

RÉPUBLIQUE DU CAMEROUN  
REPUBLIC OF CAMEROON  
*Peace – Work – Fatherland*  
**UNIVERSITÉ DE DSCHANG**  
UNIVERSITY OF DSCHANG  
*Scholae Thesaurus Dschangensis Ibi Cordum*  
P 96, Dschang (Cameroun) -Tél./Fax  
(237) 233 45 13 811  
Website: <http://www.univ-dschang.org>



**FACULTE DES SCIENCES**  
FACULTY OF SCIENCE  
**Département de Mathématiques  
et Informatique**  
Department of Mathematics and  
Computer Science  
BP 67, Dschang (Cameroun)  
E-mail : [udsrectorat@univ-dschang.org](mailto:udsrectorat@univ-dschang.org)

**Master 1 : Intelligence Artificielle**

**RSD 418 : Intergiciel pour les applications web**

**Thème:** Mise en place d'une application de gestion des ressources pour une multinationale de voyage suivant une architecture micro-service.

(Université de dschang)

3 juin 2025

Rédigé Par :

Noms et prénoms	Matricules
KENFACK FONGANG Victor Cyntiche *	CM-UDS-21SCI0555
MFENTAM Mohammed Salam	CM-UDS-21SCI0941
KOUOKAM TALLA Eugene Asaph	CM-UDS-21SCI0021
CHEGUEP Marcelle Fadhy	CM-UDS-21SCI0709

Sous la supervision de : **Pr BOMGNI Alain Bertrand.**

Année académique : **2024/2025**

Récapitulatif des tâches et des participations						
Tâches journalières		Membres du groupe et présences				
jour	Tâches	KENFACK FONGANG	MFENTAM MOHAMMED	KOUOKAM TALLA	CHEGUEP MARCELLE	
17/04/2025 08h-14H	<ul style="list-style-type: none"> <li>• Choix du thème</li> <li>• Définition de l'architecture</li> <li>• Choix des outils</li> <li>• Mise en place de la base de données</li> </ul>	✓	✓	✓		✓
19/04/2025 16h-19h	<ul style="list-style-type: none"> <li>• Mise en place du pipeline CI/CD avec Jenkins</li> </ul>	✓				
20/04/2025 08-14h	<ul style="list-style-type: none"> <li>• Mise en place des serveurs (config-server, debug-server, Proxy-server, DNS_Server, Git-Server, server web)</li> </ul>	✓	✓			
22/04/2025 11h-14h	<ul style="list-style-type: none"> <li>• Implémentation des services (employee, agency, person, automobile, role, city, salary)</li> </ul>	✓	✓	✓		✓
24/04/2025 11h-14h	<ul style="list-style-type: none"> <li>• Mise en place des serveurs (discovery-server, Spring Authorization server, DNS_Server)</li> </ul>	✓	✓			
04/05/2025 10h-14h	<ul style="list-style-type: none"> <li>• implémentation de l'interopérabilité du système</li> </ul>	✓				
05/05/2025 15h-17h	<ul style="list-style-type: none"> <li>• Déploiement et monitoring final</li> </ul>	✓				
00/05/2025 14h-17h	<ul style="list-style-type: none"> <li>• Rédaction du rapport</li> </ul>	✓	✓	✓		✓
Total	Fin du projet	8/8	5/8	3/8		3/8

## Table des matières

<b>1</b>	<b>Contexte et description du projet</b>	<b>5</b>
<b>2</b>	<b>Réalisation</b>	<b>6</b>
2.1	Architectures du système . . . . .	6
2.1.1	Architecture générale de l'application . . . . .	6
2.1.2	outils utilisés . . . . .	8
2.2	Présentation de l'application . . . . .	11
2.3	description de la base de données . . . . .	12
2.4	implémentation du serveur d'authentification et d'autorisation (Keycloak) . . . . .	12
2.5	implémentation du serveur de découverte de service (eureka) . . . . .	13
2.6	implémentation du proxy (gateway) . . . . .	15
2.7	implémentation du serveur de configuration (config-server) . . . . .	15
2.8	implémentation du serveur de configuration (config-server) . . . . .	16
2.9	implémentation du serveur d'administration (spring-admin-server) . . . . .	16
2.10	implémentation des micro-services . . . . .	17
2.10.1	implémentation du service City . . . . .	17
2.10.2	implémentation du service agency . . . . .	18
2.10.3	implémentation du service automobile . . . . .	18
2.10.4	implémentation du service role . . . . .	19
2.10.5	implémentation du service salary . . . . .	19
2.10.6	implémentation du service person . . . . .	20
2.10.7	implémentation du service employee . . . . .	20
2.10.8	Communication des micro-services . . . . .	21
2.11	implémentation des pipelines Jenkins . . . . .	21
2.12	implémentation des déploiements sur le cloud . . . . .	22
<b>3</b>	<b>référentiel github pour le code source de l'application</b>	<b>23</b>

## Table des figures

1	description globale du système . . . . .	6
2	system into microservice architecture . . . . .	11
3	base de donnée . . . . .	12
4	fonctionnement du serveur d'authentification et d'autorisation . . . . .	12
7	fonctionnement du serveur eureka . . . . .	13
8	fonctionnement d'une gateway . . . . .	15
9	fonctionnement du serveur de configuration . . . . .	15

10	fonctionnement du serveur de debogage ZipKin . . . . .	16
13	fonctionnement d'un micro-service . . . . .	17
14	city-service . . . . .	17
15	agency-service . . . . .	18
16	automobile-service . . . . .	18
17	role-service . . . . .	19
18	salary-service . . . . .	19
19	person-service . . . . .	20
20	employee-service . . . . .	20
21	Communication micro-services . . . . .	21
23	pipeline Jenkins . . . . .	21
24	Terraform management . . . . .	22

# 1 Contexte et description du projet

Mr X est propriétaire d'une multinationale de voyage distribuée au sein de plusieurs villes, ces villes sont dotées de plusieurs agences et celles-ci ont plusieurs employés, actuellement l'entreprise utilise plusieurs applications pour la gestion des employés et la flotte des bus ce qui fait que, Mr X se trouve confronté à des défis de coordination des équipes et l'optimisation des ressources, de suivi de performance et d'optimisation des ressources notamment :

- ❑ La difficulté à suivre la disponibilité des employés et des bus
- ❑ La gestion manuelle des recensements des ressources entraînant des erreurs, des retards et inefficacités
- ❑ La gestion manuelles des rapports entraînant le manque de visibilité sur les performances opérationnelles et financières ce qui rend difficile l'identification des problèmes et l'optimisation des processus
- ❑ Les applications actuelles ne permettent pas une gestion centralisée et efficace des employés et des bus ce qui entraine des conflits d'horaire et une mauvaise utilisation des ressources
- ❑ Ces dernières ne communiquent pas entre elles rendant difficile le suivi des performances globales et l'accès a des données précises compliquant la prise de décision stratégique etc.

Dans un secteur aussi compétitif et Face à ces défis la structure a décidée de créer une nouvelle application qui répondra aux besoins spécifiques de l'entreprise en matière de gestion du personnel et de la flotte des bus. L'objectif est de développer une solution plus efficace intégrée et adaptée aux exigences opérationnelles actuelles :

- ❑ **Optimisation des ressources** : maximiser l'utilisation du personnel afin de réduire les couts
- ❑ **Scalabilité** : ajout facile de nouvelles fonctionnalités
- ❑ **Suivi et reporting** : centralisation afin de faciliter les analyses pour une meilleure prise de décisions etc.

Ainsi, l'équipe technique décide de mettre sur pied une application propre à une architecture micro-service qui présente plusieurs avantages :

- ❑ **Indépendance des services** : chaque service peut être développé, testé et déployé indépendamment facilitant ainsi la maintenance et l'évolutivité
- ❑ **Scalabilité** : les services sont mis à l'échelle individuellement en fonction de la demande

- ⊗ **Centralisation et interopérabilité** : les ressources sont centralisées et les services communiquent entre eux ce qui facilite la prise de décision globale et une vue plus éclaircie des ressources
- ⊗ **Résilience** : une défaillance dans un service n'affecte pas nécessairement l'ensemble de l'application etc.

## 2 Réalisation

### 2.1 Architectures du système

#### 2.1.1 Architecture générale de l'application

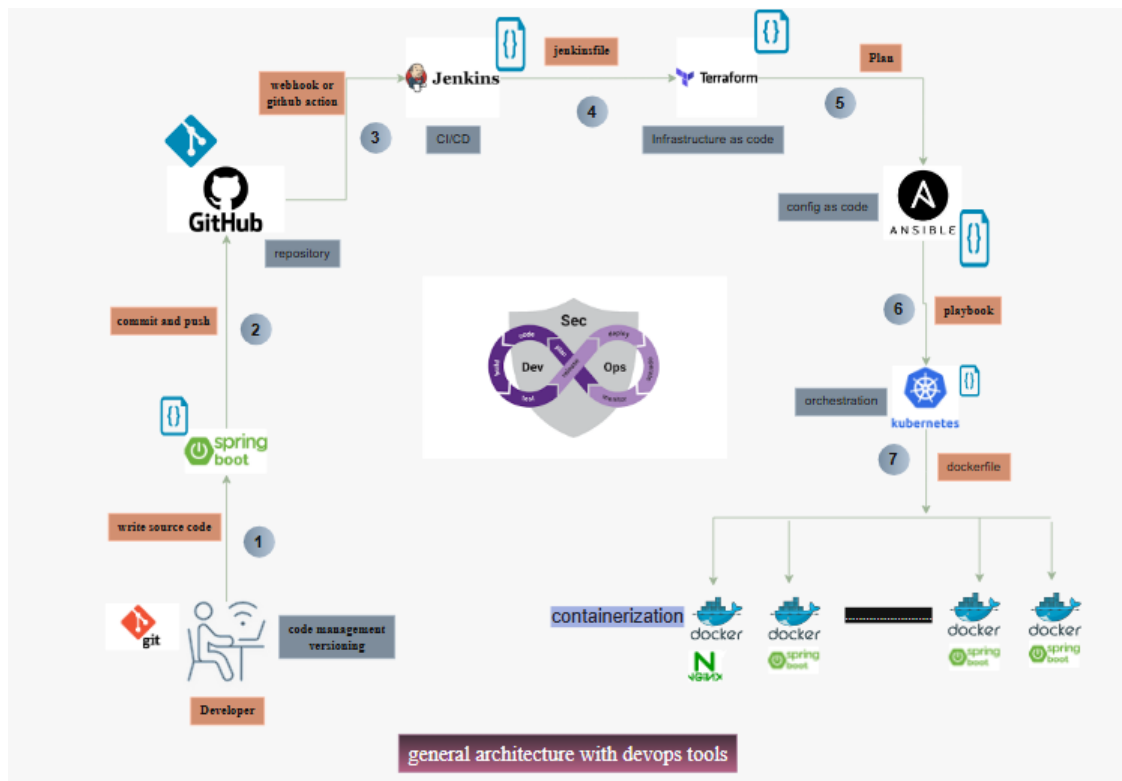










FIGURE 1 – description globale du système

C'est la description globale de l'application : on voit la communication et le fonctionnement du système, comment les composants sont reliés, où sont les bases de données, les outils de sécurité, et comment tout fonctionne ensemble pour bien gérer le personnel des différentes agences.






- ❏  **DevSecOps** : c'est une collaboration entre développeurs et équipes d'infrastructure qui intègre la sécurité directement dans le processus de développement logiciel pour automatiser et accélérer la création, le test, le déploiement et la maintenance des logiciels. le but est de créer des applications plus sûres, plus rapidement, grâce à l'automatisation et au travail collaboratif entre développeurs, sécurité et opérations.
- ❏  **Développement et Code (Git, Spring Boot, spring cloud)** : les développeurs écrivent le code source en utilisant Spring Boot et spring cloud (des frameworks Java). Git aussi est utilisé pour le contrôle de version du code, permettant de suivre les modifications et de collaborer.
- ❏  **Versionning (GitHub)** : Le code est “commité” (enregistré) et “poussé” (téléchargé) vers un référentiel GitHub qui sert de plateforme centrale pour le stockage et la collaboration autour du code.
- ❏  **Jenkins CI/CD**<sup>1</sup> : Un webhook ou une action GitHub est fait pour déclencher un pipeline d'intégration continue/déploiement continu (CI/CD) dans Jenkins qui lui va automatiser les tests, la construction et le déploiement de l'application.
- ❏  **Infrastructure as Code** : Terraform est utilisé pour définir et provisionner l'infrastructure (serveurs, réseaux, etc.) sur le cloud de manière automatisée afin de créer et de gérer facilement l'infrastructure nécessaire pour chaque site de l'entreprise.
- ❏  **Configuration as Code (Ansible)** : Ansible est utilisé pour configurer les serveurs et les applications de manière automatisée afin de garantir une configuration cohérente sur tous les sites de l'entreprise.
- ❏  **Orchestration (Kubernetes)** : Kubernetes orchestre le déploiement, la mise à l'échelle et la gestion des conteneurs Docker. Il assure aussi une haute disponibilité et une répartition de la charge sur les différents sites.
- ❏  **Conteneurisation (Docker)** : Docker est utilisé pour emballer chaque micro service dans un conteneur pour garantir sa portabilité et son exécution sur n'importe quel machine.

---







1. CI/CD : intégration continue / déploiement continue






### 2.1.2 outils utilisés

Les outils utilisés désignent l'ensemble des technologies, logiciels, frameworks ou plateformes qui seront employés pour concevoir, développer, tester, déployer et maintenir le système ou l'application.

	role	outil	description
	développement	Spring boot	Créer les micro-services ( la gestion du personnel,API,...)de façon rapide et modulaire
	développement	Spring Cloud	Gérer les fonctionnalités de configuration centralisée, le service discovery, la gestion des appels inter-services
	base de données	PostgreSQL	Stocke les données RH dans une base relationnelle de façon sécurisée et structurée
	Contrôle de version ou outils de visionnage	git	Suivi des versions du code, gestion des branches ou gestion de collaboration entre les développeurs
	référentiel	GitHub	Plateforme pour hébergement du code source avec gestion des dépôts, branches,pull requests, intégration avec Jenkins



	role	outil	description
	Intégration et déploiement continus (CI/CD)	Jenkins	Automatiser les tâches comme le build, les tests la génération des artefacts et le déploiement continu des micro services
	IAC( Infra-structures As Code)	terraform	Déployer automatiquement l'infrastructure (IaC)en local ou sur le cloud
	CAC (Configuration As Code)	Ansible	Automatiser la configuration des serveurs
	orchestration	Kubernetes	Orchestrer et gérer le déploiement,la scalabilité, la résilience et la communication des conteneurs Docker
	Conteneurisation	Docker	Conteneuriser les services pour les rendre portables
	serveur web	nginx	Servir de reverse proxy, gérer le trafic HTTP/https,

	role	outil	description
	centraliser la sécurité	keycloak	serveur d'authentification et d'autorisation pour la gestion des permissions, la génération des tokens et la sécurité globale
	proxy	spring-gateway	centraliser le routage et fournir un unique point d'entrée à l'application
	Debugage	zipkin	Surveillance et traçabilité : Suivre le parcours d'une requête entre les microservices( latence, erreurs, performances et dépendance)
	centraliser les configuration	spring-config-server	faciliter la configuration et la maintenance des micro-services
	monitoring (surveillance)	spring-admin-server	seveur pour monitoring et la maintenance des micro-services

## 2.2 Présentation de l'application

la composition du système ainsi que la communication des modules est la suivante :

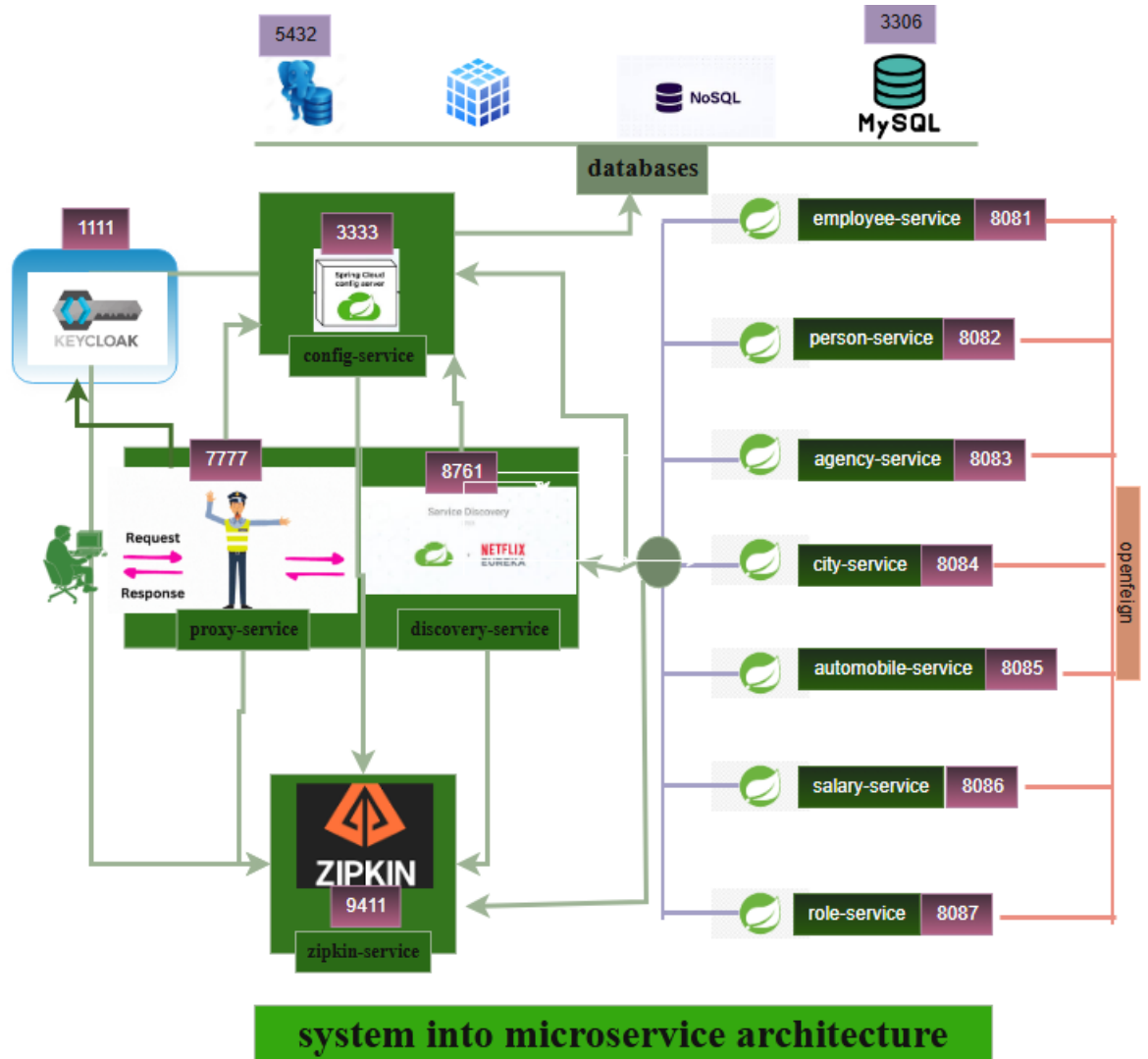


FIGURE 2 – system into microservice architecture

## 2.3 description de la base de données

on voit ici la description de nos entités et l'interopérabilité entre elles ainsi que la définition des structures de stockage

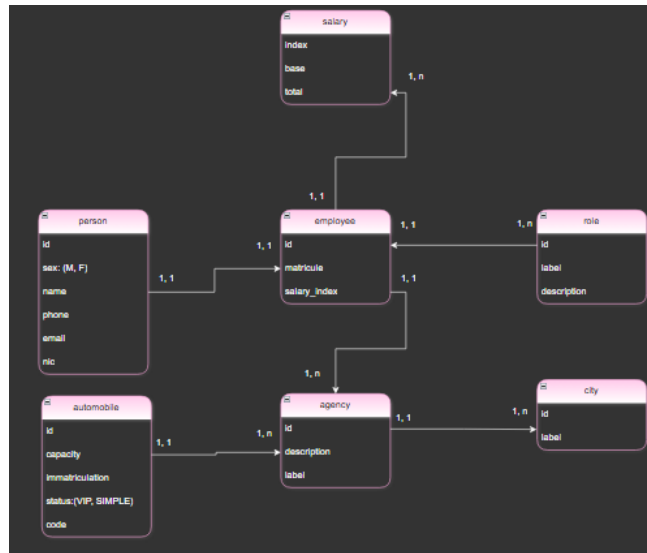


FIGURE 3 – base de donnée

## 2.4 implementation du serveur d'authentification et d'autorisation (Keycloak)

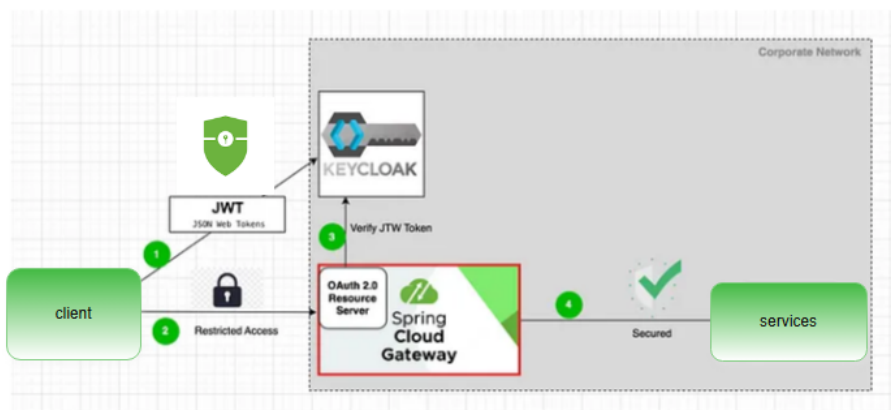
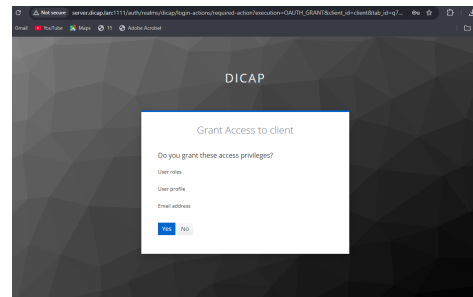
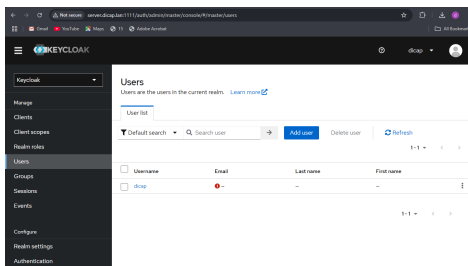
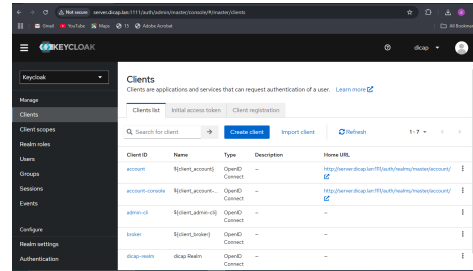
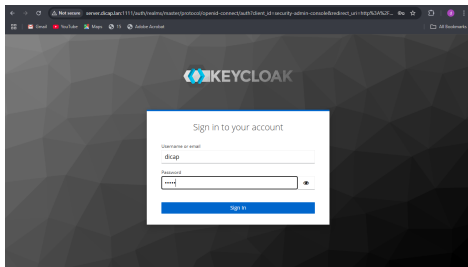


FIGURE 4 – fonctionnement du serveur d'authentification et d'autorisation



## 2.5 implémentation du serveur de découverte de service (eureka)

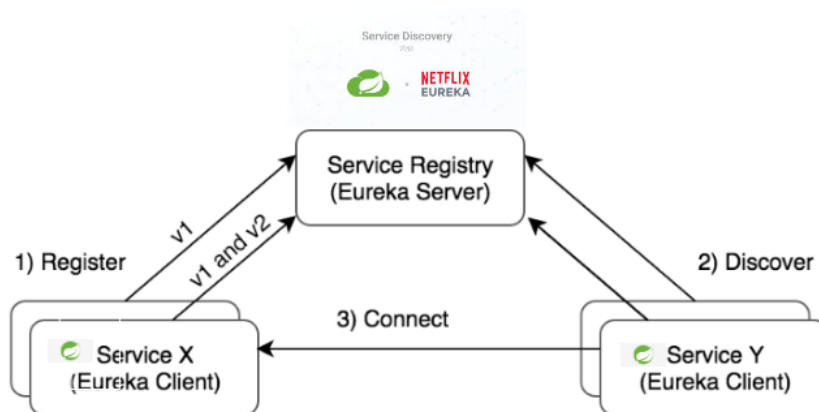


FIGURE 7 – fonctionnement du serveur eureka

← → ↺

Not secure server.dicap.lan:8761

☆

🔖

📄

👤

☰

📧 Gmail


📺 YouTube

🗺️ Maps

🕒 15

📄 Adobe Acrobat

📁 All Bookmarks

 **spring** Eureka

HOME LAST 1000 SINCE STARTUP

## System Status

Environment	test	Current time	2025-04-30T04:46:41 -0700
Data center	default	Uptime	00:11
		Lease expiration enabled	true
		Renews threshold	18
		Renews (last min)	40

## DS Replicas

[localhost](#)

Instances currently registered with Eureka

## 2.6 implémentation du proxy (gateway)

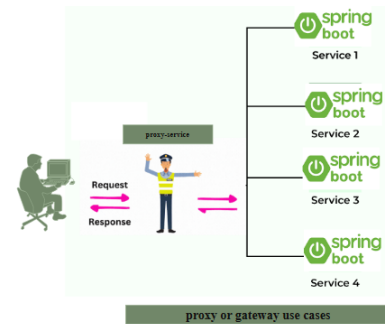


FIGURE 8 – fonctionnement d'une gateway

## 2.7 implémentation du serveur de configuration (config-server)

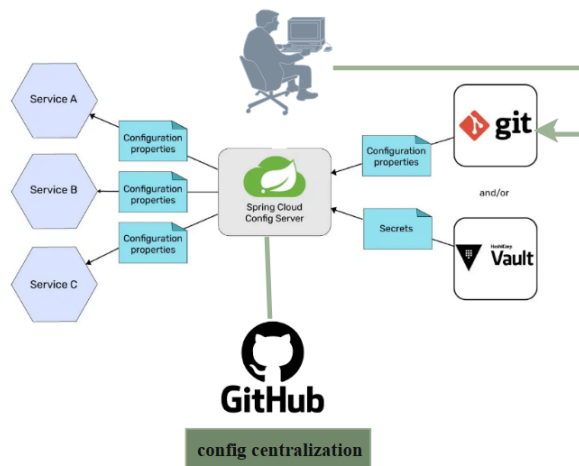


FIGURE 9 – fonctionnement du serveur de configuration

## 2.8 implémentation du serveur de configuration (config-server)

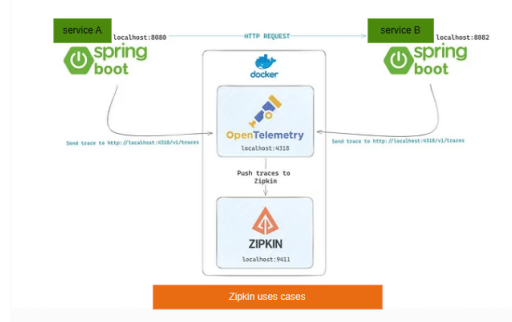
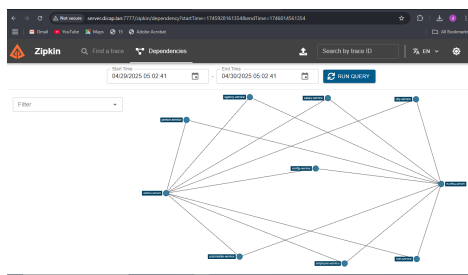
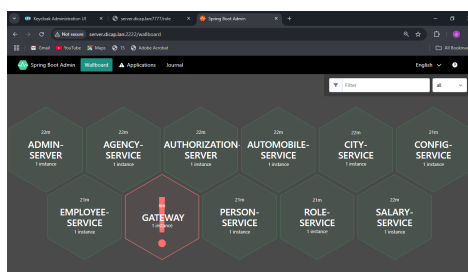


FIGURE 10 – fonctionnement du serveur de débogage ZipKin



Root	Start Time	Spans	Duration
salary-service: http get	a few seconds ago (04/03 05:04:45.976)	2	14.45ms
person-service: http put	a few seconds ago (04/03 05:04:46.973)	2	14.45ms
agency-service: http get	a few seconds ago (04/03 05:04:46.973)	2	14.45ms
city-service: http get	a few seconds ago (04/03 05:04:46.973)	2	14.45ms
role-service: http get	a few seconds ago (04/03 05:04:46.973)	2	14.45ms
employee-service: http get	a few seconds ago (04/03 05:04:46.973)	2	14.45ms
counta-server: http post	a few seconds ago (04/03 05:04:46.973)	1	14.45ms
counta-server: http post	a few seconds ago (04/03 05:04:46.973)	1	14.45ms
counta-server: http put	a few seconds ago (04/03 05:04:46.973)	1	14.45ms

## 2.9 implémentation du serveur d'administration (spring-admin-server)



Name	Status	Version
ADMIN-SERVICE	UP	1.0.0
AGENCY-SERVICE	UP	1.0.0
AUTHORIZATION-SERVICE	UP	1.0.0
AUTOMOBILE-SERVICE	UP	1.0.0
CITY-SERVICE	UP	1.0.0
CONFIG-SERVICE	UP	1.0.0
EMPLOYEE-SERVICE	UP	1.0.0
PERSON-SERVICE	UP	1.0.0
ROLE-SERVICE	UP	1.0.0
SALARY-SERVICE	UP	1.0.0



## 2.10 implémentation des micro-services

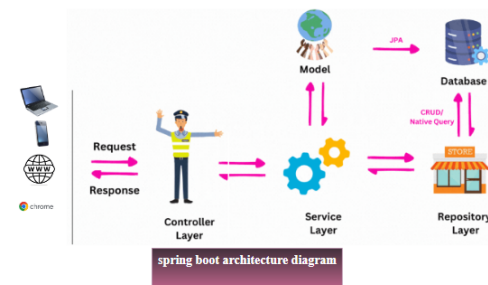


FIGURE 13 – fonctionnement d'un micro-service

### 2.10.1 implémentation du service City

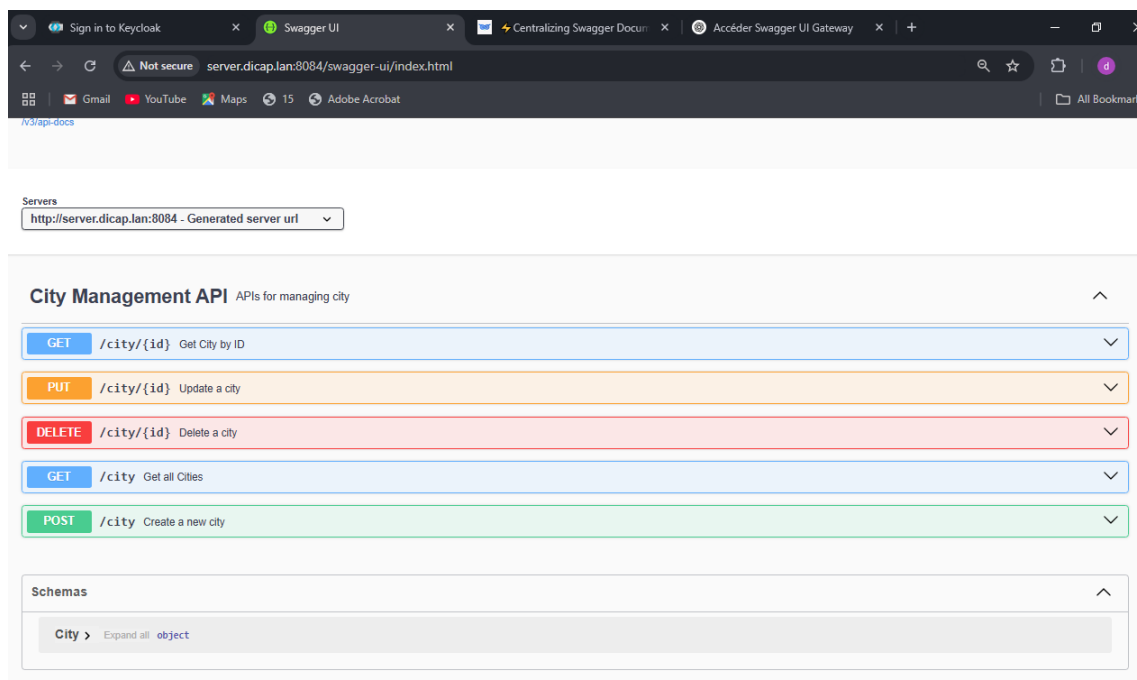


FIGURE 14 – city-service

## 2.10.2 implémentation du service agency

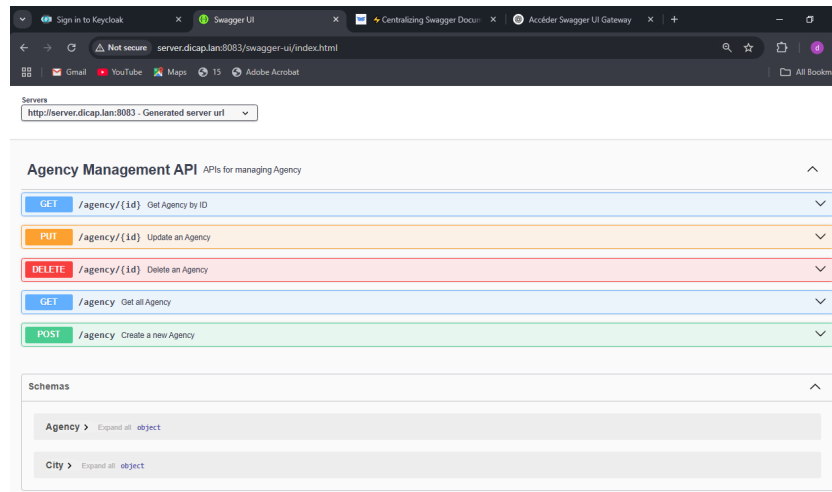


FIGURE 15 – agency-service

## 2.10.3 implémentation du service automobile

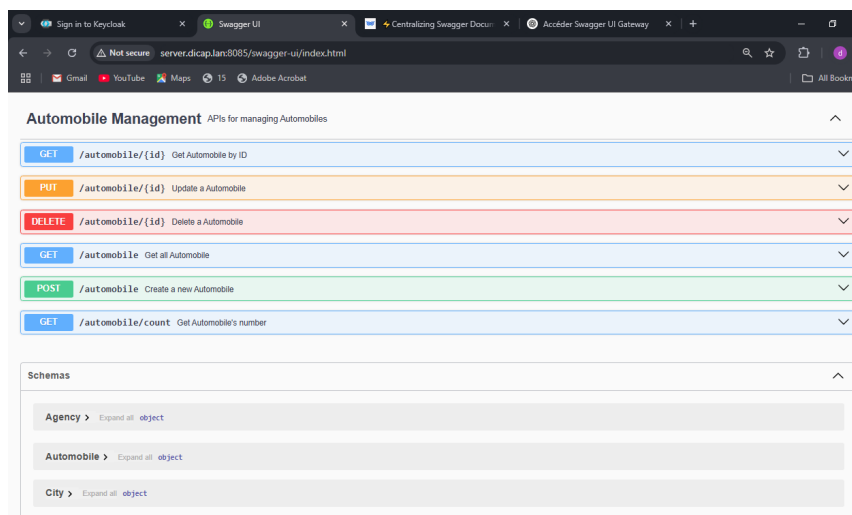


FIGURE 16 – automobile-service

## 2.10.4 implémentation du service role

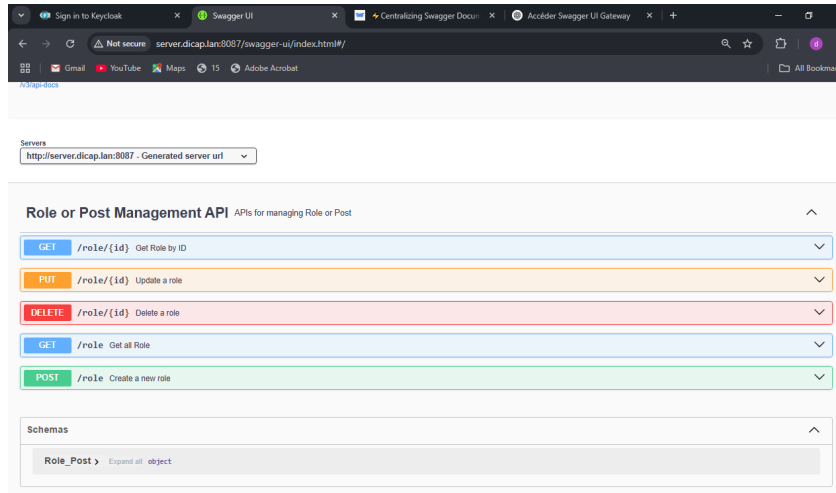


FIGURE 17 – role-service

## 2.10.5 implémentation du service salary

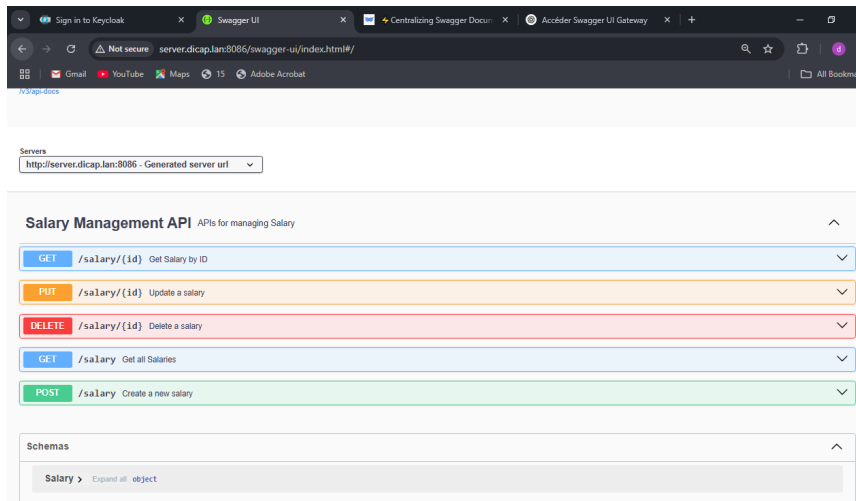


FIGURE 18 – salary-service

## 2.10.6 implémentation du service person

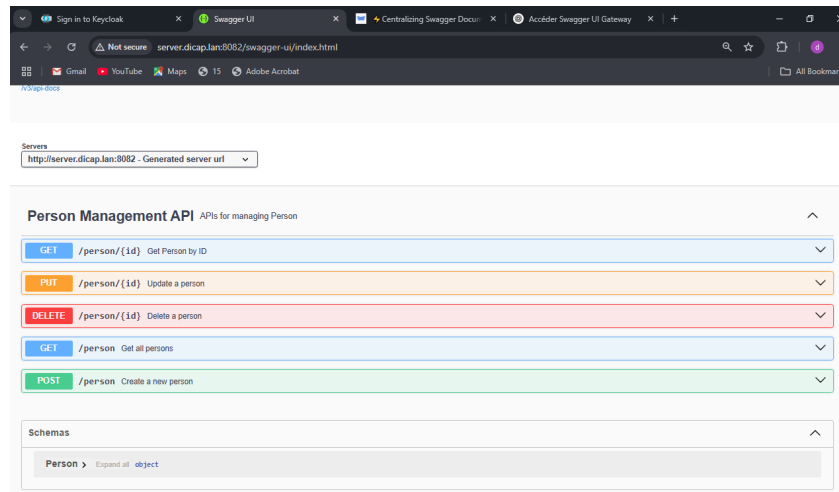


FIGURE 19 – person-service

## 2.10.7 implémentation du service employee

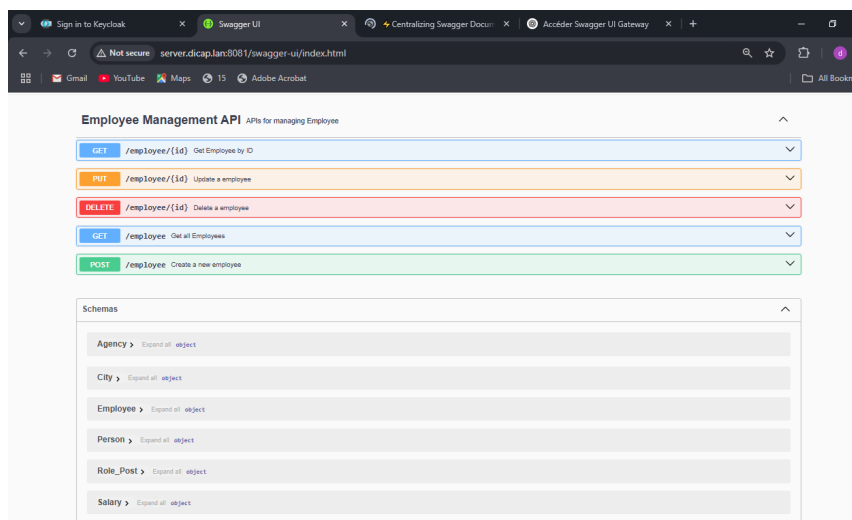


FIGURE 20 – employee-service

## 2.10.8 Communication des micro-services

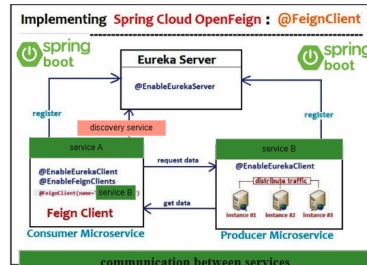


FIGURE 21 – Communication micro-services

## 2.11 implementation des pipelines Jenkins

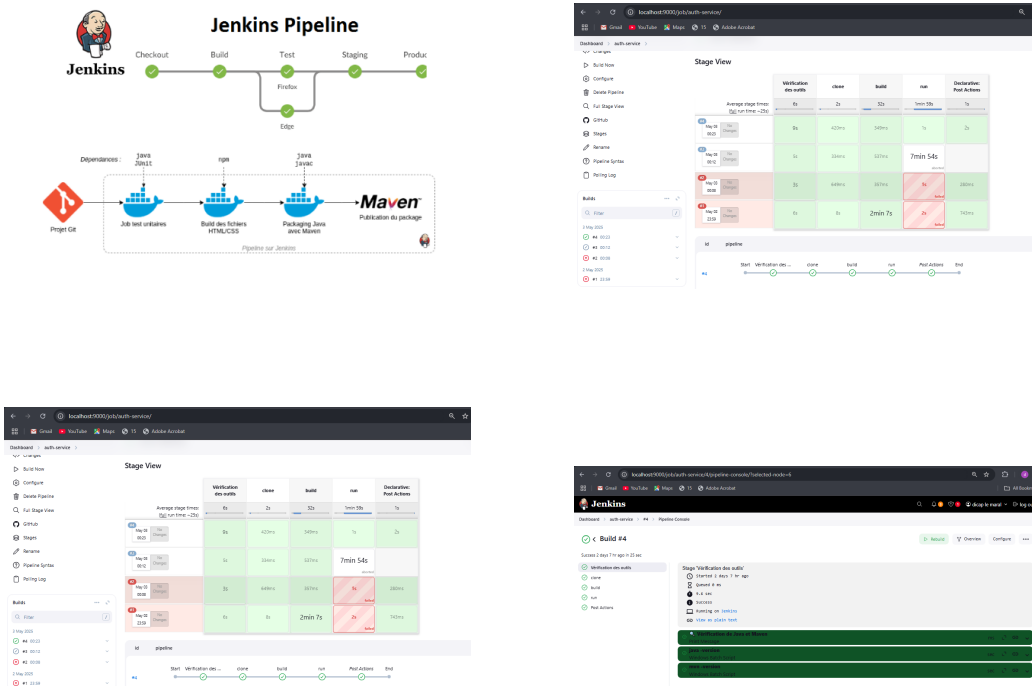


FIGURE 23 – pipeline Jenkins

## 2.12 implementation des déploiements sur le cloud



FIGURE 24 – Terraform management

NAME	STATUS	PROVIDER	REGION	DEPLOYED
admin-service	Deployed	Default	Oregon	3d
agency-service	Deployed	Default	Oregon	3d
auth-service	Deployed	Default	Oregon	3d
automobile-service	Deployed	Default	Oregon	3d
city-service	Deployed	Default	Oregon	3d
config-service	Deployed	Default	Oregon	3d
discovery-service	Deployed	Default	Oregon	3d
employee-service	Deployed	Default	Oregon	3d

(a) Déploiement sur le cloud

NAME	UNIT	VALUE	PROVIDER	REGION	DEPLOYED
admin-service	Default	1270	Oregon	3d	Deployed
agency-service	Default	1270	Oregon	3d	Deployed
auth-service	Default	1270	Oregon	3d	Deployed
automobile-service	Default	1270	Oregon	3d	Deployed
city-service	Default	1270	Oregon	3d	Deployed
config-service	Default	1270	Oregon	3d	Deployed
discovery-service	Default	1270	Oregon	3d	Deployed
employee-service	Default	1270	Oregon	3d	Deployed

(b) monitoring des services

### 3 référentiel github pour le code source de l'application

1. <https://github.com/cynticho/auth-service>
2. <https://github.com/cynticho/config-middleware-repoe>
3. <https://github.com/cynticho/agency-service>
4. <https://github.com/cynticho/person-service>
5. <https://github.com/cynticho/role-service>
6. <https://github.com/cynticho/config-service>
7. <https://github.com/cynticho/salary-service>
8. <https://github.com/cynticho/employee-service>
9. <https://github.com/cynticho/automobile-service>
10. <https://github.com/cynticho/admin-service>
11. <https://github.com/cynticho/gateway-service>
12. <https://github.com/cynticho/debug-service>
13. <https://github.com/cynticho/discovery-service>