# Architecture Microservices et intégration continue

Salah Gontara
2022-23

# Sommaire

- Architecture monolithique
- Que sont les microservices?
- Intégration

# Monolithes

- Applications classiques que vous connaissez tous
  - Tomcat:

# Monolithes

# Application monolithique

Avantages:

- Simplicité de mise en place
- Une seule codebase
- Efficience à petite échelle
- Latence applicative

# Application monolithique

Inconvénients:

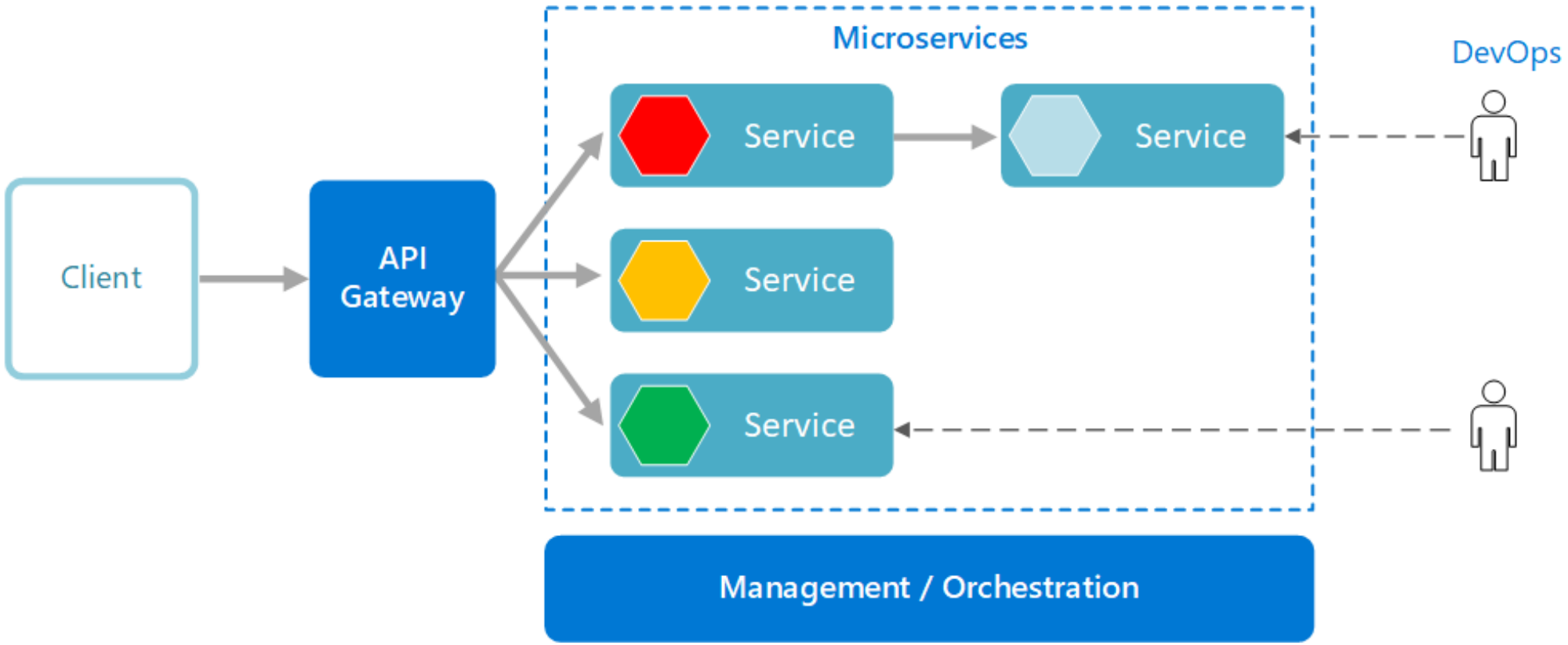
- Mauvaise application de la modularité
- Temps de build long
- Déploiement de toute l'application (downtime,failure)
- Mauvaise mise à l'échelle (vertical scaling)

# Evolution de l’industrie

- Domain-Driven Design
- Intégration continue
- Virtualisation à la demande
- Automatisation des infrastructures
- Equipes de développement autonomes
- Mise à l'échelle des systèmes (scaling)

# Microservices

- Petit et focalisé sur une et une seule responsabilité



# Single Responsibility Principle

- Fait partie des 5 principes SOLID :
- S : Single responsibility principle
- 0 : Open/closed principle
- L : Liskov substitution principle
- I : Interface segregation principle
- D : Dependency inversion principle

# Caractéristiques (1/2)

- Application autonome
- Organisé autour des capacités métier
- Produit et non plus projet
- Endpoints évolués et canaux de communication pauvres
- Gouvernance décentralisée
- Management de donnée décentralisée
- Automatisation de l'infrastructure
- Design For Failure

# Caractéristiques (2/2)

- Les microservices doivent pouvoir être :
  - remplacés de manière indépendante
  - mis à jour de manière indépendante

# La Communication asynchrone

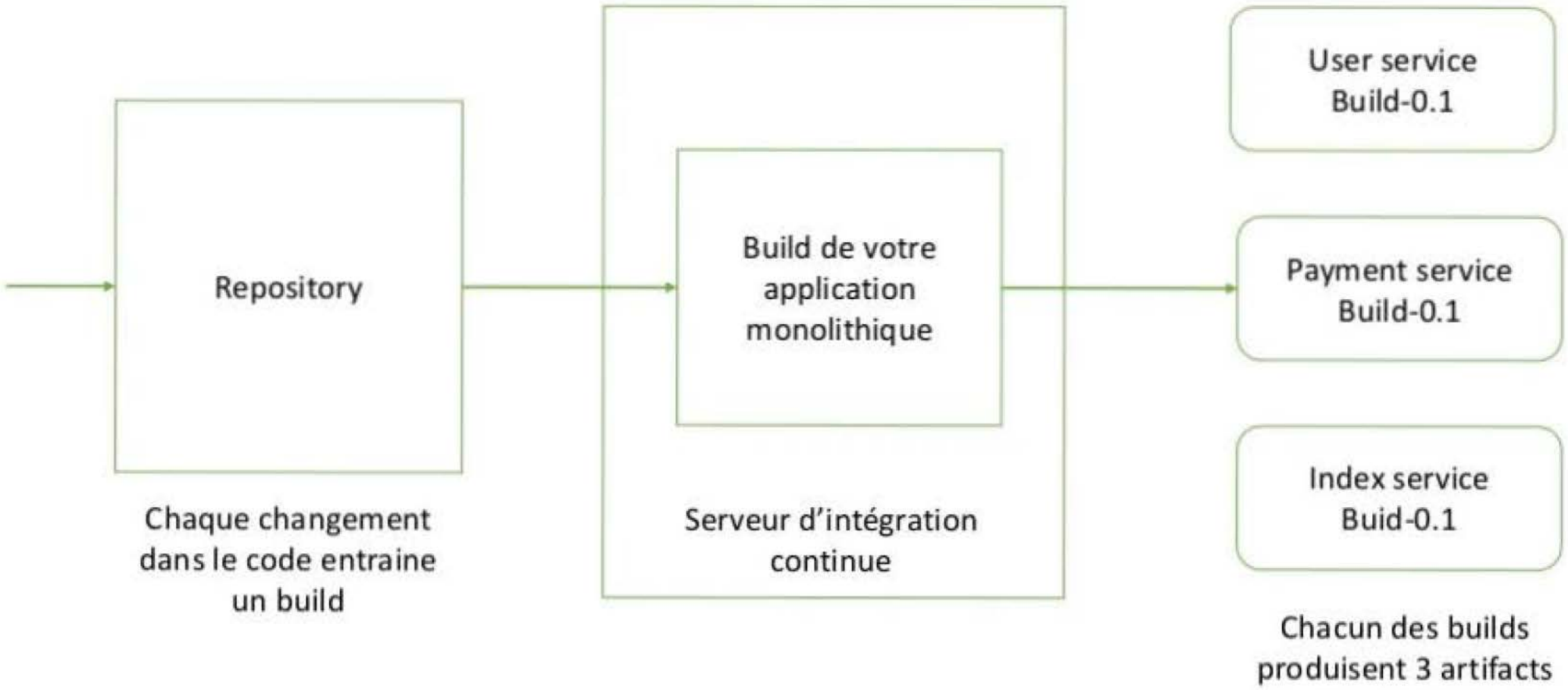
Event-Based communication :

- Publication d'événement
- Abonnement à la réception d'événement

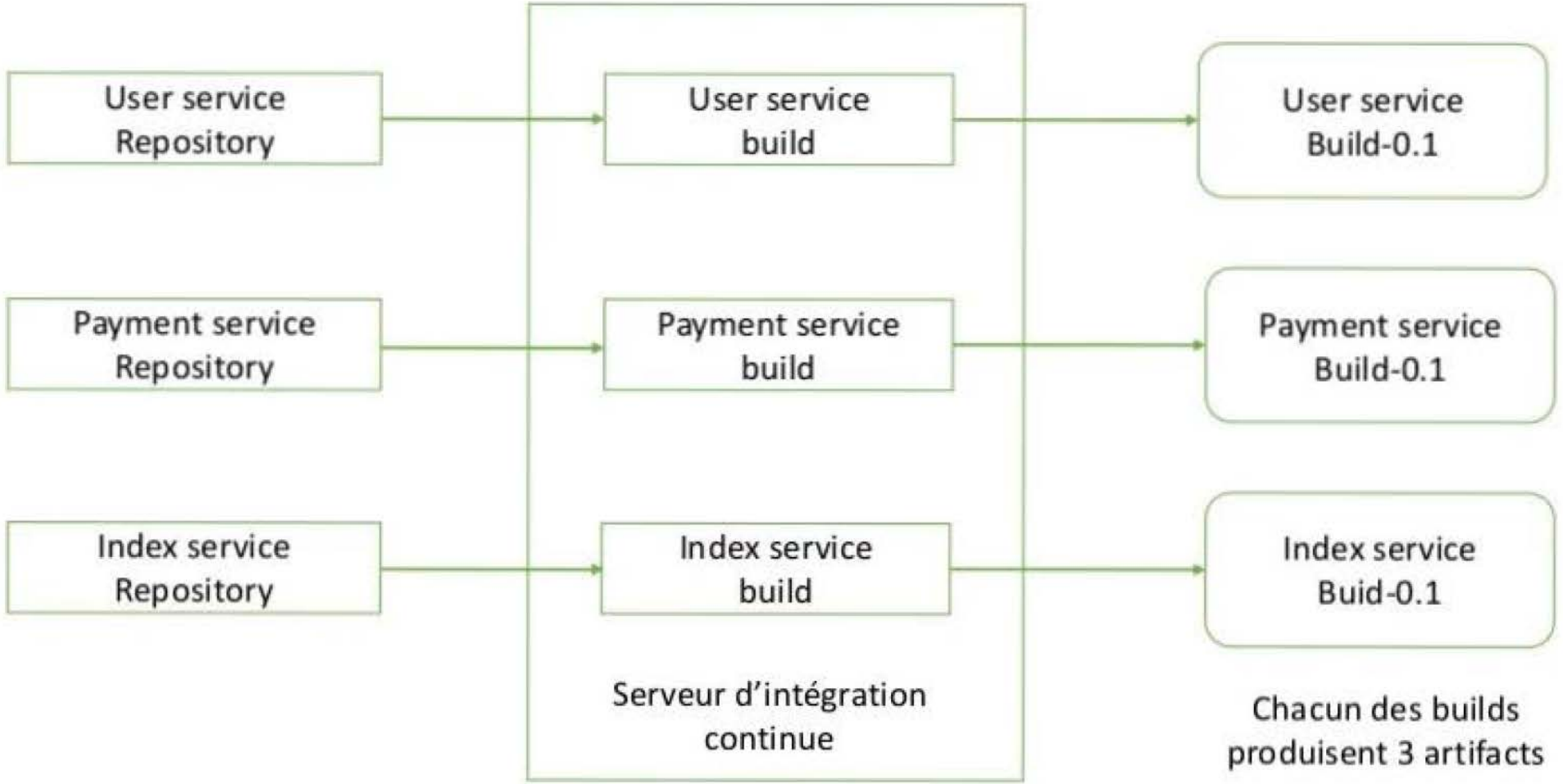
Maturité des APIs :

- Niveau 0 : Une seule URI (SOAP,XML,RPC)
- Niveau 1: Plusieurs URI, un seul verbe
- Niveau 2 : Plusieurs URI, plusieurs verbes (CRUD)
- Niveau 3 : Hypermedia

# Intégration continue monolithique



# Intégration continue en microservices



# Intégration avec Jenkins (1/2)

- L'idée principale derrière cela est d'utiliser des conditions intégrées pour différents modules d'un monorepo quand:

  *{ changeset "*path/to/module-example/.*" }*

- Par exemple, imaginez que votre monorepo se compose de trois modules : frontend, backend/web et backend/api

```
. (monorepo-example)
├── backend
│   ├── api
│   └── web
├── frontend
└── Jenkinsfile
```

# Intégration avec Jenkins (2/2)

Un fichier Jenkinsfile valide serait similaire à :

```
pipeline {
    agent none
    stages {
        stage('Build Frontend') {
            agent { docker { image 'my-node-agent' } }
            when {
                changeset "**/frontend/*.*"
                beforeAgent true
            }
            steps {
                dir('frontend') {
                  sh 'npm install'
                  sh '...'
                }
            }
        }
        stage('Build Web') {
            agent { docker { image 'my-maven-agent' } }
            when {
                changeset "**/backend/web/*.*"
                beforeAgent true
            }
            steps {
               dir ('backend/web') {
                sh 'mvn -B -DskipTests clean package'
                sh '...'
               }
            }
        }
        stage('Build REST API') {
            agent { docker { image 'my-golang-agent' } }
            when {
                changeset "**/backend/api/*.*"
                beforeAgent true
            }
            steps {
               dir ('backend/api') {
                 sh 'go build'
                 sh '...'
               }
            }
        }
    }
}
```