DOCUMENT D'ARCHITECTURE TECHNIQUE (DAT)

AgentOps - Plateforme Micro-SaaS d'Automatisation IA pour Développeurs

1. Synthèse Exécutive pour l'Architecture

Vue d'Ensemble

AgentOps est conçu comme une **plateforme d'orchestration IA distribuée** combinant une architecture microservices moderne avec des capacités d'intelligence artificielle avancées. L'architecture privilégie la **scalabilité horizontale**, la **résilience** et la **souveraineté des données** tout en maintenant une complexité opérationnelle maîtrisée pour une équipe lean.

Principes Architecturaux Fondamentaux

- 1. **Hybrid Cloud-First avec Self-Hosting Capability**: Architecture cloud-native déployable sur infrastructure managée (DigitalOcean/AWS) ou on-premise
- 2. Event-Driven Architecture : Communication asynchrone via events et queues pour découplage et résilience
- 3. API-First Design: Tous les services exposent des APIs RESTful/GraphQL documentées et versionnées
- 4. Observability by Design: Logging, monitoring et tracing intégrés dès la conception
- 5. Security in Depth: Sécurité à chaque couche (réseau, application, données)

Choix Technologiques Majeurs et Justifications

ComposantTechnologie ChoisieJustification Stratégique Backend API Laravel 12 (PHP 8.4) • Expertise founder existante (time-to-market)

Horizon, Vapor)

Fixedlente DX pour prototypage rapide

Horizon, Vapor)

Horizon, Vapor)

Fixedlente DX pour prototypage rapide

Horizon, Vapor)

Horizon, Vapor)

Fixedlente DX pour prototypage rapide

Horizon, Vapor)

Horizon, Va

Architecture en Chiffres (Capacité Cible Phase 1-3)

MétriquePhase 1 (M0-3)Phase 2 (M3-9)Phase 3 (M9-18) Utilisateurs Concurrents 1001 00010 000 Workflows/jour 1 00010 00010 000 Latence API (p95) < 500ms< 300ms< 200ms Uptime SLA 95%99%99.9% Data Volume 10 GB100 GB1 TB Infrastructure Cost < 500 \$/mois< 2K \$/mois< 10K \$/mois

2. Exigences Clés (Issues du PRD et de la Vision)

2.1. Exigences Fonctionnelles Critiques

EF-1: Workflow IA Autonome End-to-End

Priorité: P0 (Bloquant MVP)

Description : Le système doit permettre l'orchestration complète d'un workflow : analyse repo \rightarrow génération code \rightarrow exécution tests \rightarrow déploiement CI/CD, sans intervention humaine.

Critères de Validation Technique :

- Temps d'exécution workflow complet : < 10 minutes (p95)
- Taux de réussite : > 85% (workflows complétés sans erreur)
- Capacité à gérer 10 workflows parallèles par utilisateur

Contraintes:

- Idempotence : re-exécution du même workflow = même résultat
- · Rollback automatique en cas d'échec déploiement

EF-2 : Code Intelligence Map (Analyse Sémantique)

Priorité: P0 (Bloquant MVP)

Description : Génération automatique d'un graphe interactif des dépendances d'un projet (classes, services, modèles, migrations).

Critères de Validation Technique :

- Parsing d'un repo Laravel standard (50 fichiers) : < 30 secondes
- Graphe stocké en format exploitable (Neo4j ou JSON Graph)
- Mise à jour incrémentale (détection changements Git)

Contraintes:

- Support multi-langage (PHP, JavaScript minimum en Phase 1)
- Scalabilité : repos jusqu'à 500 fichiers en Phase 1

EF-3: Intégrations Git Providers (GitLab/GitHub)

Priorité: P0 (Bloquant MVP)

Description : Connexion OAuth avec GitLab et GitHub pour lecture/écriture repos, création branches/MR, déclenchement pipelines.

Critères de Validation Technique :

- OAuth flow complet : < 60 secondes
- Webhooks : réception événements (push, MR) en temps réel
- Rate limiting respecté (5000 reg/h GitHub, 300 reg/min GitLab)

Contraintes :

- Stockage sécurisé tokens (encryption at rest)
- Refresh automatique tokens expirés

EF-4: LLM Router Multi-Modèles

Priorité: P1 (Post-MVP, critique Phase 2)

Description : Service intelligent routant les requêtes vers le LLM optimal (GPT-4, Mistral, Claude, Ollama) selon contexte et coûts.

Critères de Validation Technique :

- Latence décision routing : < 50ms
- Réduction coûts API: > 50% vs mono-modèle
- · Fallback automatique si modèle indisponible

Contraintes:

- · Circuit breaker (retry logic avancée)
- Monitoring coûts temps réel par modèle

EF-5: Real-Time Workflow Monitoring (WebSocket)

Priorité: P1 (Post-MVP)

Description: Dashboard temps réel affichant progression workflows via WebSocket (logs, étapes, statuts).

Critères de Validation Technique :

- Latence broadcast: < 200ms
- Support 100 connexions WebSocket concurrentes (Phase 1)
- Persistence logs: 30 jours minimum

2.2. Exigences Non-Fonctionnelles Critiques

ENF-1: Performance et Latence

Cibles Mesurables:

Endpoint/ActionLatence Cible (p95)Throughput CibleGET /api/projects< 100ms500 req/sPOST /api/workflows (création)< 200ms100 req/sWorkflow complet (analyse → deploy)< 10 min10 workflows/min (cluster)WebSocket message delivery< 200ms1000 msg/sCode Intelligence parsing< 30s (50 fichiers)N/A

Stratégies:

- Caching agressif (Redis): repos parsés, résultats LLM
- Pagination systématique (max 100 items/page)
- Database indexing optimisé (queries < 50ms)
- CDN pour assets statiques (Cloudflare)

ENF-2: Scalabilité

Modèle de Croissance :

PhaseUtilisateursWorkflows/jourInfrastructure Phase 1 1001 0002 nodes (API) + 1 node (Worker) + 1 node (DB) Phase 2 1 00010 0004 nodes (API) + 3 nodes (Worker) + 1 node (DB + replicas) Phase 3 10 000100 00010+ nodes (K8s autoscaling) + DB cluster

Stratégies de Scaling :

Horizontal Scaling:

- API Stateless (sessions Redis) → scaling linéaire
- Workers découplés (queue-based) \rightarrow ajout nodes selon backlog
- DB read replicas (PostreSQL streaming replication)

Vertical Scaling (Court terme):

- Optimisation queries (EXPLAIN ANALYZE systématique)
- · Connection pooling (PgBouncer)
- · Indexes covering queries critiques

Bottlenecks Identifiés et Solutions:

BottleneckSeuil CritiqueSolution DB Write Throughput 1000 TPSSharding par tenant_id (Phase 3) LLM API Rate Limits Variable par providerQueue prioritization + retry exponential backoff WebSocket Connections 10K connections/nodeRedis Pub/Sub + multi-node broadcast Storage (repos clonés) 1 TBS3-compatible storage + TTL cleanup (7 jours)

ENF-3: Sécurité

Modèle de Menaces (STRIDE Analysis) :

MenaceVecteur d'AttaqueContrôle de Sécurité Spoofing Token forgeryJWT signing (RS256), short TTL (1h), refresh tokens Tampering Code injection via LLMInput sanitization, output validation, sandboxed execution Repudiation Actions non-traçablesAudit logs immuables (PostgreSQL + WORM storage) Information Disclosure Tokens en clairEncryption at rest (AES-256), TLS 1.3 in transit Denial of Service Rate abuseRate limiting (Redis), CAPTCHA, WAF (Cloudflare) Elevation of Privilege RBAC bypassMulti-tenant isolation stricte, principe least privilege

Exigences Détaillées :

AUTH-1: Authentification & Autorisation

• JWT (RS256) avec rotation clés hebdomadaire

- Refresh tokens stockés hashed (bcrypt, cost 12)
- MFA obligatoire pour actions sensibles (delete project, change billing)
- RBAC: 4 rôles (Owner, Admin, Developer, Viewer)

SEC-1: Encryption

- · At Rest:
 - DB: PostgreSQL native encryption (AES-256)
 - o Secrets (API tokens): Vault ou AWS KMS
 - Backups : Encrypted (GPG)
- In Transit:
 - o TLS 1.3 obligatoire (API, WebSocket)
 - Certificate pinning (mobile apps futur)

SEC-2: Network Security

- VPC isolé (subnets privés pour DB/Workers)
- Firewall rules : whitelist IPs (API publique), deny all (DB)
- DDoS protection (Cloudflare)

SEC-3: Application Security

- OWASP Top 10 compliance
- Dependency scanning (Snyk, Dependabot)
- · Secret scanning (git-secrets, TruffleHog)
- Penetration testing (annuel en Phase 2+)

SEC-4 : Compliance (Roadmap)

- GDPR (Phase 1): Consent management, data portability, right to deletion
- SOC 2 Type II (Phase 3, M+18)
- ISO 27001 (Phase 4, optionnel)

ENF-4: Disponibilité (Availability)

SLA Cibles:

PhaseUptime SLADowntime Max/moisRTORPO | Phase 1 | 95%36 heures4h24h | Phase 2 | 99%7.2 heures1h6h | Phase 3 | 99.9%43 minutes15min1h

Stratégies High Availability:

HA-1: Redondance Infrastructure

- Multi-AZ deployment (2 zones minimum)
- Load balancer avec health checks (HAProxy/ALB)
- DB: Master-Replica avec automatic failover (Patroni/Stolon)

HA-2: Backup & Recovery

- DB: Backups quotidiens automatiques (retention 30 jours)
- Incremental backups horaires (WAL archiving PostgreSQL)
- Disaster recovery drills trimestriels

HA-3: Graceful Degradation

- · Circuit breakers (Hystrix pattern)
- Feature flags (LaunchDarkly/unleash) pour désactivation features non-critiques

• Mode dégradé : UI en read-only si backend instable

ENF-5: Observabilité et Monitoring

Stratégie Observability (3 Piliers):

1. Metrics (Prometheus + Grafana)

- Infrastructure: CPU, RAM, Disk, Network (node_exporter)
- Application : Request rate, error rate, duration (RED method)
- · Business: Workflows created, success rate, MRR

Dashboards Clés:

- · System Health (infrastructure)
- API Performance (latency, throughput, errors)
- · Workflow Analytics (success rate, avg duration)
- Business Metrics (signups, conversions, churn)

2. Logs (ELK Stack ou Loki)

- Structured logging (JSON format)
- · Correlation IDs (request tracing)
- Retention: 30 jours (logs applicatifs), 90 jours (audit logs)

3. Tracing (Jaeger/Zipkin)

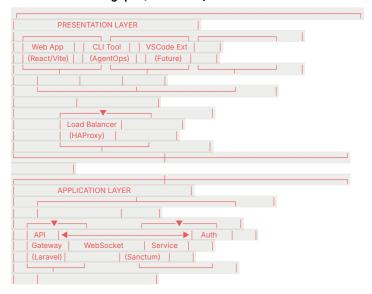
- · Distributed tracing pour workflows multi-services
- Span attribution (quel service prend du temps)
- · Critical paths analysis

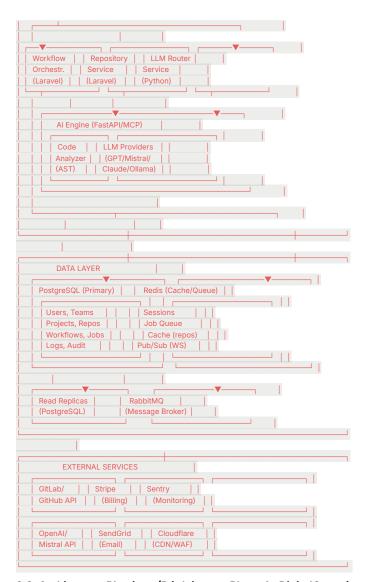
Alerting (PagerDuty/Opsgenie):

AlertConditionSeverityResponse TimeAPI Down5xx > 50% sur 2minCritical5 minHigh Latencyp95 > 1s sur 5minWarning15 minDB Connection Pool ExhaustedActive connections > 90%Critical5 minDisk Space Low< 10% freeWarning1hFailed Workflows> 20% failures sur 10minWarning15 min

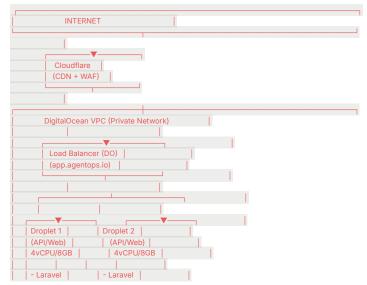
3. Diagramme d'Architecture de Haut Niveau

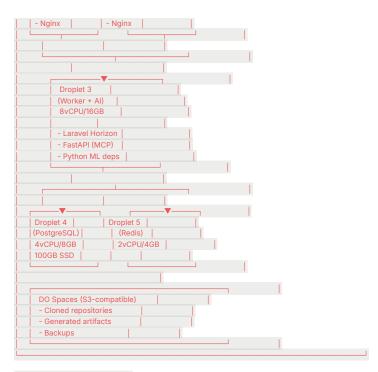
3.1. Architecture Logique (4 Couches)





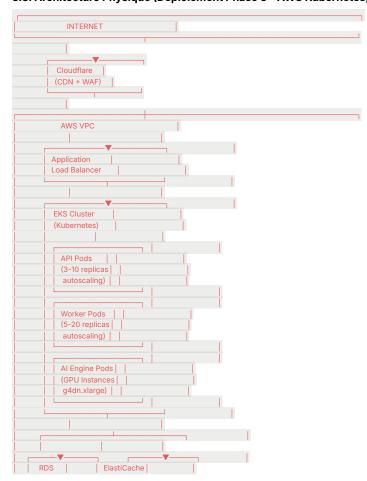
3.2. Architecture Physique (Déploiement Phase 1 - DigitalOcean)

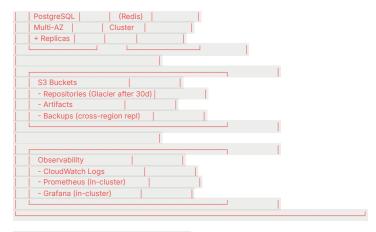




Total Cost Phase 1: ~\$400/month

3.3. Architecture Physique (Déploiement Phase 3 - AWS Kubernetes)





Total Cost Phase 3: ~\$5K-10K/month (autoscaling)

4. Choix Technologiques (Technology Stack)

4.1. Frontend

Technologie Principale: React 18 + TypeScript

Justification:

- Maturité : Écosystème le plus mature (bibliothèques, tooling, communauté)
- Performance : Virtual DOM optimisé, code splitting natif, Concurrent Mode
- Composants Réutilisables : Facilite construction design system (shadcn/ui)
- TypeScript: Type safety critique pour app complexe (réduction bugs 15-30%)
- Hiring: Largest talent pool (facilite recrutement futur)

Build Tool: Vite

- Hot Module Replacement (HMR) instantané (< 100ms)
- Build optimisé (Rollup under the hood)
- · Configuration minimale (vs Webpack)

UI Framework: Tailwind CSS + shadcn/ui

- Tailwind: Utility-first, cohérence design, bundle size optimisé (tree-shaking)
- shadcn/ui: Composants accessibles (WCAG 2.1), customisables, copy-paste friendly

State Management:

- React Query (TanStack Query): Server state (caching, refetching, optimistic updates)
- Zustand : Client state (légèr, moins boilerplate que Redux)

Routing: React Router v6

· Nested routes, lazy loading, data loaders

WebSocket Client: Socket.io-client

· Auto-reconnect, fallback polling, room management

Alternative Considérée : Next.js

- Avantages : SSR/SSG, API routes intégrées, image optimization
- Désavantages : Complexité accrue (unnecessary pour SPA), vendor lock-in (Vercel)
- Décision : Différé à Phase 2 si SEO public devient critique

4.2. Backend API

Technologie Principale: Laravel 12 (PHP 8.4)

Justification Stratégique:

- Time-to-Market: Expertise founder → 3x plus rapide que learning new stack
- Écosystème Mature :
 - Sanctum: Auth API simple (JWT alternative)
 - Horizon: Queue monitoring UI intégré
 - Telescope: Debug toolbar production-ready
 - o Cashier: Stripe integration turnkey
- Developer Experience : Migrations, seeders, factories, Eloquent ORM
- Communauté: 2e framework backend le plus populaire (après Node.js), hiring facilité
- Performance: PHP 8.4 (JIT compiler) → performance comparable à Node.js pour I/O-bound tasks

Architecture Pattern: Service-Repository

php

```
// Service Layer (business logic)
class WorkflowOrchestrationService {
    public function _construct(
        private RepositoryService $repoService,
        private LLMRouterService $llmRouter,
        private CodeAnalyzer $analyzer
    ) {}

    public function executeWorkflow(Workflow $workflow): WorkflowResult {
        // Orchestration logic
    }
}

// Repository Layer (data access)

class WorkflowRepository {
    public function findActiveByUser(User $user): Collection {
        return Workflow:where('user_id', $user→id)
        → where('status', 'active')
        → with(['steps', 'logs'])
        → get();
    }
}
```

API Design: RESTful + GraphQL (Phase 2)

- Phase 1: REST pur (simplicité)
- Phase 2 : GraphQL pour queries complexes (éviter N+1 problems)

Versioning API: Header-based (Accept: application/vnd.agentops.v1+json)

Alternative Considérée : Node.js (NestJS)

- Avantages : Même langage frontend/backend, performance async excellente
- Désavantages : Learning curve, ecosystem moins mature pour auth/billing
- Décision : Réévaluation en Phase 3 si besoin microservices purs

4.3. Al Engine

Technologie Principale: FastAPI (Python 3.12)

Justification:

- Standard ML/AI: Ecosystem Python imbattable (Langchain, transformers, tiktoken)
- Performance Async : Comparable à Node.js (event loop ASGI)

- Type Hints: Pydantic validation automatique (request/response)
- OpenAPI Auto-generation : Documentation API gratuite
- **Isolation**: Service indépendant → scaling séparé du backend principal

Architecture MCP (Model Context Protocol):

python

```
# FastAPI endpoints
@app.post("/api/ai/analyze")
async def analyze_repository(request: AnalyzeRequest) \rightarrow AnalyzeResponse:
   # AST parsing (tree-sitter)
 ast = await parse_codebase(request.repo_path)
  # Generate knowledge graph
 graph = build_dependency_graph(ast)
 return AnalyzeResponse(graph=graph, metadata=...)
@app.post("/api/ai/generate")
async def generate_code(request: GenerateRequest) \rightarrow GenerateResponse:
   # LLM Router decision
  model = llm_router.select_model(request.context, request.task_type)
   # Prompt engineering
 prompt = build_prompt(request.context, request.task)
   # LLM call with retry
 code = await call_llm_with_retry(model, prompt)
 return GenerateResponse(code=code, model_used=model)
```

ML Libraries:

- Langchain: LLM orchestration, chains, agents
- tiktoken: Token counting (cost estimation)
- tree-sitter: AST parsing multi-langage
- transformers: Local model inference (Ollama support)

Alternative Considérée : Go

- Avantages: Performance brute, concurrency native, single binary deploy
- Désavantages : Ecosystem ML inexistant
- Décision : Go pour services critiques perf (e.g. LLM Router standalone) en Phase 3

4.4. Base de Données Principale

Technologie: PostgreSQL 16

Justification:

- ACID Compliance : Critique pour facturation, workflows transactionnels
- Performance : Excellente pour queries complexes (JOINs, aggregations)
- Extensions Puissantes :
 - o pg_vector: Embeddings storage (semantic search Phase 2)
 - **pg_cron**: Scheduled jobs in-database
 - pgAudit: Audit logging conforme
- JSON Support : Flexibilité schema (workflow metadata, LLM responses)
- Mature Ecosystem: Tooling, monitoring, backup solutions

Schema Design Principles:

• Multi-Tenancy : tenant_id (team_id) sur toutes les tables → Row-Level Security (RLS)

- Soft Deletes : deleted_at nullable (GDPR right-to-erasure → hard delete après 90j)
- Audit Trail: Tables _audit miroirs (triggers automatiques)

Indexing Strategy:

sql

- Composite indexes pour queries fréquentes
CREATE INDEX idx_workflows_user_status
ON workflows(user_id, status) WHERE deleted_at IS NULL;
- Partial indexes pour queries spécifiques
CREATE INDEX idx_workflows_active
ON workflows(created_at DESC) WHERE status = 'active';
- GIN index pour JSON queries
CREATE INDEX idx_workflow_metadata
ON workflows USING GIN (metadata jsonb_path_ops);

Partitioning (Phase 3):

- Table logs partitionnée par mois (automatique via pg_partman)
- Table workflows partitionnée par team_id (10K+ teams)

Alternative Considérée : MongoDB

- Avantages : Schema flexibility, horizontal scaling natif
- Désavantages : Transactions complexes, pas de foreign keys
- Décision : PostgreSQL JSON + JSONB couvre 90% des use cases NoSQL

4.5. Cache & Queue

Technologie: Redis 7 (Cluster Mode)

Justification Multi-Usage:

- 1. Cache (GET/SET < 1ms)
- Repos parsés (TTL 1h)
- User sessions (stateless API)
- Rate limiting counters (INCR atomic)
- LLM responses (deduplication)

2. Job Queue (Laravel Horizon)

php

3. Pub/Sub (WebSocket Broadcasting)

php

```
// Laravel Event
broadcast(new WorkflowProgressUpdated($workflow));

// Redis channels
PUBLISH workflow.123.progress '{"step": "testing", "progress": 65}'
```

4. Session Store (Stateless API)

- Distributed sessions (multi-node API)
- · Token blacklist (logout/revoke)

Cluster Configuration (Phase 2+):

- 3 master nodes (sharding automatique)
- 3 replica nodes (read scaling)
- · Sentinel pour auto-failover

Persistence Strategy:

- AOF (Append-Only File): fsync every second (balance durability/perf)
- RDB snapshots : every 5 minutes (backup)

Alternative Considérée : Memcached

- Avantages : Légèrement plus rapide (single-purpose cache)
- Désavantages : Pas de persistence, pas de data structures avancées
- Décision : Redis plus polyvalent (queue + cache + pub/sub)

4.6. Message Broker

Technologie: RabbitMQ

Justification:

- Reliability: Garanties de livraison (ack/nack), persistent queues
- Dead Letter Queues: Retry logic automatique (exponential backoff)
- Routing Complex: Topic exchanges pour orchestration fine
- Management UI: Monitoring queues temps réel

Use Cases AgentOps:

1. Workflow Orchestration

```
Exchange: workflows.fanout

→ Queue: workflow.analyze (Worker 1-3)

→ Queue: workflow.generate (Worker 4-6)

→ Queue: workflow.test (Worker 7-9)

→ Queue: workflow.deploy (Worker 10)
```

2. Priority Queues

Queue: workflows.high_priority (paying users, SLA < 5min)
Queue: workflows.normal (free tier, best effort)

3. Delayed Messages (Retry Logic)

Alternative Considérée : AWS SQS/SNS

- Avantages: Fully managed, unlimited scaling
- Désavantages : Vendor lock-in, latency plus élevée (cloud API calls)
- Décision: RabbitMQ Phase 1-2 (self-hosted), migration AWS SQS Phase 3 (optionnel)

4.7. Conteneurisation & Orchestration

Phase 1-2: Docker + Docker Compose

Justification:

• Simplicité : Déploiement one-command (docker-compose up -d)

- Portabilité : Dev/staging/prod parity parfaite
- Self-Hosting: Clé pour souveraineté utilisateur

docker-compose.yml Structure:

yaml

```
services:
 image: agentops/api:latest
 deploy:
  replicas: 2
 depends_on: [postgres, redis]
worker:
 image: agentops/api:latest
 command: php artisan horizon
 deploy:
  replicas: 3
ai-engine:
 image: agentops/ai-engine:latest
 deploy:
  limits:
   cpus: '4'
   memory: 8G
postgres:
 image: postgres:16-alpine
  - postgres_data:/var/lib/postgresql/data
redis:
 image: redis:7-alpine
command: redis-server --appendonly yes
```

Phase 3: Kubernetes (AWS EKS)

Justification Migration:

- Auto-Scaling: HPA (Horizontal Pod Autoscaler) basé sur CPU/custom metrics
- Rolling Updates : Zero-downtime deploys
- Resource Optimization : Bin packing, node autoscaling
- Multi-Tenancy: Namespaces par environnement

Architecture K8s:

yaml

```
# Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
name: api
spec:
replicas: 3
 strategy:
 type: RollingUpdate
  rollingUpdate:
  maxSurge: 1
  maxUnavailable: 0
 template:
   containers:
   - name: api
    image: agentops/api:v1.2.3
    resources:
    requests:
     cpu: 500m
     memory: 1Gi
     limits:
     cpu: 2000m
     memory: 4Gi
```



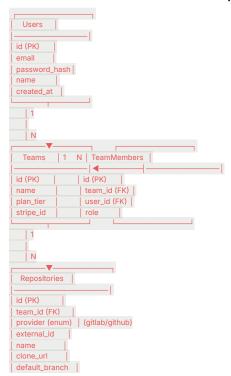
HPA Configuration:

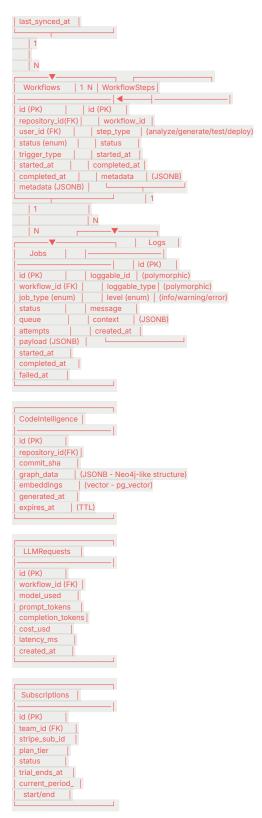
yaml



5. Architecture des Données (Data Architecture)

5.1. Schéma de Haut Niveau (Entités Principales)





5.2. Stratégie Multi-Tenancy

Approche : Shared Database, Shared Schema avec Row-Level Security (RLS) Justification :

- Phase 1-2 : Simplicité opérationnelle (1 DB, 1 schema)
- Cost-Effective: Pas de DB par tenant (overhead)
- Performance : Indexes optimisés globalement

Implementation PostgreSQL RLS:

sql

```
- Enable RLS sur table sensible

ALTER TABLE workflows ENABLE ROW LEVEL SECURITY;

- Policy: Users can only see their team's workflows

CREATE POLICY team_isolation ON workflows USING (team_id = current_setting('app.current_team_id')::int);

- Application layer (Laravel middleware)

public function handle($request, Closure $next) { $teamId = $request→user()→currentTeam→id; DB::statement("SET app.current_team_id = ?", [$teamId]); return $next($request);
```

Migration vers DB par Tenant (Phase 3, optionnel):

- Trigger: > 1000 teams ou compliance stricte (finance, health)
- Stratégie : Routing layer (team_id → DB connection pool)

5.3. Partitioning Strategy (Phase 3)

Table logs: Partitioning Temporel (Range Partitioning)

Problème: Croissance exponentielle (100M+ rows en Phase 3)

Solution: Partitions mensuelles automatiques

sql

```
- Partition parent

CREATE TABLE logs (id BIGSERIAL, loggable_type VARCHAR(255), loggable_id BIGINT, level VARCHAR(20), message TEXT, context JSONB, created_at TIMESTAMP NOT NULL

) PARTITION BY RANGE (created_at);

- Partitions automatiques (pg_partman extension)

SELECT partman.create_parent( 'public.logs', 'created_at', 'native', 'monthly');

- Retention automatique (suppression > 90 jours)

UPDATE partman.part_config

SET retention = '90 days', retention_keep_table = false

WHERE parent_table = 'public.logs';
```

Table workflows: Partitioning par Team (List Partitioning)

Trigger: > 10 000 teams

Solution: Partitions par tranche de team_id

sql

```
CREATE TABLE workflows (
id BIGSERIAL,
team_id INT NOT NULL,
-- other columns
) PARTITION BY LIST (team_id);
-- Exemple : 10 partitions pour 10K teams
CREATE TABLE workflows_p0 PARTITION OF workflows
FOR VALUES IN (SELECT generate_series(1, 1000));
CREATE TABLE workflows_p1 PARTITION OF workflows
FOR VALUES IN (SELECT generate_series(1001, 2000));
```

5.4. Backup & Recovery Strategy

Backups Automatisés

1. PostgreSQL (pg_basebackup + WAL archiving)

bash

```
# Full backup quotidien (3h du matin UTC)
0 3 * * * * pg_basebackup -D /backups/$(date +\%Y\%m\%d) -Ft -z -P
# WAL archiving continu (Point-in-Time Recovery)
archive_command = 'cp %p /wal_archive/%f'
```

Retention Policy:

- Full backups: 30 jours (local) + 90 jours (S3 Glacier)
- WAL archives: 7 jours

2. Redis (RDB + AOF)

conf

```
# RDB snapshot every 5 min if 100+ keys changed
save 300 100
# AOF fsync every second
```

3. Application Data (Repositories clonés)

- S3 Lifecycle Policy: Archive vers Glacier après 30 jours
- Cleanup automatique : Suppression repos non-utilisés > 90 jours

Disaster Recovery Plan

```
RTO (Recovery Time Objective): 1h (Phase 2), 15min (Phase 3)
RPO (Recovery Point Objective): 6h (Phase 2), 1h (Phase 3)
```

DR Runbook (Scénario : Corruption DB) :

- 1. **Detection**: Alert monitoring (Sentry/Grafana) → 5min
- 2. **Isolation**: Basculement traffic vers page maintenance → 5min
- 3. **Recovery**: Restore dernier backup + replay WAL → 30-45min
- 4. **Validation**: Smoke tests automatisés → 5min
- 5. **Switchback**: Redirection traffic vers DB restaurée → 5min

DR Drills: Trimestriel (Phase 2+)

6. Déploiement et Opérations (DevOps & Deployment)

6.1. Stratégie d'Hébergement

Phase 1-2: DigitalOcean (Managed Droplets + Spaces)

Justification:

- Simplicité : UI intuitive, docs excellentes, support réactif
- Coûts Prévisibles : Pricing flat (pas de surprises AWS)
- Performance: Datacenters worldwide, latency < 50ms (EU/US)
- Managed Services : Load Balancer, Spaces (S3-compatible), Managed DB (optionnel Phase 2)

Infrastructure Code (Terraform):

hcl

```
# terraform/main.tf
resource "digitalocean_droplet" "api" {
count = 2
name = "api-${count.index + 1}"
size = "s-2vcpu-4gb"
image = "docker-20-04"
region = "ams3"
```

```
ssh_keys = [var.ssh_key_fingerprint]
tags = ["production", "api"]
resource "digitalocean_loadbalancer" "public" {
name = "agentops-lb"
forwarding_rule {
 entry_protocol = "https"
  entry_port = 443
  target_protocol = "http"
 target_port = 80
  certificate_id = digitalocean_certificate.main.id
healthcheck {
 port = 80
 protocol = "http"
 path = "/health"
droplet_ids = digitalocean_droplet.api[*].id
resource "digitalocean_spaces_bucket" "repos" {
name = "agentops-repositories" region = "ams3"
lifecycle_rule {
 enabled = true
 expiration {
  days = 90
```

Phase 3: AWS (EKS + RDS + ElastiCache + S3)

Justification Migration:

- Scaling Illimité: HPA, Cluster Autoscaler, serverless options (Fargate)
- Services Managés Premium : RDS Multi-AZ, ElastiCache Cluster, Aurora Serverless
- Global Footprint: 30+ regions, CloudFront CDN intégré
- Compliance: SOC 2, ISO 27001, HIPAA ready (entreprise clients)

Migration Path:

- 1. M+15: Proof-of-Concept EKS (staging environment)
- 2. M+16: Blue-Green deployment (gradual traffic shift)
- 3. M+17: Full cutover (DNS switch)
- 4. M+18: Decommission DigitalOcean

AWS Architecture (Terraform):

hcl

```
# EKS Cluster
module "eks" {
    source = "terraform-aws-modules/eks/aws"
    version = "~> 19.0"
    cluster_name = "agentops-prod"
    cluster_version = "1.28"

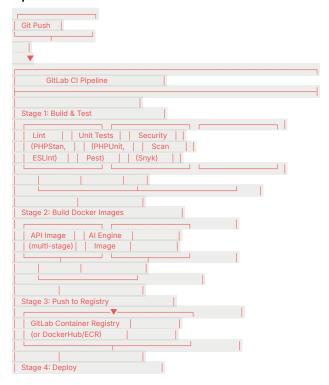
    vpc_id = module.vpc.vpc_id
    subnet_ids = module.vpc.private_subnets

    eks_managed_node_groups = {
        api = {
```

```
min_size = 3
max_size = 20
   desired_size = 5
   instance_types = ["t3.large"]
  workers = {
min_size = 5
max_size = 50
   desired_size = 10
   instance_types = ["c5.2xlarge"]
  ai_gpu = {
  min_size = 2
max_size = 10
   desired_size = 3
   instance_types = ["g4dn.xlarge"]
# RDS PostgreSQL Multi-AZ
resource "aws_db_instance" "main" {
identifier = "agentops-prod"
engine = "postgres"
engine_version = "16.1"
instance_class = "db.r6g.2xlarge"
allocated_storage = 500
max_allocated_storage = 2000 # Autoscaling storage
multi_az
                  = true
backup_retention_period = 30
backup_window = "03:00-04:00"
maintenance_window = "sun:04:00-sun:05:00"
enabled_cloudwatch_logs_exports = ["postgresql", "upgrade"]
  # Read replicas
replicate_source_db = aws_db_instance.main.id
```

6.2. CI/CD Pipeline (GitLab CI)

Pipeline Architecture



```
\mathsf{SSH} \to \mathsf{Production} \; \mathsf{Servers}
  docker-compose pull && up
Stage 5: Smoke Tests
  Health Checks (curl /health)
  Critical Flow Tests
```

.gitlab-ci.yml Configuration

yaml

```
# .gitlab-ci.yml
stages:
- build
 - test
 - security
 - build-images
  deploy
- smoke-test
DOCKER_DRIVER: overlay2
DOCKER_TLS_CERTDIR: ""
# === STAGE 1: BUILD & TEST ===
stage: build
image: php:8.4-cli
script:
 - composer install --no-dev --prefer-dist
  - vendor/bin/phpstan analyze --level=8
  - vendor/bin/php-cs-fixer fix --dry-run --diff
 cache:
 paths:
   - vendor/
only:
 - merge_requests
- main
lint:frontend:
stage: build
image: node:20-alpine
script:
 - cd frontend
  - npm ci
  - npm run lint
  - npm run type-check # TypeScript
cache:
 paths:
   - frontend/node_modules/
only:
 - merge_requests
- main
test:unit:
stage: test
image: php:8.4-fpm
services:
  - postgres:16-alpine
  - redis:7-alpine
 DB_HOST: postgres
  REDIS_HOST: redis
 script:
  - composer install
  - cp .env.testing .env
  - php artisan key:generate
  - php artisan migrate --seed
   vendor/bin/pest --coverage --min=80
 artifacts:
 reports:
```

coverage_report:

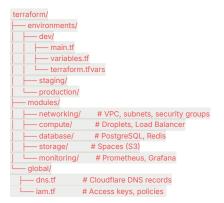
```
coverage_format: cobertura
    path: coverage.xml
coverage: '/^\s*Lines:\s*\d+.\d+\%/'
test:integration:
stage: test
image: php:8.4-fpm
  - postgres:16-alpine
  - redis:7-alpine
script:
  - composer install
 - php artisan test --testsuite=Integration
# === STAGE 2: SECURITY ===
security:scan:
stage: security
image: snyk/snyk:php
script:
 - snyk test --severity-threshold=high
  - snyk monitor # Send results to Snyk dashboard
 allow_failure: true # Don't block pipeline on medium vulns
only:
- main
secret:scan:
stage: security
image: trufflesecurity/trufflehog:latest
 - trufflehog git file://. --only-verified
only:
 - merge_requests
 - main
# === STAGE 3: BUILD DOCKER IMAGES ===
build:api:
stage: build-images
image: docker:24-dind
services:
  - docker:24-dind
before_script:
  - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
  - docker build -f docker/Dockerfile.api -t $CI_REGISTRY_IMAGE/api:$CI_COMMIT_SHA .
  - docker tag $CI_REGISTRY_IMAGE/api:$CI_COMMIT_SHA $CI_REGISTRY_IMAGE/api:latest
  - docker push $CI_REGISTRY_IMAGE/api:$CI_COMMIT_SHA
  - docker push $CI_REGISTRY_IMAGE/api:latest
only:
- main
build:ai-engine:
stage: build-images
image: docker:24-dind
services:
  - docker:24-dind
  - docker build -f docker/Dockerfile.ai -t $CI_REGISTRY_IMAGE/ai-engine:$CI_COMMIT_SHA ./ai-engine
  - docker push $CI_REGISTRY_IMAGE/ai-engine:$CI_COMMIT_SHA
only:
- main
# === STAGE 4: DEPLOY ===
deploy:production:
stage: deploy
image: alpine:latest
before_script:
  - apk add --no-cache openssh-client
  - eval $(ssh-agent -s)
- echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
  - mkdir -p ~/.ssh
  - chmod 700 ~/.ssh
  - ssh-keyscan -H $PRODUCTION_SERVER >> ~/.ssh/known_hosts
 script:
```

ssh deploy@\$PRODUCTION_SERVER << 'EOF'

```
cd /opt/agentops
    docker-compose up -d --remove-orphans
    docker system prune -f
  EOF
 environment:
 name: production
 url: https://app.agentops.io
  - main
when: manual # Manual approval required# === STAGE 5: SMOKE TESTS ===
smoke:test:
stage: smoke-test
image: curlimages/curl:latest
 - sleep 30 # Wait for containers to be healthy
  - curl --fail https://app.agentops.io/health | exit 1
  - curl --fail https://api.agentops.io/health || exit 1
only:
- main
```

6.3. Infrastructure as Code (IaC)

Terraform Structure



Example Module (Compute)

hcl

```
# modules/compute/main.tf
resource "digitalocean_droplet" "api" {
count = var.api_instance_count
name = "${var.environment}-api-${count.index + 1}"
size = var.api_instance_size
image = var.droplet_image
region = var.region
ssh_keys = var.ssh_keys
user_data = templatefile("${path.module}/user_data.sh", {
 docker_compose_version = var.docker_compose_version
  environment = var.environment
tags = [
  "environment:${var.environment}",
  "role:api"
  "managed-by:terraform"
output "api_ips" {
value = digitalocean_droplet.api[*].ipv4_address
```

6.4. Configuration Management

Approche : Docker Environment Variables + Secrets Management

1. Environment-Specific Configs (.env files)

bash

```
# .env.production

APP_ENV=production

APP_DEBUG=false

APP_URL=https://app.agentops.io

DB_CONNECTION=pgsql

DB_HOST=postgres.internal

DB_DATABASE=agentops_prod

DB_USERNAME=agentops

DB_PASSWORD=${DB_PASSWORD} # Injected by secrets manager

REDIS_HOST=redis.internal

REDIS_PASSWORD=${REDIS_PASSWORD}

# LLM API Keys (rotated monthly)

OPENAI_API_KEY=${OPENAI_API_KEY}

ANTHROPIC_API_KEY=${ANTHROPIC_API_KEY}
```

2. Secrets Management (HashiCorp Vault - Phase 2)

bash

```
# Fetch secrets at runtime
vault kv get -field=db_password secret/agentops/production/database
vault kv get -field=stripe_secret secret/agentops/production/stripe
```

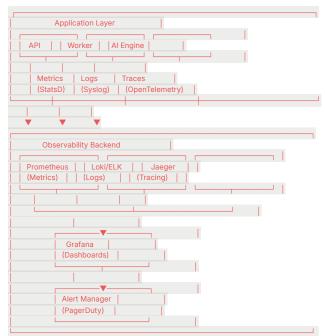
3. Feature Flags (LaunchDarkly/Unleash - Phase 2)

php

```
// Toggle features without deploy
if (Features::enabled('Ilm-router-v2')) {
    return $this→IlmRouterV2→route($request);
}
return $this→IlmRouterV1→route($request);
```

6.5. Monitoring & Alerting Stack

Architecture Observability



Prometheus Configuration

yaml

```
# prometheus.yml
scrape_interval: 15s
evaluation_interval: 15s
scrape_configs:
 - job_name: 'api'
  static_configs:
   - targets: ['api-1:9090', 'api-2:9090']
 metrics_path: '/metrics'
 - job_name: 'postgres'
  static_configs:
   - targets: ['postgres-exporter:9187']
 - job_name: 'redis'
 static_configs:
- targets: ['redis-exporter:9121']
alerting:
alertmanagers:
static_configs:
- targets: ['alertmanager:9093']
rule_files:
- 'alerts/*.yml'
```

Alert Rules

yaml

```
# alerts/api.yml
groups:
 - name: api_alerts
 interval: 30s
   - alert: APIHighErrorRate
    expr:
      rate(http_requests_total{status=~"5.."}[5m])
      rate(http_requests_total[5m])
     ) > 0.05
    for: 2m
    annotations:
     summary: "High API error rate ({{ $value | humanizePercentage }})"
     description: "API error rate is above 5% for 2 minutes"
   - alert: APIHighLatency
    expr:
     histogram_quantile(0.95,
      rate(http_request_duration_seconds_bucket[5m])
    for: 5m
    labels:
     severity: warning
    annotations:
     summary: "High API latency (p95: {{ $value }}s)"
   - alert: WorkflowFailureSpike
     rate(workflows_failed_total[10m]) > 0.2
    for: 5m
    labels:
     severity: warning
     summary: "Workflow failure rate spiking"
```

Grafana Dashboards

Dashboard 1: System Health

- CPU/RAM/Disk usage (all nodes)
- Network I/O

· Docker container health

Dashboard 2: API Performance

- Request rate (req/s)
- Error rate (%)
- Latency distribution (p50, p95, p99)
- Throughput (MB/s)

Dashboard 3: Workflow Analytics

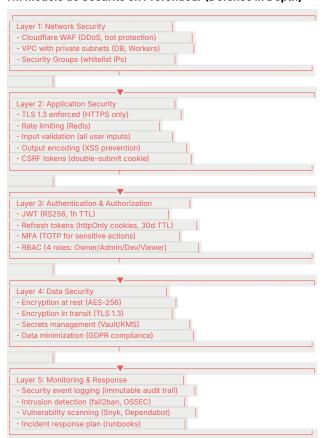
- · Workflows created/completed/failed (time series)
- · Average workflow duration
- Step-by-step breakdown (analyze/generate/test/deploy times)
- Queue depth (RabbitMQ)

Dashboard 4: Business Metrics

- Signups (daily/weekly/monthly)
- Active users (DAU/WAU/MAU)
- · MRR trend
- · Conversion funnel

7. Considérations de Sécurité

7.1. Modèle de Sécurité en Profondeur (Defense in Depth)



7.2. Authentication & Authorization (AuthN/AuthZ)

JWT Implementation

Token Structure:

```
json
```

```
// Access Token (1h TTL)
{

"header": {
    "alg": "RS256",
    "typ": "JWT"
},
    "payload": {
    "sub": "user:12345",
    "team_id": 67,
    "role": "developer",
    "permissions": ["workflows:create", "workflows:read"],
    "iat": 1698765432,
    "exp": 1698769032
},
    "signature": "..."
}
```

Token Lifecycle:

```
// Laravel AuthController
public function login(LoginRequest $request): JsonResponse {
 $credentials = $request→validated();
  if (!Auth::attempt($credentials)) {
   throw new AuthenticationException('Invalid credentials');
  $user = Auth::user();
   // Generate access token (short-lived)
  $accessToken = $user→createToken(
   name: 'access_token'
    expiresAt: now() →addHour()
  ) → plainTextToken;
   // Generate refresh token (long-lived, httpOnly cookie)
  $refreshToken = $user→createToken(
    name: 'refresh_token',
    expiresAt: now() →addDays(30)
 ) → plainTextToken;
  return response() → json([
    'access_token' ⇒ $accessToken,
    'token_type' ⇒ 'Bearer',
    'expires_in' ⇒ 3600
  ])→cookie(
    name: 'refresh_token',
    value: $refreshToken,
    minutes: 43200, // 30 days
   httpOnly: true,
    secure: true,
    sameSite: 'strict'
 );
// Token refresh endpoint
public function refresh(Request $request): JsonResponse {
  $refreshToken = $request > cookie('refresh_token');
  if (!$refreshToken) {
   throw new AuthenticationException('No refresh token');
   // Validate and rotate refresh token
  $user = PersonalAccessToken::findToken($refreshToken)?→tokenable;
  if (!$user) {
    throw new AuthenticationException('Invalid refresh token');
```

```
// Revoke old tokens
$user→tokens()→delete();
// Issue new tokens
return $this→login( /*...*/ );
```

Role-Based Access Control (RBAC)

```
Roles & Permissions Matrix:

PermissionViewerDeveloperAdminOwnerworkflows:read  vvv workflows:create  vvv workflows:delete  vvv team:invite  vvv team:invite  vvv team:remove_member  vvv v team:remove
```

7.3. Sécurisation des Secrets

return response() → json(null, 204);

HashiCorp Vault Integration (Phase 2)

Architecture:



Configuration:

```
// config/vault.php
return [
    'addr' ⇒ env('VAULT_ADDR', 'https://vault.internal:8200'),
    'token' ⇒ env('VAULT_TOKEN'), // Bootstrap token
    'mount' ⇒ 'secret',
    'path' ⇒ env('APP_ENV') . '/agentops',
];

// Service Provider
public function boot() {
    $vault = new VaultClient(config('vault'));

// Fetch secrets at boot
```

```
$secrets = $vault→read('database');

config([
    'database.connections.pgsql.password' → $secrets['password'],
    ]);
}
```

Secret Rotation (Automated):

hash

```
# Cron job (weekly)
0 2 ** 0 /usr/local/bin/rotate-secrets.sh

# rotate-secrets.sh#!/bin/bash
NEW_PASSWORD=$(openssl rand -base64 32)

# Update Vault
vault kv put secret/production/database password=$NEW_PASSWORD

# Update Database
psql -c "ALTER USER agentops PASSWORD '$NEW_PASSWORD';"

# Restart application (rolling restart)
kubect! rollout restart deployment/api
```

7.4. Protection contre les Vulnérabilités OWASP Top 10

1. Injection (SQL, Command, LDAP)

Mitigation:

• Prepared Statements (Eloquent ORM utilise PDO prepared statements)

php

```
// ✓ SAFE: Parameterized query
Workflow::where('user_id', $userId)→get();
// ✗ UNSAFE: String concatenation
DB::select("SELECT * FROM workflows WHERE user_id = " . $userId);
```

• Input Validation (Laravel Form Requests)

php

2. Broken Authentication

Mitigation:

• Rate Limiting (login attempts)

php

```
// Middleware

RateLimiter::for('login', function (Request $request) {
    return Limit::perMinute(5)→by($request→ip());
});
```

Account Lockout (after 10 failed attempts)

```
if ($user→failed_login_attempts >= 10) {
  throw new TooManyLoginAttemptsException(
   'Account locked. Contact support.'
  );
}
```

• MFA for Sensitive Actions

php

```
// Delete project requires MFA
if (l$user→verifyMFACode($request→mfa_code)) {
throw new InvalidMFACodeException();
}
```

3. Sensitive Data Exposure

Mitigation:

• Encryption at Rest (Laravel Crypt)

php

```
// Model attribute casting class User extends Model { protected $casts = [ 'api_token' ⇒ 'encrypted', 'github_access_token' ⇒ 'encrypted', } }
```

• TLS 1.3 Enforced (Nginx config)

nginx

```
server {
    listen 443 ssl http2;

    ssl_protocols TLSv1.3;
    ssl_ciphers 'TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384';
    ssl_prefer_server_ciphers off;

# HSTS
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
}
```

• Data Minimization (log only necessary data)

php

```
// X UNSAFE: Logging full request
Log::info('User login', ['request' → $request→all()]);

// SAFE: Log only safe data
Log::info('User login', [
    'user_id' → $user→id,
    'ip' → $request→ip(),
    // NO password, NO tokens
]);
```

4. XML External Entities (XXE)

Mitigation:

• Disable External Entities (si parsing XML nécessaire)

php

```
libxml_disable_entity_loader(true);

$xml = simplexml_load_string($xmlString, 'SimpleXMLElement', LIBXML_NOENT);
```

Note: AgentOps n'utilise pas XML (JSON only), risque minimisé.

5. Broken Access Control

Mitigation:

• Authorization Checks Systématiques (Laravel Policies)

```
// TOUJOURS vérifier ownership
public function show(Workflow $workflow): JsonResponse {
    if ($workflow->team_id !== auth() -> user() -> currentTeam->id) {
        abort(403, 'Unauthorized');
    }
```

```
return response()→json($workflow);
}

// OU utiliser Laravel Policy
$this→authorize('view', $workflow);
```

• Row-Level Security (PostgreSQL RLS) (cf. section 5.2)

6. Security Misconfiguration

Mitigation:

· Disable Debug Mode in Production

php

//.env.production
APP_DEBUG=false
APP_ENV=production

• Security Headers (Nginx)

nginx

```
add_header X-Frame-Options "SAMEORIGIN" always;
add_header X-Content-Type-Options "nosniff" always;
add_header X-XSS-Protection "1; mode=block" always;
add_header Referrer-Policy "strict-origin-when-cross-origin" always;
add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline';" always;
```

• Dependency Updates (automated via Dependabot)

7. Cross-Site Scripting (XSS)

Mitigation:

• Output Encoding (React escapes by default)

jsx

```
// 

SAFE: React escapes automatically

<div>{user.name}</div>

// 

WUNSAFE: dangerouslySetInnerHTML

<div dangerouslySetInnerHTML={{_html: user.bio}} />
```

• Content Security Policy (CSP) (cf. Security Headers ci-dessus)

8. Insecure Deserialization

Mitigation:

• Avoid PHP unserialize() (utiliser JSON)

php

```
// ✓ SAFE
$data = json_decode($jsonString, true);
// ✓ UNSAFE
$data = unserialize($serializedString);
```

9. Using Components with Known Vulnerabilities

Mitigation:

• Automated Scanning (CI pipeline)

yaml

```
# .gitlab-ci.yml (cf. section 6.2)
security:scan:
script:
- snyk test --severity-threshold=high
```

• Regular Updates (composer update, npm update)

10. Insufficient Logging & Monitoring

Mitigation:

- Audit Trail Complet (cf. section 6.5)
- · Security Events Logging

php

```
// Log security events
Log::channel('security')→warning('Failed login attempt', [
    'ip' ⇒ $request→ip(),
    'email' ⇒ $request→email,
    'user_agent' ⇒ $request→userAgent(),
]);
```

• Alerting sur Anomalies (Sentry, Prometheus Alerts)

8. Risques et Solutions de Mitigation au niveau Technique

8.1. Matrice des Risques Techniques

IDRisqueProbabilitéImpactScoreMitigation RT-1 Performance LLM API (latency spikes)Élevée (70%)Majeur P1 LLM Router + circuit breaker + fallback cache RT-2 DB bottleneck (writes)Moyenne (40%)Majeur P1 Connection pooling, read replicas, sharding (Phase 3) RT-3 Vendor lock-in (cloud provider)Faible (20%)Critique P2 Infrastructure as Code (Terraform), Docker (portabilité) RT-4 Data loss (corruption DB)Faible (10%)Critique P1 Backups automatisés, PITR, DR drills RT-5 Security breach (API tokens leaked)Moyenne (30%)Critique P0 Secret scanning, rotation automatique, Vault RT-6 Scaling costs (unexpected growth)Moyenne (50%)Majeur P2 Autoscaling policies, cost monitoring, usage-based pricing RT-7 Third-party API downtime (GitLab/GitHub)Moyenne (40%)Modéré P2 Retry logic, circuit breaker, graceful degradation RT-8 Concurrency bugs (race conditions)Faible (15%)Majeur P2 Database transactions, locks, idempotency keys RT-9 Technical debt accumulationÉlevée (80%)Modéré P2 Code reviews, refactoring sprints (10% time), tech radar RT-10 Monitoring blind spotsMoyenne (50%)Majeur P1 Comprehensive observability (metrics/logs/traces), chaos engineering

8.2. Détails des Mitigations Prioritaires

RT-1: Performance LLM API (Latency Spikes)

Problème:

- LLM APIs (OpenAI, Anthropic) ont des latences variables (500ms-10s)
- Rate limits stricts (10 reg/min pour certains tiers)
- · Coûts explosifs si mal optimisé

Solution Multi-Couche:

1. LLM Router Intelligent

python

```
# ai-engine/Ilm_router.py
class LLMRouter:
 def select_model(self, context: dict, task_type: str) → str:
     # Decision tree based on context
    if task_type == "code_generation" and context["complexity"] == "low":
     return "mistral-7b" # Cheaper, faster
    elif task_type == "code_generation" and context["complexity"] == "high":
     return "gpt-4-turbo" # More accurate
    elif task_type == "refactor"
     return "claude-3-haiku" # Good balance
    return "gpt-3.5-turbo" # Default
 async def call_with_fallback(self, model: str, prompt: str) → str:
      return await self.providers[model].generate(prompt)
    except RateLimitError:
       # Fallback to cheaper model
      fallback_model = self.get_fallback(model)
      return await self.providers[fallback_model].generate(prompt)
    except TimeoutError:
       # Use cached similar prompt result
      return self.cache.get_similar(prompt)
```

2. Circuit Breaker Pattern

python

```
from pybreaker import CircuitBreaker

IIm_breaker = CircuitBreaker(
    fail_max=5,
    timeout_duration=60, # Open circuit for 60s after 5 failures
    expected_exception=LLMAPIError
)

@IIm_breaker
async def call_IIm_api(model: str, prompt: str) → str:
    # API call
pass
```

3. Response Caching (Redis)

python

```
# Cache similar prompts (cosine similarity > 0.95)
cache_key = f"llm:{hash_prompt(prompt)}"
if cached := redis.get(cache_key):
    return cached

response = await call_llm_api(model, prompt)
redis.setex(cache_key, 3600, response) # TTL 1h
```

RT-5: Security Breach (API Tokens Leaked)

Problème:

- Tokens GitLab/GitHub/Stripe commités par erreur en Git
- Tokens exposés dans logs/errors
- Tokens volés via XSS/phishing

Solution Multi-Couche:

1. Secret Scanning (Prevention)

yaml

```
# .gitlab-ci.yml
secret:scan:
image: trufflesecurity/trufflehog:latest
script:
- trufflehog git file://. --only-verified --fail
# Bloque merge si secrets détectés
```

2. Token Rotation Automatique

bash

```
# Cron job (tous les 30 jours)

0 0 1 * * /usr/local/bin/rotate-github-token.sh

# rotate-github-token.sh#!/bin/bash# 1. Generate new token via GitHub API

NEW_TOKEN=$(curl -X POST https://api.github.com/app/installations/.../access_tokens)

# 2. Update Vault

vault kv put secret/prod/github token=$NEW_TOKEN

# 3. Rollling restart (zero downtime)
kubectl rollout restart deployment/api

# 4. Revoke old token (grace period 24h)
sleep 86400 && curl -X DELETE https://api.github.com/installations/.../tokens/$OLD_TOKEN
```

3. Incident Response Plan

Scénario: GitHub token leaked on public repo

ÉtapeActionResponsableSLA1. DetectionAlert (GitHub Secret Scanning, TruffleHog)Automated < 5min2. ContainmentRe voke token immédiatement via GitHub APIOn-call engineer < 15min3. EradicationPurge Git history (BFG Repo-Cleaner)E

ngineer< 1h4. RecoveryGenerate + deploy new tokenEngineer< 30min5. Post-MortemDocument incident, update proce duresTeam< 48h

RT-10: Monitoring Blind Spots

Problème:

- Métriques collectées mais pas actionnables
- · Alerts trop bruyantes (alert fatigue)
- Pas de visibilité end-to-end (workflow complet)

Solution: Observability-Driven Development

1. Distributed Tracing (Jaeger)

php

```
// Laravel Middleware
public function handle($request, Closure $next) {
     $tracer = app(Tracer::class);
      $span = $tracer→startSpan('http.request', [
             'http.method' \Rightarrow $request\rightarrowmethod(),
               \verb|'http.url'| \Rightarrow \verb| $request \rightarrow fullUrl()|,
      $response = $next($request);
      $span→setTag('http.status_code', $response→status());
      $span→finish();
      return $response;
 // Trace workflow complet
public function executeWorkflow(Workflow $workflow) {
      $span = $tracer→startSpan('workflow.execute');
      \adjust{sanalyzeSpan} = \adj
      $this→analyzeRepository($workflow→repository);
      $analyzeSpan→finish();
      $generateSpan = $tracer→startSpan('workflow.generate', ['child_of' → $span]);
      $this→generateCode($workflow);
      $generateSpan→finish();
         // ... test, deploy spans
      $span→finish();
```

2. SLO-Based Alerting (vs threshold-based)

yaml

```
# Prometheus alert rule
groups:
- name: slo_alerts
rules:
# SLO: 99% of API requests < 500ms
- alert: SLOViolation_APILatency
expr: |
(
histogram_quantile(0.99,
rate(http_request_duration_seconds_bucket[5m])
) > 0.5
)
for: 10m # Tolerate brief spikes
annotations:
summary: "SLO violation: 99th percentile latency > 500ms"
```

3. Business Metrics Dashboards

sql

```
- Custom Grafana query (Postgres datasource)
SELECT date_trunc('hour', created_at) as time, COUNT(*) FILTER (WHERE status = 'completed') as successful, COUNT(*) FILTER (WHERE status = 'failed') as failed,
 AVG(EXTRACT(EPOCH FROM (completed_at - started_at))) as avg_duration_seconds
 FROM workflows
 WHERE created_at > now() - interval '24 hours'
 GROUP BY 1
ORDER BY 1;
## 9. Plan de Migration et Évolution de l'Architecture
 ### Phase 1 → Phase 2 (Mois 3-9
 **Changements Architecturaux :**
  Composant | État Actuel (Phase 1) | État Cible (Phase 2) | Migration Path |
  **Hébergement** | DO Droplets (5 nodes) | DO Managed DB + Droplets (8 nodes) | - Migrate DB to Managed PostgreSQL<br/>br>- Add read replicas |
  **Caching** | Redis single-node | Redis Cluster (3 masters, 3 replicas) | - Setup cluster<br/>
- Gradual traffic shift |
  **Secrets** | .env files | HashiCorp Vault | - Deploy Vault<br>- Migrate secrets progressively
  **Monitoring** | Basic (Prometheus + Grafana) | Full stack (+ Jaeger, ELK) | - Add tracing instrumentation<br>- Deploy ELK stack |
  **CI/CD** | GitLab CI (basic) | GitLab CI + canary deploys | - Implement blue-green<br/>- Add smoke tests |
 ### Phase 2 → Phase 3 (Mois 9-18)
 **Migration vers AWS EKS :**
 **Semaine 1-4: Proof of Concept**
 - Deploy staging environment sur EKS
 - Load testing (compare perf DO vs AWS)
 - Cost analysis (TCO 12 mois)
 **Semaine 5-8 : Préparation**
 - Terraform infrastructure AWS
 CI/CD pipeline adapté (EKS deploy)
 - Runbooks migration
  *Semaine 9-12: Migration Progressive**
 Week 9: 10% traffic → AWS (canary users)
 Week 10: 30% traffic → AWS
 Week 11: 70% traffic → AWS
 Week 12: 100% traffic → AWS
```

Rollback Plan:

- DNS TTL court (5 min)
- · Health checks automated
- · Red button: instant switch back to DO

10. Annexes

10.1. Glossaire Technique

TermeDéfinitionAST (Abstract Syntax Tree)Représentation arborescente du code source, utilisée pour l'analyse sémant iqueCircuit BreakerPattern de résilience : coupe automatiquement les appels à un service défaillantHPA (Horizontal Pod Autoscaler)Kubernetes : scaling automatique basé sur métriques (CPU, custom)PITR (Point-in-Time Recovery)Restaura tion DB à un instant précis (via WAL replay)RLS (Row-Level Security)PostgreSQL : politiques de sécurité au niveau ligne (isolation multi-tenant)RBAC (Role-Based Access Control)Modèle de contrôle d'accès basé sur rôles utilisateurSLO (Ser vice Level Objective)Objectif quantifiable (ex: 99% requêtes < 500ms)WAL (Write-Ahead Logging)PostgreSQL : journali sation pour durabilité et réplication

10.2. Références et Standards

Sécurité:

- OWASP Top 10 (2021): https://owasp.org/www-project-top-ten/
- NIST Cybersecurity Framework : https://www.nist.gov/cyberframework

Architecture:

• 12-Factor App : https://12factor.net/

• Microservices Patterns (Chris Richardson): https://microservices.io/patterns/

DevOps:

- GitLab CI Best Practices : https://docs.gitlab.com/ee/ci/
- Kubernetes Production Best Practices : https://learnk8s.io/production-best-practices

10.3. Outils et Services Recommandés

CatégorieOutilUsageCoût (Phase 1)HostingDigitalOceanInfrastructure\$400/moisDNS/CDNCloudflareWAF, CDN, DNSGr atuit (Pro: \$20/mois)MonitoringSentryError tracking\$26/moisMonitoringGrafana CloudMetrics (alternative self-hosted) Gratuit (< 10K metrics)SecretsVault (self-hosted Phase 2)Secrets management\$0 (self-hosted)CI/CDGitLabCI/CD pipeli nesGratuit (self-hosted runners)EmailSendGridTransactional emails\$15/mois (40K emails)PaymentsStripeBilling2.9% + \$0.30/transaction

Total Phase 1: ~\$500/mois

11. Validation et Prochaines Étapes

11.1. Checklist de Validation Architecture

☐ Per	rformance: Load testing confirme targets latence/throughput
☐ Séc	curité : Penetration testing (Phase 2) ou audit code (Phase 1)
☐ Sca	alabilité : Stress testing confirme capacité 10x traffic
☐ Ob	servability: Dashboards couvrent 100% des flows critiques
□ DR	: Disaster recovery drill réussi (RTO/RPO respectés)
☐ Doo	cumentation : Runbooks à jour pour incidents courants
☐ Cos	st: Coûts infrastructure < 20% MRR

11.2. Prochaines Étapes (Action Items)

Semaine 1-2 (Immédiat) :

- 1. Setup repository Git + infrastructure IaC (Terraform)
- 2. Provision DigitalOcean droplets (dev environment)
- 3. Deploy PostgreSQL + Redis + RabbitMQ (Docker Compose)
- 4. Configure CI/CD pipeline GitLab (lint + tests)

Semaine 3-4 (Sprint 1):

- 5. Implémenter Auth API (Laravel Sanctum)
- 6. Créer schéma DB initial (migrations + seeders)
- 7. Setup monitoring basic (Prometheus + Grafana)
- 8. Deploy staging environment

Mois 2 (Sprint 2-3):

- 9. Intégrations GitLab/GitHub (OAuth + webhooks)
- 10. Al Engine FastAPI (endpoints /analyze, /generate)
- 11. Workflow orchestration (Laravel jobs + RabbitMQ)

Mois 3 (Sprint 4 + Launch):

- 12. Frontend React (dashboard + workflow viewer)
- 13. Stripe integration (billing)
- 14. Security hardening (penetration testing)
- 15. MVP Launch (Product Hunt)

12. Conclusion

Synthèse Exécutive

L'architecture proposée pour AgentOps combine **pragmatisme** (Laravel + PostgreSQL éprouvés) et **innovation** (Al Engine FastAPI, LLM Router). Elle est conçue pour :

- 1. Livrer Rapidement : MVP en 30 jours (stack familière, Docker Compose simplifié)
- 2. Scaler Progressivement: De 100 users (DO Droplets) à 10K+ (AWS EKS) sans refonte
- 3. Maintenir la Sécurité : Defense in depth, secrets management, audit trail complet
- 4. Rester Abordable: \$500/mois (Phase 1) → \$10K/mois (Phase 3) avec autoscaling

Risques Résiduels Acceptés

- Technical Debt : Certains raccourcis Phase 1 (pas de Vault, monitoring basique) seront remboursés Phase 2
- Vendor Lock-in Partiel : Laravel écosystème (acceptable car open-source + large communauté)
- Complexité Opérationnelle : Croît avec scale (mitigé par IaC + runbooks)

Facteurs de Succès Critique

- 1. Obsession Observability: Instrumenter dès le début (pas après les incidents)
- 2. Security by Design: Pas une "feature" Phase 2, mais un principe foundational
- 3. Iterate Based on Data: Load testing réguliers, pas d'assumptions sur perf

Document préparé par : Lead Software Architect (Al Assistant)

Date: 23 octobre 2025

Version: 1.0

Statut : Architecture approuvée — Implémentation en cours

Prochaine Revue: Post-MVP (M+3), puis trimestrielle

Annexe Finale: Architecture Decision Records (ADRs) Template

Pour chaque décision architecturale majeure future, utiliser ce template :

markdown

```
# ADR-001: Choix de PostgreSQL pour Base de Données Principale
**Date:** 2025-10-23
**Statut:** Accepté
**Décideurs:** Lead Architect, CTO
## Contexte
Besoin d'une base de données relationnelle pour workflows, users, audit logs.
PostgreSQL 16 (vs MySQL, MongoDB)
## Conséquences
**Positives:**
Extensions (pg_vector, pg_cron)
- Excellent pour queries complexes
**Négatives:**
- Scaling horizontal plus complexe que MongoDB
- Nécessite expertise DBA (Phase 3)
## Alternatives Considérées
- MySQL : Moins de features avancées
```

- MongoDB: Transactions complexes, pas de FK

Réessayer