OnCallNova

Plateforme de Gestion d'Incidents

Type PagerDuty — 100% Locale

5 Microservices | gRPC & REST | PostgreSQL + Redis | JWT Auth | Observabilité

Hackathon Opensrc — Document d'Architecture

10 février 2026

Table des matières

1 Vue d'Ensemble du Projet

Résumé Exécutif

OnCallNova est une plateforme de gestion d'incidents de production déployable **100% en local** via Docker Compose. Elle reproduit les fonctionnalités essentielles d'un outil comme PagerDuty : ingestion d'alertes, corrélation intelligente, gestion du cycle de vie des incidents, planification d'astreintes et tableau de bord temps réel — le tout orchestré par **5 microservices** communiquant en **gRPC** et exposés derrière un API Gateway **Traefik**.

1.1 Objectifs Fonctionnels

1. **Ingestion & Corrélation** — Recevoir des alertes multi-sources et les regrouper en incidents via un algorithme de fenêtre temporelle.
2. **Gestion du Cycle de Vie** — Créer, acquitter, résoudre, escalader les incidents avec traçabilité complète.
3. **Planification d'Astreintes** — Définir des équipes, des rotations et identifier l'ingénieur de garde en temps réel.
4. **Observabilité** — Métriques Prometheus, dashboards Grafana, tracing distribué Jaeger.
5. **Sécurité** — Authentification JWT, rate limiting Traefik, chiffrement des secrets.

2 Architecture Globale

2.1 Diagramme de Topologie des Services

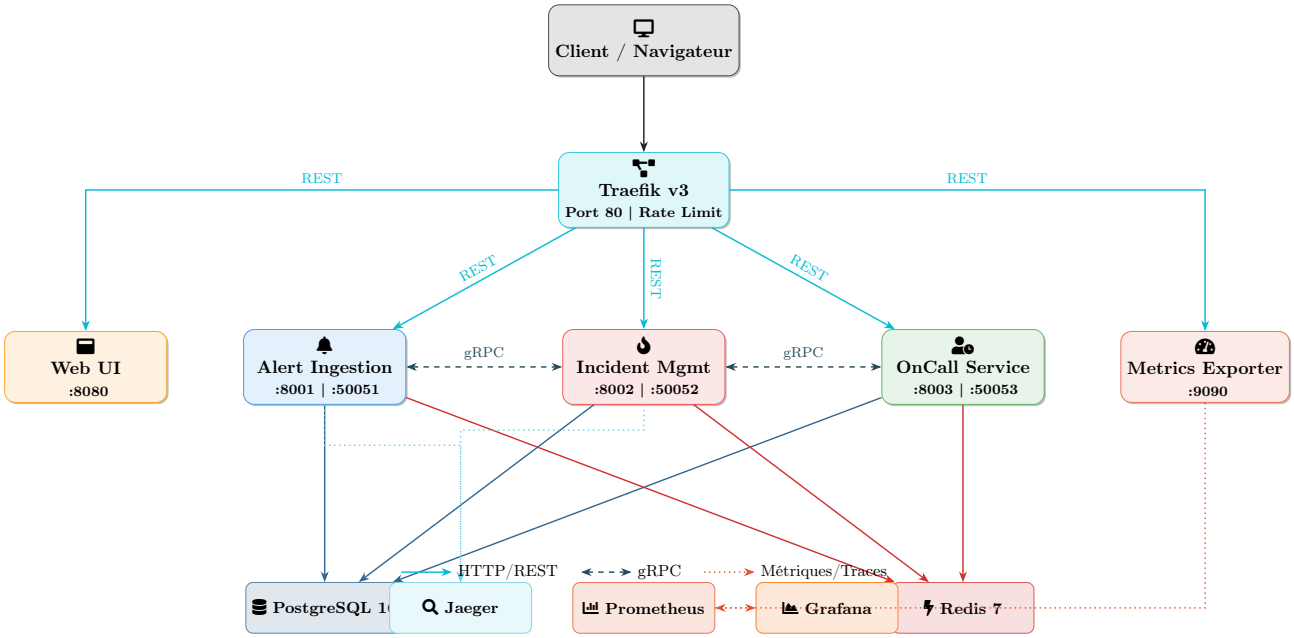

FIGURE 1 – Topologie complète des 14 conteneurs Docker de la plateforme OnCallNova

2.2 Inventaire des 14 Conteneurs

Service	Techno	Port(s)	Rôle
Alert Ingestion	Python/FastAPI	dyn / 50051	Réception alertes, corrélation, gRPC→Incident
Incident Mgmt	Python/FastAPI	8002 / 50052	CRUD incidents, webhooks, analytics
OnCall Service	Python/FastAPI	8003 / 50053	Astreintes, rotations, escalade
Web UI	Python/FastAPI	8080	Dashboard temps réel, login JWT
Metrics Exporter	Python/FastAPI	9090	Agrégation métriques custom
PostgreSQL 16	PostgreSQL	5432	Base de données relationnelle
Redis 7	Redis	6379	Cache, Pub/Sub, sessions
Traefik v3	Go	80 / 8888	API Gateway, routage, rate limiting
Prometheus	Go	9091	Collecte métriques (scrape 5s)
Grafana 10.3	Go	3000	Visualisation dashboards
Jaeger	Go	16686 / 4317	Tracing distribué (OTLP)
Loki 3.0	Go	3100	Agrégation logs
Promtail	Go	—	Collecte logs → Loki
MailHog	Go	8025 / 1025	Serveur SMTP de test local

TABLE 1 – Détail des 14 services Docker Compose

3 Architecture Détaillée par Couche

3.1 Couche 1 — API Gateway (Traefik v3)

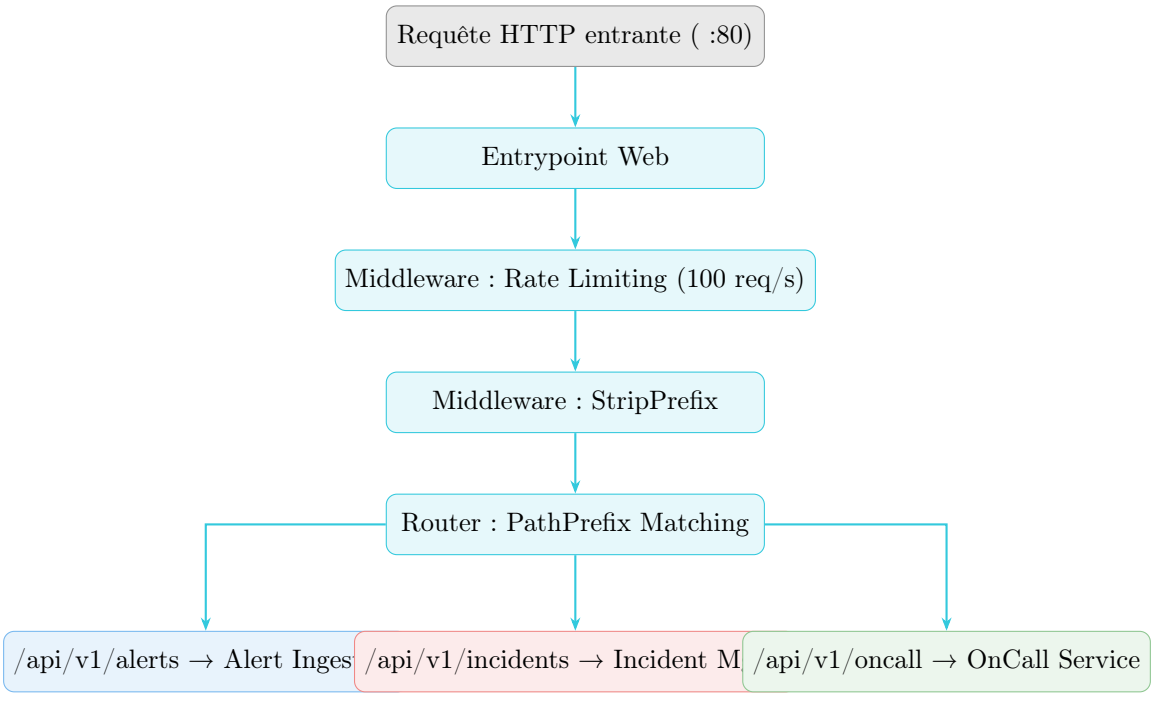

FIGURE 2 – Pipeline de traitement Traefik — du client au microservice

Justification : Traefik v3

- **Découverte automatique** via labels Docker — zéro configuration de routes manuelles.
- **Rate limiting natif** (100 req/s par IP) sans code applicatif.
- **Dashboard intégré** (:8888) pour visualiser les routes en temps réel.
- **Support scaling** : découvre automatiquement les nouvelles instances lors d'un `docker compose up -scale alert-ingestion=3`.

— **Alternatives rejetées** : Nginx (configuration statique, pas de découverte Docker), Kong (trop lourd pour un déploiement 100% local).

3.2 Couche 2 — Les 5 Microservices

3.2.1 Alert Ingestion Service

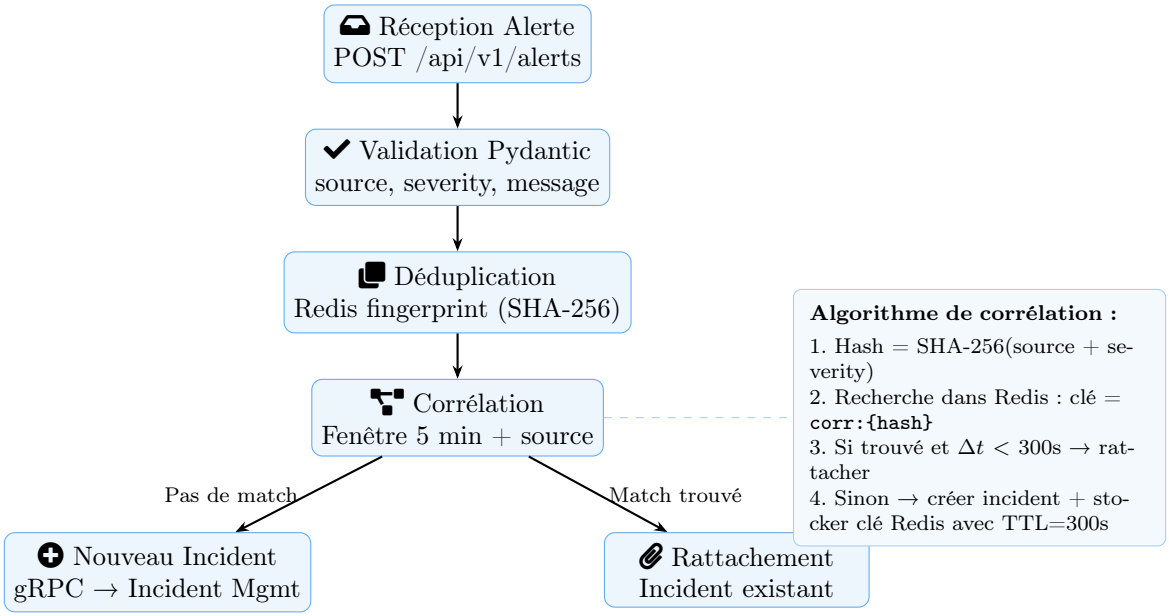

FIGURE 3 – Pipeline de traitement du service Alert Ingestion

Justification : Corrélation par fenêtre temporelle

— Réduit le **bruit d'alertes** : 100 alertes similaires en 5 minutes = 1 seul incident.
— **Redis comme cache de corrélation** : TTL natif pour l'expiration automatique de la fenêtre — $O(1)$ en lecture/écriture.
— **SHA-256 fingerprint** : collision quasi-nulle, déduplication fiable.
— Fenêtre de 5 minutes configurable via variable d'environnement `CORRELATION_WINDOW`.

3.2.2 Incident Management Service

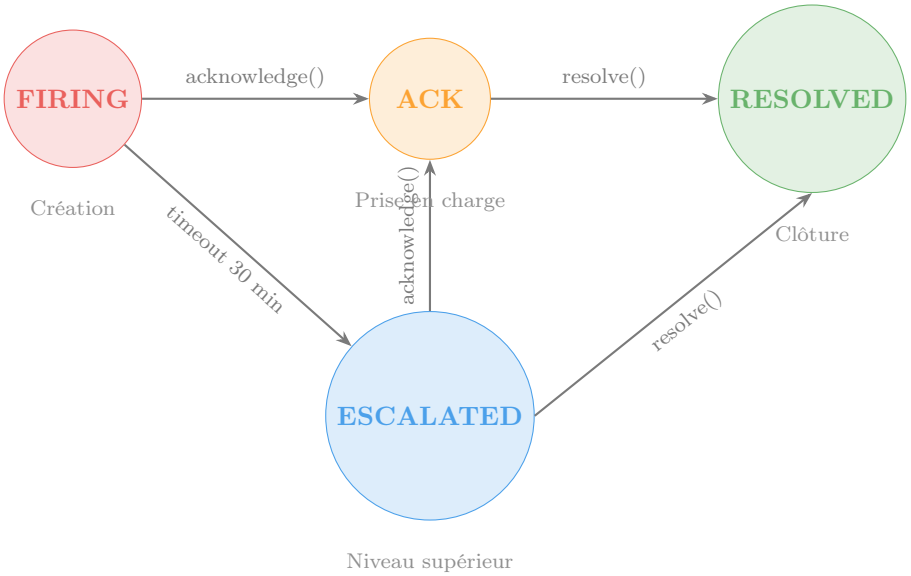

FIGURE 4 – Machine à états du cycle de vie d'un incident

Fonctionnalités clés — Incident Management

- CRUD complet (REST + gRPC)
- Machine à états FIRING → ACK → RESOLVED
- Escalade automatique (timeout 30 min)
- Historique complet (timeline)
- Calcul MTTA / MTTR automatique
- Notifications email (MailHog)
- Dispatch webhooks (HMAC-SHA256)
- Analytics (trends & MTTR distribution)
- Protection JWT sur endpoints d'écriture
- Métriques Prometheus exposées

3.2.3 OnCall Service

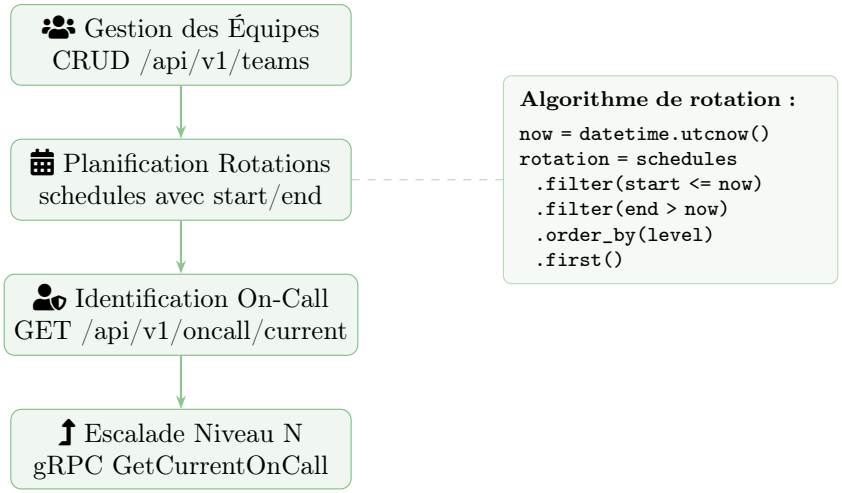

FIGURE 5 – Fonctionnement du service d'astreintes OnCall

3.2.4 Web UI

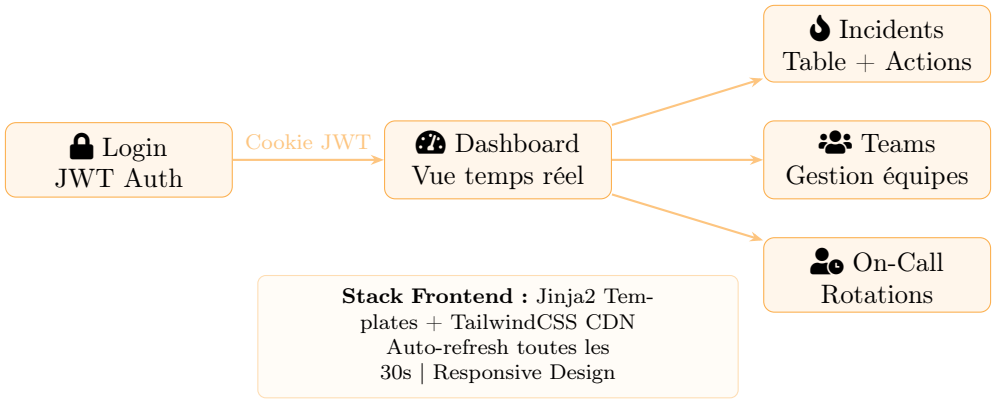

FIGURE 6 – Navigation de l'interface Web UI

3.2.5 Metrics Exporter

Métriques exposées (/metrics)

- `oncall_incidents_total` — Compteur total d'incidents (par sévérité)
- `oncall_incidents_active` — Jauge d'incidents actifs
- `oncall_mtta_seconds` — Histogramme Mean Time To Acknowledge
- `oncall_mttr_seconds` — Histogramme Mean Time To Resolve
- `oncall_alerts_ingested_total` — Compteur d'alertes ingérées
- `oncall_oncall_engineers_active` — Jauge d'ingénieurs de garde
- Format : Prometheus text exposition (scrape interval : 5s)

3.3 Couche 3 — Communication Inter-Services

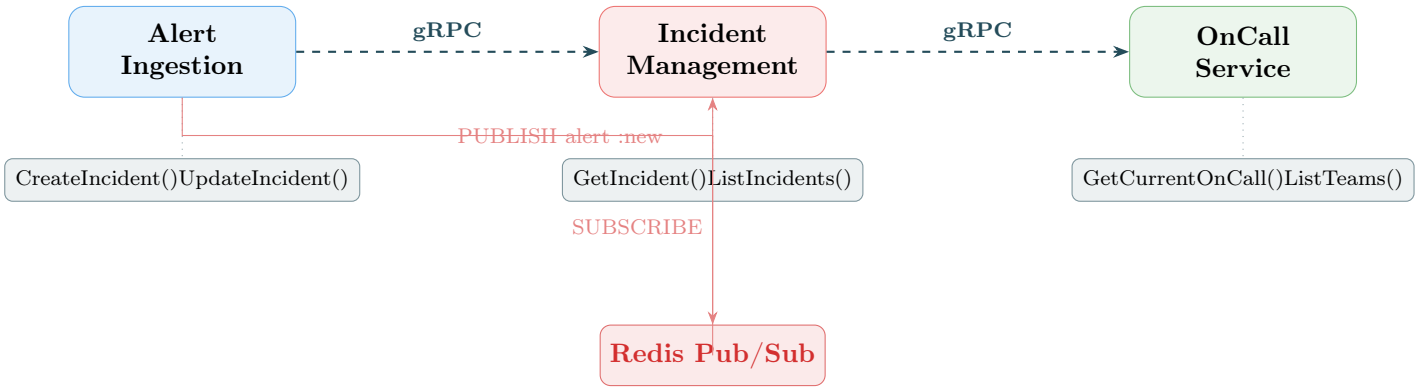

FIGURE 7 – Communication inter-services : gRPC synchrone + Redis Pub/Sub asynchrone

Justification : gRPC + Protobuf

- **Performance** : sérialisation binaire Protobuf 5-10× plus rapide que JSON.
- **Contrat fort** : fichiers `.proto` = documentation + validation automatique.
- **Streaming** : support natif du streaming bidirectionnel pour les futures évolutions.
- **Code generation** : stubs Python générés automatiquement (`grpcio-tools`).
- **Alternatives rejetées** : REST interne (overhead HTTP, pas de contrat), RabbitMQ (complexité supplémentaire, Redis suffit pour le Pub/Sub simple).

3.4 Couche 4 — Persistance des Données

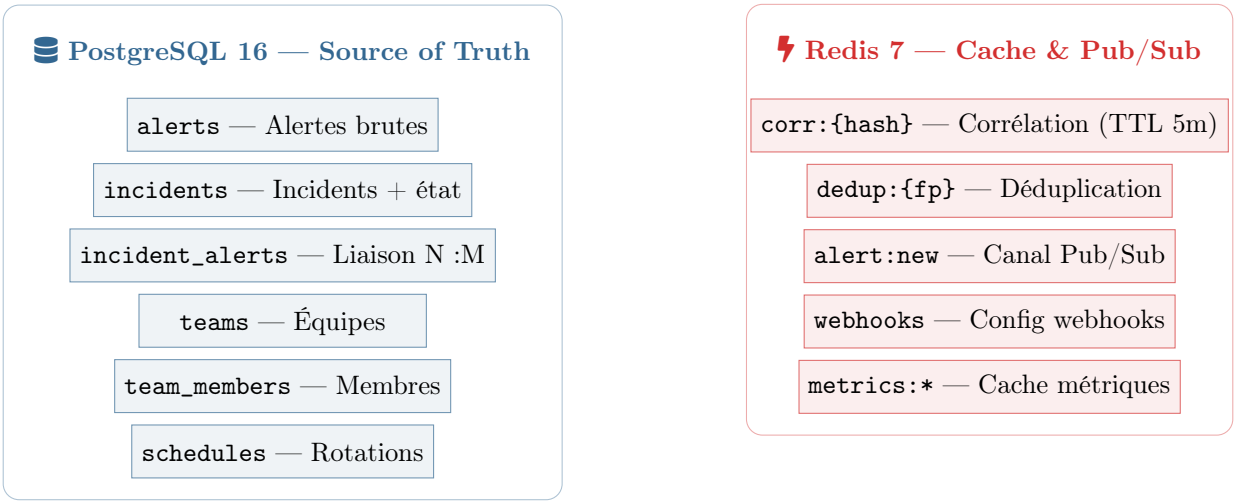

FIGURE 8 – Modèle de données — PostgreSQL (persistant) + Redis (éphémère)

Justification : PostgreSQL + Redis

- **PostgreSQL** : ACID complet, JSONB pour métadonnées flexibles, requêtes analytiques performantes (trends, MTTR).
- **SQLAlchemy Async 2.0** : ORM avec support `asyncpg` — performances optimales sans bloquer l'event loop FastAPI.
- **Redis** : latence sub-milliseconde pour la corrélation en temps réel, TTL natif pour l'expiration des fenêtres, Pub/Sub intégré.
- **Séparation des responsabilités** : PostgreSQL = source de vérité, Redis = cache transitoire.

4 Sécurité

4.1 Architecture d'Authentification JWT

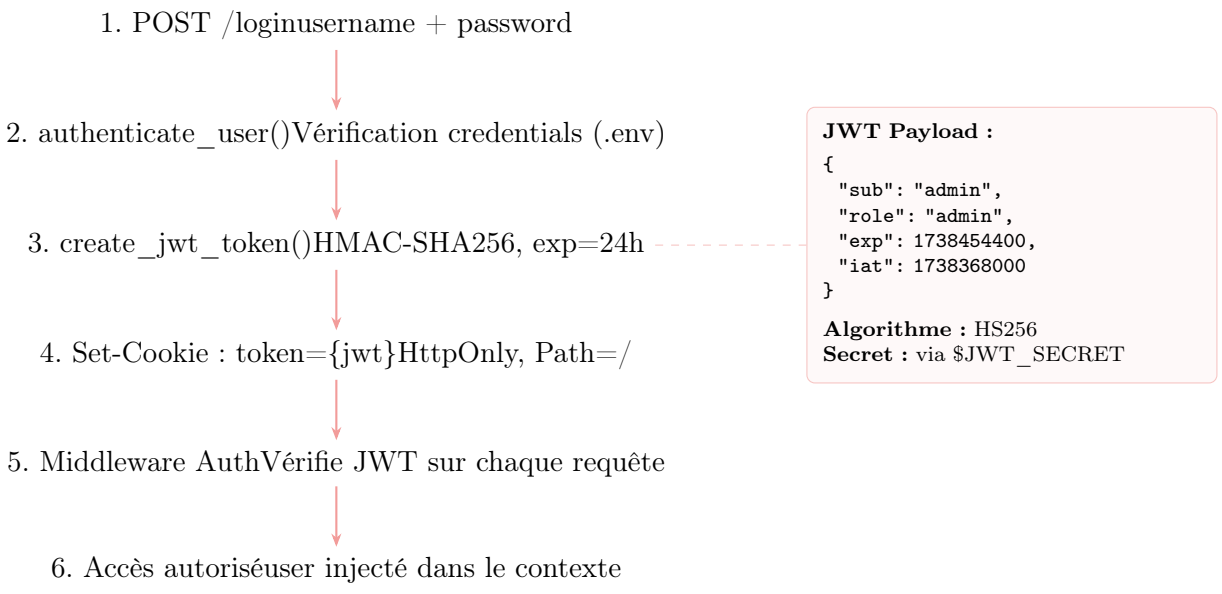

FIGURE 9 – Flux d'authentification JWT — du login à l'accès protégé

Mesures de Sécurité

- ✓ JWT HMAC-SHA256
- ✓ Expiration 24h configurable
- ✓ Secrets via variables d'env
- ✓ Rate limiting Traefik (100/s)
- ✓ Cookie HttpOnly
- ✓ Pas de credentials en dur
- ✓ Endpoints sensibles protégés
- ✓ Bearer token pour API
- ✓ Middleware global Web UI
- ✓ Support multi-utilisateurs

5 Stack d'Observabilité

5.1 Les 3 Piliers

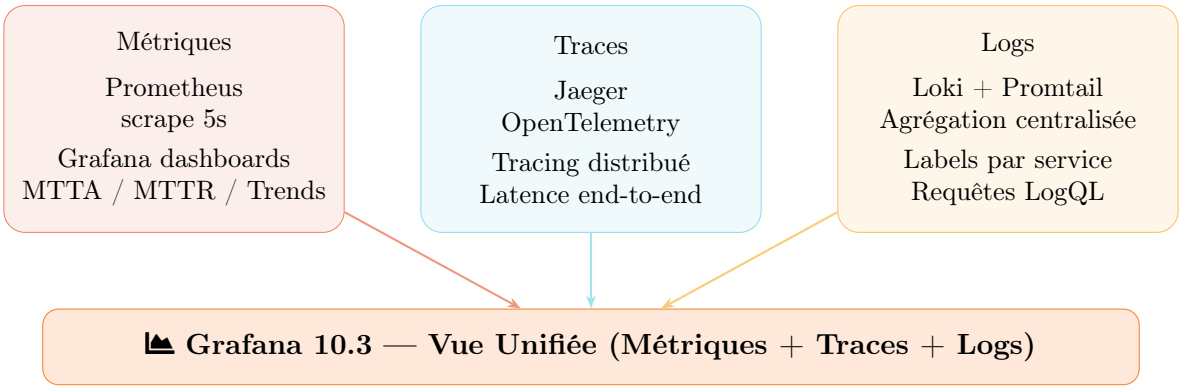

FIGURE 10 – Les 3 piliers de l'observabilité unifiés dans Grafana

Justification : Stack LGTM (Loki, Grafana, Tempo→Jaeger, Mimir→Prometheus)

- **Standard industriel** : stack Grafana Labs / CNCF, utilisée par les leaders du marché.
- **OpenTelemetry** : standard vendeur-neutre pour l'instrumentation — un seul SDK pour métriques, traces et logs.
- **Corrélation** : Grafana permet de naviguer d'une métrique → une trace → des logs d'un seul clic.
- **Alternatives rejetées** : ELK Stack (lourd, license), Datadog (SaaS payant), New Relic (pas 100% local).

6 Fonctionnalités Bonus

6.1 Vue d'ensemble

#	Bonus	Technologie	Statut	Description
1	Notifications Email	MailHog / SMTP	✓	Envoi d'emails lors des changements d'état
2	Webhooks	HMAC-SHA256	✓	Dispatch sécurisé vers URLs externes
3	Escalade Automatique	Cron + gRPC	✓	Escalade après timeout (30 min)
4	Analytics Historiques	SQL + agrégations	✓	Trends, MTTR distribution, rapports
5	Agrégation de Logs	Loki 3.0 + Promtail	∼	Code présent, limité sur Windows Docker
6	Tracing Distribué	Jaeger + OTLP	✓	Traces end-to-end inter-services
7	Scaling Horizontal	Docker Compose	✓	`-scale alert-ingestion=3`

TABLE 2 – Tableau récapitulatif des 7 fonctionnalités bonus

6.2 Bonus 1 — Notifications Email

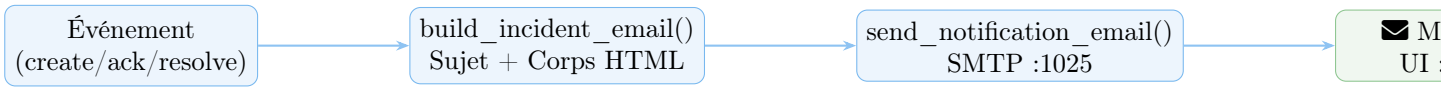

FIGURE 11 – Pipeline de notification email via MailHog

Détails techniques — Email

- **Transport** : SMTP standard (port 1025), compatible SendGrid en production.
- **MailHog** : capture tous les emails localement sans envoi réel — parfait pour le développement.
- **Templates** : emails HTML formatés avec sévérité, titre, timestamp.
- **Configuration** : `SMTP_HOST`, `SMTP_PORT`, `SMTP_FROM` via `.env`.

6.3 Bonus 2 — Webhooks Sécurisés

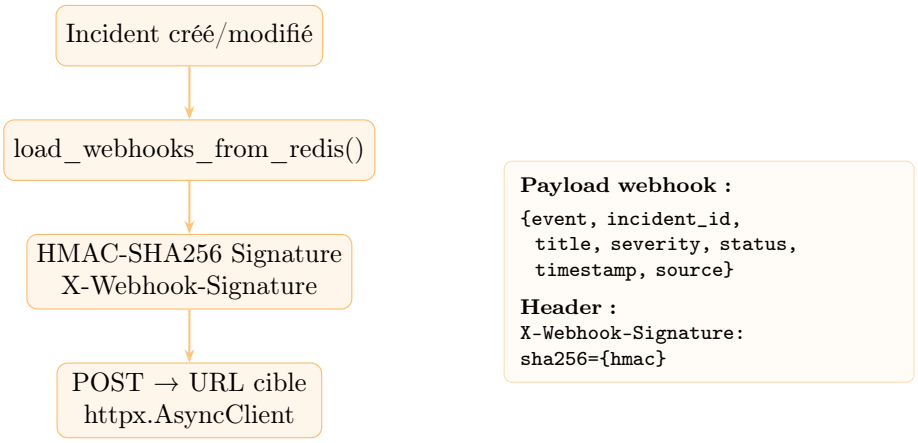

FIGURE 12 – Architecture du système de webhooks

6.4 Bonus 3 — Escalade Automatique

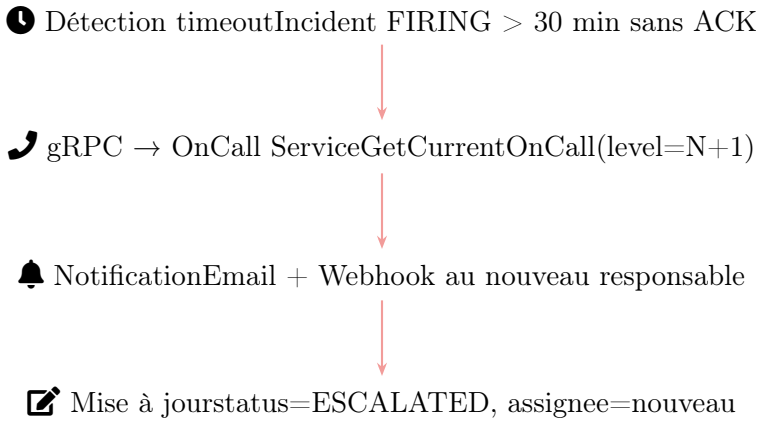

FIGURE 13 – Processus d'escalade automatique multi-niveaux

6.5 Bonus 4 — Analytics Historiques

Endpoints Analytics

- **GET /api/v1/analytics/trends**
  - Paramètres : `days` (défaut 30), `severity`
  - Retourne : incidents par jour, par sévérité, totaux
  - Requête SQL avec `GROUP BY date_trunc('day', created_at)`
- **GET /api/v1/analytics/mttr-distribution**
  - Distribution des temps de résolution
  - Calcul des percentiles P50, P90, P99
  - Segmentation par sévérité (critical, high, medium, low)

6.6 Bonus 5 — Agrégation de Logs (Loki)

Architecture Loki

- **Loki 3.0** : stockage et indexation des logs par labels (pas full-text).
- **Promtail** : agent de collecte qui scrape les logs Docker.
- **LokiHandler** (Python) : handler `logging.Handler` custom qui push directement vers l'API HTTP Loki (`/loki/api/v1/push`).
- **Labels** : `service`, `level`, `environment`.
- **Note** : fonctionnel sur Linux, limité sur Windows Docker Desktop (incompatibilité API Docker socket pour Promtail).

6.7 Bonus 6 — Tracing Distribué (Jaeger)

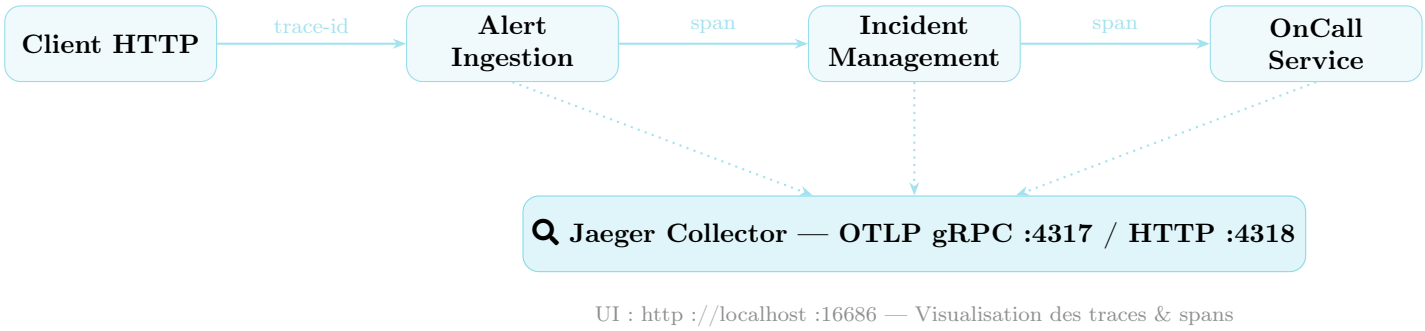

FIGURE 14 – Propagation des traces distribuées via OpenTelemetry → Jaeger

Justification : OpenTelemetry + Jaeger

- **OpenTelemetry** : standard CNCF vendeur-neutre — instrumentation une seule fois, exportation vers n'importe quel backend (Jaeger, Zipkin, Datadog...).
- **Propagation automatique** : le `trace-id` est propagé dans les headers HTTP et métadonnées gRPC entre tous les services.
- **Jaeger All-in-One** : déploiement simplifié (collector + query + UI en 1 conteneur).
- **Impact performances** : overhead $< 3\%$ grâce au sampling.

6.8 Bonus 7 — Scaling Horizontal

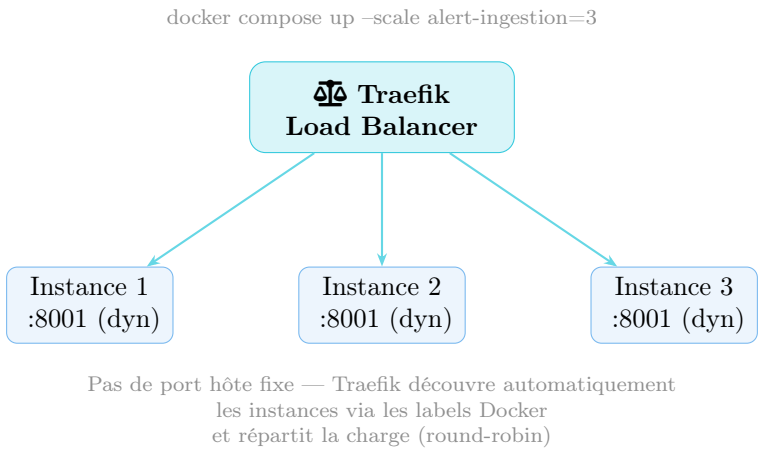

FIGURE 15 – Scaling horizontal du service Alert Ingestion via Docker Compose

Justification : Scaling via Docker Compose + Traefik

- **Zéro configuration supplémentaire** : Traefik découvre les conteneurs automatiquement via l'API Docker.
- **Port dynamique** : le service n'a pas de port hôte fixe, évitant les conflits lors du scaling.
- **Load balancing** : round-robin natif entre les instances.
- **Commande unique** : `docker compose up -d -scale alert-ingestion=3`.

7 Synthèse des Choix Techniques

Composant	Choix	Justification	Alternative rejetée
Langage	Python 3.11	Écosystème riche, async natif, prototypage rapide	Go (temps de dev), Java (verbosité)
Framework API	FastAPI 0.109	Async natif, validation Pydantic, OpenAPI auto	Flask (sync), Django (monolithique)
Communication	gRPC + Protobuf	Contrat fort, perf binaire, génération de code	REST interne (pas de contrat)
BDD	PostgreSQL 16	ACID, JSONB, performances analytiques	MongoDB (pas ACID), MySQL (moins de features)
Cache	Redis 7	Sub-ms latence, TTL natif, Pub/Sub intégré	Memcached (pas de Pub/Sub)
ORM	SQLAlchemy 2.0	Async (asyncpg), migrations, mature	Tortoise (moins mature), raw SQL
Gateway	Traefik v3	Auto-discovery Docker, rate limiting, dashboard	Nginx (config statique), Kong (lourd)
Conteneurs	Docker Compose	Orchestration simple, 100% local, reproductible	Kubernetes (overkill pour local)
Métriques	Prometheus	Pull model, PromQL puissant, standard k8s	InfluxDB (push model), StatsD
Dashboards	Grafana 10.3	Multi-datasource, alerting, communauté	Kibana (ELK only)
Tracing	Jaeger + OTel	CNCF, vendeur-neutre, all-in-one	Zipkin (moins de features), X-Ray (AWS only)
Auth	JWT HS256	Stateless, standard, simple à implémenter	OAuth2 (complexe pour local), sessions
Email test	MailHog	Capture locale, UI web, zéro config	Mailtrap (SaaS), Papercut

TABLE 3 – Matrice décisionnelle des choix technologiques

8 Déploiement & Utilisation

8.1 Architecture de Build

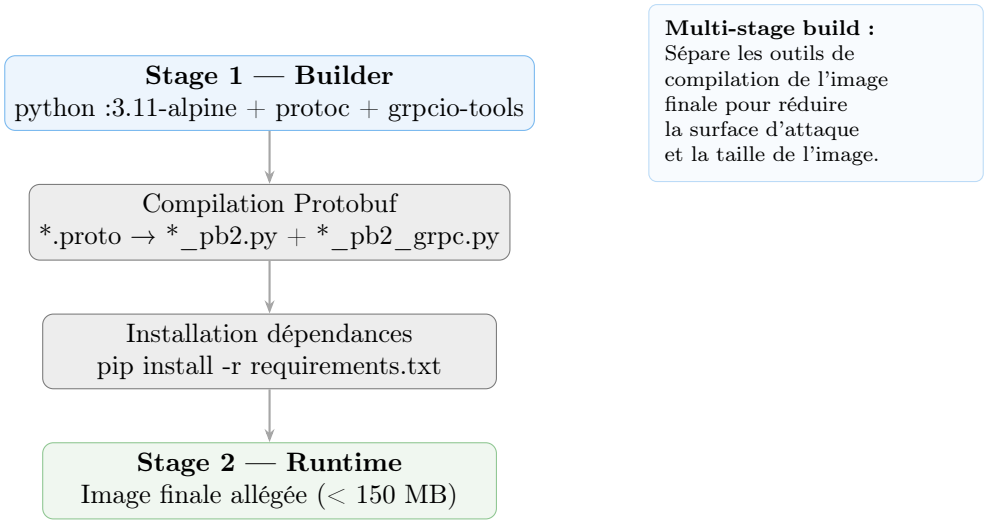

FIGURE 16 – Dockerfile multi-stage pour les microservices

8.2 Commandes de Déploiement

Lancement rapide (PowerShell — Windows)

```
# Deploiement complet (14 conteneurs)
powershell -ExecutionPolicy Bypass -File .\run.ps1 deploy

# Demo complete pour le jury (7 etapes automatisees)
powershell -ExecutionPolicy Bypass -File .\run.ps1 full-demo

# Scaling horizontal (3 instances)
powershell -ExecutionPolicy Bypass -File .\run.ps1 scale 3

# Analytics et metriques
powershell -ExecutionPolicy Bypass -File .\run.ps1 analytics
```

8.3 URLs d'Accès

Service	URL	Description
Web UI (Dashboard)	`http://localhost:8080`	Interface principale (login requis)
API Gateway (Traefik)	`http://localhost:80`	Point d'entrée API REST
Traefik Dashboard	`http://localhost:8888`	Visualisation des routes
Grafana	`http://localhost:3000`	Dashboards (admin/admin)
Prometheus	`http://localhost:9091`	Interface requêtes PromQL
Jaeger UI	`http://localhost:16686`	Tracing distribué
MailHog	`http://localhost:8025`	Emails capturés

TABLE 4 – Points d'accès de la plateforme

9 KPI & Métriques Clés

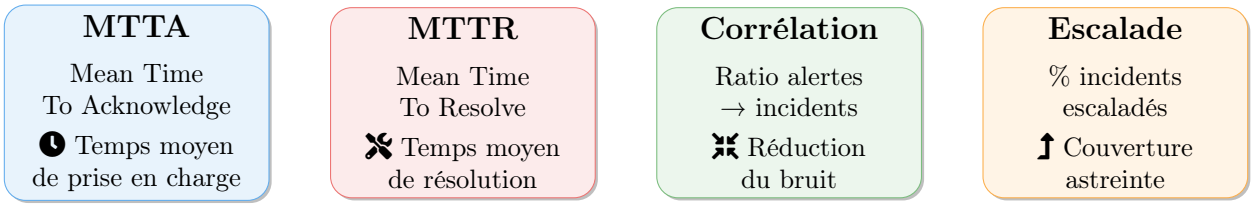

FIGURE 17 – Les 4 KPI opérationnels suivis par la plateforme

10 Conclusion

Récapitulatif

OnCallNova démontre qu'il est possible de construire une plateforme de gestion d'incidents de niveau production avec :

- **5 microservices** communicant en gRPC + REST
- **14 conteneurs** Docker Compose orchestrés par Traefik
- **Observabilité complète** : Prometheus + Grafana + Jaeger + Loki
- **7 fonctionnalités bonus** implémentées (6 pleinement fonctionnelles)
- **Sécurité** : JWT, rate limiting, secrets externalisés
- **100% local** — aucune dépendance cloud, déployable en une commande

🚀 Déploiement en une commande :

```
powershell -ExecutionPolicy Bypass -File .\run.ps1 full-demo
```