

Cloud Computing

Concepts d'applications *Cloud native*¹

Danilo Carastan dos Santos

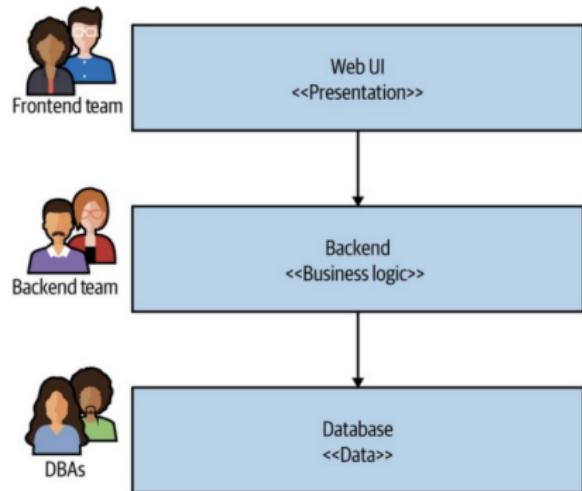
`danilo.carastan-dos-santos@univ-grenoble-alpes.fr`

2024

¹Adapté du support développé par Thomas ROPARS et Renaud LACHAIZE

Motivation : du data-centre privé (*on premises*) au Cloud

- Migration d'applications existantes
- “*Lift and Shift*” : Réhébergement entier de l'application vers un Cloud public ou privé
- Migration via IaaS ou PaaS



Tendance : Concevoir de nouvelles applications en prenant en compte dès le départ les spécificités (défis et opportunités) des plateformes et technologies Cloud modernes

Cloud Native

Une définition¹

"Les technologies Cloud Native permettent aux entreprises de construire et d'exploiter des applications élastiques dans des environnements modernes et dynamiques comme des clouds publics, privés ou bien hybrides. Les conteneurs, le maillage de services, les microservices, les infrastructures immuables et les API déclaratives illustrent cette approche. **Ces techniques permettent la mise en œuvre de systèmes faiblement couplés, à la fois résistants, pilotables et observables.** Combinés à un **robuste système d'automatisation**, ils permettent aux ingénieurs de **procéder à des modifications impactantes, fréquemment et de façon prévisible** avec un minimum de travail.

La Cloud Native Computing Foundation cherche à favoriser l'adoption de ce paradigme en encourageant et en soutenant un écosystème de projets open source et indépendants. Nous démocratisons l'état de l'art des bonnes pratiques afin de rendre l'innovation accessible à tous."

¹<https://github.com/cncf/toc/blob/main/DEFINITION.md#fran%C3%A7ais>

Cloud Native

Une autre définition¹

"A cloud native application is engineered to run on a platform and is designed for resiliency, agility, operability, and observability."

- **Résilience** : Profiter de la nature dynamique et distribuée du Cloud pour s'adapter et rester opérationnelle face à des pannes
- **Agilité** : permettre des déploiements et des interactions rapides
- **Opérabilité** : ajouter le contrôle des cycles de vie des applications depuis l'intérieur de l'application au lieu de s'appuyer sur des processus et des moniteurs externes
- **Observabilité** : fournir des informations pour répondre aux questions sur l'état de l'application

¹J. Garrisson and K. Nova. Cloud native infrastructure. O'Reilly, 2017.

Transition vers le *Cloud Native* : un exemple

- + Niveau faible, moyen et élevé de sous-traitance de responsabilités
- + Réduction de coûts liées à la gestion d'infrastructure physique/logiciel
- + Niveau faible, moyen et élevé de dépendance d'un fournisseur de Cloud

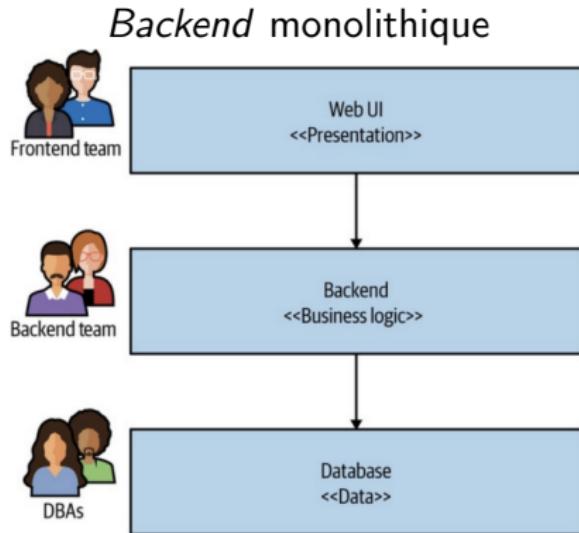
	LEVEL OF MODERNIZATION			
	ON-PREMISES	VIRTUAL MACHINES (EC2 / VMC)	CONTAINERIZATION (ECS/EKS)	SERVERLESS (Lambda/Fargate)
Application code	✓	✓	✓	✓
Data source integrations	✓	✓	✓	✓
Capacity planning and scaling	✓	✓	✓	✓
Security and network configuration	✓	✓	✓	✓
OS Software install and maintenance	✓	✓	✗	✓
Infrastructure provisioning	✓	✓	✓	✓
Physical server, storage, networking, and facilities	✓	✓	✓	✓

MANAGED BY |  CUSTOMER  AWS

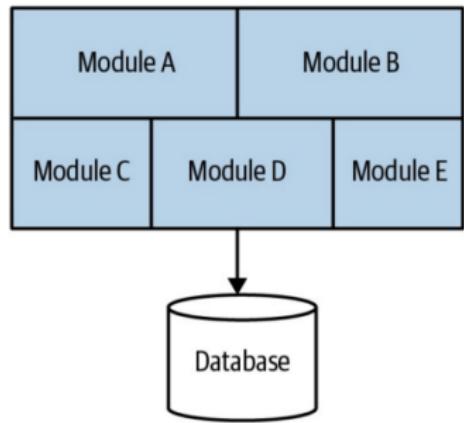
Quelques méthodologies liées

- Conteneurs, CI/CD, **Microservices**, *Serverless* (FaaS)

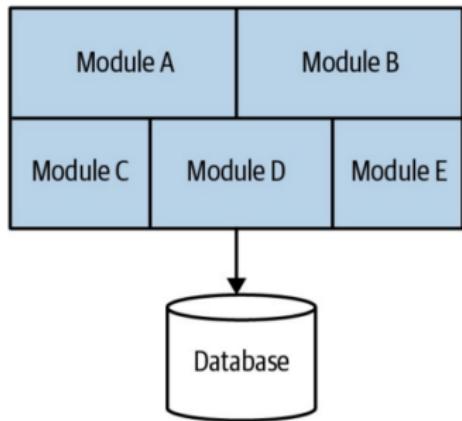
Architecture monolithique à 3 niveaux



Un niveau monolithique

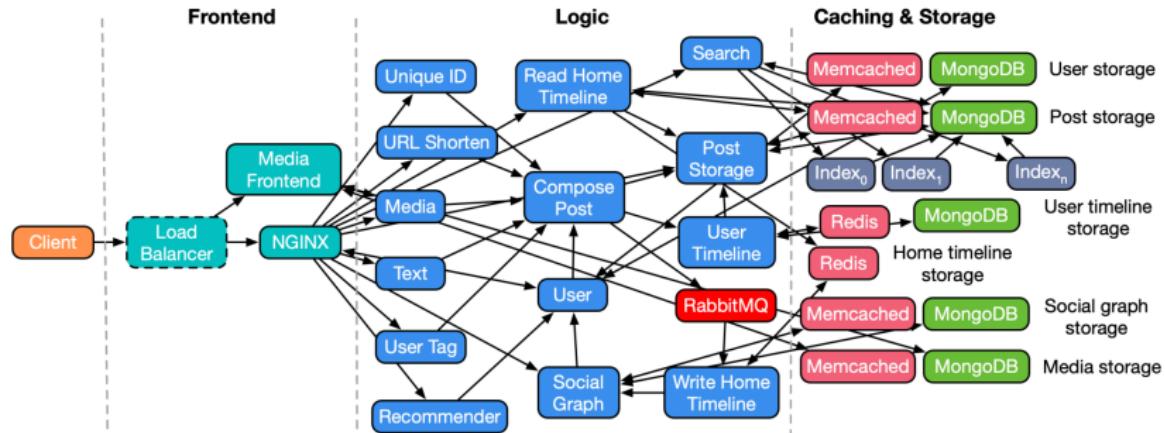


Monolithe modulaire



- Modulaire mais typiquement déployé comme un processus unique (monolithique)
- Problèmes liés :
 - ▶ Qui décide sur quoi ? Qui détient quoi ?
 - ▶ Pas d'indépendance pour mettre à jour un module

Microservices¹



- Déployer les modules de façon indépendante
- Posséder l'état de chaque module
- Aligner sur l'organisation et structure de l'activité de l'entreprise

¹Source de la figure : <https://github.com/delimitrou/DeathStarBench/tree/master/socialNetwork>

Objectives

Déployer les modules de façon indépendante :

- Modifier un service sans modifier les autres services
- Réduire le temps de déploiement de nouvelles fonctionnalités

Posséder l'état de chaque module :

- Éviter utiliser de bases de données partagées

Aligner sur l'organisation et structure de l'activité de l'entreprise

Loi de Conway (*Conway's law*)¹ :

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

Conséquences :

¹<https://martinfowler.com/bliki/ConwaysLaw.html>

Aligner sur l'organisation et structure de l'activité de l'entreprise

Loi de Conway (*Conway's law*)¹ :

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

Conséquences :

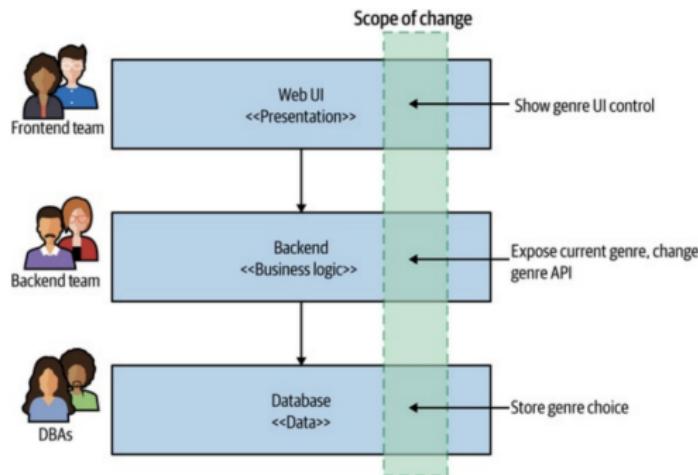
- Un petit groupe de développeurs qui travaillent dans le même bureau → vont produire un monolithe
- Organisation basée sur une expertise précise → vont produire une architecture à trois niveaux

Une architecture qui n'est pas alignée sur l'organisation des collaborateurs peut être contre productive

- Surcharge de communication
- Tension

¹<https://martinfowler.com/bliki/ConwaysLaw.html>

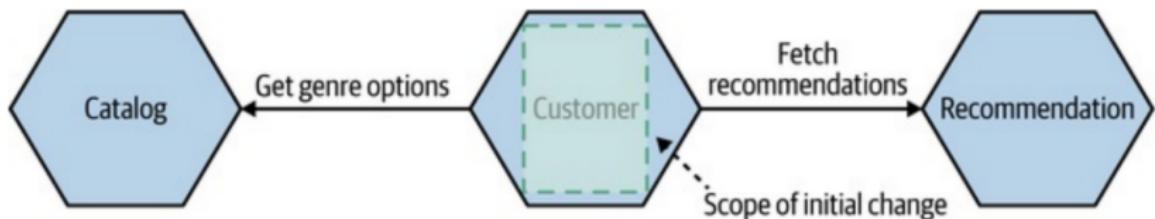
Exemple



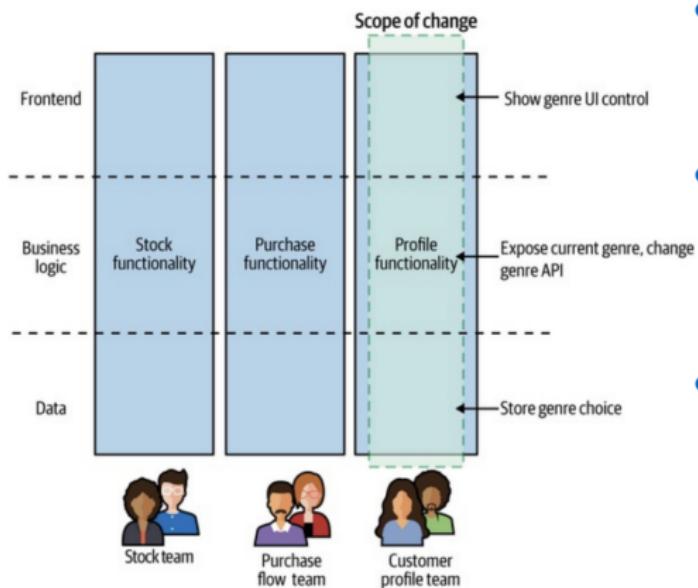
- Objectif : développer un service de *streaming* de musique
- Nous voulons ajouter une nouvelle fonctionnalité : chaque client pourra définir son genre de musique préféré dans son profil.
 - ▶ Plusieurs équipes doivent être impliquées
 - ▶ Modifications doivent être déployées dans le bon ordre

Structuration par microservices

- Délimiter les frontières de microservices en fonction du domaine d'activité
- Inspiré par l'approche *Domain-Driven Design*



Structuration par microservices



- Une petite équipe est chargée de la fonctionnalité *Profile*
- Cette équipe comprend des développeurs frontend, backend et de base de données
- Nouvelle organisation :
 - ▶ Petite équipe polyvalente (de 5 à 10 personnes)
 - ▶ Objectif : Faciliter les interactions entre les équipes et leurs membres

Technologies liées aux microservices

Communication

Communication synchrone et bloquante :

- *Remote Procedure Calls* : gRPC¹
- REST² + HTTP

Communication non bloquante par
queues/événements/producteur-consommateur :

- Kafka³, RabbitMQ⁴, ZeroMQ⁵

¹<https://grpc.io/>

²<https://www.redhat.com/fr/topics/api/what-is-a-rest-api>

³<https://kafka.apache.org/>

⁴<https://www.rabbitmq.com/>

⁵<https://zeromq.org/>

Technologies liées aux microservices

Déploiement

Conteneurs + orchestration (gestion)

- Docker¹ (conteneurs)
- Kubernetes², Docker Swarm³ (orchestration)

Services fournis par les fournisseurs de Cloud :

- ECS/EKS (AWS)
- Cloud Run, Kubernetes Engine (Google)

¹<https://www.docker.com/>

²<https://kubernetes.io/>

³<https://docs.docker.com/engine/swarm/>

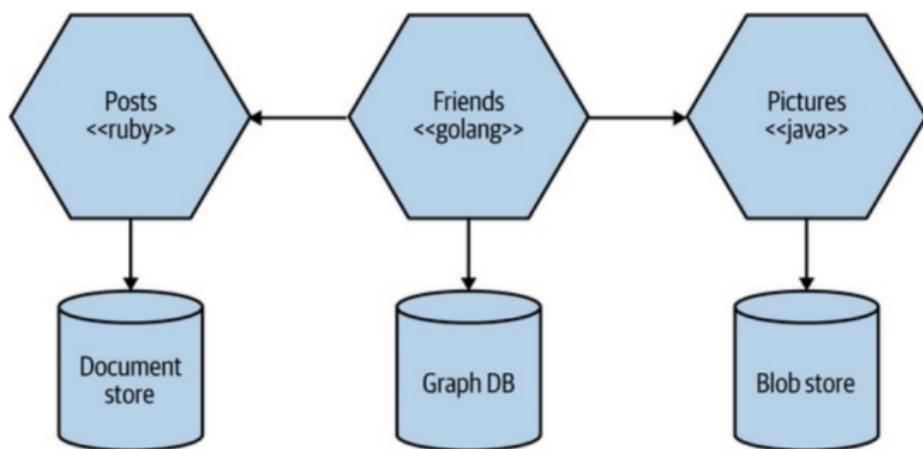
Microservices : Avantages

- Hétérogénéité technologique
- Robustesse
- Scalabilité
- Facilité de déploiement

Microservices : Avantages

Hétérogénéité technologique

- Possible implémentation en plusieurs langages/technologies/base de données
 - ▶ Les détails internes sont cachés. Il suffit de bien communiquer avec les autres services
- Cela s'aligne sur :
 - ▶ Une équipe par microservice
 - ▶ Une gestion décentralisée



Microservices : Avantages

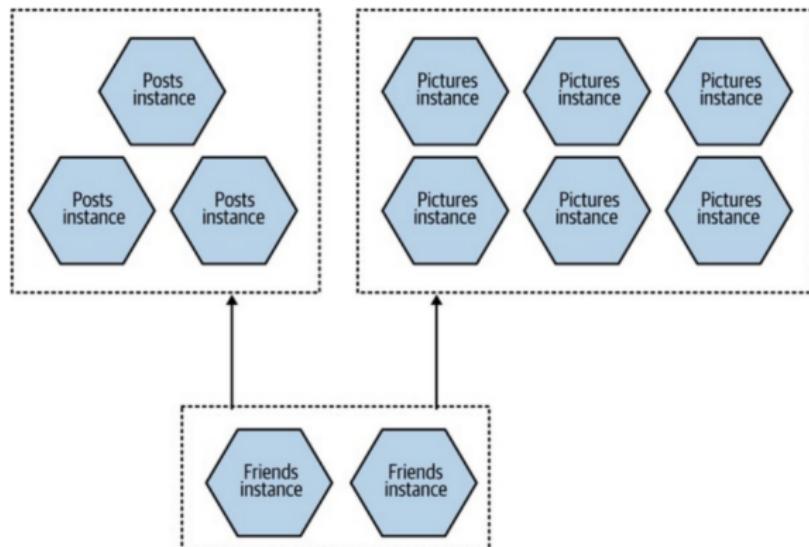
Robustesse

- Un microservice tombe en panne != l'application tombe en panne
 - ▶ Certaines parties de l'application peuvent continuer à fonctionner même lorsque certains composants tombent en panne
 - ▶ Nécessite une bonne isolation entre les composants
- Attention : de nouveaux types de pannes peuvent apparaître
 - ▶ Pannes réseau

Microservices : Avantages

Scalabilité

- Plusieurs instances d'un microservice peuvent être créées
- Cela s'aligne sur :
 - ▶ Agilité : S'adapter en fonction des fluctuations de la charge de travail
 - ▶ Résilience : Distribuer les fonctionnalités géographiquement



Microservices : Avantages

Facilité de déploiement

- Chaque microservice peut être déployé indépendamment
- Il est important pour :
 - ▶ Décentraliser la gestion
 - ▶ Mettre en place de nouvelles fonctionnalités le plus rapidement possible (agilité)
 - ▶ Pouvoir appliquer les approches DevOps :
 - Intégration continue
 - Livraison continue

Microservices : Inconvénients

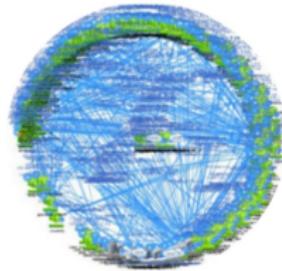
Développement et maintenance complexes :

- Hétérogénéité de technologies
- Relations complexes entre les services
 - ▶ Débogage difficile
- Difficultés liés aux bases de données distribuées
 - ▶ Gestion de l'état de l'application
- Déploiement complexe avec un grand nombre de services
 - ▶ Monitoring difficile

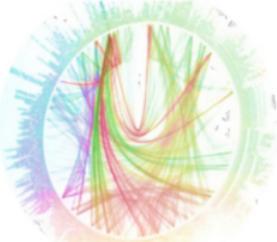
Nécessite des développeurs expérimentés :

- Une application monolithique peut être une solution plus appropriée dans certaines situations

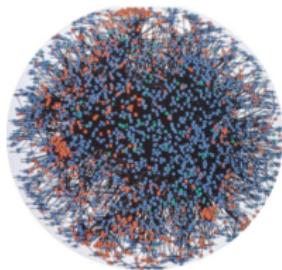
Microservices : Inconvénients¹



Netflix



Twitter



Amazon



Social Network

- Explosion de services
- Comment comprendre ?
Comment déboguer ?
- Tous les services sont-ils essentiels ?

¹Source de la figure : Gan, Yu, et al. "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems.", ASPLOS 2019.

Contenu supplémentaire

- Support de Thomas ROPARS et Renaud LACHAIZE
 - ▶ <https://m2-mosig-cloud.gitlab.io/lectures/>
 - ▶ <https://roparst.gricad-pages.univ-grenoble-alpes.fr/cloud-tutorials/m2gi-devops/>
- Qu'est-ce que le concept de "cloud natif" ? : <https://cloud.google.com/learn/what-is-cloud-native?hl=fr>
- What Are Microservices : <https://youtu.be/CdBtNQZH8a4>
- Microservices : <https://martinfowler.com/articles/microservices.html>
- Newman, Sam. Building microservices. "O'Reilly Media, Inc.", 2021.
- Burns, Brendan. Designing distributed systems: patterns and paradigms for scalable, reliable services. "O'Reilly Media, Inc.", 2018.