YOUR CAR YOUR WAY

SOMMAIRE

Objet du document

Ce document décrit l’architecture de l’application "YourCarYourWay", en détaillant les principes de conception, les composants du système, les processus de déploiement et les justifications pour l’approche choisie.

Il offre une vue d’ensemble complète de la structure globale du système ainsi que des détails opérationnels de son pipeline de déploiement.

L’objectif est d’assurer une cohérence entre les objectifs métier et la mise en œuvre technique.

Objectifs du projet

Les objectifs du projet sont :

- Expérience fluide pour la location de véhicules :
    - Fournir une plateforme intuitive permettant aux utilisateurs de rechercher, réserver et gérer leurs locations de véhicules.
    - Permettre le retrait et le retour des véhicules entre différentes agences pour plus de flexibilité.
- Architecture microservices évolutive :
    - Implémenter un système basé sur des microservices pour garantir la modularité, l'évolutivité et des déploiements indépendants.
- Pipeline de déploiement robuste :
    - Exploiter des pipelines de build et de déploiement automatisés pour garantir l'intégration et la livraison continues.
- Sécurité et surveillance :
    - Gérer de manière sécurisée les données sensibles (par ex. : détails de paiement, informations utilisateur) en utilisant les services AWS.
    - Surveiller les performances et les journaux du système pour assurer une excellence opérationnelle.

Principes d’architecture

L’architecture de l’application "YourCarYourWay" repose sur un modèle en microservices structuré selon les principes de **Domain-Driven Design** (DDD), garantissant modularité, évolutivité et indépendance des composants.

Elle est également basée sur une communication événementielle pour gérer les interactions complexes et asynchrones.

Séparation en couches fonctionnelles
L’architecture est organisée en trois couches principales :

- **Couche client** : Hébergée sur Amazon CloudFront, elle inclut les fichiers statiques (frontend Angular 17) stockés dans Amazon S3 pour une distribution rapide et efficace. Les interactions utilisateur (via API ou WebSocket) sont sécurisées et routées via un Application Load Balancer (ALB).

- **Couche backend** : Implémentée en microservices (SpringBoot 3 - Java 21) conteneurisés sur Amazon ECS, chaque microservice gère un domaine fonctionnel spécifique (par ex. : gestion des utilisateurs, réservations, paiements, support). Les communications synchrones sont exposées via des API REST, tandis que les événements métier asynchrones sont orchestrés via Amazon EventBridge.

- **Couche données** : Les informations sensibles et métier (utilisateurs, véhicules, réservations) sont stockées dans une base relationnelle PostgreSQL hébergée sur Amazon RDS, avec des principes stricts de séparation et d’isolation des données par microservice.

Communication événementielle
Les interactions complexes entre microservices sont gérées par Amazon EventBridge :

Les événements métier (ex. : confirmation de paiement, mise à jour de réservation, notification de support) sont publiés et consommés de manière asynchrone pour minimiser les dépendances directes entre microservices.

Cette approche favorise la résilience du système et permet une extension facile pour de nouveaux cas d’usage.

Architectures

L'architecture respecte le Domain-Driven-Design en séparant les responsabilités : chaque domaine métier (Gestions des utilisateurs, Support, Paiements et Locations) est géré par un microservice.

Diagramme de composants

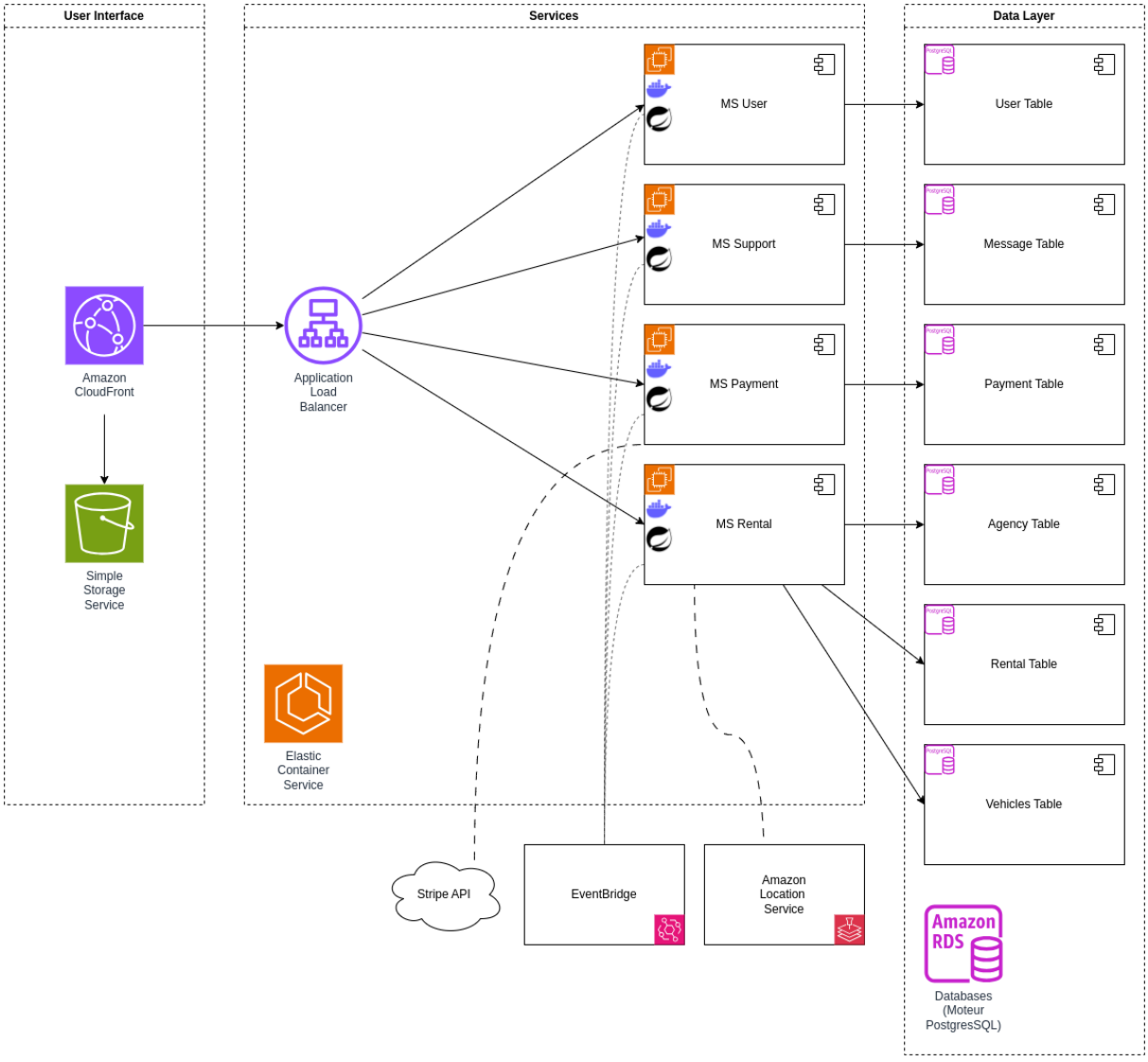


Diagramme de déploiement

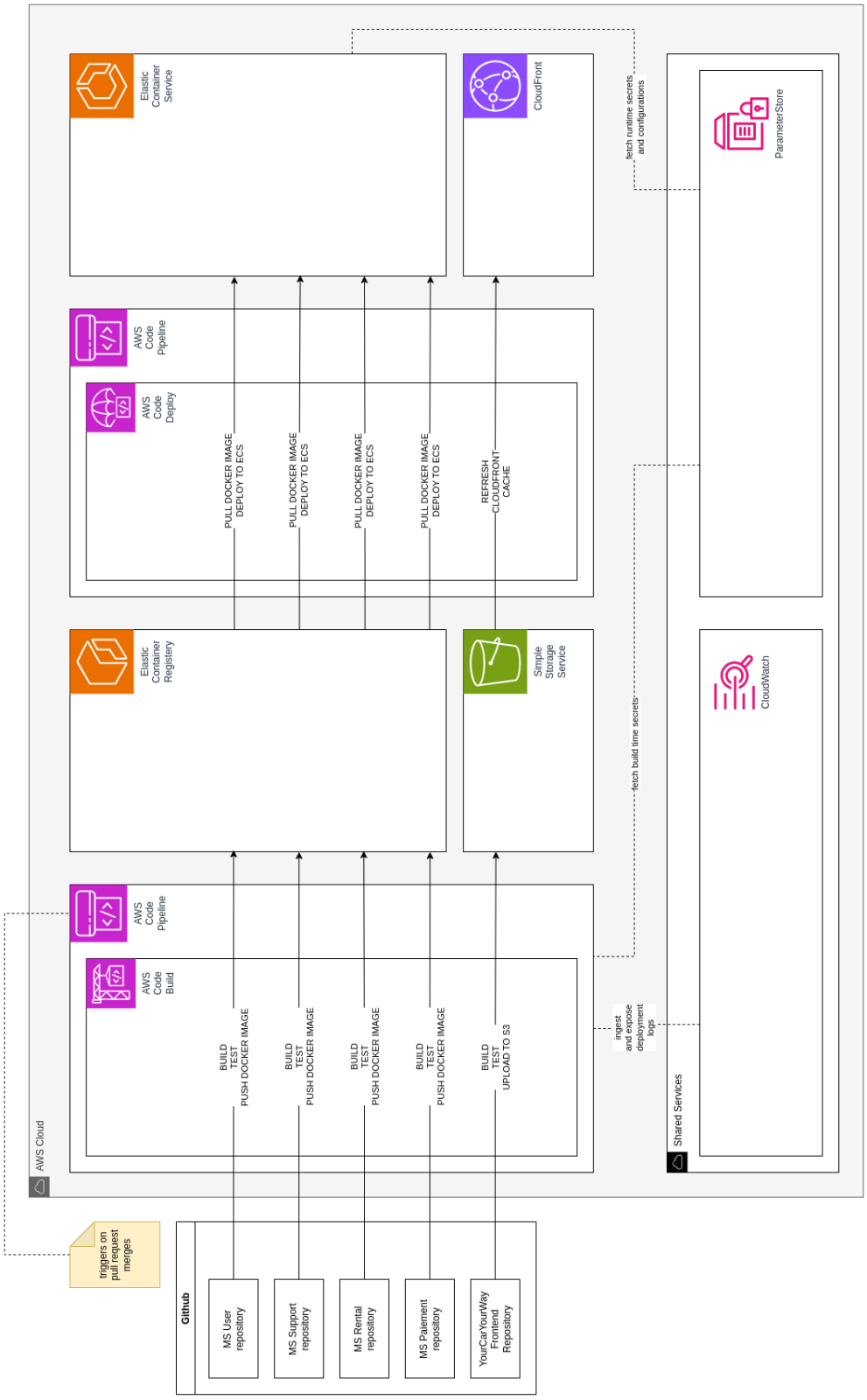
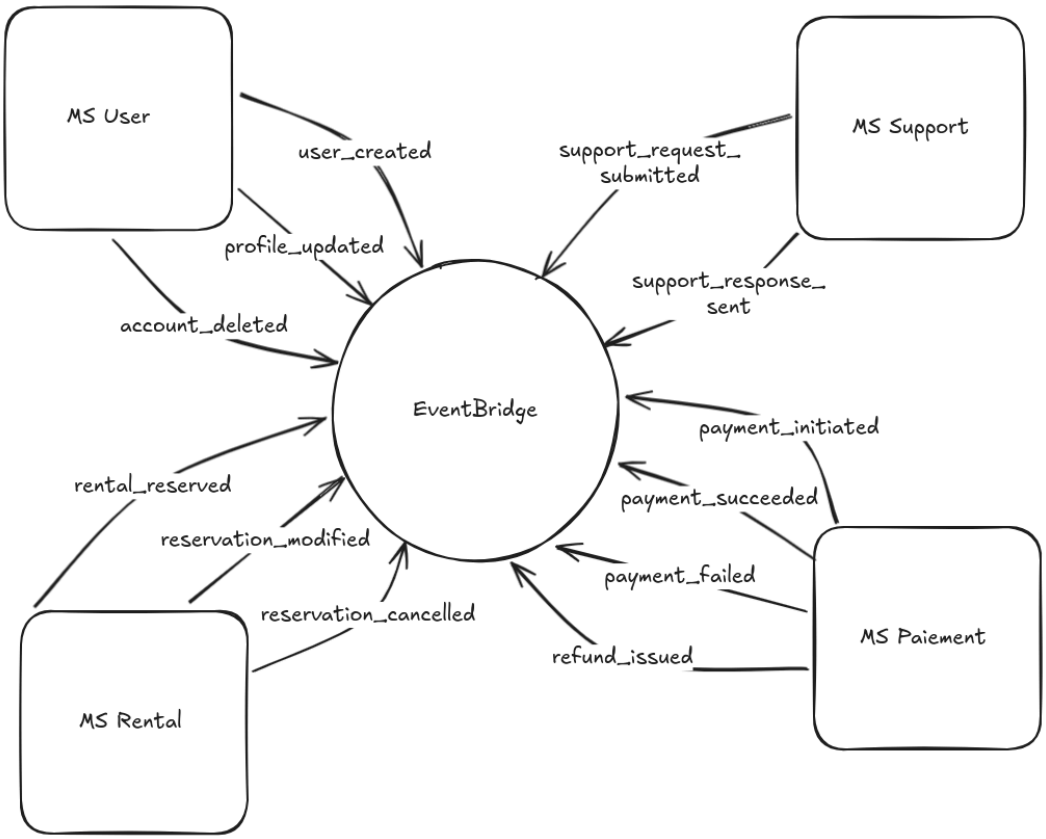


Diagramme Entité Relation

Event storming: définition des événements par domaine

Ce diagramme représente les différents événements nécéssaires à la communication entre les services.

Etant donné que ce document couvre uniquement l'application à destination des clients, les événements liés à l'usage des applications de gestion des employés des Agences de YourCarYourWay (ex: événements de notification du retour d'un véhicule dans l'agence) ne sont pas représentés dans ce diagramme.



Affichage des offres de location de véhicule

L’affichage des offres de location de véhicule est régi par un ensemble de règles métier géré par le **microservice rental**.

Rappelons le cas d’utilisation de la feature du point de vue de l’utilisateur (client)

L’utilisateur saisit les critères de recherche suivants :

- Agence de départ (ou ville de départ).
- Agence de retour (ou ville de retour, si différente).
- Date et heure de début de location.
- Date et heure de retour.
- Catégorie du véhicule

Filtrage dynamique :

L’utilisateur peut affiner les résultats via des filtres (par exemple : prix, type de véhicule).

Après validation de la recherche, l’application affiche une liste des véhicules disponibles qui répondent aux critères. Chaque offre comprend :

- La catégorie du véhicule (par exemple : berline, utilitaire).
- Le tarif total, comprenant :
    - Le coût journalier.
    - Les frais additionnels (transfert inter-agences).

Le **microservice de location** utilise trois tables SQL (agences, véhicules, réservations) pour déterminer la disponibilité des véhicules.

Des jointures entre ces tables permettent de vérifier les critères utilisateur :

- les agences de départ et de retour sont associées via leurs identifiants

- les véhicules disponibles associés à l'agence de départ sont filtrés en excluant ceux déjà réservés pour la période demandée

- si l'agence de retour est différente de l'agence de départ, les véhicules disponibles associés à l'agence de départ sont filtrés en excluant ceux déjà réservés

Le prix pour une offre de location est calculé à partir de la catégorie du véhicule et de la durée de la location.

Les frais additionnels pour une réservation avec un transfert inter-agences sont calculés à l'aide d'une requête sur Amazon Location Service, déterminant la distance exacte entre l'agence de départ et l'agence de retour.

Cette distance est utilisée pour définir les coûts de transfert en fonction des tarifs prédéfinis par kilomètre.

Le **microservice Rental** utilise ces données pour calculer dynamiquement le tarif total de la réservation et afficher les frais supplémentaires à l'utilisateur avant la validation.

Service Support

Le **service support** de l'application "YourCarYourWay" utilise Amazon Application Load Balancer (ALB) pour gérer les connexions WebSocket en temps réel entre les utilisateurs et le microservice Support.

La configuration SSL/TLS de l'ALB assure une sécurisation des échanges via des connexions `wss://`.

Enfin, un délai d'inactivité optimisé est configuré (par exemple 300 secondes) pour s'adapter aux besoins de chat tout en limitant les ressources inutilisées.

Cette implémentation offre une scalabilité, une sécurité, et une gestion efficace des connexions WebSocket pour une expérience utilisateur fluide.

Service Utilisateur

Le **service Utilisateur** est responsable de la gestion des comptes et de l'authentification des utilisateurs. Il joue un rôle central dans l'architecture de l'application YourCarYourWay en fournissant les fonctionnalités suivantes :

Gestion des comptes utilisateur

- Création, mise à jour et suppression des comptes utilisateur.
- Gestion des informations personnelles comme le nom, l'email et le mot de passe.
- Le mot de passe est hashé avec Bcrypt.

Authentification et autorisation

- Génération de JSON Web Tokens (JWT) lors de l'authentification des utilisateurs.
- Les JWT contiennent des informations essentielles pour identifier l'utilisateur et gérer ses autorisations.
- Les autres microservices valident les JWT à l'aide d'une clé publique récupérée via le ParameterStore lors du build.

Ce service centralise la gestion des utilisateurs tout en assurant une sécurité robuste et une intégration fluide avec le reste de l'écosystème.

Interfaçage avec l’api Stripe

L’interfaçage avec Stripe est géré par le **microservice Paiement**, qui assure la sécurité et la conformité **PCI-DSS*** pour toutes les transactions financières. Aucune information de carte bancaire ne sera persisté dans nos bases, les sessions stripes seront rattachés à un identifiant utilisateur.

Les étapes sont :

- **Initiation de la transaction** :

  Une fois une offre sélectionnée, le microservice envoie une requête à Stripe pour créer une session de paiement, incluant les détails du tarif et des frais applicables.

- **Suivi et statut du paiement** :

  Stripe notifie en temps réel le microservice de l’état de la transaction (succès, échec, remboursement). Ces événements sont traités via Amazon EventBridge pour synchroniser les statuts avec les autres microservices.

- **Gestion des remboursements** :

  En cas d’annulation ou de modification, le microservice envoie une requête à Stripe pour déclencher un remboursement partiel ou total.

Cette architecture garantit des paiements fiables, sécurisés et conformes, tout en offrant une expérience utilisateur fluide.

Stratégie de test

Tests des microservices

Chaque microservice de l'architecture "YourCarYourWay" intègre une batterie de tests unitaire et de tests d'intégration pour garantir la qualité du code et la fiabilité des interactions entre les différentes composantes.

- Tests unitaires avec **Junit**:
    - Les tests unitaires vérifient le bon fonctionnement des composants individuels du microservice (services, méthodes, contrôleurs).
    - Ces tests couvrent les cas fonctionnels principaux et les cas aux limites.
- Tests d'intégration avec **TestContainers** :
    - Les tests d'intégration sont réalisés à l'aide de la librairie Testcontainers, qui permet de simuler l'environnement runtime du microservice en déployant des instances Docker de dépendances (bases de données, queues de messages, services tiers).
    - Cela garantit que les interactions entre les microservices ou avec les systèmes externes (ex. : Stripe) fonctionnent correctement.

Tests du frontend Angular

Le frontend, développé en Angular 18, embarque des tests end-to-end (E2E) pour valider le bon fonctionnement de l'application dans son ensemble, depuis l'interface utilisateur jusqu'aux interactions backend.

- Tests unitaires avec **Jest** :
    - Les tests unitaires vérifient le bon fonctionnement des composants individuels du microservice (services, méthodes, contrôleurs).
    - Ces tests couvrent les cas fonctionnels principaux et les cas aux limites.
    -
- Tests End-to-End avec **Cypress** :
    - Les tests E2E vérifient le comportement global de l'application, en simulant les actions des utilisateurs (navigation, remplissage de formulaires, validations).
    - Cypress est utilisé pour automatiser ces tests, avec une exécution dans des environnements CI/CD pour valider chaque déploiement.
    - Les tests couvrent des scénarios critiques comme la connexion utilisateur, la recherche et la réservation de véhicules, ainsi que les interactions en temps réel avec le service support via WebSocket.

Intégration dans le pipeline CI/CD

- Les tests sont exécutés automatiquement dans le pipeline CI/CD.

- En cas de échec, le pipeline bloque le déploiement pour garantir la stabilité de l'application.

Cette stratégie de tests permet de détecter précocement les régressions, de renforcer la robustesse de l'application et de garantir une expérience utilisateur optimale.

Dans le cadre de la stratégie de test, **SonarCloud** sera utilisé pour assurer un contrôle continu de la qualité du code. Cet outil de qualité logicielle permettra de détecter les vulnérabilités et les mauvaises pratiques de codage tout au long du cycle de développement.

L'intégration de SonarCloud dans le pipeline CI/CD permettra :

- **Analyse automatisée** : Chaque commit déclenche une analyse automatique pour identifier les problèmes de qualité dès leur introduction.

- **Conformité aux normes de codage** : Les règles de codage adaptées au projet seront configurées dans SonarCloud pour garantir un code homogène et maintenable.

- **Rapports** : Les métriques clés (code coverage, duplication, complexité cyclomatique) seront centralisées pour une visibilité accrue sur l'état global du projet.

Justification de l’approche architecturale

L’architecture de "YourCarYourWay" repose sur une approche microservices structurée autour des principes de **Domain-Driven Design**.

Cette approche garantit une modularité, une scalabilité, et une adaptabilité optimales pour répondre aux besoins métier :

- **Séparation des responsabilités** :

  Chaque domaine métier (utilisateurs, locations, paiements, support) est géré par un microservice dédié, ce qui simplifie la gestion, le développement, et la maintenance.

- **Scalabilité** :

  - Les microservices sont déployés sur Amazon ECS, garantissant une montée en charge fluide grâce à la scalabilité automatique des conteneurs.
  - La séparation des couches (client, backend, données) permet d’optimiser les performances indépendamment des volumes de trafic.

- **Résilience et fiabilité** :

  - Les interactions asynchrones sont gérées via Amazon EventBridge, minimisant les dépendances directes entre microservices.

  - L’architecture cloud-native d’AWS offre des services managés (RDS, S3, EventBridge), réduisant les risques liés à l’infrastructure.

- **Sécurité** :
    - Les données sensibles (paiements, profils utilisateurs) sont traitées dans un environnement conforme à PCI-DSS et RGPD grâce à Stripe et à des bases de données sécurisées sur Amazon RDS.
- **Déploiement continu** :
    - L'intégration et le déploiement continus (CI/CD) via des pipelines automatisés garantissent une livraison rapide et fiable des mises à jour.

Conclusion :

Cette architecture répond aux objectifs métier en offrant une solution robuste, évolutive, et sécurisée, tout en facilitant l'innovation et la gestion opérationnelle à long terme.