INSTITUT POLYTECHNIQUE DE PARIS

**E-MSC in Cybersecurity**

# Dossier d'Architecture Technique

# –

# Projet Fast Car

**Archi - Secure By Design**


**Atelier - Equipe 2**

*Kelaty BAH*
*Jerome Clérambault*
*Stéphane Gagnor*
*Laurent Giovannoni*

# Table des matières

# 1. Contexte et objectifs du projet

Fast Car prévoit de déployer une plateforme de services connectés pour sa nouvelle génération de véhicules. Le projet vise à concevoir une architecture microservices sécurisée, scalable et industrialisée par DevSecOps.

Les exigences principales incluent :

- La collecte et le stockage de la télémétrie véhicule.
- La fourniture d'informations de météo et de qualité de l'air en fonction de la localisation.
- La synchronisation des contacts Google du conducteur.
- L'assurance de la sécurité, de la conformité, de l'observabilité et de la résilience.

L'architecture s'articule autour des composants suivants :

- Un **API Gateway** central, assurant la sécurité et l'orchestration des requêtes.
- Trois microservices dédiés : **Telemetry Service**, **Weather Service**, **Contacts Service**.
- Une base de données **PostgreSQL** pour le stockage des tokens OAuth (Contacts Service). Le rate-limiting utilise **slowapi** en mémoire .
- Une approche **Secure by Design**, intégrée au pipeline **GitLab CI/CD**.

Réalisations :

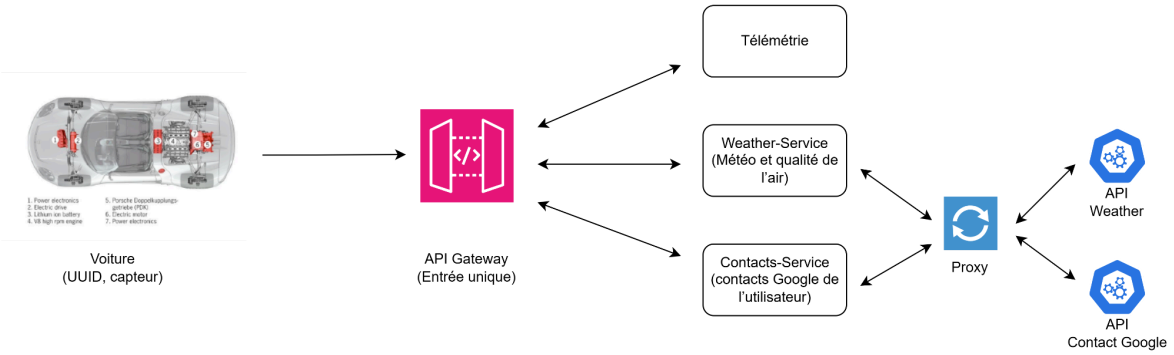
- Séparation des domaines (gateway et services) et application de contrôles de sécurité (gestion des PII, durcissement des API, observabilité).
- Industrialisation via CI/CD DevSecOps (tests, scans de sécurité, génération de SBOM, signature) pour maîtriser le risque lié à la chaîne d'approvisionnement logicielle.
- Fourniture d'une démonstration reproductible, documentée dans DEMO.md.

Périmètre technique implémenté :

- **API Gateway** (port 8000) : gestion de l'authentification (authN), protections OWASP, routage, endpoints /health et /metrics.
- **Services internes** : Telemetry (8001), Weather (8002), Contacts (8003).
- **Base de données** : PostgreSQL (5432).
- **Orchestration locale** : docker compose et Makefile.

# 2. Architecture fonctionnelle

## 2.1 Schéma d'architecture fonctionnelle




L'écosystème pourrait comporter trois types de clients en cible : Véhicule, App mobile conducteur et Dashboard web. Dans un premier temps nous implémenterons une solution embarquée dans le véhicule.

Le véhicule s'identifie via un UUID (vehicle_id) et obtient un JWT (POST /auth/token).

Cycle d'usage :

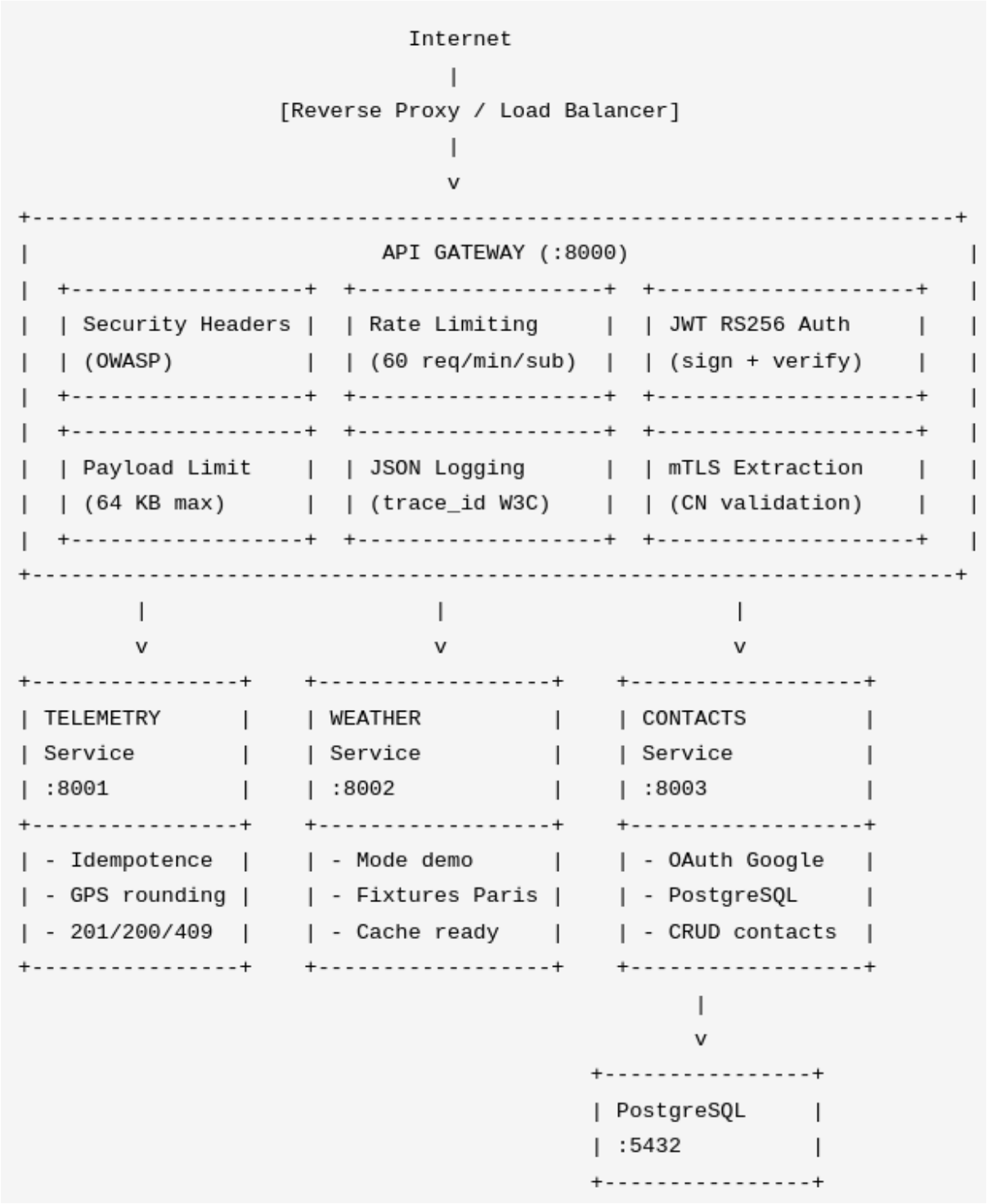
1. Le client demande un jeton via POST /auth/token → **JWT RS256** (exp 15 min).
2. Le client envoie ou récupère des données via l'API Gateway.
3. L'API Gateway applique authentification, rate-limit, validation stricte, logs corrélés et expose /metrics.
4. Les microservices internes répondent aux requêtes via des timeouts courts (2 secondes).

## 2.2 Interactions entre services

**Télémétrie** : le véhicule envoie POST /telemetry/ingest. Le service applique idempotence, normalisation et règles de stockage.
**Météo** : le client appelle GET /weather/current?lat=...&lon=.... Le service retourne des fixtures en mode démo (intégration WeatherAPI en cible).
**Contacts** : Contacts : le conducteur appelle GET /contacts/?person_fields=names,emailAddresses (mode démo avec fixtures JSON et intégration OAuth Google réalisée).

# 3. Architecture technique

## 3.1 Schéma global d’architecture



## 3.2 Choix techniques (langages, frameworks, conteneurisation)

Les microservices sont développés en Python pour la simplicité et la compatibilité avec l'architecture.
Chaque service est conteneurisé avec Docker, garantissant portabilité et isolation.
Les communications internes utilisent des API REST sécurisées.
Le monitoring et les logs sont intégrés par défaut dans chaque conteneur pour assurer la traçabilité et la supervision du système.

- Langage : Python, FastAPI.
- Conteneurisation : Docker, images minimales.
- Communication : REST
- Observabilité : Prometheus.

## 3.3 Infrastructure cible (Docker, Kubernetes, GitLab CI/CD)

- Déploiement orchestré sur **Kubernetes**.
- CI/CD GitLab : build → tests → scans sécurité → déploiement.
- API Gateway exposée via Ingress.
- Secrets gérés par GitLab variables protégées ou Vault.

## 3.4 Composants techniques

| Composant | Rôle | Points clés | Port |
|---|---|---|---|
| API Gateway | Entrée unique | JWT RS256, rate-limit, headers OWASP, validation stricte, logs JSON, /metrics | 8000 |
| Telemetry | Ingestion | 201/200/409 ; minimisation GPS (arrondi) ; schémas stricts | 8001 |
| Weather | Météo/air | Mode démo (fixtures) ; integration fournisseur possible | 8002 |
| Contacts | Contacts | Mode démo + OAuth2 (moindre privilège) ; persistance DB | 8003 |
| PostgreSQL | Stockage | PostgreSQL 16 ; réseau docker interne | 5432 |

### 3.5 Authentification multi-niveaux

```
Niveau 1: Transport (mTLS)
+---------------------------+
| Certificat client X.509   |
| Signe par CA SuperCarX    |
| CN = vehicle_id           |
+---------------------------+
            |
            v
Niveau 2: Application (JWT RS256)
+---------------------------+
| Token JWT signe RS256     |
| sub = vehicle_id          |
| exp = 15 minutes max      |
| aud = "supercarx"         |
+---------------------------+
            |
            v
Niveau 3: Cross-Validation
+---------------------------+
| CN certificat == sub JWT  |
| Verification automatique  |
+---------------------------+
```

# 4. Gestion des données

## 4.1 Flux de données

- Le véhicule envoie la télémétrie → API Gateway → Telemetry Service → PostgreSQL.
- Une demande météo → API Gateway → Weather Service → WeatherAPI.
- Une demande contacts → API Gateway → Contacts Service → Google API.

## 4.2 Données échangées

- **Télémétrie** :

  { event_id, vehicle_id, ts, metrics: { speed, gps: { lat, lon } } }

- **Météo** :

{ temperature, humidity, air_quality, timestamp }

- **Contacts** :

  { contacts: [ { name, phone, email } ] }

## 4.3 Données et minimisation PII

| Donnée | Classif. | Contrôles (MVP) |
|---|---|---|
| vehicle_id (UUID) | Interne | Validation UUID stricte ; sub du JWT |
| Télémétrie | Conf. | Schémas stricts ; idempotence (event_id) ; pas de PII en logs |
| GPS | PII | Arrondi à 4 décimales (~11 m) |
| Contacts | PII | Mode démo ; OAuth2 scope minimal (cible) ; persistance contrôlée |
| Tokens OAuth | Restreint | Jamais en clair dans le code ; variables runtime/CI protégées |
| Logs/métriques | Restreint | Logs JSON corrélés (trace_id) ; /metrics Prometheus |

## 4.4 Recommandations RRA et implémentation

| Impact | Recommandation RRA | Status | Implementation |
|---|---|---|---|
| MAXIMUM | Utiliser mTLS pour l'identification vehicule-service | Fait | Module supercarx/mtls.py, scripts PKI dans scripts/, cross-validation CN <-> JWT |

| | | | |
|---|---|---|---|
| MAXIMUM | Utiliser OAuth avec moindres privileges | Fait | Service Contacts avec Google OAuth 2.0, scope read-only pour contacts |
| MAXIMUM | Gerer les secrets avec KMS/Vault | Fait | Variables CI protegees + .env local |
| HIGH | Minimisation des donnees (Geohash localisation) | Fait | GPS arrondi 4 decimales (~11m), pas de persistance contacts par defaut |
| HIGH | Securisation API (JWT + rate limiting) | Fait | JWT RS256 (15min exp), slowapi 60 req/min par vehicle_id |
| HIGH | Logs sans PII, tracage, metriques | Fait | JSON logging avec trace_id, /metrics Prometheus, logs sans PII |
| HIGH | Supply chain CI/CD (SBOM, SCA, SAST/DAST) | Fait | Pipeline GitLab: bandit, trivy, cyclonedx-bom, ZAP DAST |

# 5. CI/CD et déploiement

Pipeline GitLab standardisé comprenant :

1. Build des images Docker.
2. Tests unitaires et intégration.
3. Analyses SAST/DAST.
4. Scan des images conteneurs.
5. Déploiement automatique (dev → test → prod).
6. Versioning et validation avant production.
7. Signature et attestation des images docker.

# 6. Sécurité et conformité (DevSecOps)

La sécurité est intégrée dès la conception.
Les images Docker sont signées et scannées avant déploiement.
Les analyses SAST et DAST sont exécutées automatiquement dans GitLab CI/CD.
Les secrets sont gérés via Vault ou des variables protégées GitLab.
Les communications interservices sont chiffrées en TLS mutuel.

Pipeline détaillée :

- Authentification et identité :
  - JWT RS256 : exp 15 min, aud=supercarx, sub=vehicle_id.
  - mTLS : support implémenté/activable (certificat X.509) ; cross-validation transport/applicatif en cible.
- Durcissement API :

    - Rate limiting : 60 req/min/vehicle_id (429).
    - Limite de payload : 64 KB.
- Validation stricte :
    - additionalProperties=false + Pydantic forbid.
    - Security headers OWASP + traçabilité x-trace-id.
- CI/CD DevSecOps :
    - ruff/black + bandit ; pytest ; pip-audit + gitleaks ; trivy ; SBOM CycloneDX ; ZAP baseline ; cosign (signature +attestation).

# 7. Supervision et maintenance

- Prometheus pour métriques (timeouts, latence, erreurs).
- Logs centralisés
- Monitoring conteneurs + règles d’alerting.