



ETNA

**FDI-LEEX**

Communauté métier Cloud & Microservices

# PLAN DE LA PRÉSENTATION

- Notre communauté métier
- Objectif du protocole d'expérimentation
- Méthodologie & critères d'évaluation
- Nos tests réalisés (Python & Java)
- Résultats (internes + benchmarks externes)
- Limites, apprentissages & perspectives

# GROUPE



- Elyesse BEN HADJ
- Aurélien GUENEBEM KOSSI
- Beviryon ISSANGA NGOULOU
- Luca BRUNET
- Mattéo TOLLA
- Sandro BAKURADZE

# Objectifs de l'expérimentation

1 ————— 2 ————— 3 ————— 4 ————— 5

## STEP

Etablir un protocole d'expérimentation

Critère de comparaison pour chaque techno, repartir les taches pour chaque binome

## STEP

Développer une architecture avec 3 micro services

user-service  
message-service  
file-service

## STEP

Dockerisé l'architecture  
Dockerfile pour chaque micro-service, et utilisation de docker-compose

## STEP

Intégré une techno d'API Gateway  
Spring-Cloud, Traefik, Kong, etc

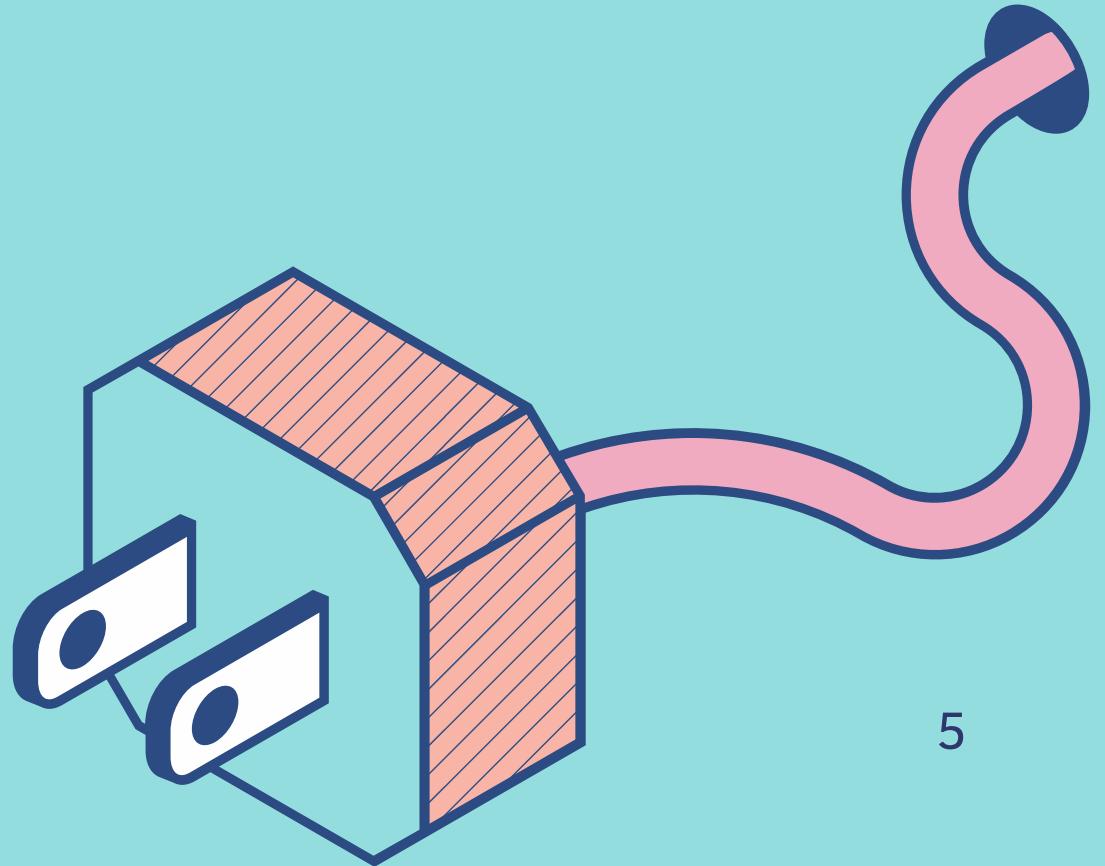
## STEP

Comparer les solutions de chaque groupe bânone

Facilité d'intégration, Performance, Logs, Simplicité de configuration

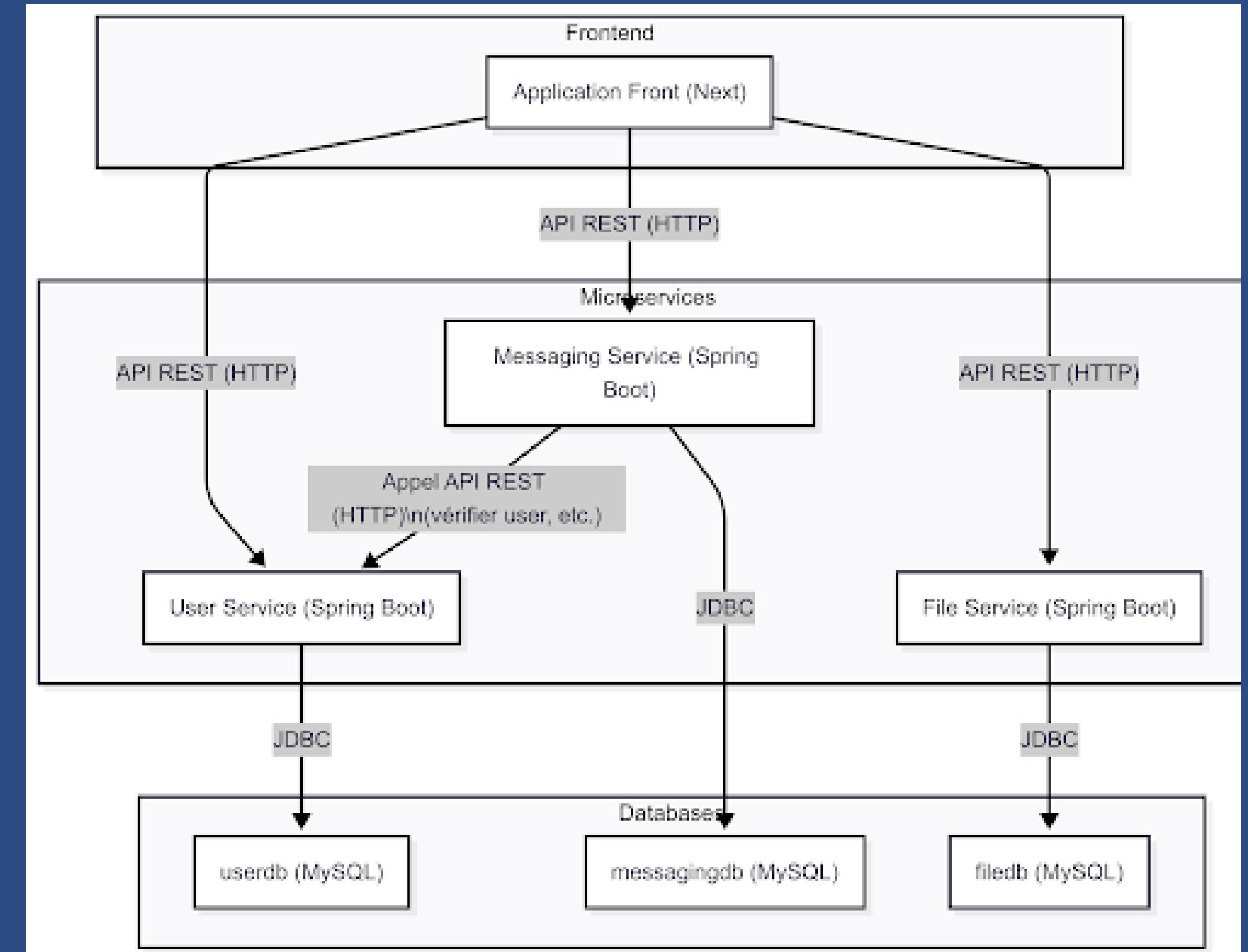
# Communauté métier Cloud & Microservices

- Objectifs : Comprendre, expérimenter, comparer
- Méthodologies : déploiement local, conteneurisation, orchestration
- Outils utilisés : Docker, Compose, Spring Boot, FastAPI, Traefik, etc.



# Architecture existante

- 3 SERVICES : USER-SERVICE, FILE-SERVICE, MESSAGES-SERVICE
- 1 BASE DE DONNÉES POUR CHAQUE SERVICE
- DÉPLOIEMENT LOCAL (DOCKER COMPOSE)





# Méthodologie

BINÔMES, ENVIRONNEMENTS LOCAUX,  
TECHNOLOGIES COMPARÉES

- Travail en binômes (6 étudiants → 3 binômes)
- Implémentation complete et partielle :
  - Binôme 1 : Python FastAPI + Traefik + Docker-Swarm
  - Binôme 2 : Java Spring + Spring-Cloud + Docker-Compose
  - Binôme 3 : Java Spring + Kong-Gateway + Kubernetes
- Intégration partielle / tests via Postman
- Complément par benchmarks disponibles en ligne



## CRITÈRES D'ÉVALUATION

- Intégration : Facilité de mise en place, documentation
- Performance : Latence, temps de réponse
- Configuration : Complexité des fichiers, clarté
- Observabilité : Logs, monitoring
- Expérience développeur : prise en main, outils, debug

# API Gateways

- Point d'entrée unique pour exposer les microservices
- Permet le routage, l'authentification, le load-balancing
- Simplifie l'accès aux services dans une architecture distribuée
- Différents niveaux de complexité selon la technologie utilisée (Traefik, Kong, Spring Cloud Gateway)
- Peut offrir des outils de sécurité, monitoring, rate-limiting...

# Orchestrateurs

- Gèrent le cycle de vie des conteneurs dans l'environnement
- Automatisent le déploiement, la mise à l'échelle et la résilience des services
- Compose = simple pour dev local
- Swarm = léger mais peu utilisé aujourd'hui
- Kubernetes = complet mais plus complexe à maîtriser

# Technos testées : API Gateways



# Python & Traefik

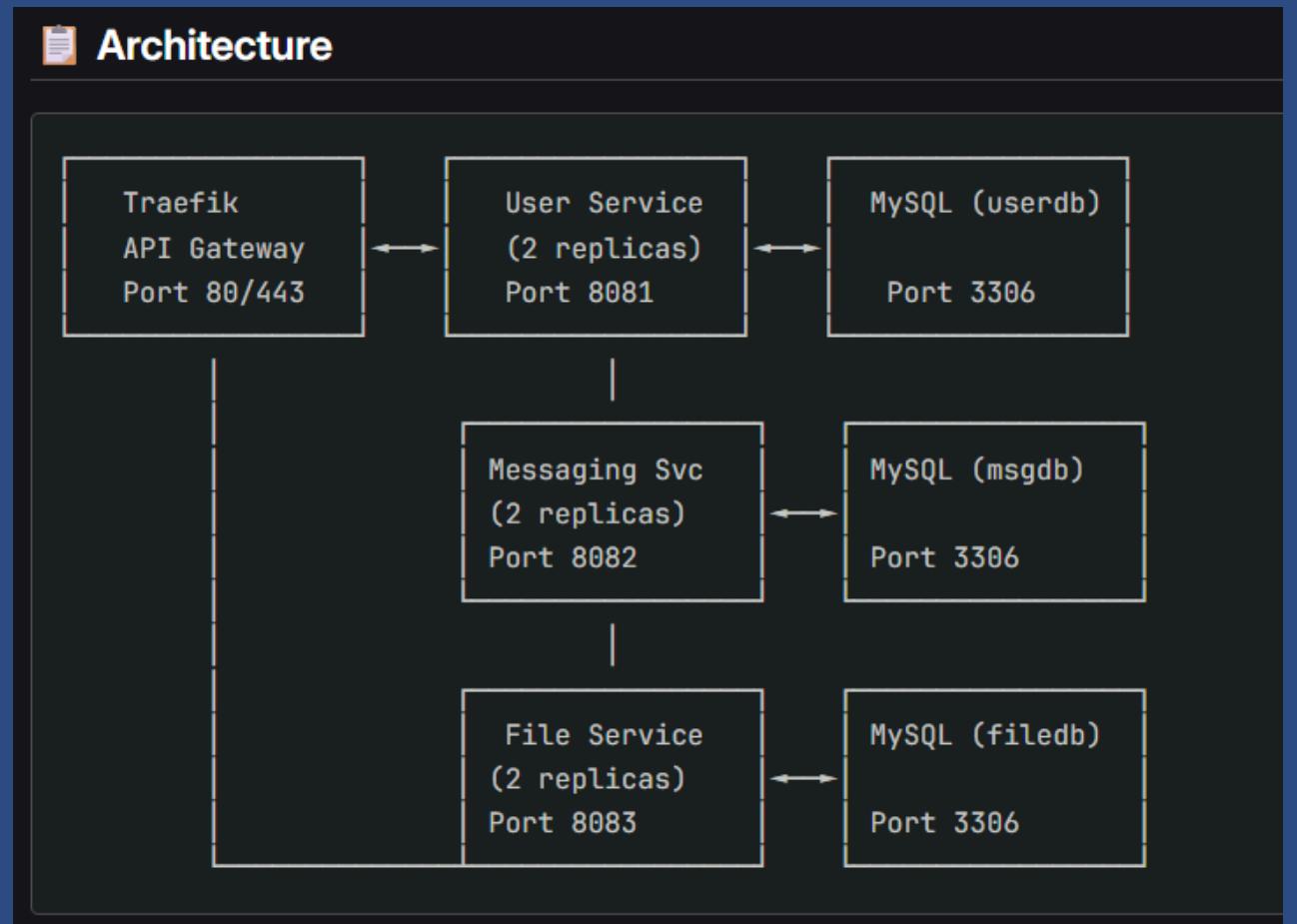
BASE FASTAPI + DOCKERISÉE

## Orchestrateur Docker Swarm

Déploiement distribué localement avec docker stack deploy

Facile à initialiser, mais moins populaire aujourd'hui

## Api Gateway Traefik



- Configuration via labels dans le docker-compose.yml
- Détection automatique des services
- Interface web de monitoring utile

## Difficultés rencontrées

- Configuration réseau dans Swarm
- Exposition des ports / services dynamiques
- Logs et erreurs pas toujours explicites

# Java + Spring Cloud Gateway

ARCHITECTURE SPRING  
BOOT + SPRING CLOUD  
GATEWAY

## Orchestrator Docker-Compose

Déploiement simple et rapide pour développement local  
Très adapté aux services Spring Boot

## Spring Cloud Gateway

- Routing fluide via fichiers YAML
- Très bonne intégration Spring (annotations, auto-discovery)
- Config centralisée et réutilisable

## Difficultés rencontrées

- Configuration dynamique plus rigide
- Gestion du reload sans redémarrage délicate
- Fichier application.yml peut devenir verbeux

# Java + Kong

ARCHITECTURE SPRING  
BOOT + KONG

## Orcheteur Kubernetes

Déploiement via Minikube avec YAML + kubectl  
Très puissant mais demande une bonne maîtrise

### Kong

- Architecture modulaire avec plugins
- Supporte authentification, logging, rate-limiting
- Adapté aux environnements multi-langages

### Difficultés rencontrées

- Installation initiale complexe (Kong + DB + Ingress)
- Beaucoup de fichiers YAML à maintenir
- Monitoring à configurer manuellement



# Critères d'évaluation

TABLEAU RÉCAPITULATIF

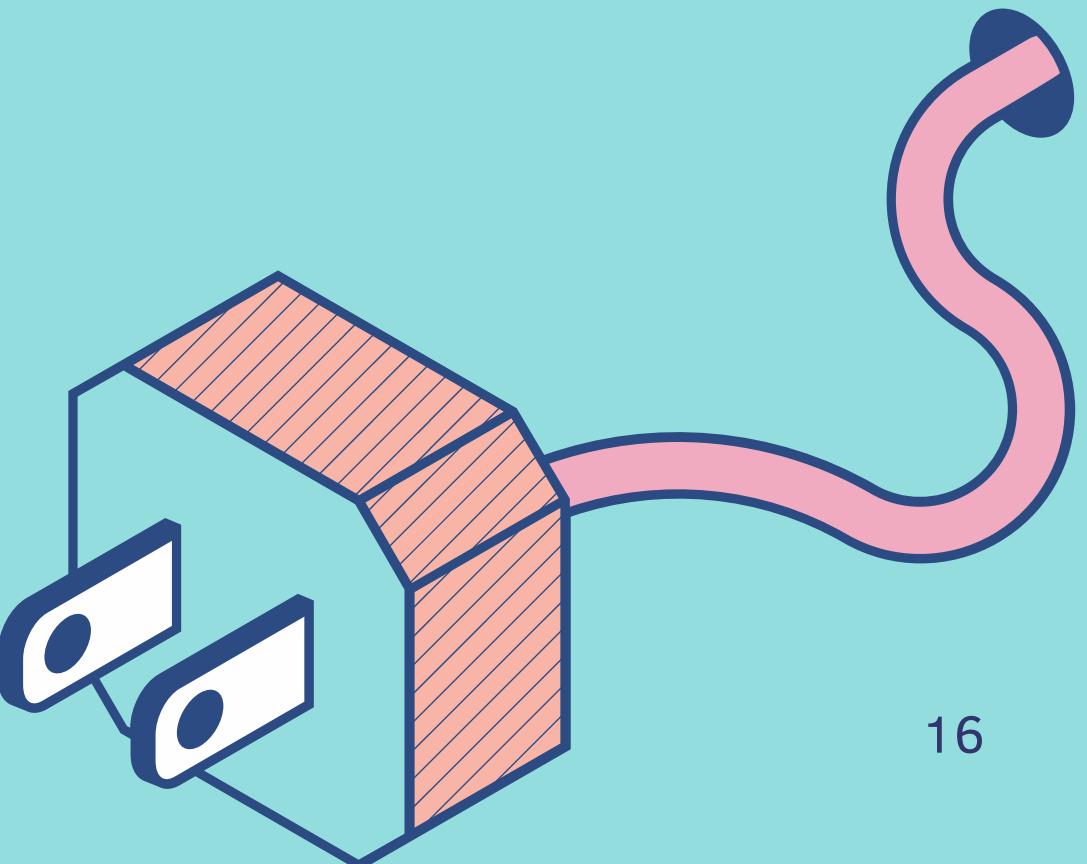
# Critères d'évaluation

TABLEAU RÉCAPITULATIF

CRITÈRE	BINÔME 1	BINÔME 2	BINÔME 3
Facilité d'intégration	Configuration via labels, rapide	Intégration Spring native très fluide	Installation plus lourde, mais plugins puissants
Simplicité de configuration	Peu de fichiers, mais attention au réseau Swarm	YAML clair dans Spring, peut devenir verbeux	Besoin de beaucoup de manifests YAML
Courbe d'apprentissage	Facile pour dev Python	Naturel si déjà Spring Dev	Complexe (K8s + Kong + plugins)
Performance	Bon temps de réponse local	Stable, mais dépend du JVM startup	Très performant, plugins optionnels ajoutables
Observabilité / Logs	Dashboard Traefik utile, logs simples	Logs disponibles via Spring Boot Actuator	Requiert setup manuel (ex. Grafana + Prometheus)

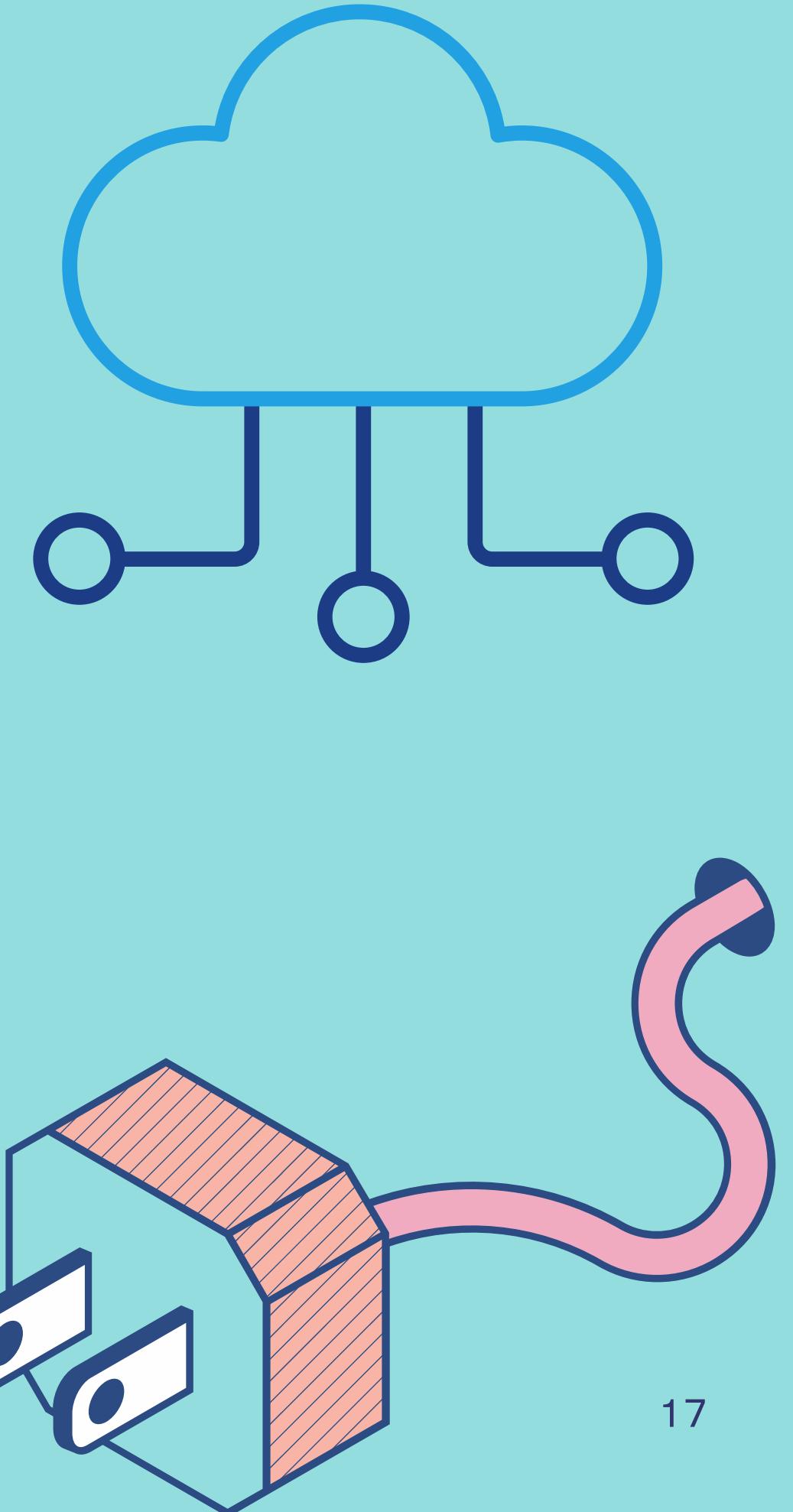
# Benchmark API Gateways

Gateway	RPS (req/s)	Latence P95	Remarques
Traefik	~160 000	~2 ms	Très performant, facile à configurer
Kong	96k – 138k	~4–5 ms	Solide même avec plugins
Spring Cloud Gateway	20k – 50k	>30 ms (P99)	Idéal pour écosystème Spring



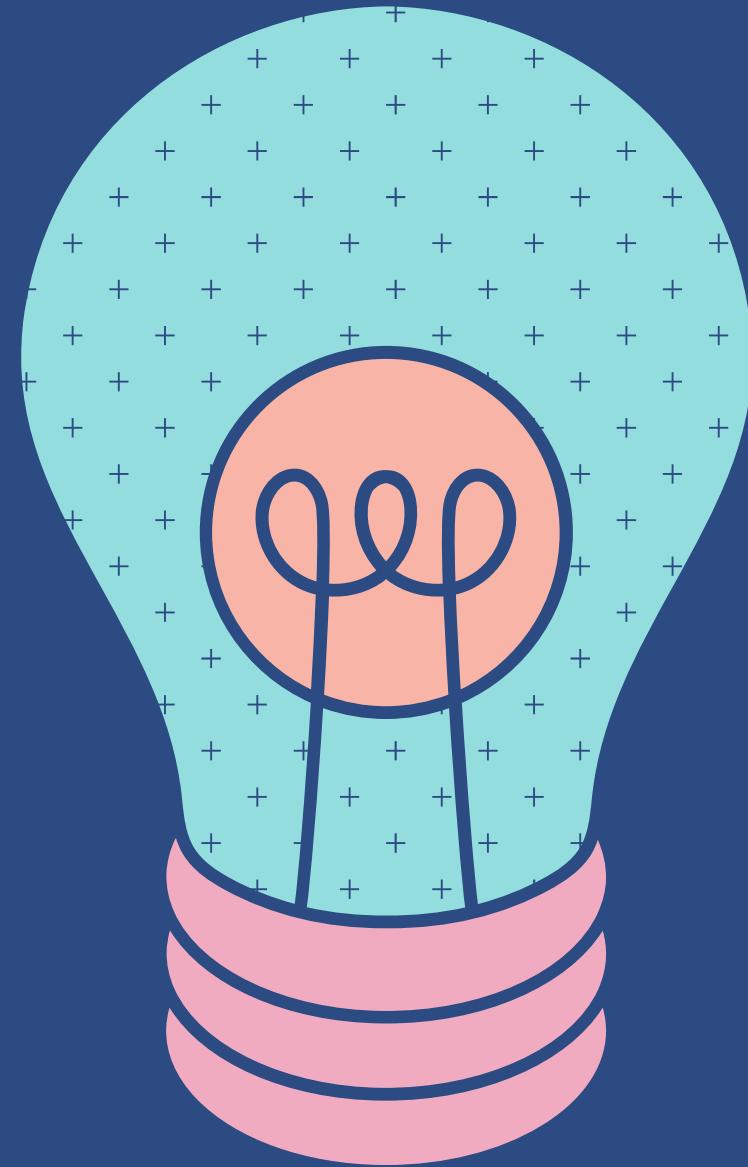
# Benchmark Orchestrateurs

Orchestrateur	Facilité	Scalabilité	Commentaires
Docker Compose	★★★★★	★	Parfait pour développement local
Docker Swarm	★★★	★★★	Léger, simple, mais peu maintenu
Kubernetes	★★	★★★★★	Très puissant, mais plus complexe



# Perspectives

- FINIR LES DÉPLOIEMENTS COMPLETS (GATEWAY + ORCHESTRATION)
- AJOUT D'UN MONITORING (PROMETHEUS, GRAFANA)
- PIPELINE CI/CD (GITHUB ACTIONS)
- INTÉGRATION D'UN FRONTEND MINIMAL
- TEST DE MONTÉE EN CHARGE
- DÉPLOIEMENT SUR UN CLOUD (HORS PÉRIMÈTRE PRINCIPAL)



# Merci !

# Place aux questions !

FDI-LEEX

