



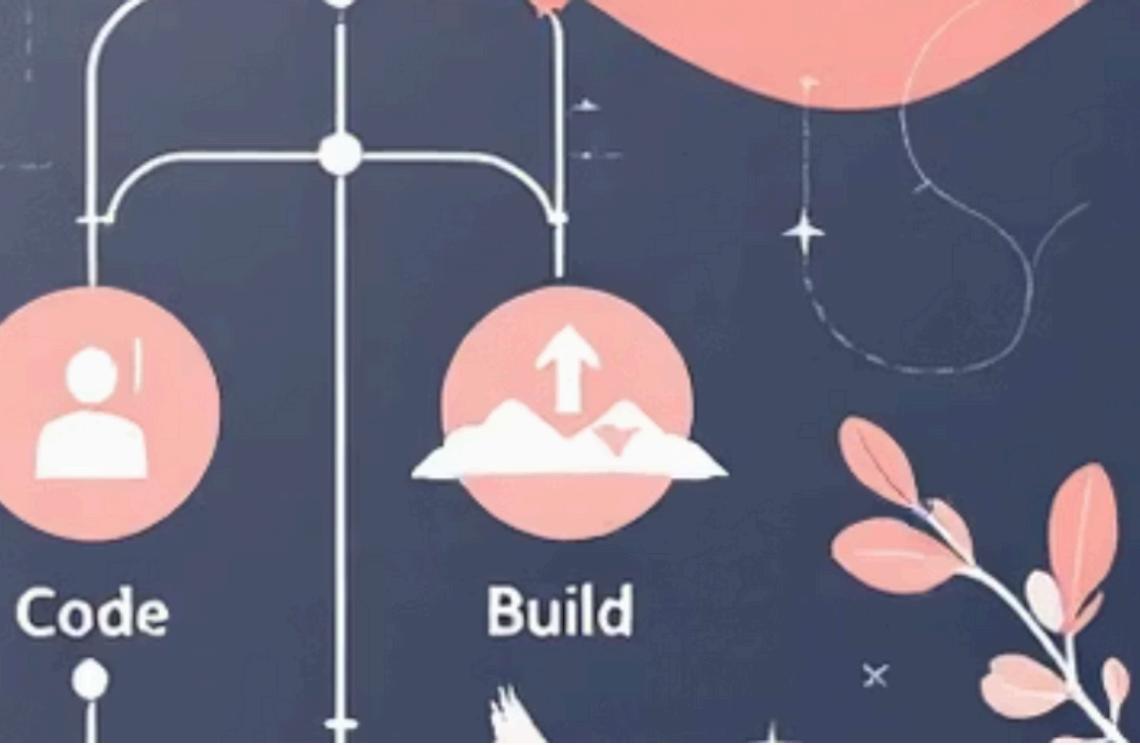
# LifyTP – Conteneurisation et orchestration d'une application microservices

Projet universitaire DevOps – Docker & Kubernetes

Raoul DRAGUS, Maelis NOHE ESTRADA, Reda AIT TALEB, Francklin VIRGINIE

ESILV, 2026

# DevOps



DEVOPS

CLOUD-NATIVE

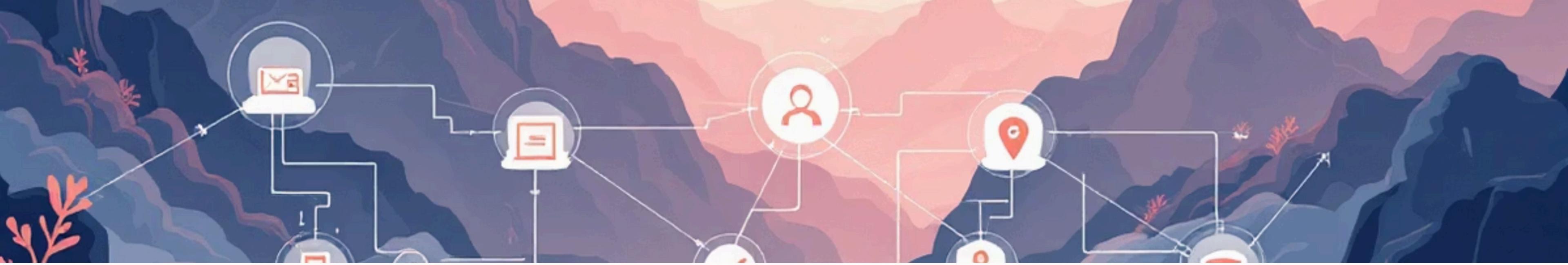
## Contexte et objectifs du projet

### Contexte moderne

Les applications cloud-native représentent l'avenir du développement logiciel. Elles offrent une scalabilité horizontale, une résilience accrue et une maintenance simplifiée grâce à l'architecture microservices.

### Objectifs pédagogiques

- Maîtriser les principes de la conteneurisation
- Déployer des microservices avec Docker
- Orcherstrer avec Kubernetes
- Automatiser via pipelines CI/CD



# Architecture globale

Service d'authentification

Gestion sécurisée des utilisateurs et sessions

Service d'événements

Orchestration et gestion des activités

Service de messagerie

Communication temps réel entre utilisateurs

## Infrastructure de support

PostgreSQL

Base de données relationnelle pour la persistance

Redis

Cache distribué et gestion de sessions

MinIO

Stockage objet compatible S3



# Conteneurisation avec Docker

1

## Dockerfiles optimisés

Chaque microservice possède son propre Dockerfile, garantissant une image légère et spécifique aux besoins du service.

2

## Environnements isolés

Les conteneurs offrent une isolation complète, éliminant les conflits de dépendances et assurant la reproductibilité.

3

## Images optimisées

Utilisation de multi-stage builds (lorsque pertinent) pour réduire la taille des images et améliorer les performances de déploiement.

4

## Orchestration locale

Docker Compose permet de lancer l'ensemble de l'écosystème avec une seule commande pour le développement.

# Environnement de développement

01

Orchestration multi-conteneurs

Coordination automatique de tous les services et leurs dépendances

02

Réseaux et volumes dédiés

Isolation réseau et persistance des données garanties

03

Démarrage simplifié

Une seule commande pour lancer l'environnement complet

04

Isolation totale

Séparation complète de l'application principale Lify



`npm run lifytp:start` – Lance l'ensemble de l'infrastructure en quelques secondes



# Orchestration Kubernetes



## Namespace dédié

Isolation logique via le namespace lifytp pour une séparation claire des ressources du cluster.



## Deployments

Gestion déclarative des microservices avec versioning et rollback automatique en cas d'échec.



## Services ClusterIP

Communication inter-services sécurisée via DNS interne et load balancing automatique.

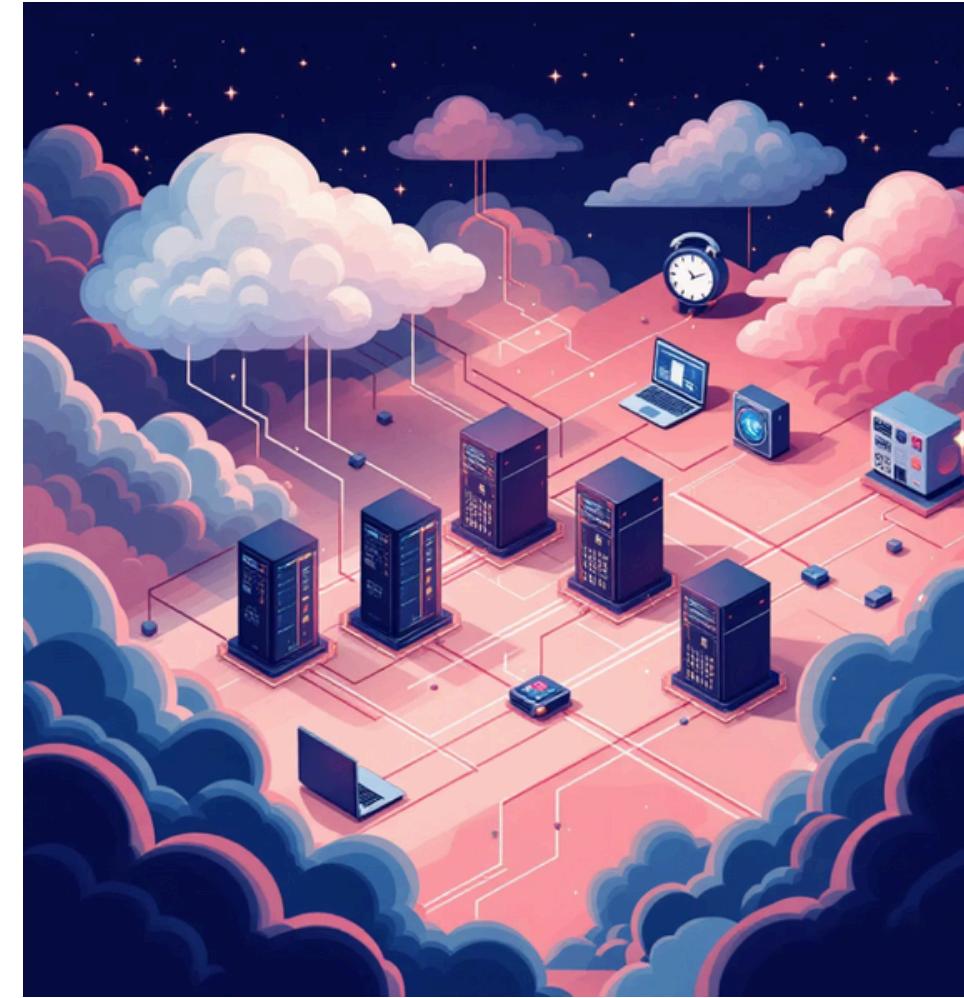


## StatefulSets

PostgreSQL déployé avec identité stable et stockage persistant garantissant l'intégrité des données.

L'architecture permet une **scalabilité horizontale** dynamique en fonction de la charge applicative.

# Haute disponibilité et résilience



## Réplicas multiples

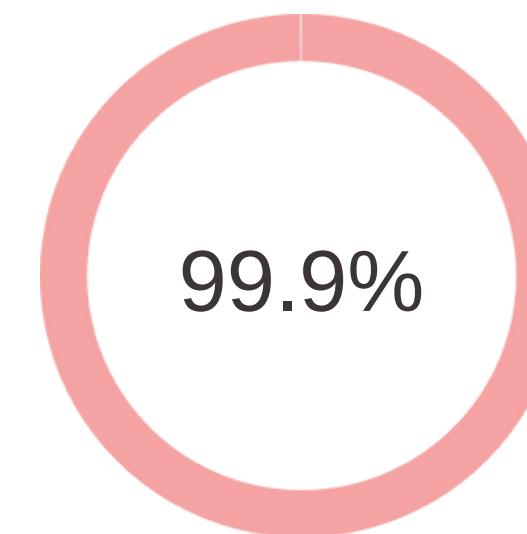
Chaque microservice fonctionne avec plusieurs instances pour garantir la continuité de service.

## Auto-healing intelligent

Kubernetes détecte automatiquement les pods défaillants et les recrée instantanément sans intervention manuelle.

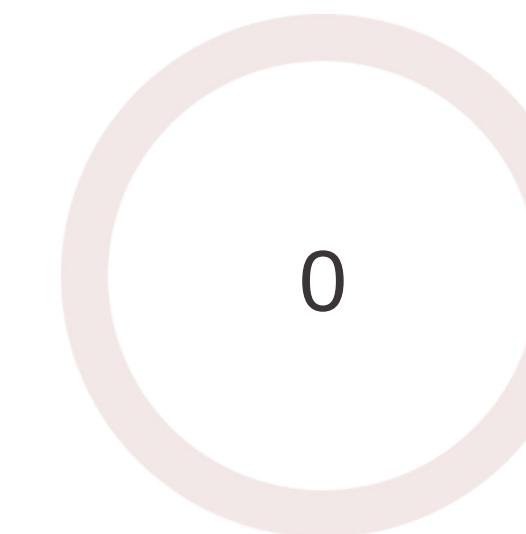
## Rolling updates

Les mises à jour progressives assurent un déploiement sans interruption de service, avec rollback automatique.



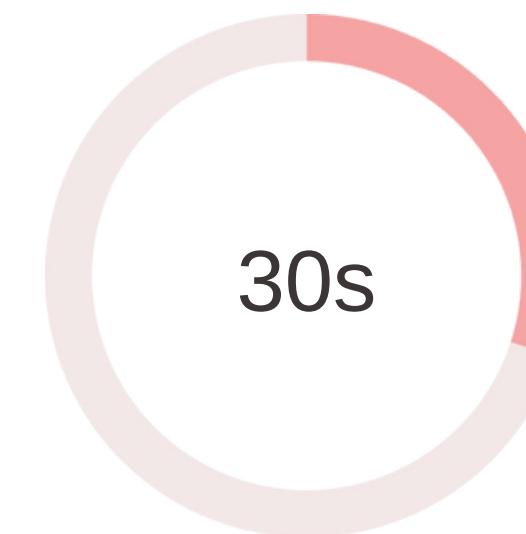
Disponibilité

Taux de disponibilité cible



Downtime

Interruption lors des déploiements



Récupération

Temps moyen de restauration

# Pipeline CI/CD automatisé



## Intégration continue

- Validation automatique du code
- Tests unitaires et d'intégration
- Build des images Docker

## Génération d'images

- Construction optimisée
- Tagging sémantique
- Push vers le registry

## Déploiement continu

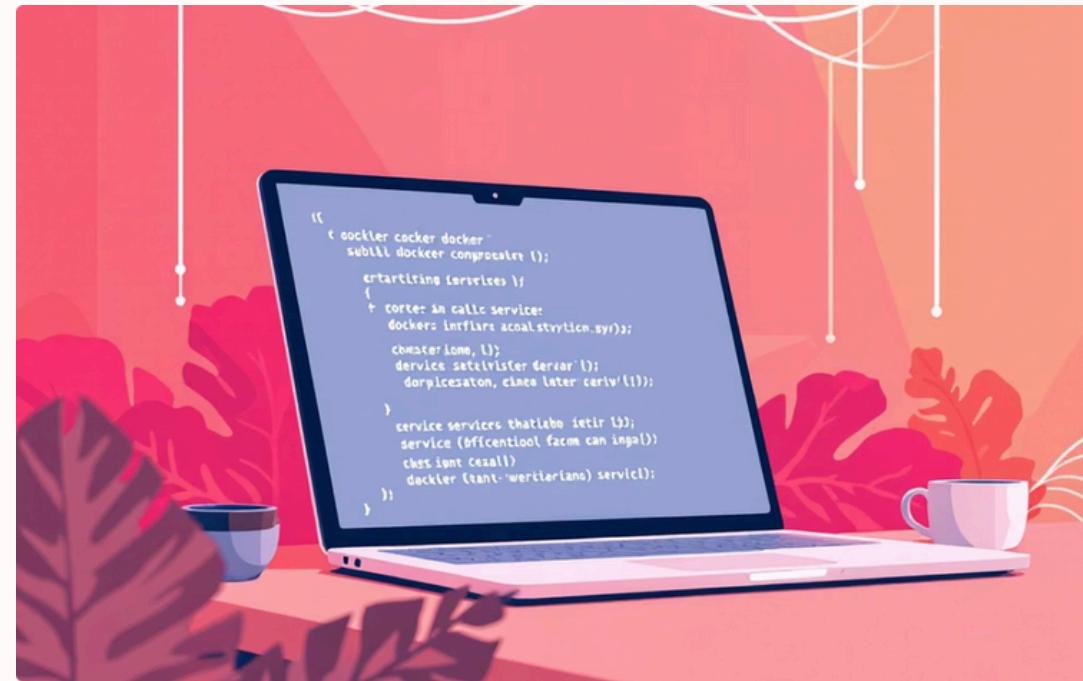
- Déploiement sur Kubernetes
- Mise à jour progressive
- Vérification de santé



**GitHub Actions** orchestre l'ensemble du pipeline de manière transparente et automatique

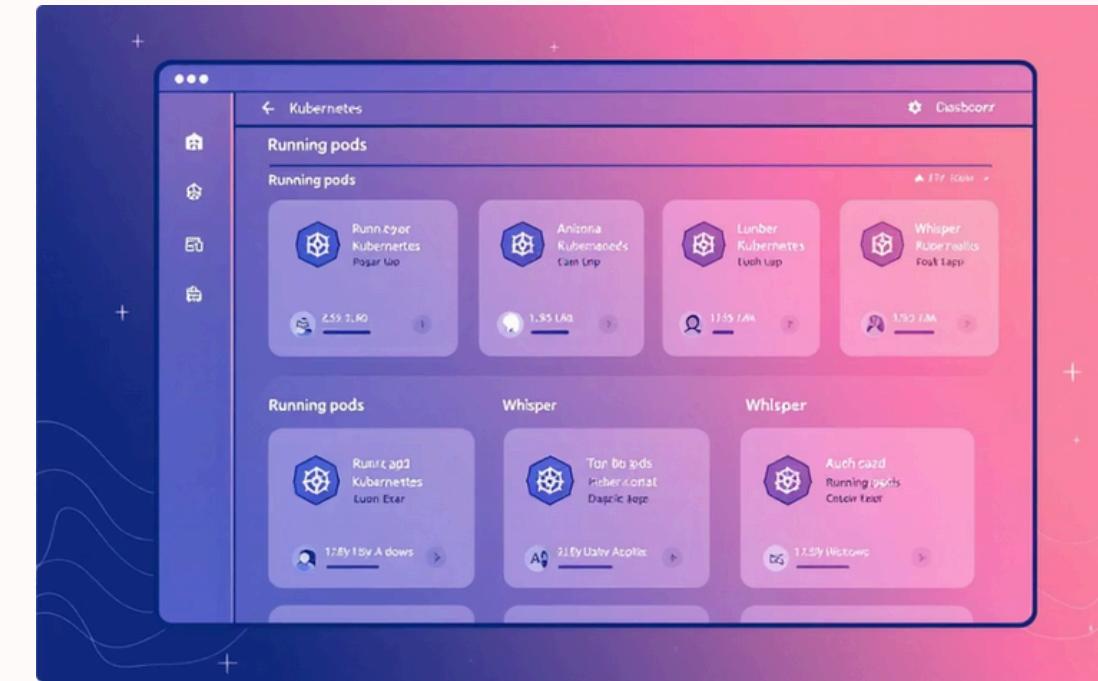


# Démonstrations et validation



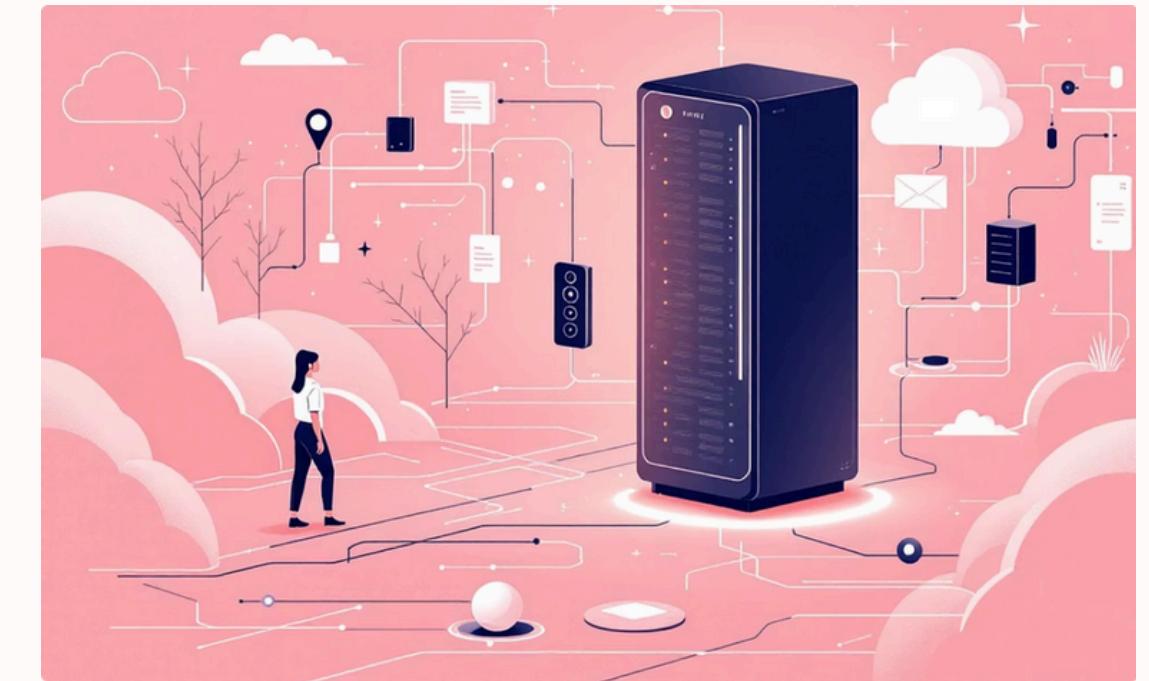
## Exécution Docker Compose

Lancement de l'environnement complet de développement en local



## Déploiement Kubernetes

Visualisation des pods et services actifs dans le cluster



## Auto-healing en action

Récupération automatique après simulation de panne d'un pod

## Rolling update

Démonstration de mise à jour sans interruption de service avec rollback en cas d'erreur

## Tests API

Vérification des endpoints de santé et validation des flux de communication entre services

# Conclusion et perspectives

## Réalisations

Implémentation réussie d'une architecture cloud-native complète avec maîtrise des concepts Docker, Kubernetes et CI/CD

## Compétences acquises

Conteneurisation, orchestration, automatisation des déploiements et gestion de la haute disponibilité

## Améliorations futures



### Monitoring et observabilité

Intégration de Prometheus, Grafana et ELK Stack pour la supervision complète



### Renforcement sécuritaire

Mise en place de Network Policies, secrets management et scanning de vulnérabilités



### Patterns avancés

Service mesh, circuit breakers et stratégies de déploiement canary/blue-green

