

**Rapport projet**

**Gestion des comptes**

**Binôme : Nadia Ajmaoui**

**Amina Belkiyal**

# **Introduction**

Ce rapport se concentre sur l'architecture micro services adoptée pour notre application bancaire. L'architecture micro services, caractérisée par sa flexibilité et sa scalabilité, a été mise en œuvre avec succès pour répondre aux défis actuels du développement logiciel.

Notre application est structurée autour de trois micro services distincts **: le Service Compte**, le **Service Transaction**, et le **Service Utilisateur**. Chaque service interagit avec le service de découverte Eureka et transite par la passerelle API Zuul, facilitant ainsi une expérience utilisateur fluide.

Cette étude examine les composantes clés de l'architecture micro services, mettant en évidence les avantages qu'elle offre. Nous explorons également les implications sur des aspects tels que la résilience et la gestion des données.

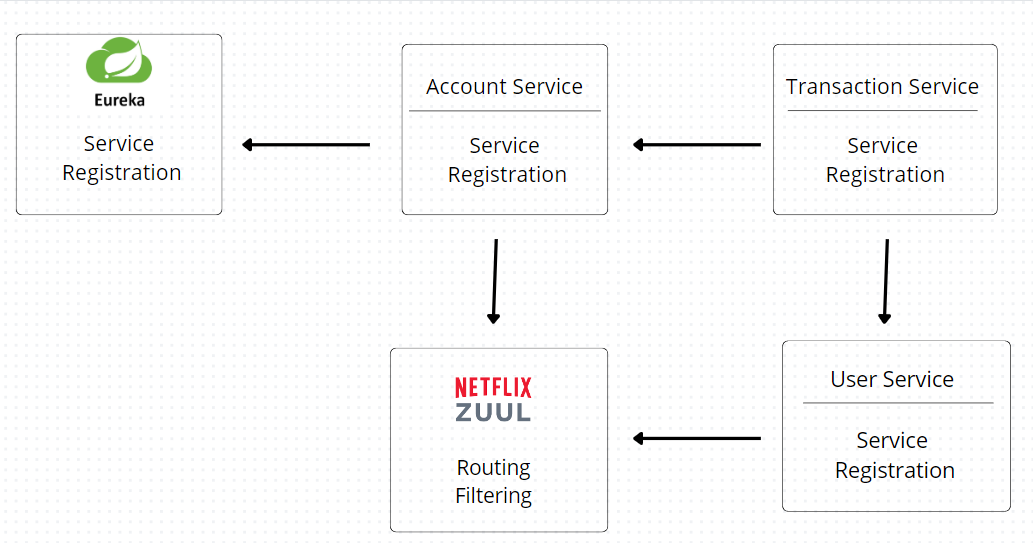
L'objectif est de fournir une compréhension claire des choix architecturaux, de leurs avantages, et des considérations futures. Cette analyse vise à éclairer les décisions de développement logiciel et à enrichir la réflexion sur l'architecture micro services pour notre application bancaire.

# **Importance de l’architecture micro services**

L'adoption de l'architecture micro services revêt une importance cruciale dans le paysage du développement logiciel contemporain. Cette approche modulaire permet de décomposer une application monolithique en composants indépendants, favorisant ainsi une évolutivité, une agilité et une maintenance accrues. Les micro services permettent aux équipes de développement de travailler de manière autonome sur des services spécifiques, accélérant ainsi le déploiement des fonctionnalités et facilitant les mises à jour continues. La scalabilité granulaire offre la flexibilité nécessaire pour répondre dynamiquement aux fluctuations de la demande, tandis que la résilience inhérente à cette architecture garantit une meilleure tolérance aux pannes. En outre, les micro services permettent une gestion efficace des technologies variées au sein d'une même application, favorisant l'innovation et la modularité. En résumé, l'adoption des micro services représente une étape cruciale pour les organisations cherchant à concilier rapidité de développement, agilité opérationnelle et capacité d'adaptation aux exigences changeantes du marché.

# **Architecture Micro services :**

## **Architecture**

****

## **Description des services :**

1. **Service Compte :**

Le micro service Compte gère les opérations liées aux comptes bancaires. Il assure la création, la consultation, la mise à jour et la suppression des comptes.

Grâce à son indépendance, il permet une gestion efficace des données liées aux comptes tout en favorisant une évolutivité séparée du reste du système.

**2. Service Transaction :**

Le micro service Transaction est dédié à la gestion des transactions financières au sein de l'application.

Il prend en charge des opérations telles que les transferts de fonds, les dépôts et les retraits. Grâce à son autonomie, ce service assure une traçabilité précise des transactions, garantissant l'intégrité des opérations financières et permettant une évolutivité adaptée à la complexité des interactions.

1. **Service Utilisateur :**

Le micro service Utilisateur gère les informations liées aux utilisateurs de l'application. En permettant une gestion décentralisée des données utilisateur, ce service assure une flexibilité dans la gestion des profils, tout en favorisant une sécurité renforcée et une maintenance simplifiée.

Chaque micro service est conçu pour fonctionner de manière indépendante, offrant des services spécifiques tout en contribuant de manière collaborative à l'ensemble de l'application. L'utilisation de ces micro services distincts, orchestrés par des composants tels que le service de découverte Eureka et la passerelle API Zuul, améliore la scalabilité, la résilience et la maintenance de l'application bancaire globale.

## **Mécanisme de communication :**

Le mécanisme de communication au sein de l'architecture micro services repose généralement sur des échanges via des API (Interfaces de Programmation Applicative) RESTful. Chaque micro service expose des points d'accès bien définis, permettant aux autres services de consommer leurs fonctionnalités. Dans le contexte de l'application bancaire, les micro services, tels que le Service Compte, le Service Transaction, et le Service Utilisateur, communiquent entre eux de manière décentralisée.

L'utilisation d'une architecture basée sur HTTP/REST facilite l'intégration et permet une communication légère entre les services. Les requêtes HTTP, telles que GET, POST, PUT, et DELETE, sont employées pour effectuer des opérations sur les ressources exposées par chaque micro service. Ces échanges sont souvent accompagnés de formats de données standard tels que JSON, favorisant une communication claire et cohérente entre les services.

La mise en œuvre de la découverte de services, comme celle fournie par Eureka, facilite la localisation dynamique des micro services. Ainsi, un micro service peut découvrir et interagir avec d'autres services sans avoir besoin de configurations statiques.

En résumé, le mécanisme de communication dans une architecture micro services s'appuie sur des principes RESTful avec des requêtes HTTP, favorisant la modularité, la flexibilité, et l'évolutivité des interactions entre les composants de l'application bancaire.

# **Conception des Micro-services :**

## **Approche de conception pour chaque service**

**1. Service Compte :**

• Responsabilité : Gestion des opérations relatives aux comptes bancaires, y compris la création, la mise à jour, la consultation, et la suppression.

•Contrat d'Interface : Exposition d'API RESTful claires pour l'accès aux informations des comptes et la réalisation d'opérations financières.

• Données : Gestion indépendante de la base de données des comptes, incluant le solde, la date de création, et le type de compte.

**2. Service Transaction :**

•Responsabilité : Gestion des transactions financières, telles que les transferts de fonds, les dépôts, et les retraits.

•Contrat d'Interface : Définition d'API RESTful pour l'initiation et la consultation des transactions, assurant la traçabilité des opérations.

• Données : Stockage indépendant des détails de transaction, garantissant l'intégrité et la cohérence des mouvements financiers.

1. **Service Utilisateur :**

•Responsabilité : Gestion des informations liées aux utilisateurs

•Contrat d'Interface : Exposition d'API pour l'inscription, l'authentification, et la gestion des profils.

•Données : Gestion autonome de la base de données utilisateur.

# **CI/CD avec Jenkins :**

Jenkins est un outil open source d'intégration continue et de déploiement continu (CI/CD) qui joue un rôle central dans le processus de développement logiciel. Son principal objectif est d'automatiser les différentes phases du cycle de vie d'un projet, de la compilation du code source à son déploiement en passant par les phases de test.

## **Processus et configuration :**

Ce pipeline ci-dessous automatise le processus de construction, création d'une image Docker et exécution de notre projet.

* **Étapes :**

1. Git Clone : Clone le dépôt du projet depuis la branche 'master'.

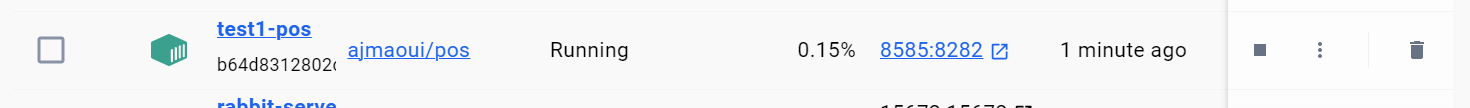
2. Build : Utilise Maven pour nettoyer et installer les dépendances du projet.

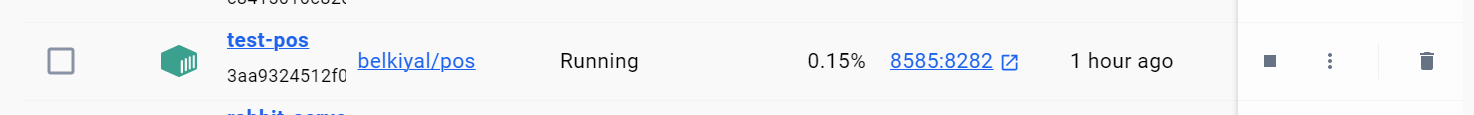
3. Création de l'image Docker : Construit une image Docker nommée 'belkiyal/pos' à partir du répertoire du projet.

4. Exécution : Lance le conteneur Docker nommé 'test-pos' en mode détaché, en associant le port 8585 sur l'hôte au port 8282 dans le conteneur.

pipeline { agent any tools { maven 'maven' } stages { stage('Git Clone') { steps { script { checkout([$class: 'GitSCM', branches: [[name: 'master']], userRemoteConfigs: [[url: 'https://github.com/Aminabelkiyal/projet-mic.git']]]) } } } stage('Build') { steps { script { // Navigate into the project directory before running Maven commands dir('POV-JAVA') { bat 'mvn clean install' } } } } stage('Create Docker Image') { steps { dir('POV-JAVA') { bat 'docker build -t belkiyal/pos .' } } } stage('Run') { steps { script { bat "docker run --name test-pos -d -p 8585:8282 belkiyal/pos" } } } }}

## 





## 

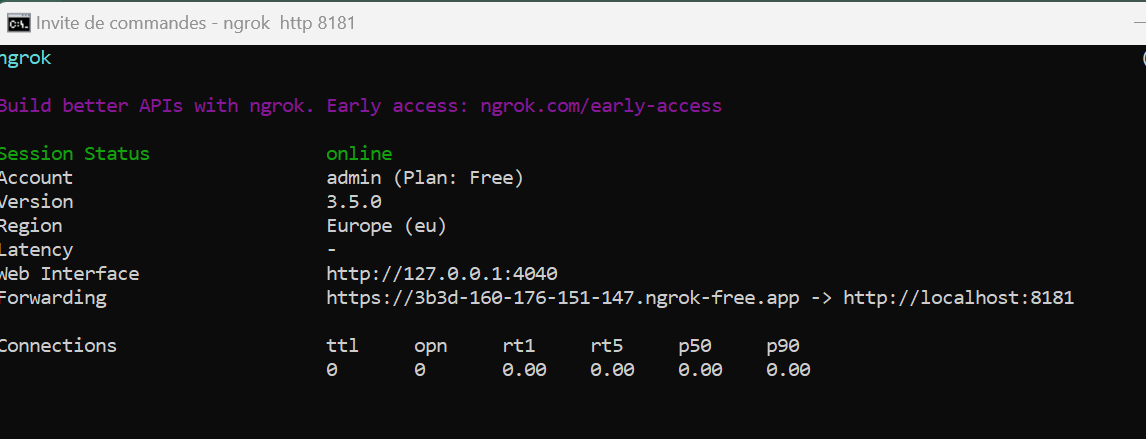
# **Déploiement Automatique :**

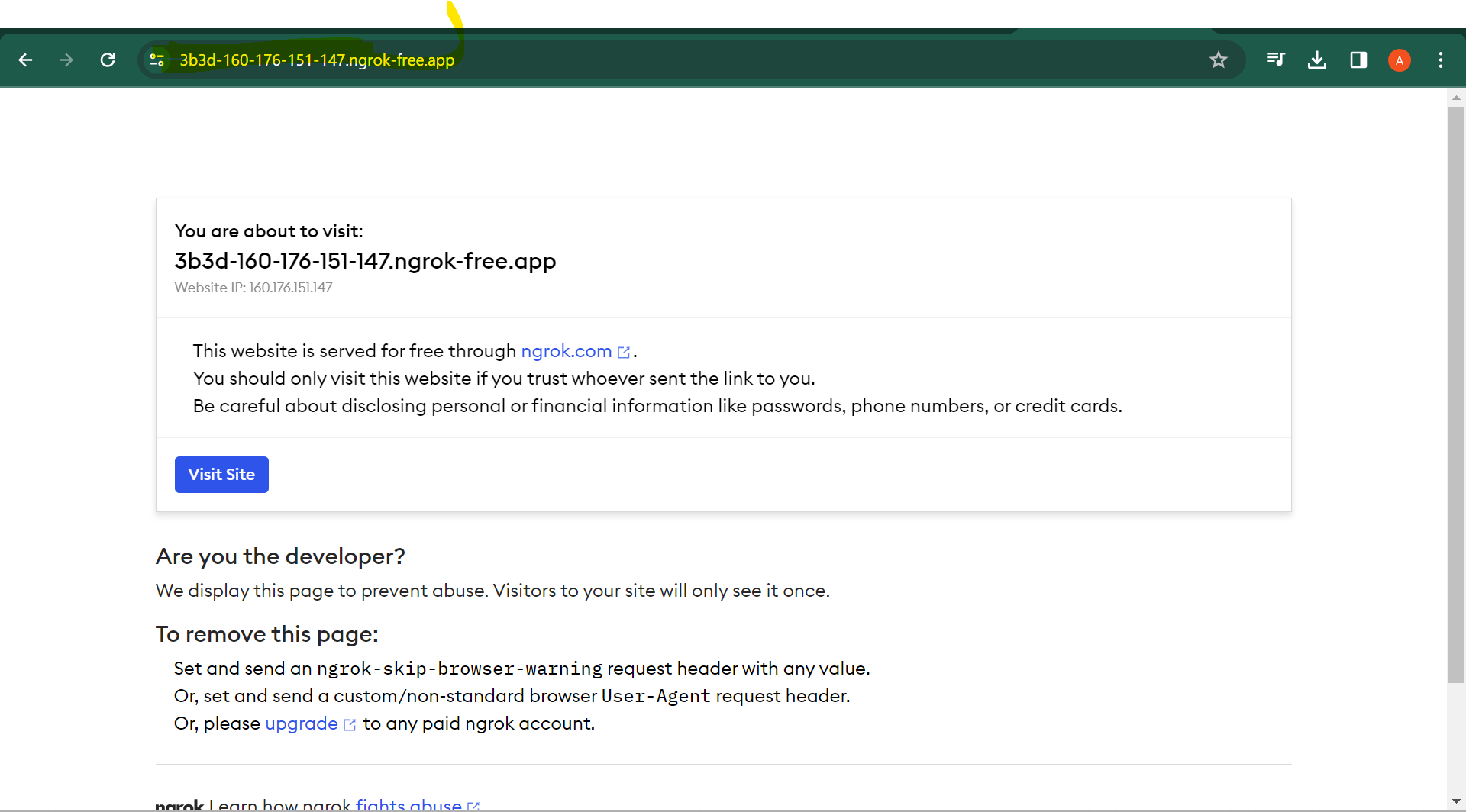
## **Ngrok**

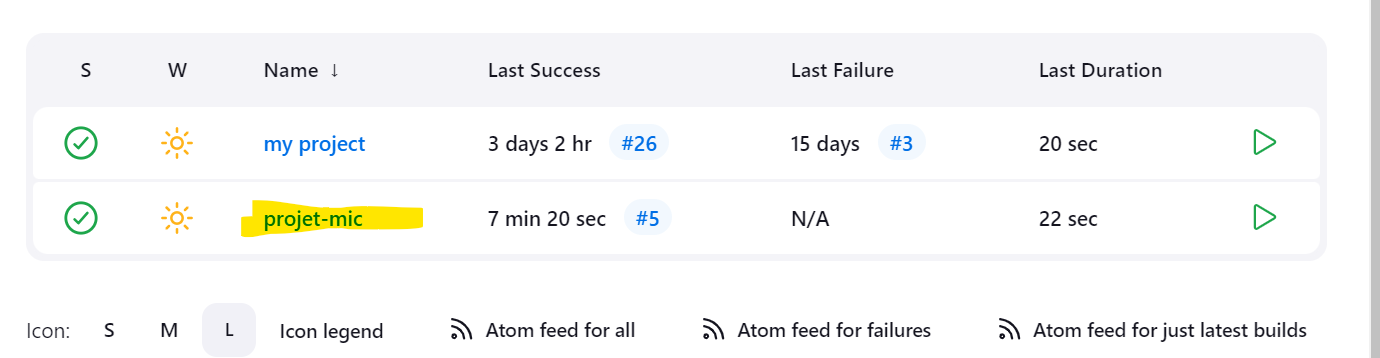
Ngrok est un outil de tunneling qui permet de rendre des services locaux accessibles depuis Internet en créant un tunnel sécurisé.

Il facilite le partage d'applications en développement ou de serveurs locaux avec d'autres utilisateurs, même à travers des r31éseaux privés ou des pares-feux.

Ngrok est souvent utilisé dans le développement web pour permettre un accès distant aux services locaux.

****

****

****

# **Conclusion**

En résumé, l'adoption de l'architecture micro services dans notre application, soutenue par l'utilisation de Jenkins pour l'intégration continue, offre une approche flexible, évolutive et facilement maintenable.

Chaque micro service, spécialisé dans des responsabilités distinctes, favorise la modularité et la résilience, tandis que Jenkins automatise le processus de développement, améliorant ainsi la qualité du code.

Cette combinaison crée un écosystème agile, prêt à répondre aux exigences changeantes du développement logiciel.