République Tunisienne
Ministère de l'Enseignement Supérieur
Université de Tunis El Manar
Institut Supérieur d'Informatique

Rapport de Projet

Conception de Systèmes Complexes

Système de Gestion et d'Optimisation de Livraison

Réalisé par

Chiheb ELLEFI
Ibtihel Khmili
Rayen Fehri
Chadha Ammar

Année Universitaire : 2025–2026

Cours : Conception de Systèmes Complexes
Niveau : 2ème Année Ingénieur

Table des matières

1 Introduction générale

1.1 Contexte de la logistique moderne

L'industrie de la logistique et de la livraison connaît une croissance exponentielle, alimentée par l'essor du commerce électronique et les attentes croissantes des consommateurs en matière de rapidité et d'efficacité. Les entreprises de livraison font face à des défis majeurs : optimisation des itinéraires, affectation intelligente des chauffeurs, gestion des contraintes de temps et de capacité, tout en minimisant les coûts opérationnels.

Dans ce contexte, l'automatisation et l'intelligence artificielle jouent un rôle crucial pour améliorer la performance opérationnelle. Les algorithmes d'optimisation permettent de réduire les distances parcourues, d'équilibrer la charge de travail entre les chauffeurs, et de respecter les fenêtres de temps de livraison.

1.2 Objectif du projet

Ce projet vise à concevoir et développer un système complet de gestion et d'optimisation de livraison basé sur une architecture microservices moderne et déployé sur une infrastructure cloud scalable. Le système permettra aux entreprises de livraison d'automatiser l'affectation des chauffeurs, de calculer les itinéraires optimaux, et de suivre les livraisons en temps réel.

1.3 Problématique

Comment développer un système automatisé capable d'optimiser les livraisons en temps réel, tout en garantissant la scalabilité, la fiabilité et une expérience utilisateur fluide pour les gestionnaires, chauffeurs et clients, en utilisant une infrastructure cloud moderne ?

2 Description du projet

2.1 Contexte

Le système de gestion de livraison vise à résoudre les problèmes suivants :

— Affectation manuelle des livraisons aux chauffeurs, source d'inefficacité
— Routes non optimisées entraînant des coûts de carburant élevés
— Manque de visibilité en temps réel sur la localisation des chauffeurs
— Déséquilibre de la charge de travail entre les chauffeurs
— Difficulté à respecter les fenêtres de temps de livraison
— Infrastructure on-premise coûteuse et difficile à maintenir

2.2 Objectifs

Le système proposé vise à atteindre les objectifs suivants :

— Automatiser l'affectation intelligente des livraisons aux chauffeurs
— Calculer les itinéraires optimaux en minimisant la distance et le temps
— Assurer le suivi en temps réel de la position des chauffeurs
— Équilibrer la charge de travail entre les chauffeurs disponibles
— Respecter les contraintes de capacité des véhicules et les fenêtres horaires
— Fournir des analyses et rapports de performance
— Déployer sur une infrastructure cloud évolutive et résiliente

2.3 Portée du projet

2.3.1 Version MVP (Minimum Viable Product)

— Authentification et gestion des utilisateurs
— Gestion des véhicules et chauffeurs
— Affectation intelligente des livraisons
— Calcul d'itinéraires optimaux
— Applications web pour clients et gestionnaires
— Application mobile pour chauffeurs
— Déploiement sur AWS avec Docker Compose

2.3.2 Améliorations futures

— Suivi en temps réel avec WebSocket
— Tableau de bord analytique avancé avec Machine Learning
— Communication asynchrone complète avec RabbitMQ
— Déploiement Kubernetes (EKS) avec auto-scaling
— Système de notifications multi-canaux (SMS, Email, Push)
— Analyse prédictive des délais de livraison
— Intégration avec systèmes ERP externes

3 Conception

3.1 Spécification des besoins

3.1.1 Identification des acteurs

— **Gestionnaire (Manager)** : Gère les chauffeurs et véhicules, crée et assigne les livraisons, génère les plannings, consulte les tableaux de bord via l'application web Angular.
— **Chauffeur** : Consulte les livraisons assignées, accède aux itinéraires, met à jour le statut, partage sa position en temps réel via l'application mobile (React Native/Flutter/Native).
— **Client (Vendeur)** : Crée des demandes de livraison, consulte l'historique, suit le statut des commandes, gère ses préférences via le portail web Angular.
— **Administrateur** : Gère tous les utilisateurs, configure les paramètres système, accède aux métriques d'infrastructure, gère les clés API et les services cloud.

3.1.2 Besoins fonctionnels

— **Gestion des livraisons** : CRUD des livraisons, adresses, fenêtres horaires, suivi statut, historique.
— **Optimisation et affectation** : Affectation automatique basée sur algorithmes, calcul d'itinéraires optimaux, gestion des contraintes de capacité, équilibrage de charge, re-optimisation dynamique.
— **Suivi et tracking** : GPS temps réel, calcul ETA dynamique, géofencing, historique des déplacements.
— **Analytics et reporting** : Tableaux de bord interactifs, KPI en temps réel, rapports de performance, analyse d'efficacité opérationnelle.
— **Applications utilisateur** :

— Portail web client (Angular)
— Application web gestionnaire (Angular)
— Application mobile chauffeur (React Native/Flutter/Kotlin-Swift)

3.1.3 Besoins non fonctionnels

— **Scalabilité** : 300+ utilisateurs simultanés, 10 000+ livraisons/jour, auto-scaling horizontal via Kubernetes HPA.
— **Performance** : Calcul itinéraire < 2s, API REST < 500ms, tracking temps réel < 1s.
— **Fiabilité** : 99.9% disponibilité (SLA), gestion robuste des erreurs, retry automatique, tolérance aux pannes.
— **Sécurité** : TLS 1.3, JWT avec rotation, RBAC granulaire, chiffrement des données au repos et en transit, audit trail complet.
— **Ergonomie** : Interface intuitive et responsive, support multi-plateforme (web, mobile iOS/Android).
— **Cloud-native** : Déploiement sur AWS, utilisation de services managés, infrastructure as code.

3.1.4 Diagrammes de cas d'utilisation

La figure suivante présente une vue globale des interactions entre les acteurs et le système.

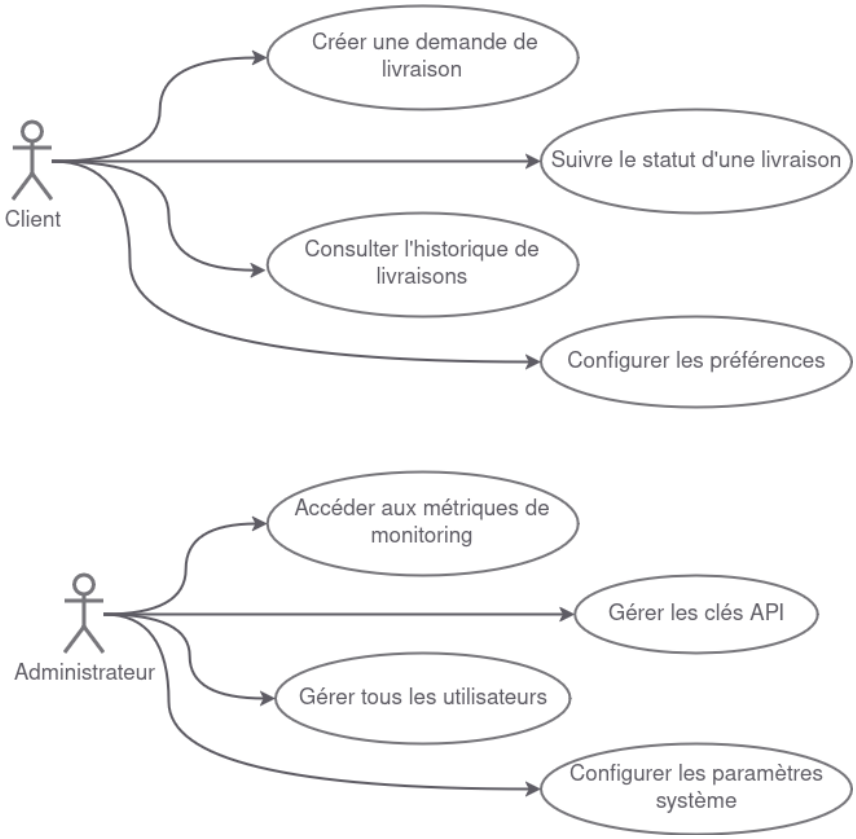


FIGURE 1 – Diagramme de cas d'utilisation du système

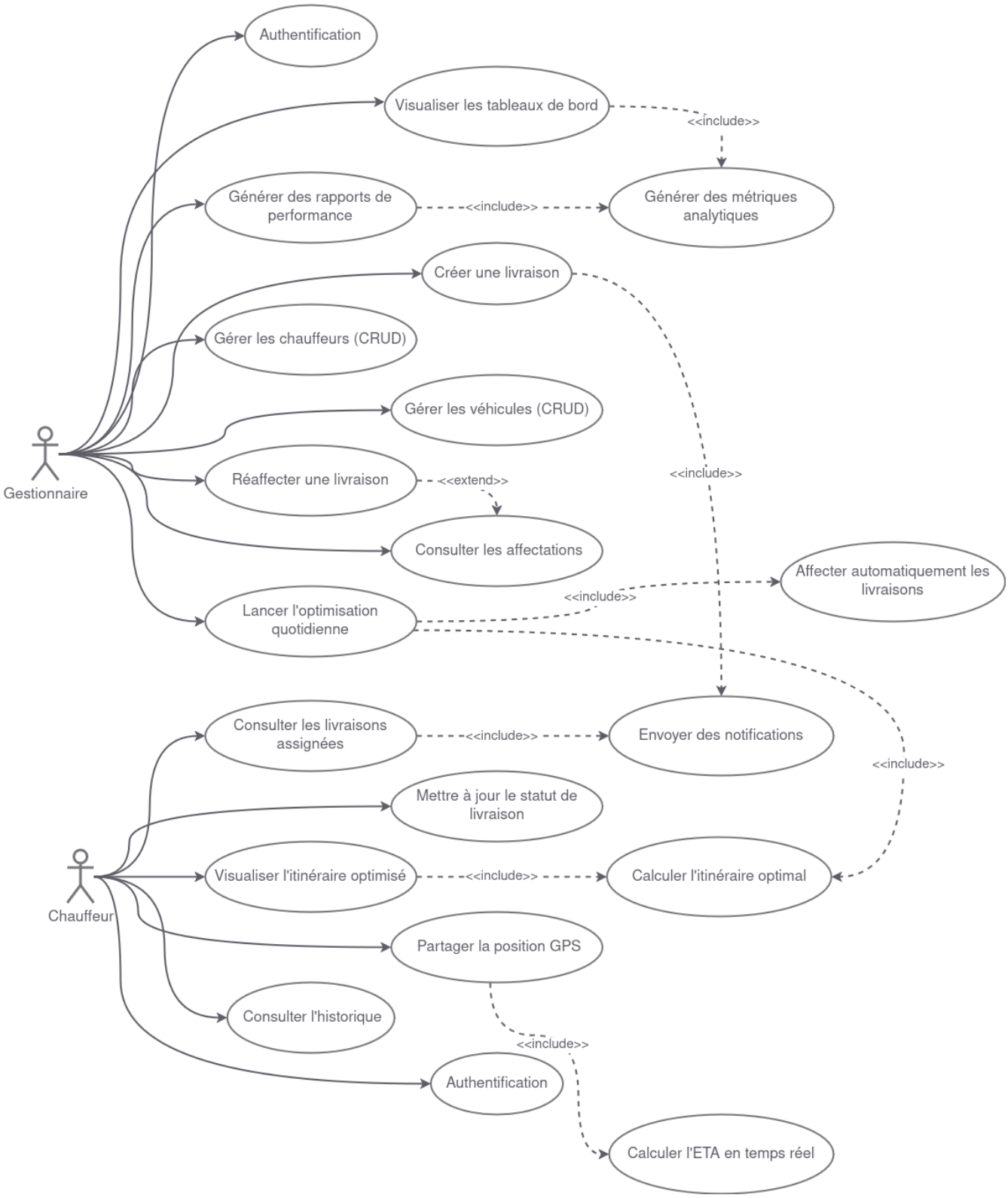


Figure 2 – Diagramme de cas d'utilisation du système

4 Architecture du système

4.1 Architecture de haut niveau

Le système adopte une architecture microservices cloud-native moderne pour assurer la scalabilité, la maintenabilité et la résilience. Chaque service est indépendant, conteneurisé avec Docker, déployable séparément sur Kubernetes (Amazon EKS), et communique via des API REST/gRPC et des événements asynchrones.

L'infrastructure est entièrement hébergée sur Amazon Web Services (AWS) pour garantir une haute disponibilité, une scalabilité élastique et une réduction des coûts opérationnels.

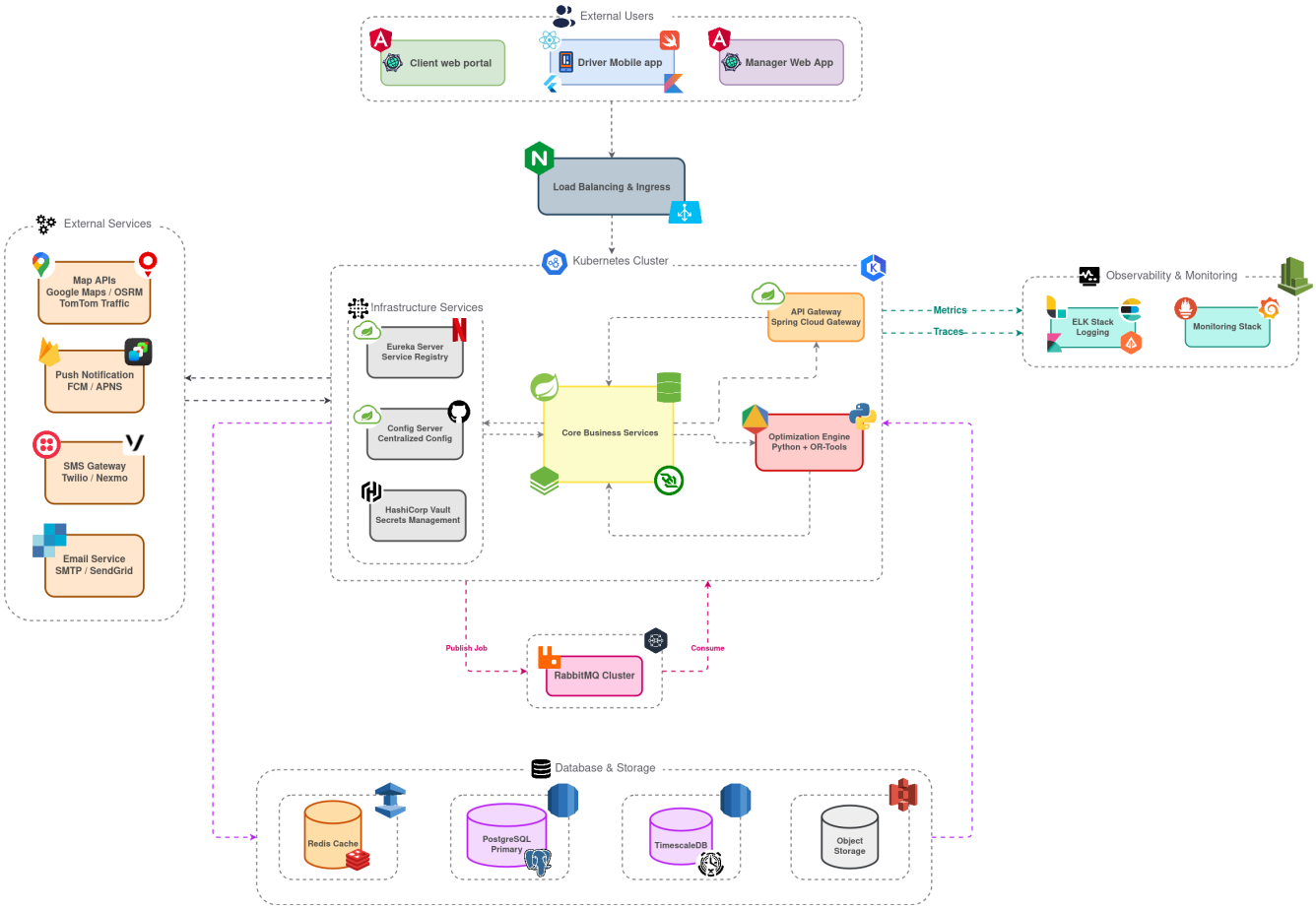


FIGURE 3 – Architecture globale du système

4.2 Composants principaux

- **API Gateway** : Point d'entrée unique (Spring Cloud Gateway)
- **Services métier** : Microservices applicatifs (Spring Boot)
- **Moteur d'optimisation** : Service Python avec OR-Tools de Google
- **Couche de données** : Amazon RDS PostgreSQL, TimescaleDB, Amazon ElastiCache Redis
- **Message Broker** : Amazon MQ (RabbitMQ managé)
- **Services d'infrastructure** : Authentification, découverte de services, configuration centralisée
- **Stack d'observabilité** : ELK Stack, Prometheus, Grafana, Zipkin, Amazon CloudWatch

— **Applications clientes** :
  — Portail web Angular (Client & Manager)
  — Application mobile (React Native, Flutter, ou Native Kotlin/Swift)

4.2.1 Services AWS utilisés

Service AWS	Icône	Utilisation	Justification
Amazon EKS		Orchestration Kubernetes des microservices	Scalabilité automatique, haute disponibilité, gestion simplifiée
Amazon RDS PostgreSQL		Base de données relationnelle principale	Service managé, backups automatiques, multi-AZ
Amazon ElastiCache Redis		Cache mémoire distribué	Haute performance, réplication automatique
Amazon MQ (RabbitMQ)		Message broker pour communication asynchrone	Service managé, haute disponibilité
Amazon S3		Stockage objets (logs, rapports, exports)	Durabilité 99.999999999%, versioning, lifecycle policies
Amazon CloudWatch		Monitoring, logs, métriques	Intégration native AWS, alertes automatiques

4.3 Décomposition des services

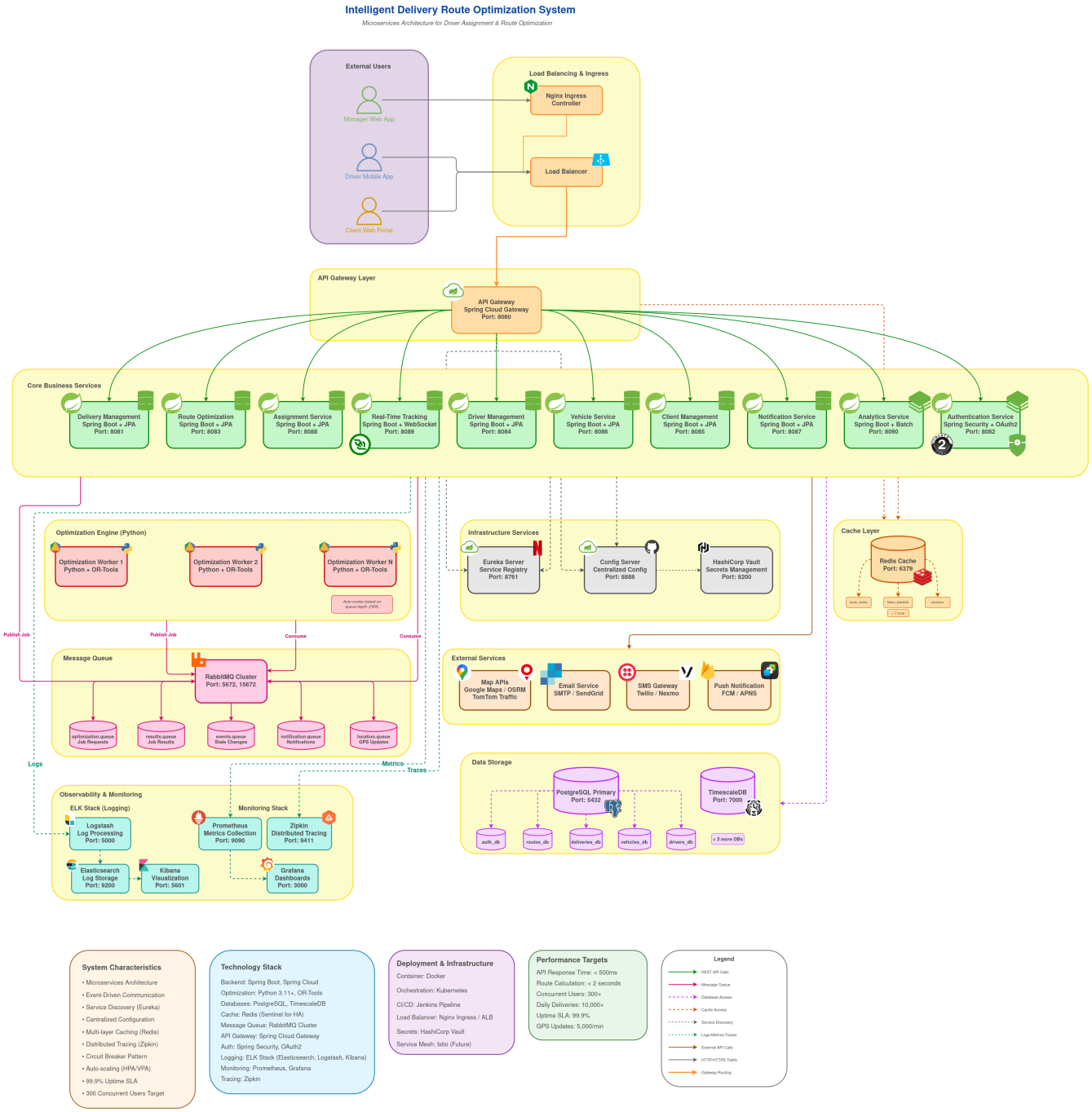


FIGURE 4 – Architecture logique simplifiée

4.3.1 Services métier principaux

— **Delivery Management Service** : Gestion du cycle de vie complet des livraisons (création, modification, suppression, consultation)

— **Route Optimization Service** : Orchestration de l'optimisation des itinéraires, interface avec le moteur Python OR-Tools

— **Assignment Service** : Gestion des affectations chauffeur-livraison, équilibrage de charge, gestion des contraintes

- **Real-Time Tracking Service** : Suivi GPS en temps réel, calcul ETA dynamique, géofencing
- **Driver Management Service** : Gestion des chauffeurs (CRUD, disponibilité, shifts, performance)
- **Vehicle Service** : Gestion de la flotte de véhicules (CRUD, maintenance, capacités, statut)
- **Client Management Service** : Gestion des clients vendeurs (CRUD, préférences, historique)
- **Notification Service** : Notifications multi-canaux
- **Analytics Service** : Génération d'insights, rapports de performance, tableaux de bord

4.3.2 Moteur d'optimisation

Optimization Worker Service (Python + OR-Tools)

- Exécution des algorithmes d'optimisation de routage (VRP - Vehicle Routing Problem)
- Gestion des contraintes complexes :
  - Fenêtres de temps de livraison
  - Capacités des véhicules (poids, volume)
  - Compétences requises des chauffeurs
  - Temps de service et de déplacement
  - Priorités des livraisons
- Technologies : Python 3.11+, OR-Tools, RabbitMQ

4.3.3 Services d'infrastructure

- **Authentication Service** : Gestion JWT, OAuth2
- **API Gateway** : Spring Cloud Gateway avec rate limiting et circuit breaker
- **Service Registry** : Eureka Server pour la découverte de services
- **Configuration Service** : Spring Cloud Config avec backend Git

4.4 Couche de données

- **PostgreSQL** :
  - Base relationnelle principale
- **TimescaleDB** :
  - Séries temporelles (positions GPS des chauffeurs)
- **Redis** :
  - Cache mémoire distribué
  - Session storage (tokens JWT, sessions utilisateurs)
  - Cache des itinéraires calculés

4.5 Communication inter-services

- **Amazon MQ (RabbitMQ)** :
  - Communication asynchrone entre microservices
- **REST API** :
  - Communication synchrone (environnement développement)
  - Documentation automatique avec Swagger/OpenAPI
- **gRPC** :
  - Communication synchrone haute performance (production)
  - Protocol Buffers pour sérialisation efficace

4.6 Stack d'observabilité

- **ELK Stack (sur EKS)** :
  - Elasticsearch : Indexation et recherche de logs
  - Logstash : Collecte et transformation de logs
  - Kibana : Visualisation et analyse
- **Prometheus + Grafana (sur EKS)** :
  - Prometheus : Collecte de métriques temps réel
  - Grafana : Dashboards interactifs
  - Alertmanager : Gestion des alertes
- **Zipkin (sur EKS)** :
  - Distributed tracing entre microservices
  - Analyse des latences et goulots d'étranglement
  - Visualisation des flux de requêtes

4.7 Services externes

- **APIs de cartographie** :
  - TomTom API : Calcul d'itinéraires, matrices de distance
  - Google Maps API : Géocodage, reverse geocoding
  - OSRM (Open Source Routing Machine) : Alternative open-source
- **Services de notifications** :
  - Amazon SES : Envoi d'emails transactionnels
  - Amazon SNS : SMS et notifications push
  - Twilio (optionnel) : SMS et appels vocaux

4.8 Applications clientes

4.8.1 Portail web Angular

Client Web Portal & Manager Web Application

- **Framework** : Angular
- **Fonctionnalités** :
  - Dashboard interactif avec KPI temps réel
  - Gestion des livraisons (CRUD, filtres, recherche)
  - Visualisation des itinéraires sur carte
  - Suivi temps réel des chauffeurs
  - Génération de rapports et exports
  - Gestion des utilisateurs et permissions (Manager uniquement)

4.8.2 Application mobile chauffeur

Options technologiques :

Option 1 : React Native

- **Avantages** : Code partagé iOS/Android, large communauté, performance native

Option 2 : Flutter

- **Avantages** : Performance excellente, UI cohérente, hot reload rapide

Option 3 : Native (Kotlin + Swift)

- **Android** : Kotlin, Jetpack Compose, Coroutines, Room
- **iOS** : Swift, SwiftUI, Combine, CoreData
- **Avantages** : Performance maximale, accès complet aux APIs natives
- **Inconvénient** : Développement et maintenance séparés

Fonctionnalités communes :

- Authentification biométrique (Face ID/Touch ID)
- Liste des livraisons assignées du jour
- Navigation GPS turn-by-turn vers destinations
- Mise à jour statut livraison (en transit, livrée, échec)
- Scan de codes-barres/QR codes
- Signature électronique du client
- Photo de preuve de livraison
- Partage automatique de position GPS en arrière-plan
- Notifications push en temps réel
- Mode offline avec synchronisation

5 Réalisation

5.1 Technologies utilisées

5.1.1 Backend – Services Spring Boot

Technologie	Logo	Rôle
Spring Boot		Framework principal pour les microservices
Spring Cloud Gateway		API Gateway avec routage dynamique
Spring Security + OAuth2		Authentification et autorisation
Spring Data JPA		Accès aux données PostgreSQL
Spring WebSocket		Communication temps réel
Spring Cloud Config		Configuration centralisée des microservices
Spring Cloud Netflix Eureka		Service de découverte des microservices

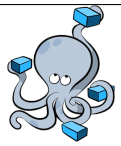
5.1.2 Moteur d'optimisation

Technologie	Logo	Rôle
Python		Langage principal du moteur d'optimisation
OR-Tools		Bibliothèque d'optimisation de Google

5.1.3 Bases de données

Technologie	Logo	Rôle
PostgreSQL		Base de données relationnelle principale
TimescaleDB		Extension pour séries temporelles
Redis		Cache et session store

5.1.4 Infrastructure

Technologie	Logo	Rôle
Docker		Conteneurisation des services
Docker Compose		Orchestration locale (développement)
Kubernetes		Orchestration en production

5.1.5 Messaging et Communication inter-services

Technologie	Logo	Rôle
RabbitMQ		Message broker pour communication asynchrone
REST (Spring Web)		Communication synchrone via API HTTP
gRPC		Communication synchrone haute performance entre microservices

5.1.6 Monitoring & Logging

Technologie	Logo	Rôle
ELK Stack		Centralisation et visualisation des logs
Prometheus		Collecte de métriques
Grafana		Dashboards et visualisation
Zipkin		Distributed tracing

6 Conclusion

Ce projet de système de gestion et d'optimisation de livraison représente une solution moderne et complète pour les entreprises de logistique. En adoptant une architecture microservices, nous garantissons la scalabilité, la maintenabilité et la résilience du système.

Les points forts du système incluent :

— Architecture microservices découplée et scalable

— Optimisation intelligente avec OR-Tools de Google

— Suivi en temps réel des chauffeurs et livraisons

— Observabilité complète (logs, métriques, tracing)

— Sécurité robuste avec JWT et OAuth2

— Pipeline CI/CD automatisé

— Haute disponibilité et tolérance aux pannes

La version MVP fournit les fonctionnalités essentielles pour démarrer les opérations, tandis que les améliorations futures permettront d'étendre les capacités du système avec des fonctionnalités avancées comme l'application mobile pour chauffeurs, les notifications multi-canaux, et les analyses prédictives.

Ce système est conçu pour évoluer avec les besoins de l'entreprise et peut supporter une croissance significative grâce à son architecture scalable et ses capacités d'auto-scaling.

A Glossaire

Microservices Architecture où une application est composée de petits services indépendants, chacun exécutant un processus unique.

OR-Tools Bibliothèque open-source d'optimisation développée par Google pour résoudre des problèmes de routage, d'ordonnancement, etc.

API Gateway Point d'entrée unique qui gère les requêtes vers les microservices.

Circuit Breaker Patron de conception pour prévenir les cascades de défaillances dans les systèmes distribués.

gRPC Framework RPC haute performance développé par Google utilisant HTTP/2 et Protocol Buffers.

TimescaleDB Extension de PostgreSQL optimisée pour les séries temporelles.

RabbitMQ Message broker open-source implémentant le protocole AMQP.

JWT JSON Web Token, standard pour les tokens d'authentification.

RBAC Role-Based Access Control, contrôle d'accès basé sur les rôles.

ELK Stack Elasticsearch, Logstash, Kibana - suite pour la gestion et visualisation de logs.

Prometheus Système de monitoring et d'alerte open-source.

Grafana Plateforme de visualisation et d'analyse de métriques.

Zipkin Système de traçage distribué open-source.

Docker Plateforme de conteneurisation.

Kubernetes Orchestrateur de conteneurs open-source.

CI/CD Continuous Integration/Continuous Deployment.