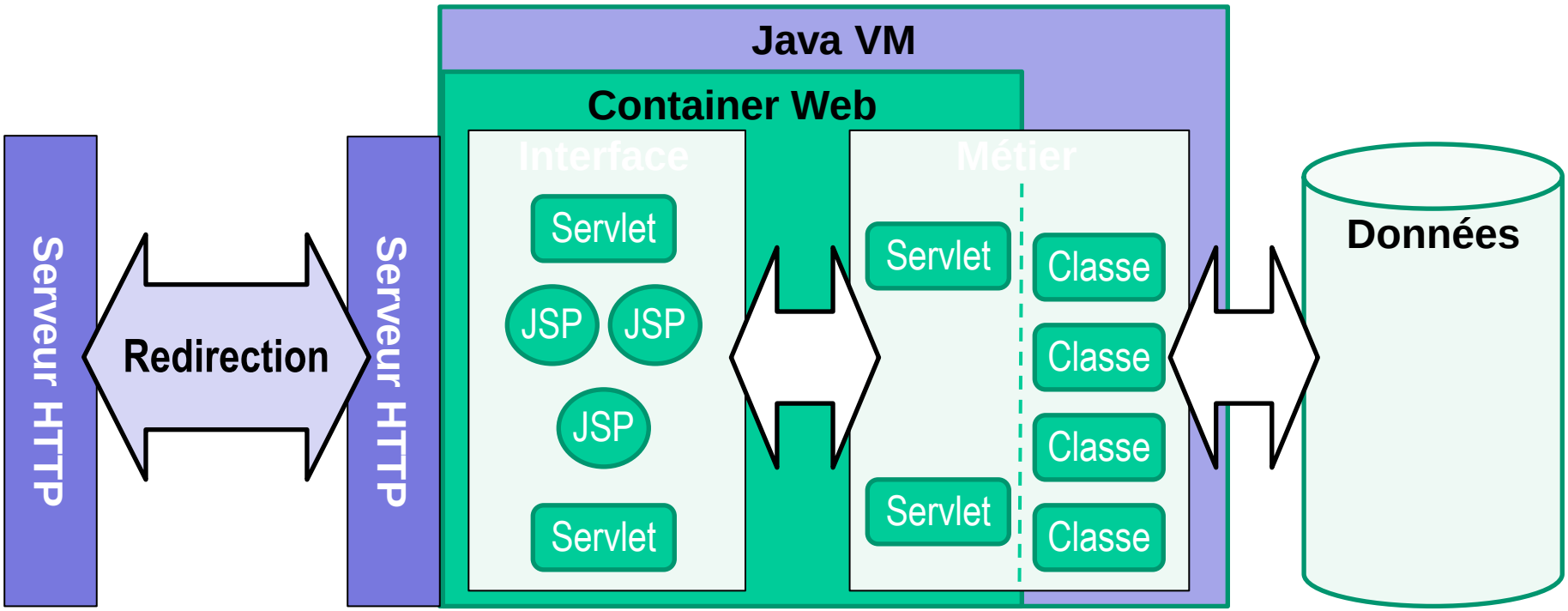
Conception d'Applications Hétérogènes Distribuées

Lionel Médini

Septembre-novembre 2015

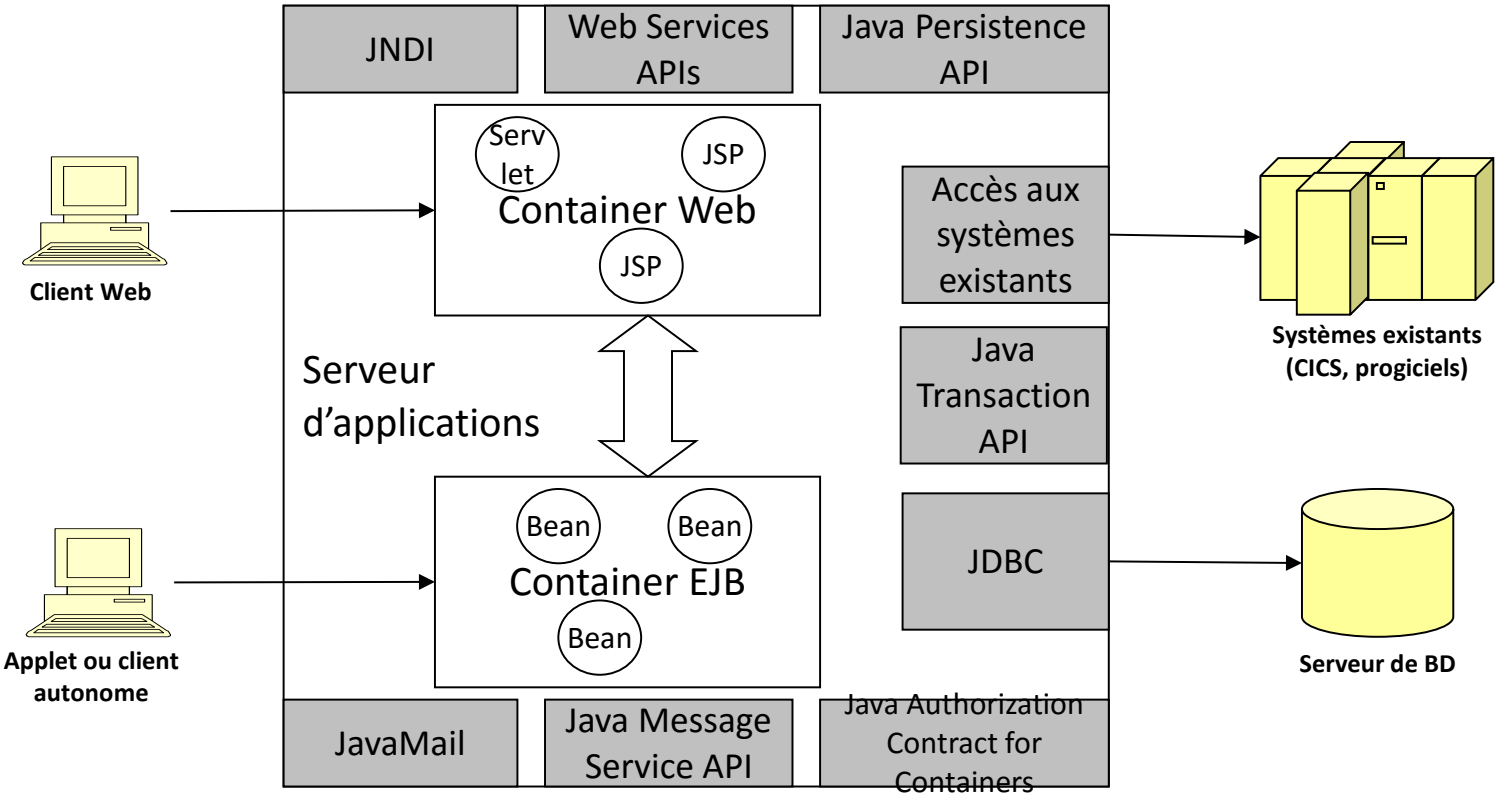
Position du problème

- 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

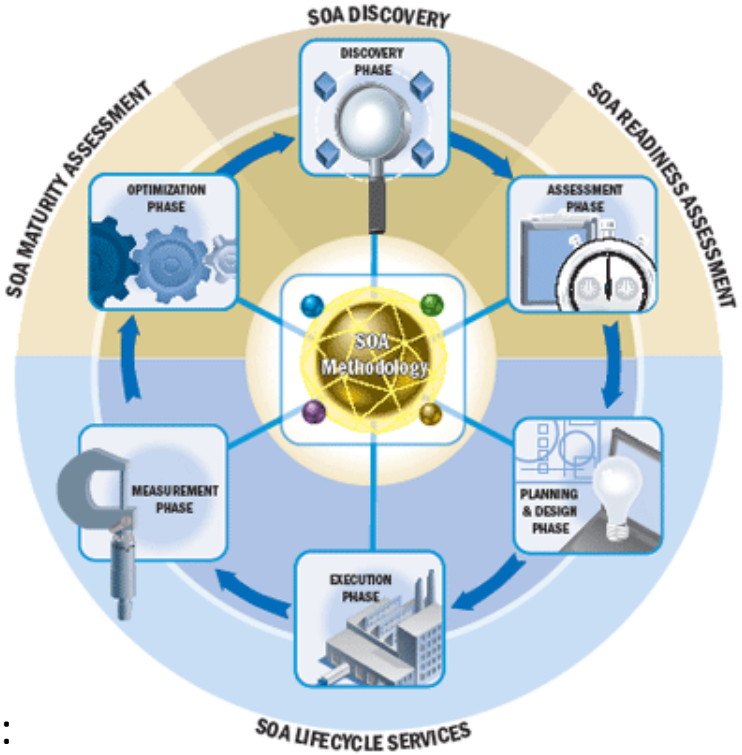
- L'architecture « monolithique »
  - Inconvénients
    - Technologies homogènes
      - Types d'objets / composants pas toujours adaptés aux besoins métier
      - Intégration de composants dans d'autres technos difficile
      - Choix technologiques à long terme
    - Framework
      - Peut devenir 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

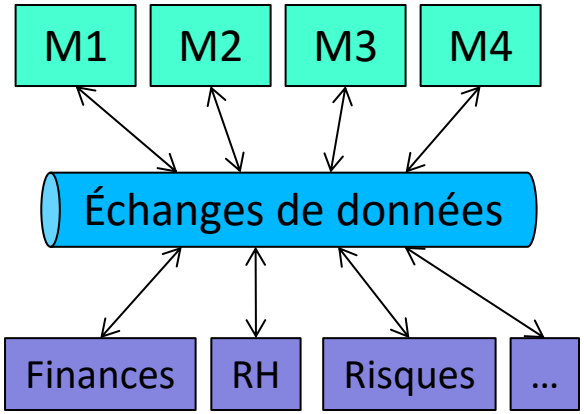
Position du problème

- 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)
      - Liberté des choix technologiques
      - Possibilité de redéployer composant par composant
      - Indépendance des équipes de dév
    - Interfaces de communication réseau
      - Pérennité des standards
      - Distribution possible
    - « Scalability by design »

Position du problème

- Les architectures orientées-services
  - Inconvénients
    - Complexité
      - Des mécanismes de communications
      - Des architectures applicatives
    - Plus adaptées
      - Aux environnements distribués
      - Aux échanges de services entre plusieurs applications

Microservices : définition

- Concepts
  - « Domain-Driven Design » -> « Bounded context »
    - 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

    ➔ Un microservice est un « running bounded context »
  - « Dumb pipe, smart endpoint »
    - Le bus reste le plus simple possible (ne contient plus de métier)
    - La logique de communication est dans les services et non entre eux

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. »

Microservices : définition

- 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

Microservices : définition

- La gestion centralisée de l'application est réduite au minimum
  - Communication par mécanismes « légers »
    - Opérateur | (pipe)
    - REST
  - Les services peuvent utiliser des mécanismes de stockage différents
  - L'« intelligence » de l'application est répartie dans les services et non dans un médium de communication centralisé (ESB)

Microservices : définition

- 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
    - Meilleure isolation des / tolérance aux bugs

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

Microservices

- 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/

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…
  - Cloud
    - NetFlix, Cloud Foundry, Amazon, MS Azure…

Docker

- Rappels
  - Un environnement dédié à l'exécution de microservices
    - Images (PAI)
    - Conteneurs (PAI)
    - Image repository (PAI)

Docker

- Rappels
  - Un outil de communication et d'isolation de conteneurs
    - Système de fichier en couches
    - Volumes (PAI)
    - Liens
    - Réseaux
      - Bridge
      - None
      - Host
      - User-defined

Docker

- Rappels
  - Des outils de configuration d'applications
    - « Mono-conteneur »
      - Dockerfile
    - « Mono-hôte »
      - Compose
    - Clusters
      - Machine

Docker

- Rappels
  - Des outils de pilotage et de déploiement
    - Swarm
      - Découverte
      - Stratégies
      - Filtres

Conclusion

- Une forme d'architecture
  - À mi-chemin entre architecture monolithique et SOA
  - Avec des mécanismes de communication simples
  - Qui reprend des patterns connus
- Un outil prédominant
  - Qui facilite le déploiement
  - Dédié à la scalabilité et à la performance
  - Complet
  - Destiné à des hôtes / stacks homogènes

Références

- http://microservices.io/
- http://martinfowler.com/articles/microservices.html
- http://alistair.cockburn.us/Patterns
- https://en.wikipedia.org/wiki/Scalability
- http://blog.xebia.fr/2015/03/16/microservices-des-pieges/
- http://docs.docker.com/
- https://hub.docker.com/