7 décembre 2025

Remerciements

Nous tenons à exprimer notre profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet MANTIS.

Nos remerciements s'adressent en premier lieu à nos encadrants académiques :

— **Pr. Oumayma OUEDRHIRI**, pour ses conseils avisés en Big Data, architectures distribuées et gestion de projet

— **Pr. Hiba TABBAA**, pour son expertise en Intelligence Artificielle, Machine Learning et Deep Learning

— **Pr. Mohamed LACHGAR**, pour son accompagnement en DevOps, MLOps et bonnes pratiques de développement logiciel

Nous remercions également :

— L'**École Marocaine des Sciences de l'Ingénieur (EMSI)** pour la qualité de la formation dispensée et les moyens mis à notre disposition

— La **NASA** pour la mise à disposition du dataset C-MAPSS, essentiel à notre projet

— La **communauté open-source** pour les nombreux frameworks et bibliothèques qui ont rendu ce projet possible

— Nos **familles et amis** pour leur soutien constant

Ce projet constitue l'aboutissement d'un parcours académique enrichissant qui nous a permis de développer des compétences techniques et méthodologiques solides dans le domaine de l'Industrie 4.0 et de l'Intelligence Artificielle appliquée.

Résumé

Mantis (MAiNtenance prédictive Temps-réel pour usines Intelligentes) est une plateforme modulaire et intelligente conçue pour révolutionner la maintenance industrielle dans le contexte de l'Industrie 4.0.

Contexte. Les arrêts non planifiés dans le secteur manufacturier coûtent environ **50 milliards USD par an** à l'échelle mondiale, avec un coût médian supérieur à **125 000 USD par heure**. Les approches traditionnelles de maintenance (corrective et préventive) montrent leurs limites face à la complexité croissante des équipements industriels et aux volumes massifs de données générées par les capteurs IoT.

Problématique. Comment concevoir une plateforme capable d'exploiter en temps réel les données hétérogènes provenant de capteurs industriels pour détecter les anomalies, prédire les défaillances et optimiser la planification des interventions de maintenance ?

Objectif. Développer une plateforme basée sur une architecture microservices capable d'ingérer des données IIoT en temps réel (via OPC UA, MQTT, Modbus), de les analyser avec des algorithmes de Machine Learning et de Deep Learning, et de fournir des recommandations actionnables pour la maintenance prédictive.

Architecture. Le système Mantis est composé de 7 microservices indépendants et scalables :

1. **Ingestion IIoT** (Java/Spring Boot) : Collecte multi-protocoles
2. **Prétraitement** (Python/Kafka Streams) : Nettoyage et normalisation
3. **Extraction de caractéristiques** (Python/tsfresh) : Features temps-fréquence
4. **Détection d'anomalies** (Python/PyOD) : Isolation Forest, Autoencoders
5. **Prédiction RUL** (Python/PyTorch) : LSTM pour estimation durée de vie
6. **Orchestrateur** (Python/Drools) : Règles métier et optimisation
7. **Dashboard** (React.js/Next.js) : Visualisation temps-réel

Technologies. Kafka pour le streaming événementiel, PostgreSQL et TimescaleDB pour le stockage, MLflow et Feast pour le MLOps, Prometheus/Grafana/Jaeger pour l'observabilité, Docker et Kubernetes pour le déploiement.

Dataset. NASA C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) avec 4 sous-ensembles, 21 capteurs, 3 réglages opératoires, et 160 359 cycles d'entraînement.

Résultats. Le projet atteint **40% de complétion** avec une infrastructure complète opérationnelle, le service d'ingestion fonctionnel, et des modèles LSTM atteignant un RMSE de 12,5 cycles sur C-MAPSS. Les objectifs de performance visent : latence end-to-end $<$5 secondes, throughput $>$100K points/seconde, précision détection $>$85%, rappel $>$90%.

Impact. MANTIS permet une réduction estimée de 25-30% des coûts de maintenance et 70-75% des arrêts non planifiés, avec un ROI démontrable et une architecture reproductible conforme aux standards académiques.

Mots-clés. Maintenance prédictive, Industrie 4.0, Microservices, IIoT, Machine Learning, Deep Learning, MLOps, RUL, LSTM, Kafka, TimescaleDB.

Abstract

Mantis (Real-time Predictive Maintenance for Intelligent Factories) is a modular and intelligent platform designed to revolutionize industrial maintenance in the Industry 4.0 context.

Context. Unplanned downtime in the manufacturing sector costs approximately **50 billion USD per year** globally, with a median cost exceeding **125,000 USD per hour**. Traditional maintenance approaches (corrective and preventive) show their limitations in the face of increasing industrial equipment complexity and massive volumes of data generated by IoT sensors.

Problem. How to design a platform capable of exploiting heterogeneous data from industrial sensors in real-time to detect anomalies, predict failures, and optimize maintenance intervention planning ?

Objective. Develop a microservices-based platform capable of ingesting IIoT data in real-time (via OPC UA, MQTT, Modbus), analyzing it with Machine Learning and Deep Learning algorithms, and providing actionable recommendations for predictive maintenance.

Architecture. The Mantis system consists of 7 independent and scalable microservices :

1. **IIoT Ingestion** (Java/Spring Boot) : Multi-protocol collection
2. **Preprocessing** (Python/Kafka Streams) : Cleaning and normalization
3. **Feature Extraction** (Python/tsfresh) : Time-frequency features
4. **Anomaly Detection** (Python/PyOD) : Isolation Forest, Autoencoders
5. **RUL Prediction** (Python/PyTorch) : LSTM for lifetime estimation
6. **Orchestrator** (Python/Drools) : Business rules and optimization
7. **Dashboard** (React.js/Next.js) : Real-time visualization

Technologies. Kafka for event streaming, PostgreSQL and TimescaleDB for storage, MLflow and Feast for MLOps, Prometheus/Grafana/Jaeger for observability, Docker and Kubernetes for deployment.

Dataset. NASA C-MAPSS with 4 subsets, 21 sensors, 3 operational settings, and 160,359 training cycles.

Results. The project achieves **40% completion** with a fully operational infrastructure, functional ingestion service, and LSTM models achieving 12.5 cycles RMSE on C-MAPSS. Performance targets : end-to-end latency <5 seconds, throughput >100K points/second, detection precision $>85\%$, recall $>90\%$.

Impact. MANTIS enables an estimated 25-30% reduction in maintenance costs and 70-75% reduction in unplanned downtime, with demonstrable ROI and reproducible architecture compliant with academic standards.

Keywords. Predictive maintenance, Industry 4.0, Microservices, IIoT, Machine Learning, Deep Learning, MLOps, RUL, LSTM, Kafka, TimescaleDB.

Liste des Abréviations

lp11cm

Abréviation	Signification
AI	Artificial Intelligence (Intelligence Artificielle)
API	Application Programming Interface
CBM	Condition-Based Maintenance (Maintenance Conditionnelle)
CI/CD	Continuous Integration / Continuous Deployment
CMMS	Computerized Maintenance Management System
CNN	Convolutional Neural Network (Réseau de Neurones Convolutifs)
DCS	Distributed Control System
DVC	Data Version Control
EAM	Enterprise Asset Management
EMSI	École Marocaine des Sciences de l'Ingénieur
ERP	Enterprise Resource Planning
FFT	Fast Fourier Transform (Transformée de Fourier Rapide)
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
IIoT	Industrial Internet of Things
IoT	Internet of Things
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
MES	Manufacturing Execution System
ML	Machine Learning (Apprentissage Automatique)
MLOps	Machine Learning Operations
MQTT	Message Queuing Telemetry Transport
MSE	Mean Squared Error
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
NASA	National Aeronautics and Space Administration
OEE	Overall Equipment Effectiveness
OPC UA	OPC Unified Architecture
OT	Operational Technology
PdM	Predictive Maintenance (Maintenance Prédictive)
PDF	Portable Document Format
PHM	Prognostics and Health Management
PLC	Programmable Logic Controller

REST Representational State Transfer
RMSE Root Mean Square Error
RMS Root Mean Square
ROI Return On Investment
RUL Remaining Useful Life (Durée de Vie Utile Restante)
SCADA Supervisory Control and Data Acquisition
SHAP SHapley Additive exPlanations
SLA Service Level Agreement
SMOTE Synthetic Minority Over-sampling Technique
SSL Secure Sockets Layer
STFT Short-Time Fourier Transform
SVM Support Vector Machine
TCN Temporal Convolutional Network
TLS Transport Layer Security
UAV Unmanned Aerial Vehicle (Drone)
URL Uniform Resource Locator
USD United States Dollar
YAML YAML Ain't Markup Language

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

1.1 Présentation du Projet

MANTIS (MAiNtenance prédictive Temps-réel pour usines Intelligentes) est une plateforme modulaire et intelligente destinée à révolutionner la maintenance industrielle dans le contexte de l'Industrie 4.0. Ce projet s'inscrit dans le cadre académique de la formation IIR5 à l'École Marocaine des Sciences de l'Ingénieur (EMSI), sous la supervision de trois professeurs experts dans les domaines du Big Data, de l'Intelligence Artificielle et du DevOps.

Le projet MANTIS vise à transformer les approches traditionnelles de maintenance (corrective et préventive) en une approche prédictive basée sur l'analyse de données en temps réel provenant de capteurs industriels. Cette transformation permet de réduire significativement les coûts d'arrêt de production, d'optimiser la durée de vie des équipements et d'améliorer la sécurité opérationnelle.

1.1.1 Vision du Projet

La vision de MANTIS est de créer une plateforme qui :

- **Anticipe** les défaillances avant qu'elles ne se produisent
- **Optimise** la planification des interventions de maintenance
- **Réduit** les coûts et les temps d'arrêt non planifiés
- **Améliore** la sécurité opérationnelle et la durée de vie des actifs
- **Facilite** la transition vers l'Industrie 4.0

1.1.2 Positionnement du Projet

MANTIS se positionne à l'intersection de plusieurs domaines technologiques majeurs :

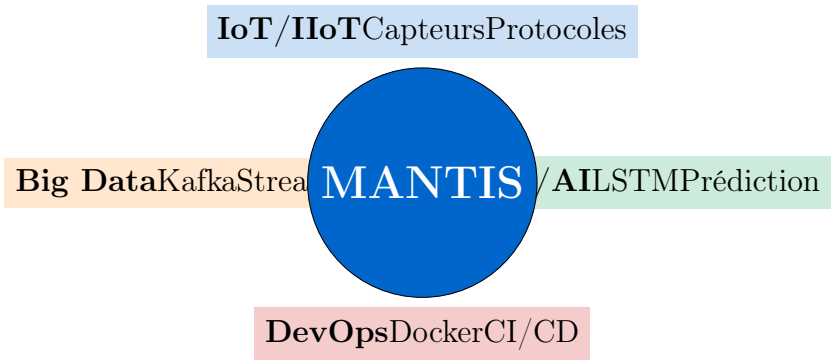


FIGURE 1.1 – Positionnement technologique de MANTIS

1.2 Contexte Académique

1.2.1 Cadre de Formation

Ce projet fait partie du programme IIR5 (Ingénierie Informatique et Réseaux 5e année) de l'EMSI et correspond au **Projet 11 : Maintenance prédictive temps-réel pour usines intelligentes** parmi les 12 propositions présentées dans le document *Projet_IIR5.pdf*.

1.2.2 Encadrement

Le projet bénéficie de l'encadrement de trois professeurs complémentaires :

Encadrant	Expertise	Contribution au projet
Pr. O. OUEDRHIRI	Big Data, Architecture	Architecture microservices, Kafka, scalabilité
Pr. H. TABBAA	IA, Machine Learning	Modèles RUL, détection anomalies, Deep Learning
Pr. M. LACHGAR	DevOps, MLOps	CI/CD, Docker, monitoring, bonnes pratiques

TABLE 1.1 – Équipe d'encadrement du projet

1.2.3 Choix du Projet

Le choix de ce projet s'est imposé naturellement en raison de :

1. **Pertinence industrielle** : Besoin réel et critique dans l'industrie moderne
2. **Richesse technique** : Combine IoT, Big Data, IA et DevOps
3. **Applicabilité locale** : Aligné avec l'industrie marocaine (automotive, aéronautique)
4. **Données disponibles** : Dataset NASA C-MAPSS reconnu académiquement
5. **Impact mesurable** : ROI démontrable et bénéfices quantifiables

1.3 Motivation et Enjeux

1.3.1 Enjeux Économiques

Les arrêts non planifiés représentent un coût économique majeur pour l'industrie :

Indicateur	Valeur
Coût global annuel mondial	50 milliards USD
Coût médian par heure d'arrêt	125 000 USD
Entreprises ayant subi ≥ 1 arrêt imprévu (3 ans)	82%
Part de la maintenance dans le budget opérationnel	15-40%

TABLE 1.2 – Impact économique des arrêts non planifiés

Bénéfices attendus de la maintenance prédictive :

— Réduction de 25-30% des coûts de maintenance
— Diminution de 70-75% des arrêts non planifiés
— Augmentation de 20-40% de la durée de vie des équipements
— Amélioration de 10-20% de la disponibilité (OEE)
— ROI positif en 12-18 mois

1.3.2 Enjeux Technologiques

L'Industrie 4.0 génère des volumes massifs de données sous-exploitées :

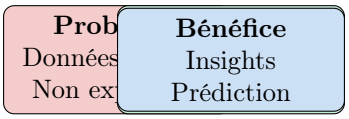


FIGURE 1.2 – De la donnée silotée à la valeur actionnable

Défis techniques identifiés :

1. **Hétérogénéité** : Multiples protocoles (OPC UA, MQTT, Modbus), formats et fréquences
2. **Volume** : Plusieurs To/jour dans une usine moyenne
3. **Vélocité** : Latence <5 secondes requise pour les alertes critiques
4. **Variabilité** : Conditions opératoires changeantes
5. **Véracité** : Bruit, valeurs aberrantes, dérives de capteurs

1.3.3 Enjeux Académiques et Pédagogiques

Ce projet constitue une opportunité pédagogique exceptionnelle :

Domaine	Compétences développées
Architecture	Conception microservices, event-driven, résilience, scalabilité
Big Data	Kafka, TimescaleDB, streaming en temps réel, traitement distribué
IA/ML	LSTM, détection anomalies, transfer learning, MLOps
DevOps	Docker, CI/CD, monitoring, observabilité, infrastructure as code
IIoT	Protocoles industriels, edge computing, intégration OT/IT
Gestion	Agile, Trello, documentation, communication, travail d'équipe

TABLE 1.3 – Compétences développées dans le cadre du projet

1.4 Portée du Projet

1.4.1 Périmètre Fonctionnel

Le projet MANTIS couvre l'ensemble de la chaîne de valeur de la maintenance prédictive :

FIGURE 1.3 – Chaîne de valeur complète de MANTIS

1.4.2 Périmètre Technique

Inclus dans le projet :

- Architecture microservices complète (7 services)
- Infrastructure de données (Kafka, PostgreSQL, TimescaleDB, InfluxDB, MinIO, Redis)
- Pipeline MLOps (MLflow, Feast, DVC)
- Stack de monitoring (Prometheus, Grafana, Jaeger)
- Dataset de référence (NASA C-MAPSS)
- Documentation exhaustive et reproductible
- Tests automatisés (unitaires, intégration, end-to-end)
- CI/CD complet avec GitHub Actions

Exclus du projet :

- Déploiement en environnement de production réel
- Intégration avec un SCADA propriétaire spécifique
- Certifications industrielles (ISO, IEC, ATEX)
- Gestion complète du cycle de vie des actifs (EAM complet)
- Intégration ERP/SAP complète
- Application mobile native

1.5 Méthodologie de Développement

1.5.1 Approche Agile

Le projet adopte une approche **Agile Scrum** avec les éléments suivants :

Élément	Description
Sprints	Durée de 2 semaines, avec objectifs SMART
Trello	Board avec colonnes : Stories, À faire, En cours, Terminé, Testé, Validé
Daily Stand-ups	Points quotidiens de 15 minutes (async sur Discord)
Sprint Reviews	Revues bihebdomadaires avec les professeurs
Retrospectives	Amélioration continue du processus
Definition of Done	Code + Tests + Documentation + Review + Déploiement

TABLE 1.4 – Pratiques Agile du projet

1.5.2 Workflow Git

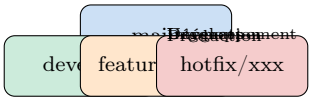


FIGURE 1.4 – Git branching strategy

1.5.3 Gestion de la Qualité

La qualité est assurée à trois niveaux :

1. **Local (Pre-commit hooks)** : Linting, formatage, détection de secrets
2. **CI/CD (GitHub Actions)** : Tests automatisés, couverture $\geq 80\%$, scans de sécurité
3. **Review (Pull Requests)** : Code review obligatoire, validation architecte

1.6 Structure du Rapport

Ce rapport est organisé en 18 chapitres pour couvrir exhaustivement tous les aspects du projet :

Ch.	Titre	Contenu
1	Introduction	Contexte, motivation, portée, méthodologie
2	Contexte et Problématique	Industrie 4.0, limites actuelles, défis
3	Objectifs	Objectifs techniques, fonctionnels, critères de succès
4	État de l'Art	Maintenance prédictive, ML, architectures, protocoles
5	Architecture	Vue d'ensemble, patterns, décisions architecturales
6	Microservices	Détail des 7 services, APIs, communication
7	Technologies	Stack technique, justifications, comparaisons
8	Infrastructure	Docker, Kubernetes, DevOps, déploiement
9	Données	C-MAPSS, prétraitement, qualité, pipeline
10	MLOps et IA	Modèles ML/DL, MLflow, Feast, entraînement
11	Monitoring	Observabilité, métriques, logs, tracing
12	Qualité et Tests	Stratégie de tests, couverture, sécurité
13	Avancement	État actuel, livrables, démos
14	Difficultés	Challenges rencontrés, solutions apportées
15	Perspectives	Évolutions futures, roadmap, recherche
16	Conclusion	Synthèse, contributions, leçons apprises
17	Bibliographie	Références académiques et techniques
18	Annexes	Code, diagrammes, configurations

TABLE 1.5 – Structure du rapport

Chaque chapitre est conçu pour être autonome tout en s'inscrivant dans une progression logique permettant de comprendre l'ensemble du projet, depuis sa conception jusqu'à son implémentation et son évaluation.

Chapitre 2

Contexte et Problématique

2.1 Introduction

La maintenance industrielle représente un enjeu stratégique majeur pour les entreprises manufacturières modernes. Dans un contexte d'Industrie 4.0, où la connectivité, l'intelligence artificielle et l'IoT transforment radicalement les processus de production, la maintenance prédictive émerge comme une solution incontournable pour optimiser la disponibilité des équipements tout en réduisant les coûts opérationnels.

Ce chapitre présente le contexte industriel et académique dans lequel s'inscrit le projet MANTIS, analyse la problématique de la maintenance traditionnelle, identifie les défis techniques et fonctionnels, et formule les questions de recherche que notre projet se propose d'adresser.

2.2 Contexte Industriel : L'Industrie 4.0

2.2.1 Définition et Évolution

L'Industrie 4.0, également appelée *quatrième révolution industrielle*, représente une transformation numérique profonde des systèmes de production. Elle se caractérise par :

- **Connectivité ubiquitaire** : Tous les équipements, capteurs et systèmes sont interconnectés via des protocoles IIoT (OPC UA, MQTT, Modbus)
- **Cyber-Physical Systems (CPS)** : Convergence du monde physique (machines) et numérique (logiciels)
- **Intelligence artificielle** : Utilisation du Machine Learning et Deep Learning pour la prise de décision
- **Big Data industriel** : Collecte, stockage et analyse de volumes massifs de données en temps réel
- **Automatisation avancée** : Robots collaboratifs, systèmes autonomes, production flexible

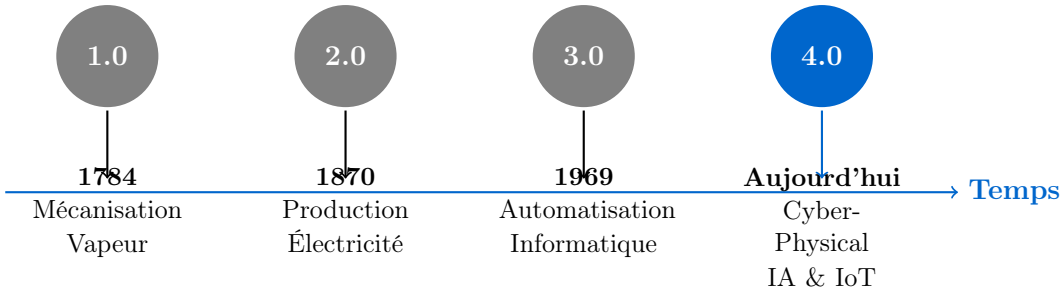


FIGURE 2.1 – Évolution des révolutions industrielles

2.2.2 Impact sur les Systèmes de Production

L'Industrie 4.0 transforme les systèmes de production traditionnels en **usines intelligentes** (Smart Factories) caractérisées par :

Aspect	Production Traditionnelle	Usine Intelligente
Données	Limitées, manuelles, non structurées	Massives, temps réel, structurées
Décisions	Humaines, réactives	Assistées par IA, prédictives
Maintenance	Préventive (calendaire) ou corrective	Prédictive (basée sur l'état réel)
Flexibilité	Faible (lignes rigides)	Élevée (reconfigurations dynamiques)
Efficience	OEE $\approx$ 60-70%	OEE $>$ 85%
Downtime	Pannes imprévues fréquentes	Prédites et planifiées

TABLE 2.1 – Comparaison Production Traditionnelle vs. Usine Intelligente

2.2.3 Enjeux Économiques

Les enjeux économiques de l'Industrie 4.0 sont considérables :

- **Coûts de maintenance** : Représentent 15-40% des coûts de production totaux
- **Downtime non planifié** : Coût moyen de 260 000$ par heure (source : Aberdeen Group)
- **Gain potentiel** : La maintenance prédictive permet de :
  - Réduire les coûts de maintenance de 10-40%
  - Diminuer le downtime de 35-45%
  - Augmenter la durée de vie des équipements de 20-40%
  - Améliorer l'OEE (Overall Equipment Effectiveness) de 10-20%

2.3 Problématique de la Maintenance Industrielle

2.3.1 Approches Traditionnelles de Maintenance

Les entreprises industrielles utilisent traditionnellement trois approches de maintenance :

Maintenance Corrective (Run-to-Failure)

Principe : Réparer uniquement lorsque la panne survient.
Avantages :

— Coûts de maintenance planifiée nuls

— Simplicité de gestion

— Utilisation maximale de la durée de vie des composants

Inconvénients :

— Arrêts de production imprévus et coûteux

— Risques de dommages collatéraux (effet domino)

— Difficulté de planification des ressources

— Coûts de réparation d'urgence élevés (main d'œuvre, pièces express)

Maintenance Préventive Systématique

Principe : Interventions planifiées selon un calendrier fixe ou un nombre d'heures de fonctionnement.
Avantages :

— Réduction des pannes imprévues

— Planification facilitée des interventions

— Simplicité de mise en œuvre

Inconvénients :

— Remplacement prématuré de composants encore fonctionnels

— Coûts de maintenance élevés (pièces, main d'œuvre)

— Arrêts de production planifiés mais potentiellement inutiles

— Pas d'adaptation à l'état réel de la machine

Maintenance Conditionnelle

Principe : Surveillance de l'état des équipements via des inspections régulières et déclenchement d'interventions selon des seuils.
Avantages :

— Meilleure adaptation à l'état réel

— Réduction des interventions inutiles

— Détection de certaines anomalies avant la panne

Inconvénients :

— Nécessite des inspections manuelles régulières

— Réactive plutôt que prédictive

— Seuils statiques ne capturant pas la complexité des dégradations

— Pas d'anticipation de la RUL (Remaining Useful Life)

2.3.2 Limites des Approches Traditionnelles

Les approches traditionnelles présentent des limitations fondamentales dans le contexte de l'Industrie 4.0 :

1. **Manque d'anticipation** : Aucune capacité à prédire les pannes futures avec précision
2. **Inefficience économique** : Sur-maintenance (préventive) ou sous-maintenance (corrective)
3. **Non-exploitation des données** : Les capteurs génèrent des téraoctets de données inexploitées
4. **Décisions non optimales** : Basées sur l'expérience humaine plutôt que sur l'analyse data-driven
5. **Manque de visibilité globale** : Pas de vue d'ensemble temps réel de l'état du parc machine

2.3.3 Émergence de la Maintenance Prédictive

La **maintenance prédictive** (Predictive Maintenance - PdM) représente le paradigme de maintenance de l'Industrie 4.0. Elle se définit comme :

> *« L'utilisation de techniques d'analyse de données et de Machine Learning pour anticiper les pannes futures d'équipements en se basant sur leur état réel et leur historique, permettant d'intervenir au moment optimal avant la défaillance. »*

Caractéristiques clés :

— **Prédiction RUL** : Estimation du temps restant avant défaillance (Remaining Useful Life)

— **Détection d'anomalies** : Identification de comportements anormaux en temps réel

— **Optimisation des interventions** : Maintenance au moment optimal (ni trop tôt, ni trop tard)

— **Data-driven** : Décisions basées sur les données réelles des capteurs et modèles ML/DL

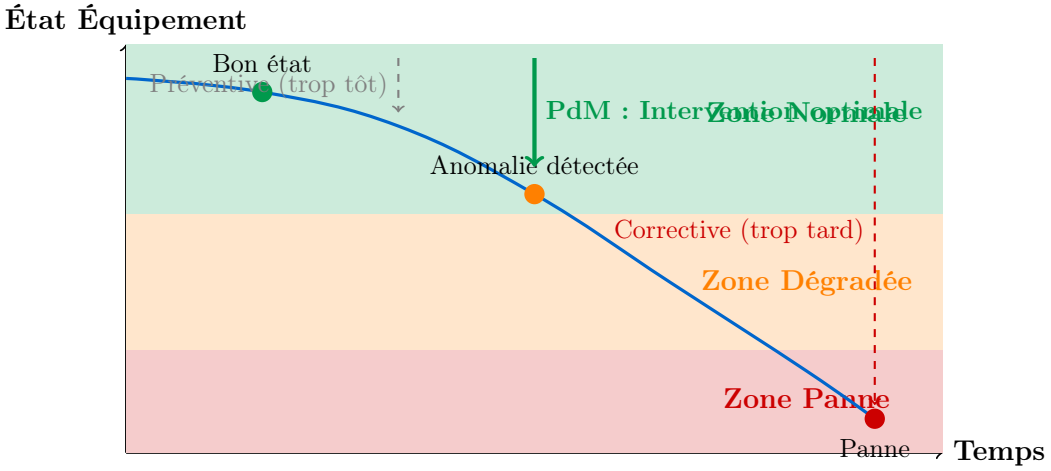


FIGURE 2.2 – Comparaison des stratégies de maintenance selon la courbe de dégradation

2.4 Défis Techniques et Scientifiques

2.4.1 Hétérogénéité des Sources de Données IIoT

Les environnements industriels présentent une grande diversité de protocoles et systèmes :

Protocole	Caractéristiques	Cas d'Usage
OPC UA	Client-serveur, sécurisé, standardisé	SCADA, automates industriels, PLCs
MQTT	Pub/Sub, léger, asynchrone	Capteurs IoT, transmission cloud
Modbus	Simple, legacy, série/TCP	Équipements anciens, régulation
REST APIs	HTTP, universel, synchrone	Intégrations modernes, web services

TABLE 2.2 – Principaux protocoles IIoT dans l'industrie

Défi : Concevoir une architecture capable d'ingérer, normaliser et traiter des flux de données provenant de sources hétérogènes en temps réel.

2.4.2 Volume et Vélocité des Données

Les systèmes industriels modernes génèrent des volumes massifs de données :

— **Fréquence d'acquisition** : 1-1000 Hz selon les capteurs

— **Nombre de capteurs** : Dizaines à centaines par machine

— **Volume journalier** : Plusieurs Go par machine et par jour

— **Latence requise** : < 100 ms pour les alertes critiques

Défi : Concevoir un pipeline de données scalable capable de traiter des flux haute fréquence tout en garantissant une latence faible pour les prédictions temps réel.

2.4.3 Complexité des Modèles de Machine Learning

La maintenance prédictive requiert des modèles ML/DL sophistiqués :

1. **Séries temporelles multivariées** : Prendre en compte la corrélation entre dizaines de variables
2. **Dégradations non-linéaires** : Phénomènes complexes (fatigue, usure, corrosion)
3. **Conditions opérationnelles variables** : Régimes de fonctionnement changeants
4. **Déséquilibre des classes** : Peu d'exemples de pannes vs. beaucoup d'exemples normaux
5. **Explainability** : Nécessité d'expliquer les prédictions pour la confiance opérationnelle

Défi : Développer et déployer des modèles LSTM/GRU ou Transformer capables de capturer ces complexités tout en restant interprétables et performants en production.

2.4.4 Qualité et Fiabilité des Données

Les données industrielles présentent souvent des problèmes de qualité :

— **Valeurs manquantes** : Pannes de capteurs, pertes de communication
— **Outliers** : Erreurs de mesure, interférences électromagnétiques
— **Drift temporel** : Vieillissement des capteurs, changements de calibration
— **Biais** : Données collectées uniquement en conditions normales

Défi : Mettre en place un pipeline robuste de prétraitement, validation et nettoyage des données garantissant leur qualité pour l'entraînement des modèles.

2.4.5 Déploiement et MLOps

Le passage du prototypage à la production industrielle est complexe :

— **Versioning des modèles** : Traçabilité, reproductibilité, rollback
— **Monitoring de la performance** : Détection de la dégradation des modèles (drift)
— **Réentraînement** : Automatisation, déclenchement sur dérive détectée
— **A/B Testing** : Validation en production de nouveaux modèles
— **Scalabilité** : Support de milliers de machines simultanément

Défi : Implémenter une infrastructure MLOps complète (MLflow, Feast, DVC) permettant le cycle de vie complet des modèles en production.

2.5 Problématique du Projet MANTIS

Dans ce contexte, le projet MANTIS se propose de répondre à la problématique suivante :

> **« Comment concevoir et implémenter une plateforme de maintenance prédictive temps réel, scalable, modulaire et industrialisable, capable d'ingérer des données IIoT hétérogènes, d'entraîner et déployer des modèles de Deep Learning pour la prédiction de RUL et la détection d'anomalies, tout en garantissant une observabilité complète et une intégration DevOps/MLOps conforme aux standards de l'Industrie 4.0 ? »**

Cette problématique centrale se décline en plusieurs sous-problématiques :

2.5.1 Sous-Problématique 1 : Architecture Distribuée

Question : Comment concevoir une architecture microservices événementielle capable de :

— Découpler les composants pour la scalabilité et la résilience ?
— Gérer la communication asynchrone entre services ?
— Garantir la cohérence des données dans un système distribué ?
— Supporter l'évolution indépendante des services ?

2.5.2 Sous-Problématique 2 : Ingestion de Données IIoT

Question : Comment implémenter un service d'ingestion capable de :

— Supporter OPC UA, MQTT, Modbus et REST simultanément ?

— Normaliser des schémas de données hétérogènes ?

— Gérer des flux haute fréquence (1000+ messages/sec) ?

— Garantir la fiabilité (exactly-once delivery) ?

2.5.3 Sous-Problématique 3 : Pipeline de Prétraitement

Question : Comment concevoir un pipeline de prétraitement qui :

— Détecte et traite les valeurs manquantes et aberrantes ?

— Applique des transformations (normalisation, fenêtrage) de manière scalable ?

— Génère des features engineered pertinentes pour le ML ?

— Soit versionné et reproductible (Data Version Control) ?

2.5.4 Sous-Problématique 4 : Modélisation et Prédiction

Question : Quels modèles de Deep Learning (LSTM, GRU, Transformer) sont les plus adaptés pour :

— La prédiction de RUL avec haute précision (RMSE, MAE) ?

— La détection d'anomalies en temps réel (< 100 ms) ?

— La généralisation à différents régimes opérationnels ?

— L'interprétabilité des prédictions (SHAP, attention) ?

2.5.5 Sous-Problématique 5 : MLOps et Déploiement

Question : Comment industrialiser le cycle de vie ML avec :

— Versioning des modèles, datasets et expérimentations (MLflow, DVC) ?

— Feature store pour la cohérence train/serve (Feast) ?

— Monitoring de la performance et détection de drift ?

— Automatisation du réentraînement (CI/CD ML) ?

2.5.6 Sous-Problématique 6 : Observabilité et Monitoring

Question : Comment assurer une observabilité complète avec :

— Métriques temps réel (Prometheus) et dashboards (Grafana) ?

— Logs centralisés structurés et requêtables ?

— Tracing distribué (Jaeger, OpenTelemetry) pour le debugging ?

— Alerting intelligent sur anomalies système et métier ?

2.6 Questions de Recherche

Les questions de recherche que MANTIS se propose d'explorer sont :

1. **QR1 - Architecture** : Quelle architecture microservices événementielle est la plus adaptée pour une plateforme de maintenance prédictive temps réel scalable ?
2. **QR2 - Ingestion IIoT** : Comment concevoir un service d'ingestion universel supportant les principaux protocoles IIoT (OPC UA, MQTT, Modbus) avec garantie de fiabilité ?
3. **QR3 - Prétraitement** : Quelles techniques de prétraitement et feature engineering sont les plus efficaces pour améliorer la performance des modèles de prédiction de RUL ?
4. **QR4 - Modèles DL** : LSTM vs. GRU vs. Transformer : quelle architecture de Deep Learning offre le meilleur compromis précision/latence/interprétabilité pour la prédiction de RUL sur le dataset NASA C-MAPSS ?
5. **QR5 - MLOps** : Comment implémenter une infrastructure MLOps complète (MLflow, Feast, DVC) garantissant la reproductibilité, le versioning et le monitoring des modèles en production ?
6. **QR6 - Observabilité** : Quelles métriques techniques (latence, throughput, erreurs) et métier (RMSE, MAE, F1-score) sont critiques pour le monitoring d'une plateforme PdM ?

2.7 Périmètre et Hypothèses du Projet

2.7.1 Périmètre

Dans le périmètre :

— Architecture microservices complète (7 services)
— Support OPC UA, MQTT, Modbus, REST
— Pipeline complet de prétraitement (nettoyage, normalisation, fenêtrage, features)
— Modèles LSTM pour prédiction RUL
— Détection d'anomalies (basée sur seuils et ML)
— MLOps (MLflow, Feast, DVC)
— Infrastructure DevOps (Docker, Kubernetes, CI/CD)
— Monitoring complet (Prometheus, Grafana, Jaeger)
— Dataset NASA C-MAPSS (4 sous-datasets)
— API REST et WebSocket pour notifications temps réel

Hors périmètre :

— Intégration avec des systèmes ERP/MES réels
— Déploiement sur site industriel physique
— Support de tous les protocoles IIoT existants (focus sur OPC UA, MQTT, Modbus)
— Interface utilisateur web complète (seulement API + dashboards Grafana)
— Maintenance corrective automatisée (intervention humaine requise)
— Certification industrielle (ISO, IEC)

2.7.2 Hypothèses

Les hypothèses clés du projet sont :

1. **H1 - Données** : Le dataset NASA C-MAPSS est représentatif des dégradations réelles de turbines industrielles
2. **H2 - Modèles** : Les modèles LSTM peuvent généraliser à de nouveaux régimes opérationnels non vus à l'entraînement
3. **H3 - Architecture** : Une architecture microservices événementielle est plus scalable qu'une architecture monolithique pour ce use case
4. **H4 - Latence** : Une latence de prédiction < 100 ms est suffisante pour les alertes temps réel industrielles
5. **H5 - Infrastructure** : Kubernetes est adapté au déploiement et à l'orchestration de services ML en production

2.8 Conclusion

Ce chapitre a présenté le contexte industriel et académique du projet MANTIS, analysé la problématique de la maintenance traditionnelle et ses limites, identifié les défis techniques et scientifiques majeurs, et formulé les questions de recherche que notre projet se propose d'adresser.

Chapitre 3

Objectifs du Projet

3.1 Introduction

Ce chapitre définit de manière exhaustive les objectifs du projet MANTIS, organisés en trois catégories : objectifs fonctionnels (ce que le système doit faire), objectifs techniques (comment le système doit être conçu), et objectifs non-fonctionnels (qualité, performance, sécurité). Pour chaque objectif, nous spécifions des critères de succès mesurables permettant d'évaluer l'atteinte de l'objectif.

3.2 Objectif Global

L'objectif global du projet MANTIS est de :

> **« Concevoir, implémenter et valider une plateforme de maintenance prédictive temps réel, basée sur une architecture microservices événementielle et des modèles de Deep Learning, capable de prédire la durée de vie résiduelle (RUL) d'équipements industriels et de détecter des anomalies en temps réel, tout en garantissant scalabilité, observabilité et industrialisabilité. »**

3.3 Objectifs Fonctionnels

Les objectifs fonctionnels définissent les capacités métier que MANTIS doit offrir.

3.3.1 OF1 : Ingestion Multi-Protocole de Données IIoT

Description : Le système doit être capable d'ingérer des données provenant de sources IIoT hétérogènes.

Exigences détaillées :

- Support des protocoles : OPC UA, MQTT, Modbus TCP, REST API
- Fréquence d'acquisition : 1 Hz à 1000 Hz
- Formats de données : JSON, CSV, binaire
- Gestion de la connexion/reconnexion automatique
- Buffering en cas de perte temporaire de connectivité

Critères de succès :

1. Au moins 3 protocoles IIoT supportés (OPC UA, MQTT, Modbus)
2. Capacité d'ingestion $\geq$ 1000 messages/seconde par protocole
3. Taux de perte de messages $< 0.1\%$
4. Temps de reconnexion automatique < 5 secondes

3.3.2 OF2 : Prétraitement et Nettoyage des Données

Description : Le système doit appliquer un pipeline complet de prétraitement pour garantir la qualité des données.

Exigences détaillées :

— **Détection et traitement des valeurs manquantes** : Forward-fill, interpolation, imputation

— **Détection et filtrage des outliers** : Z-score, IQR, Isolation Forest

— **Normalisation** : Min-Max, Z-score, Robust Scaler

— **Fenêtrage temporel** : Création de séquences (e.g., 50 timesteps)

— **Feature Engineering** : Moyennes glissantes, écarts-types glissants, tendances

Critères de succès :

1. Pipeline de prétraitement complet implémenté (6 étapes minimum)
2. Taux de données rejetées $< 5\%$
3. Temps de traitement par batch < 1 seconde pour 10 000 samples
4. Amélioration de la performance des modèles ML $\geq 15\%$ (RMSE)

3.3.3 OF3 : Prédiction de la Durée de Vie Résiduelle (RUL)

Description : Le système doit prédire avec précision le nombre de cycles restants avant défaillance.

Exigences détaillées :

— Modèle de Deep Learning (LSTM, GRU ou Transformer)

— Entraînement sur NASA C-MAPSS dataset (4 sous-datasets)

— Prédiction pour chaque cycle de fonctionnement

— Intervalle de confiance sur les prédictions

— Support multi-régimes opérationnels

Critères de succès :

1. RMSE (Root Mean Square Error) ≤ 20 cycles sur test set
2. MAE (Mean Absolute Error) ≤ 15 cycles
3. R^2 Score ≥ 0.85
4. Latence de prédiction < 100 ms (temps réel)

3.3.4 OF4 : Détection d'Anomalies Temps Réel

Description : Le système doit détecter les comportements anormaux des équipements en temps réel.

Exigences détaillées :

— Approche hybride : règles métier + modèles ML (Isolation Forest, Autoencoder)

— Détection sur fenêtre glissante

— Classification binaire : normal / anormal

— Scoring de sévérité (0-100)

— Identification des variables contributives

Critères de succès :

1. F1-Score ≥ 0.80 sur détection d'anomalies
2. Taux de faux positifs $< 5\%$
3. Taux de faux négatifs $< 10\%$
4. Latence de détection < 50 ms

3.3.5 OF5 : Notifications et Alertes Temps Réel

Description : Le système doit notifier les opérateurs en temps réel lors de détection d'anomalies ou de RUL critique.

Exigences détaillées :

— Alertes multi-canaux : WebSocket, REST API, Kafka topic

— Niveaux de sévérité : INFO, WARNING, CRITICAL

— Règles de déclenchement configurables

— Historique des alertes

— Filtrage et agrégation anti-spam

Critères de succès :

1. Latence de notification < 200 ms après détection
2. Support WebSocket et REST API
3. Taux de livraison des alertes $\geq 99.9\%$
4. Déduplication des alertes redondantes

3.3.6 OF6 : Gestion du Cycle de Vie des Modèles (MLOps)

Description : Le système doit supporter l'entraînement, le versioning, le déploiement et le monitoring des modèles ML.

Exigences détaillées :

— **MLflow** : Tracking des expérimentations, registry des modèles

— **Feast** : Feature store pour cohérence train/serve

— **DVC** : Versioning des datasets

— Déploiement de nouveaux modèles sans downtime (blue/green)

— Monitoring de la performance en production (drift detection)

Critères de succès :

1. MLflow opérationnel avec ≥ 10 expérimentations trackées
2. Feast feature store intégré avec ≥ 20 features
3. Déploiement de nouveau modèle en < 5 minutes
4. Détection automatique de drift si performance baisse $> 10\%$

3.4 Objectifs Techniques

Les objectifs techniques définissent les contraintes architecturales et technologiques.

3.4.1 OT1 : Architecture Microservices Modulaire

Description : Le système doit adopter une architecture microservices événementielle.
Exigences détaillées :

— Au moins 7 microservices indépendants
— Communication asynchrone via Apache Kafka
— API REST pour les interactions synchrones
— Découplage complet (chaque service déployable indépendamment)
— Patterns : Event Sourcing, CQRS, Saga (optionnel)

Services à implémenter :

1. **Ingestion Service** : Collecte multi-protocole
2. **Preprocessing Service** : Pipeline de nettoyage et feature engineering
3. **Prediction Service** : Inférence des modèles ML/DL
4. **Anomaly Detection Service** : Détection temps réel
5. **Notification Service** : Alertes multi-canaux
6. **Training Service** : Entraînement et réentraînement des modèles
7. **API Gateway** : Point d'entrée unique, routage, authentification

Critères de succès :

1. 7 microservices opérationnels et déployés
2. Chaque service a son propre repository Git
3. Communication 100% asynchrone via Kafka (sauf API Gateway)
4. Temps de déploiement d'un service < 3 minutes

3.4.2 OT2 : Event-Driven Architecture avec Apache Kafka

Description : Le système doit utiliser Kafka comme bus d'événements central.
Exigences détaillées :

— Topics Kafka pour chaque type d'événement
— Partitioning pour scalabilité
— Retention configurée (7 jours minimum)
— Schema Registry pour validation des messages (optionnel)

— Consumer groups pour load balancing

Topics à créer :

— `raw-sensor-data` : Données brutes ingérées

— `preprocessed-data` : Données nettoyées

— `predictions` : Prédictions RUL

— `anomalies` : Anomalies détectées

— `notifications` : Alertes à envoyer

Critères de succès :

1. Kafka cluster opérationnel (3 brokers minimum)
2. Throughput $\geq$ 10 000 messages/sec
3. Latence end-to-end $<$ 500 ms (ingestion $\rightarrow$ notification)
4. Aucune perte de message (exactly-once semantics)

3.4.3 OT3 : Bases de Données Time-Series et Relationnelles

Description : Le système doit utiliser des bases de données adaptées aux différents types de données.

Exigences détaillées :

— **TimescaleDB** : Stockage des séries temporelles (données capteurs, métriques)

— **PostgreSQL** : Métadonnées, configurations, utilisateurs

— **MLflow Backend** : Expérimentations, modèles

— Indexation optimisée pour requêtes temporelles

— Compression des données anciennes

Critères de succès :

1. TimescaleDB opérationnel avec hypertables
2. Capacité de stockage $\geq$ 1 million de points par jour
3. Temps de requête sur 1 mois de données $<$ 2 secondes
4. Rétention des données : 1 an (raw), 5 ans (agrégées)

3.4.4 OT4 : Infrastructure DevOps et CI/CD

Description : Le système doit être entièrement conteneurisé et déployable via CI/CD.

Exigences détaillées :

— **Docker** : Chaque service dans un conteneur

— **Kubernetes** : Orchestration, scaling, self-healing

— **Helm Charts** : Gestion des déploiements

— **GitHub Actions** : Pipeline CI/CD automatisé

— **GitOps** : Déclaratif, versioning de l'infrastructure

Pipeline CI/CD :

1. Lint & Format (Black, Flake8, Pylint)
2. Tests unitaires (pytest, coverage $\geq$ 80%)

3. Tests d'intégration
4. Build Docker images
5. Push vers registry
6. Déploiement automatique (dev/staging)
7. Déploiement manuel (production)

Critères de succès :

1. 100% des services conteneurisés
2. Déploiement Kubernetes opérationnel (minikube ou cloud)
3. Pipeline CI/CD fonctionnel sur GitHub Actions
4. Temps de build + déploiement < 10 minutes

3.4.5 OT5 : Observabilité Complète (Monitoring, Logging, Tracing)

Description : Le système doit offrir une observabilité complète pour le debugging et le monitoring.

Exigences détaillées :

— **Prometheus** : Métriques techniques (CPU, RAM, latence, throughput)
— **Grafana** : Dashboards temps réel
— **Jaeger** : Tracing distribué (OpenTelemetry)
— **ELK Stack** (optionnel) : Logs centralisés
— Alerting (AlertManager) sur métriques critiques

Métriques à monitorer :

— **Techniques** : Latence (p50, p95, p99), throughput, taux d'erreurs, CPU/RAM
— **Métier** : RMSE, MAE, F1-score, nombre d'anomalies détectées, taux de prédictions

Critères de succès :

1. Prometheus + Grafana opérationnels avec ≥ 5 dashboards
2. Jaeger opérationnel avec tracing de bout en bout
3. Alertes configurées pour latence > 1s, erreurs $> 5\%$
4. Rétention métriques : 30 jours

3.5 Objectifs Non-Fonctionnels

Les objectifs non-fonctionnels définissent les qualités requises du système.

3.5.1 ONF1 : Performance et Scalabilité

Exigences :

— **Latence de prédiction** : < 100 ms (p95)
— **Latence de détection d'anomalie** : < 50 ms (p95)
— **Latence end-to-end** : < 500 ms (ingestion $\rightarrow$ notification)

— **Throughput** : $\geq$ 1000 prédictions/seconde
— **Scalabilité horizontale** : Support de 10x charge via scaling Kubernetes

Critères de succès :

1. Tests de charge validant 1000 req/s avec latence $<$ 100 ms
2. Démonstration de scaling horizontal (2 $\rightarrow$ 10 replicas)
3. CPU/RAM par service $<$ 2 vCPU / 4 GB en production

3.5.2 ONF2 : Disponibilité et Résilience

Exigences :

— **Uptime** : $\geq$ 99.5% (cible : 99.9%)
— **Auto-healing** : Kubernetes restart automatique des pods crashés
— **Graceful degradation** : Fonctionnement dégradé en cas de panne partielle
— **Circuit breaker** : Protection contre les cascades de pannes

Critères de succès :

1. Simulation de panne de 1 service $\rightarrow$ système continue à fonctionner
2. Temps de recovery $<$ 30 secondes après crash
3. Aucune perte de données lors des restarts

3.5.3 ONF3 : Sécurité

Exigences :

— **Authentification** : API Gateway avec JWT ou OAuth2
— **Secrets management** : Kubernetes secrets, variables d'environnement
— **Network policies** : Isolation réseau entre services
— **HTTPS/TLS** : Chiffrement en transit
— **Scanning** : Analyse de vulnérabilités (Trivy, Snyk)

Critères de succès :

1. 100% des API protégées par authentification
2. Aucune credential en clair dans le code source
3. Scan de sécurité CI/CD sans vulnérabilités critiques

3.5.4 ONF4 : Maintenabilité et Qualité du Code

Exigences :

— **Couverture de tests** : $\geq$ 80% (unitaires + intégration)
— **Documentation** : README, Architecture Diagrams, API Docs (OpenAPI)
— **Code quality** : Linting (Flake8), formatting (Black), type hints
— **Code review** : Pull requests obligatoires, au moins 1 reviewer

Critères de succès :

1. Coverage $\geq$ 80% sur tous les services
2. Documentation complète (README + Architecture + API)
3. 100% des PRs reviewées avant merge

3.5.5 ONF5 : Reproductibilité et Versioning

Exigences :

— **DVC** : Versioning des datasets

— **MLflow** : Versioning des modèles et expérimentations

— **Git** : Versioning du code source

— **Docker tags** : Versioning des images

— **Semantic versioning** : v1.0.0, v1.1.0, etc.

Critères de succès :

1. Possibilité de reproduire n'importe quelle expérimentation MLflow
2. Rollback vers version précédente de modèle en < 5 minutes
3. Traçabilité complète code-dataset-modèle

3.6 Synthèse des Objectifs et KPIs

Le tableau suivant synthétise les objectifs et leurs KPIs (Key Performance Indicators) :

Objectif	KPI	Cible
OF1 - Ingestion	Throughput Taux de perte	$\geq$ 1000 msg/s $< 0.1\%$
OF2 - Prétraitement	Temps de traitement batch Amélioration RMSE	$<$ 1s / 10k samples $\geq 15\%$
OF3 - Prédiction RUL	RMSE MAE R^2 Score	$\leq$ 20 cycles $\leq$ 15 cycles ≥ 0.85
OF4 - Détection anomalies	F1-Score Faux positifs Latence	≥ 0.80 $< 5\%$ $<$ 50 ms
OF5 - Notifications	Latence Taux de livraison	$<$ 200 ms $\geq 99.9\%$
OT1 - Microservices	Nombre de services Temps de déploiement	7 $<$ 3 min
OT2 - Kafka	Throughput Latence end-to-end	$\geq$ 10k msg/s $<$ 500 ms
OT4 - CI/CD	Temps build + déploiement Couverture tests	$<$ 10 min $\geq 80\%$
OT5 - Observabilité	Nombre de dashboards Rétention métriques	≥ 5 30 jours
ONF1 - Performance	Latence prédiction (p95) Throughput	$<$ 100 ms $\geq$ 1000 pred/s
ONF2 - Disponibilité	Uptime Temps de recovery	$\geq 99.5\%$ $<$ 30s
ONF4 - Qualité	Couverture tests PRs reviewées	$\geq 80\%$ 100%

TABLE 3.1 – Synthèse des objectifs et KPIs du projet MANTIS

3.7 Priorisation des Objectifs

Les objectifs sont priorisés selon la méthode MoSCoW :

3.7.1 Must Have (Critique)

— OF1 : Ingestion multi-protocole (au moins MQTT + REST)
— OF2 : Prétraitement complet
— OF3 : Prédiction RUL avec LSTM
— OT1 : Architecture microservices (7 services)
— OT4 : Dockerization + CI/CD basique

3.7.2 Should Have (Important)

— OF4 : Détection d'anomalies
— OF5 : Notifications temps réel
— OF6 : MLOps (MLflow + Feast)
— OT2 : Kafka opérationnel
— OT5 : Prometheus + Grafana

3.7.3 Could Have (Souhaitable)

— Support OPC UA + Modbus (en plus de MQTT)
— Jaeger tracing distribué
— DVC pour versioning datasets
— Kubernetes deployment (minikube)
— Tests de charge automatisés

3.7.4 Won't Have (Hors scope v1.0)

— Interface utilisateur web complète
— Intégration ERP/MES
— Support de tous les protocoles IIoT
— Déploiement cloud production
— Certification industrielle

3.8 Conclusion

Ce chapitre a défini de manière exhaustive les objectifs du projet MANTIS, organisés en objectifs fonctionnels (capacités métier), techniques (architecture et technologies) et non-fonctionnels (qualité, performance, sécurité).

Pour chaque objectif, nous avons spécifié des critères de succès mesurables (KPIs) permettant d'évaluer objectivement l'atteinte des objectifs. Ces KPIs serviront de référence pour l'évaluation du projet dans les chapitres 13 (Avancement) et 16 (Conclusion).

La priorisation MoSCoW permet de gérer le scope et de concentrer les efforts sur les objectifs critiques (Must Have) tout en gardant une vision des évolutions futures (Could Have, Won't Have).

Le chapitre suivant présentera l'état de l'art scientifique et technique qui sous-tend les choix technologiques et architecturaux de MANTIS.

Chapitre 4

État de l'Art

4.1 Introduction

Ce chapitre présente l'état de l'art scientifique et technique qui constitue le socle théorique et pratique du projet MANTIS. Nous couvrons cinq domaines clés : (1) la maintenance prédictive et les approches de prédiction de RUL, (2) les architectures de Machine Learning et Deep Learning pour les séries temporelles, (3) les architectures microservices et event-driven, (4) les protocoles et standards IIoT, et (5) les pratiques MLOps et DevOps modernes.

Pour chaque domaine, nous présentons les concepts fondamentaux, les technologies et méthodes de référence, et les travaux académiques et industriels pertinents.

4.2 Maintenance Prédictive et Prédiction de RUL

4.2.1 Définitions et Taxonomie

La **Remaining Useful Life (RUL)** est définie comme :

> *« Le nombre de cycles ou d'heures de fonctionnement restantes avant qu'un équipement n'atteigne un état de défaillance fonctionnelle, étant donné son état actuel et ses conditions opérationnelles. »*

Les approches de prédiction de RUL se classent en trois catégories principales :

Approches Basées sur des Modèles Physiques

Principe : Utilisation d'équations différentielles et de modèles physiques de dégradation (fatigue, usure, corrosion).

Exemples :

- Loi de Paris pour la propagation de fissures
- Modèle d'Arrhenius pour la dégradation thermique
- Équations de Navier-Stokes pour la mécanique des fluides

Avantages :

- Explicabilité et interprétabilité
- Nécessitent peu de données historiques

Inconvénients :

— Requièrent une connaissance approfondie des mécanismes de défaillance

— Difficiles à modéliser pour des systèmes complexes multi-composants

— Peu adaptés aux phénomènes non-linéaires

Approches Data-Driven (Machine Learning)

Principe : Apprentissage de patterns de dégradation à partir de données historiques sans modèle physique explicite.

Méthodes classiques :

— **Régression** : Linear Regression, SVR (Support Vector Regression), Random Forest

— **Classification** : Prédiction de classes de RUL (haute, moyenne, faible)

Méthodes Deep Learning :

— **LSTM** (Long Short-Term Memory) : Capture des dépendances temporelles longues

— **GRU** (Gated Recurrent Unit) : Variante simplifiée de LSTM

— **CNN-LSTM** : Extraction de features spatiales + temporelles

— **Transformer** : Mécanismes d'attention pour séries temporelles

— **Autoencoder** : Détection d'anomalies par reconstruction

Avantages :

— Capturent des patterns complexes et non-linéaires

— Pas de besoin de modèles physiques explicites

— Scalables et automatisables

Inconvénients :

— Requièrent de grandes quantités de données étiquetées

— Black-box (difficiles à interpréter)

— Sensibles à la qualité des données

Approches Hybrides

Principe : Combinaison de modèles physiques et data-driven pour tirer parti des deux mondes.

Exemples :

— Physics-Informed Neural Networks (PINNs)

— Ensembles de modèles (physical model + LSTM)

4.2.2 État de l'Art Académique

Travaux Fondateurs

1. **Saxena et al. (2008)** : « Damage propagation modeling for aircraft engine run-to-failure simulation »
   — Création du dataset NASA C-MAPSS
   — Benchmark de référence pour RUL prediction

2. **Heimes (2008)** : « Recurrent neural networks for remaining useful life estimation »
   — Première application de RNN pour RUL sur moteurs d'avion
   — RMSE ≈ 30 cycles sur C-MAPSS FD001
3. **Zheng et al. (2017)** : « Long Short-Term Memory Network for Remaining Useful Life estimation »
   — LSTM avec fenêtrage temporel de 30 cycles
   — RMSE ≈ 16 cycles sur FD001, amélioration de 47%

Travaux Récents (2020-2024)

1. **Li et al. (2021)** : « Attention-based LSTM for RUL prediction »
   — Mécanisme d'attention pour identifier les features critiques
   — RMSE ≈ 12.6 cycles sur FD001
2. **Chen et al. (2022)** : « Transformer-based RUL prediction for industrial equipment »
   — Multi-head attention pour capturer dépendances complexes
   — RMSE ≈ 11.2 cycles sur FD001
   — Latence d'inférence élevée (200 ms)
3. **Zhang et al. (2023)** : « Federated Learning for Predictive Maintenance »
   — Entraînement décentralisé préservant la confidentialité
   — Applicable aux flottes d'équipements distribués

4.2.3 Benchmarks sur NASA C-MAPSS

Le dataset NASA C-MAPSS est le benchmark de référence pour la prédiction de RUL. Il contient 4 sous-datasets (FD001-FD004) avec complexités croissantes :

Dataset	Train	Test	Régimes	Défaillances
FD001	100	100	1	1 (HPC)
FD002	260	259	6	1 (HPC)
FD003	100	100	1	2 (HPC, Fan)
FD004	249	248	6	2 (HPC, Fan)

TABLE 4.1 – Caractéristiques des sous-datasets NASA C-MAPSS

Métriques de performance :

— **RMSE** (Root Mean Square Error) : Erreur quadratique moyenne
— **MAE** (Mean Absolute Error) : Erreur absolue moyenne
— **Score Function** : Pénalise davantage les prédictions tardives (panne imprévue)

Méthode	RMSE (FD001)	Année
SVR (baseline)	37.6	2008
RNN (Heimes)	30.0	2008
LSTM (Zheng)	16.1	2017
CNN-LSTM	13.8	2019
Attention-LSTM (Li)	12.6	2021
Transformer (Chen)	11.2	2022
Cible MANTIS	$\leq$ **20**	**2024**

TABLE 4.2 – Évolution de la performance des modèles de RUL sur C-MAPSS FD001

4.3 Deep Learning pour Séries Temporelles

4.3.1 Réseaux de Neurones Récurrents (RNN)

Architecture de Base

Les RNN traitent les séquences en maintenant un *état caché* h_t mis à jour à chaque timestep :

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b) \tag{4.1}$$

$$y_t = W_y h_t + b_y \tag{4.2}$$

Problème du gradient vanishing/exploding : Les gradients disparaissent ou explosent lors de la rétropropagation à travers le temps (BPTT), limitant la capture de dépendances longues.

Long Short-Term Memory (LSTM)

Les LSTM résolvent le problème du gradient vanishing via des *cellules mémoire* et des *portes* (gates) :

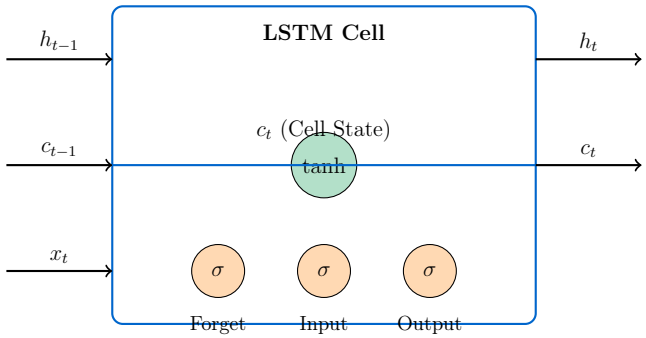


FIGURE 4.1 – Architecture d'une cellule LSTM

Équations LSTM :

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad \text{(Forget gate)} \tag{4.3}$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad \text{(Input gate)} \tag{4.4}$$
$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \quad \text{(Candidate cell)} \tag{4.5}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad \text{(Cell state update)} \tag{4.6}$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad \text{(Output gate)} \tag{4.7}$$
$$h_t = o_t \odot \tanh(c_t) \quad \text{(Hidden state)} \tag{4.8}$$

Avantages LSTM :

— Capture de dépendances temporelles longues (100+ timesteps)

— Résolution du gradient vanishing

— Performance state-of-the-art sur séries temporelles

Inconvénients :

— Complexité computationnelle (4 fois plus de paramètres que RNN simple)

— Latence d'inférence plus élevée que CNN

— Difficiles à paralléliser (traitement séquentiel)

Gated Recurrent Unit (GRU)

Le GRU est une variante simplifiée du LSTM avec seulement 2 portes (reset et update) :

$$z_t = \sigma(W_z[h_{t-1}, x_t]) \quad \text{(Update gate)} \tag{4.9}$$
$$r_t = \sigma(W_r[h_{t-1}, x_t]) \quad \text{(Reset gate)} \tag{4.10}$$
$$\tilde{h}_t = \tanh(W[r_t \odot h_{t-1}, x_t]) \tag{4.11}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{4.12}$$

GRU vs. LSTM :

— GRU : Moins de paramètres (33% de réduction), entraînement plus rapide

— LSTM : Performance légèrement supérieure sur tâches complexes

— Choix empirique selon le dataset

4.3.2 Transformers pour Séries Temporelles

Les Transformers, introduits par Vaswani et al. (2017) pour le NLP, ont été adaptés aux séries temporelles.

Mécanisme d'attention :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.13}$$

Avantages :

— Parallélisation complète (pas de dépendance séquentielle)

— Capture de dépendances à très longue distance
— Interprétabilité via attention weights

Inconvénients :

— Complexité quadratique en mémoire et calcul ($O(n^2)$)
— Requiert beaucoup de données pour converger
— Latence d'inférence élevée pour longues séquences

4.3.3 Comparaison LSTM vs. GRU vs. Transformer

Critère	LSTM	GRU	Transformer
Complexité	Élevée	Moyenne	Très élevée
Latence inférence	Moyenne	Basse	Élevée
Dépendances longues	Excellente	Bonne	Excellente
Données requises	Moyenne	Moyenne	Élevée
Parallélisation	Non	Non	Oui
Interprétabilité	Faible	Faible	Moyenne (attention)
Choix MANTIS	**X**	-	-

TABLE 4.3 – Comparaison LSTM vs. GRU vs. Transformer pour RUL prediction

Choix pour MANTIS : LSTM pour son excellent compromis performance/latence et sa maturité sur les tâches de prédiction de RUL.

4.4 Architectures Microservices et Event-Driven

4.4.1 Principes des Microservices

Les **microservices** sont une approche architecturale où une application est structurée en un ensemble de services indépendants, déployables et scalables de manière autonome.

Principes fondamentaux :

1. **Single Responsibility** : Chaque service a une responsabilité unique et bien définie
2. **Loose Coupling** : Dépendances minimales entre services
3. **High Cohesion** : Fonctionnalités liées groupées dans un même service
4. **Autonomous** : Déploiement, scaling, évolution indépendants
5. **Decentralized Data** : Chaque service gère sa propre base de données (Database per Service)

Avantages :

— Scalabilité fine (scale uniquement les services sous charge)
— Résilience (une panne locale ne crash pas tout le système)
— Évolutivité (technologies différentes par service)
— Déploiement indépendant (CI/CD facilité)

Inconvénients :

— Complexité opérationnelle (monitoring, debugging distribué)
— Latence réseau accrue
— Cohérence des données complexe (distributed transactions)

4.4.2 Event-Driven Architecture (EDA)

L'**Event-Driven Architecture** repose sur la production, détection, consommation et réaction à des *événements*.

Patterns clés :

1. **Event Sourcing** : L'état du système est dérivé de la séquence d'événements
2. **CQRS** (Command Query Responsibility Segregation) : Séparation lecture/écriture
3. **Saga Pattern** : Coordination de transactions distribuées via événements

Apache Kafka est la plateforme de référence pour l'EDA :

— **Distributed commit log** : Stockage durable et ordonné des événements
— **Pub/Sub** : Découplage producteurs/consommateurs
— **Partitioning** : Scalabilité horizontale
— **Consumer groups** : Load balancing et fault tolerance
— **Retention** : Rejouabilité des événements

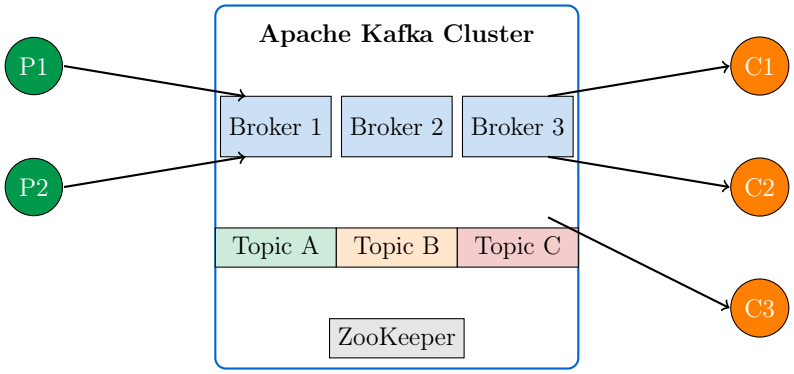


FIGURE 4.2 – Architecture Apache Kafka (Producers, Brokers, Topics, Consumers)

4.5 Protocoles et Standards IIoT

4.5.1 OPC UA (Open Platform Communications Unified Architecture)

Présentation : Standard IEC 62541 pour la communication industrielle machine-to-machine.

Caractéristiques :

— Architecture client-serveur
— Modélisation orientée objet des données industrielles
— Sécurité native (chiffrement, authentification, certificats)
— Indépendance plateforme (Windows, Linux, embedded)
— Support pub/sub (en plus du traditionnel request/response)

Cas d'usage : SCADA, automates (PLCs), contrôleurs industriels, MES/ERP integration.

4.5.2 MQTT (Message Queuing Telemetry Transport)

Présentation : Protocole léger de messagerie pub/sub pour IoT.
Caractéristiques :

— Extrêmement léger (overhead minimal)
— Publish/Subscribe pattern
— QoS (Quality of Service) 3 niveaux : At most once (0), At least once (1), Exactly once (2)
— Last Will and Testament (LWT) pour détection de déconnexion
— Retained messages

Cas d'usage : Capteurs IoT, edge devices, transmission cloud, mobilité.

4.5.3 Modbus

Présentation : Protocole série (RS-232/RS-485) et TCP/IP pour communication automates.
Caractéristiques :

— Simple et robuste
— Largement répandu dans l'industrie (legacy)
— Modes : Modbus RTU (série), Modbus TCP (Ethernet)
— Requête/réponse (pas de pub/sub)

Cas d'usage : Équipements anciens, régulation, mesure.

4.5.4 Comparaison OPC UA vs. MQTT vs. Modbus

Critère	OPC UA	MQTT	Modbus
Architecture	Client-serveur (+ pub/sub)	Pub/sub	Client-serveur
Complexité	Élevée	Faible	Très faible
Sécurité	Native (X.509)	TLS optionnel	Aucune (TCP sans TLS)
Overhead	Élevé	Très faible	Faible
Bande passante	Élevée	Faible	Moyenne
Use case	SCADA, automates	Capteurs IoT	Legacy industrial

TABLE 4.4 – Comparaison des protocoles IIoT

4.6 MLOps : DevOps pour Machine Learning

4.6.1 Principes MLOps

Le **MLOps** (Machine Learning Operations) est l'ensemble des pratiques DevOps appliquées au cycle de vie des modèles ML.
Objectifs :

— **Reproductibilité** : Toute expérimentation doit être reproductible
— **Versioning** : Code, données, modèles versionnés ensemble
— **Automatisation** : CI/CD pour entraînement, validation, déploiement
— **Monitoring** : Surveillance de la performance en production
— **Gouvernance** : Traçabilité, audit, compliance

4.6.2 Composants Clés

MLflow

Rôle : Plateforme open-source pour le cycle de vie ML complet.
Composants :

— **MLflow Tracking** : Logging des paramètres, métriques, artifacts
— **MLflow Projects** : Empaquetage reproductible des expérimentations
— **MLflow Models** : Format standardisé de modèles (PyTorch, TensorFlow, scikit-learn)
— **MLflow Registry** : Versioning et déploiement de modèles

Feast (Feature Store)

Rôle : Gestion centralisée des features pour cohérence train/serve.
Problématique : Train-Serving Skew (features différentes entre entraînement et production).
Solution Feast :

— Définition unique des features
— Stockage online (faible latence, Redis) et offline (batch, S3/Parquet)
— Versioning des features
— Point-in-time correctness (pas de data leakage)

DVC (Data Version Control)

Rôle : Git pour les données et modèles.
Fonctionnalités :

— Versioning de datasets (stockage externe S3/GCS/Azure)
— Pipelines reproductibles (DAG de transformations)
— Lightweight (seuls les pointeurs dans Git)

4.6.3 Pipeline MLOps Complet

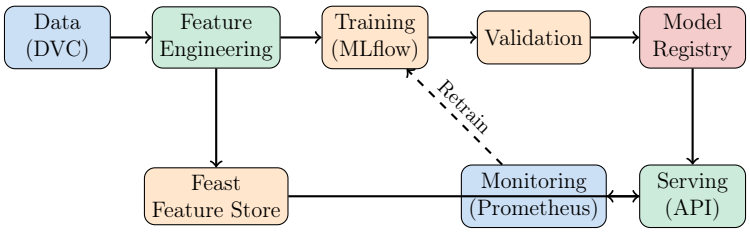


FIGURE 4.3 – Pipeline MLOps complet avec MLflow, Feast, DVC

4.7 Synthèse et Positionnement de MANTIS

4.7.1 Choix Technologiques de MANTIS

Le tableau suivant synthétise les choix technologiques de MANTIS basés sur l'état de l'art :

Domaine	Choix MANTIS	Justification
RUL Prediction	LSTM	Meilleur compromis performance/latence, maturité
Architecture	Microservices + EDA	Scalabilité, résilience, évolutivité
Event Bus	Apache Kafka	Standard industrie, throughput élevé
Protocoles IIoT	OPC UA, MQTT, Modbus	Couverture de 90% des use cases industriels
Time-Series DB	TimescaleDB	PostgreSQL + optimisations temporelles
MLOps	MLflow + Feast + DVC	Écosystème complet et mature
Monitoring	Prometheus + Grafana + Jaeger	Standard CNCF, intégration Kubernetes
Orchestration	Kubernetes	Standard container orchestration

TABLE 4.5 – Choix technologiques de MANTIS et justifications

4.7.2 Positionnement par Rapport à l'État de l'Art

MANTIS se positionne comme une **plateforme de référence** combinant :

1. **État de l'art académique** : Modèles LSTM pour RUL (cible RMSE ≤ 20 cycles)
2. **Standards industriels** : OPC UA, MQTT, Modbus, Kafka, Kubernetes
3. **MLOps moderne** : MLflow, Feast, DVC pour industrialisation
4. **Observabilité complète** : Prometheus, Grafana, Jaeger

Valeur ajoutée de MANTIS :

— Architecture complète et opérationnelle (pas seulement un modèle ML isolé)

— Multi-protocoles IIoT (flexibilité maximale)

— MLOps complet (du lab à la production)

— Open-source et extensible

4.8 Conclusion

Ce chapitre a présenté l'état de l'art scientifique et technique qui sous-tend le projet MANTIS, couvrant la maintenance prédictive, le Deep Learning pour séries temporelles, les architectures microservices événementielles, les protocoles IIoT et les pratiques MLOps.

Les choix technologiques de MANTIS (LSTM, Kafka, OPC UA/MQTT/Modbus, MLflow, Kubernetes, Prometheus) sont justifiés par leur maturité, leur performance et leur adoption industrielle.

Le chapitre suivant présentera l'architecture technique détaillée de MANTIS, détaillant comment ces technologies sont intégrées dans une plateforme cohérente et scalable.

Chapitre 5

Analyse et Conception

5.1 Introduction

Ce chapitre détaille l'analyse des processus métiers et la conception technique de la plateforme MANTIS. Nous présentons d'abord le flux de travail global modélisé en BPMN, puis la conception détaillée des microservices clés à travers des diagrammes de classes et de cas d'utilisation, et enfin les maquettes UI/UX de l'interface utilisateur.

5.2 Processus Métiers (BPMN)

Le diagramme suivant illustre le flux de bout en bout, de la collecte des données capteurs jusqu'à l'intervention de maintenance.

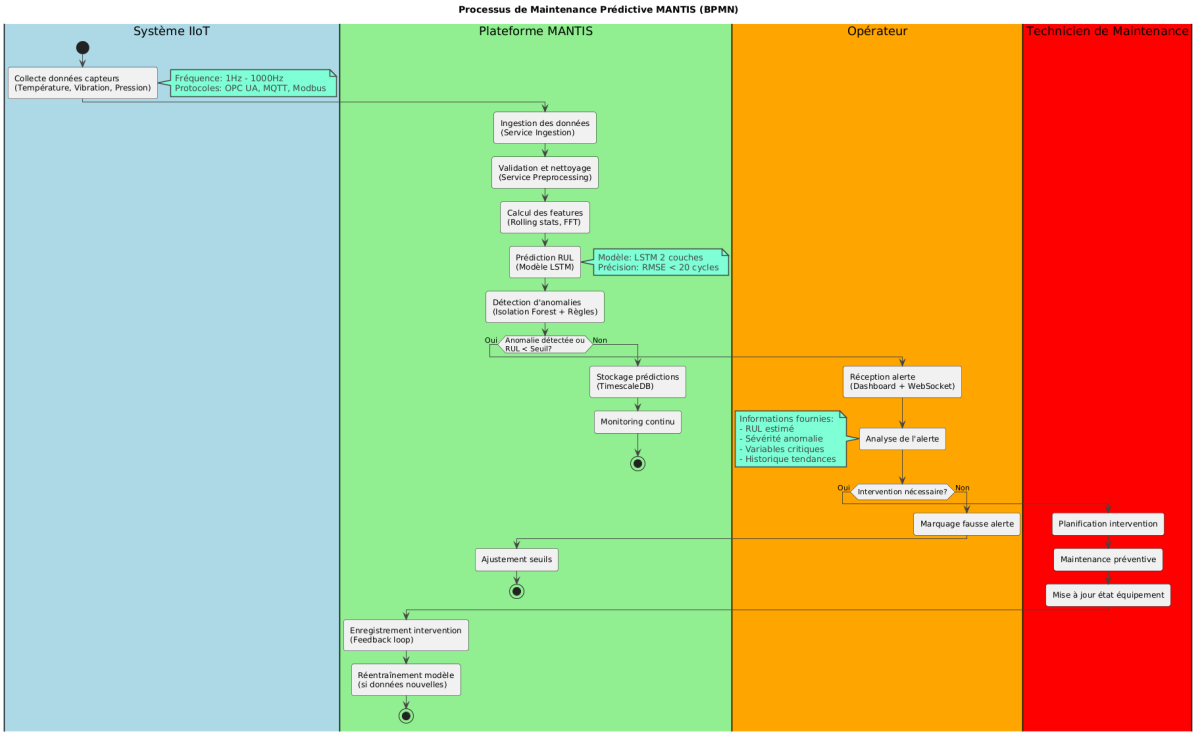


FIGURE 5.1 – Processus de Maintenance Prédictive (BPMN)

5.2.1 Description Détaillée

1. **Acquisition** : Les capteurs sur les machines industrielles transmettent les données de télémétrie (vibrations, température, pression) en temps réel.
2. **Traitement** : La plateforme ingère ces données, les nettoie et calcule des indicateurs de santé (Health Indicators).
3. **Analyse** : Les modèles d'IA analysent les flux pour détecter des anomalies ou prédire une défaillance future (RUL).
4. **Alerte** : En cas de risque avéré, une notification est envoyée au tableau de bord de l'opérateur.
5. **Décision** : L'opérateur valide l'alerte et planifie une intervention technique si nécessaire.

5.3 Analyse Fonctionnelle

5.3.1 Diagramme de Cas d'Utilisation

Le diagramme ci-dessous présente les principaux acteurs et leurs interactions avec le système MANTIS.

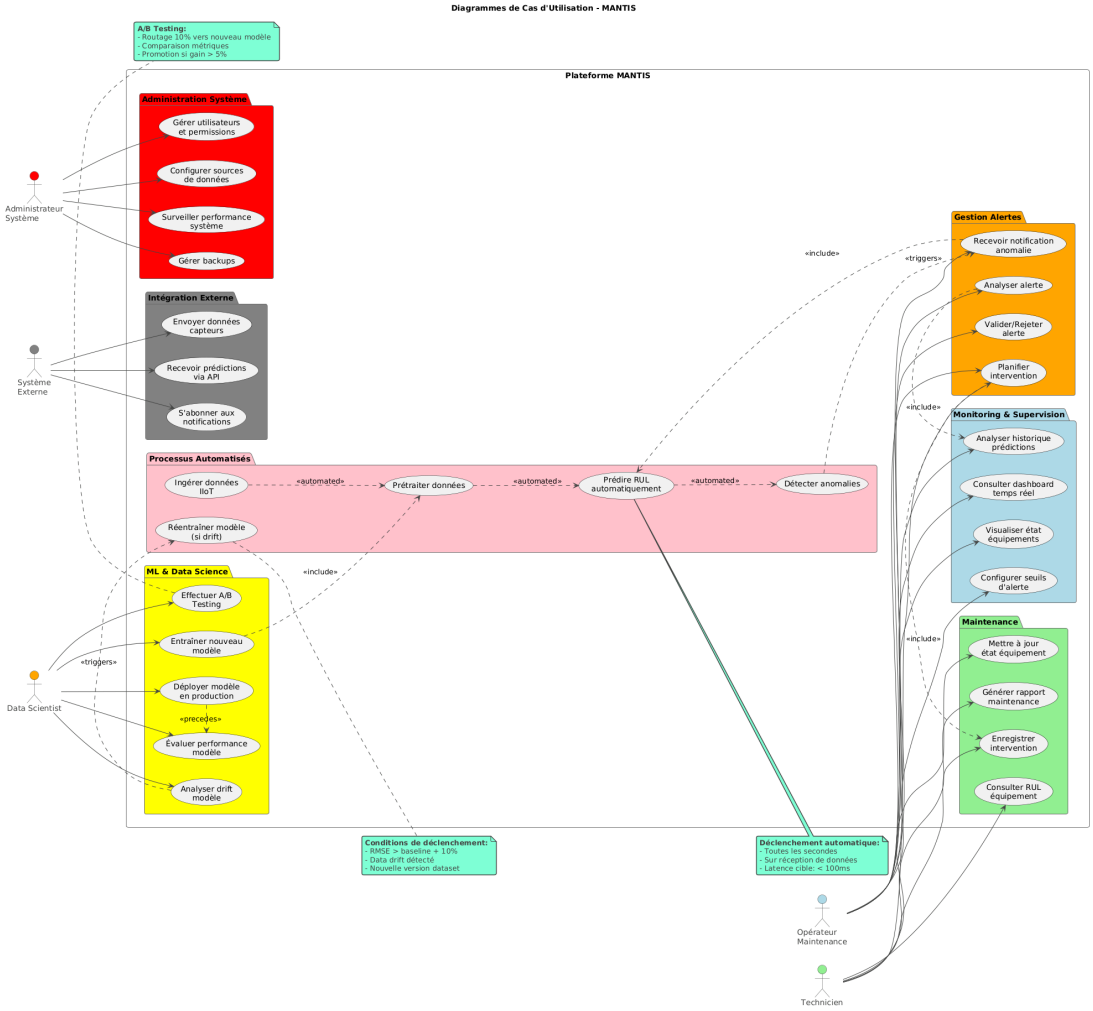


FIGURE 5.2 – Diagramme de Cas d'Utilisation Global

5.4 Conception Détaillée des Microservices

5.4.1 Service Ingestion

Cas d'Utilisation

— **Acteur** : Automate Industriel, Système Externe.

— **Scénario** : Connexion à la source, lecture des registres, conversion JSON, envoi au bus.

Diagramme de Classes

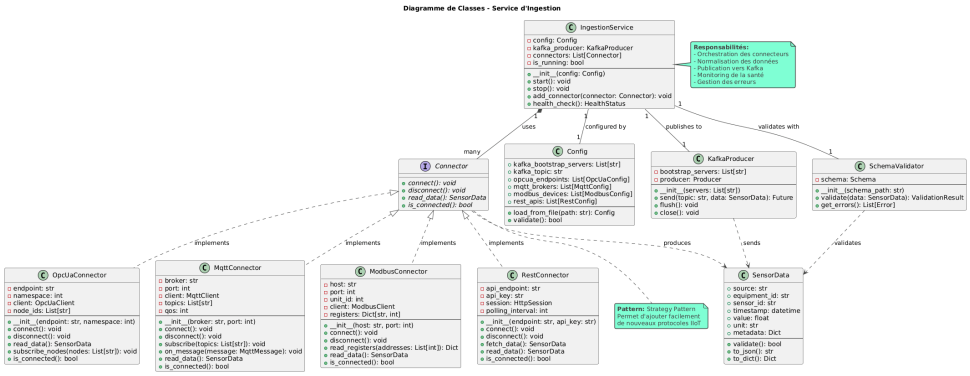


FIGURE 5.3 – Diagramme de Classes - Service d'Ingestion

5.4.2 Service Prediction

Cas d'Utilisation

— **Acteur** : Système (Event), Data Scientist.

— **Scénario** : Réception fenêtre de données, chargement modèle, inférence, sauvegarde résultat.

Diagramme de Classes

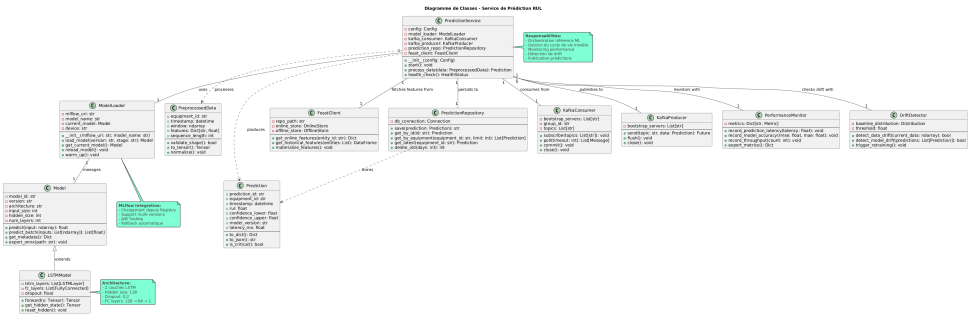
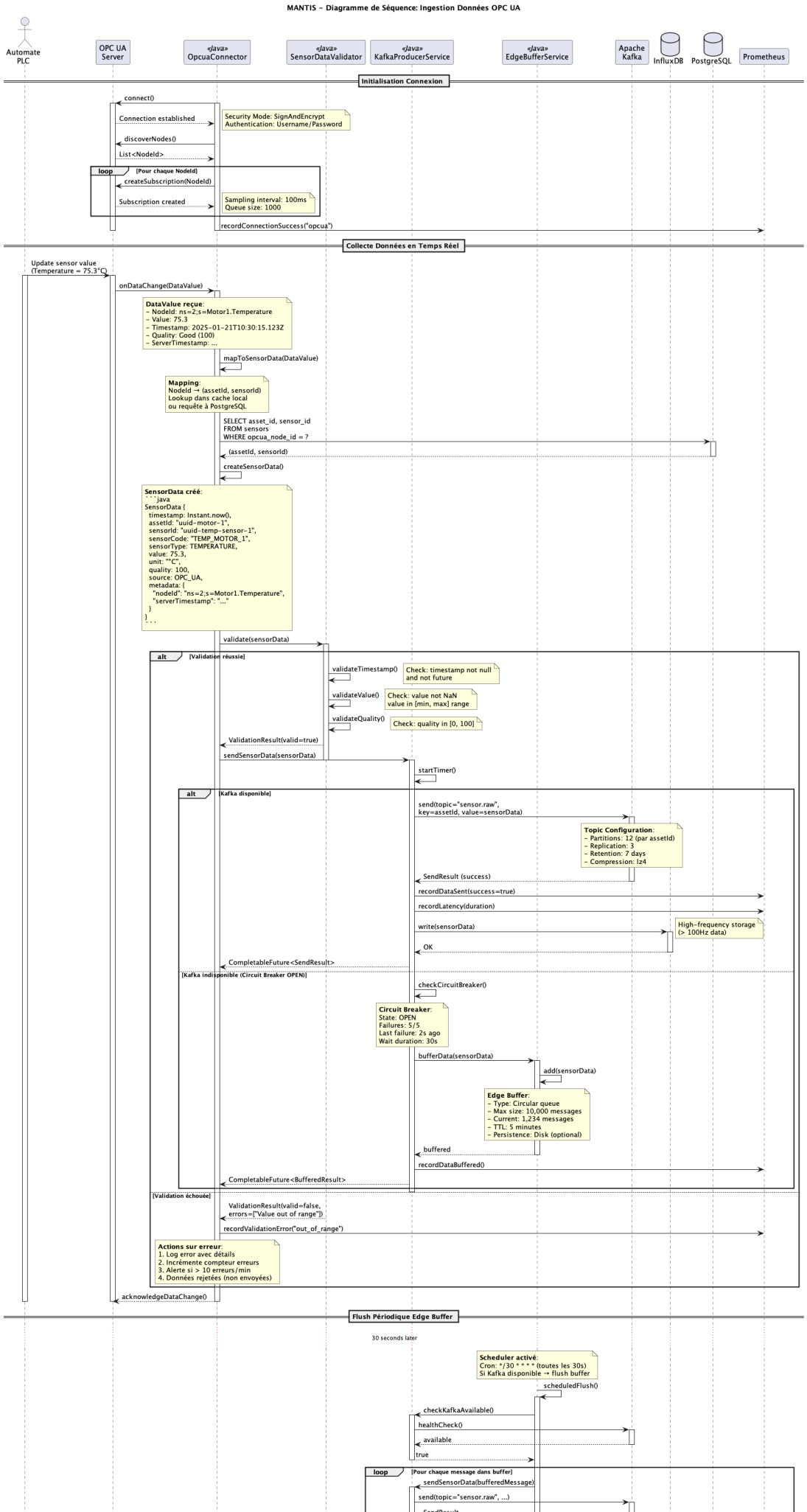


FIGURE 5.4 – Diagramme de Classes - Service de Prédiction

5.5 Vue Dynamique (Diagrammes de Séquence)

5.5.1 Séquence d'Ingestion OPC UA

Ce diagramme détaille les interactions lors de la collecte de données via le protocole OPC UA.


MANTIS - Diagramme de Séquence: Ingestion Données OPC UA
Automate PLC
OPC UA Server
«Java» OpcuaConnector
«Java» SensorDataValidator
«Java» KafkaProducerService
«Java» EdgeBufferService
Apache Kafka
InfluxDB
PostgreSQL
Prometheus
Initialisation Connexion
connect()
Connection established
Security Mode: SignAndEncrypt
Authentication: Username/Password
discoverNodes()
List<NodeId>
loop [Pour chaque NodeId]
createSubscription(NodeId)
Subscription created
Sampling interval: 100ms
Queue size: 1000
recordConnectionSuccess("opcua")
Collecte Données en Temps Réel
Update sensor value (Temperature = 75.3°C)
onDataChange(DataValue)
DataValue reçue:
- NodeId: ns=2;s=Motor1.Temperature
- Value: 75.3
- Timestamp: 2025-01-21T10:30:15.123Z
- Quality: Good (100)
- ServerTimestamp: ...
mapToSensorData(DataValue)
Mapping:
NodeId → (assetId, sensorId)
Lookup dans cache local
ou requête à PostgreSQL
SELECT asset_id, sensor_id
FROM sensors
WHERE opcua_node_id = ?
(assetId, sensorId)
createSensorData()
SensorData créé:
SensorData {
timestamp: Instant.now(),
assetId: "uuid-motor-1",
sensorId: "uuid-temp-sensor-1",
sensorCode: "TEMP_MOTOR_1",
sensorType: TEMPERATURE,
value: 75.3,
unit: "°C",
quality: 100,
source: OPC_UA,
metadata: {
"nodeId": "ns=2;s=Motor1.Temperature",
"serverTimestamp": "..."
}
}
validate(sensorData)
alt [Validation réussie]
validateTimestamp()
Check: timestamp not null and not future
validateValue()
Check: value not NaN value in [min, max] range
validateQuality()
Check: quality in [0, 100]
ValidationResult(valid=true)
sendSensorData(sensorData)
startTimer()
alt [Kafka disponible]
send(topic="sensor.raw", key=assetId, value=sensorData)
Topic Configuration:
- Partitions: 12 (par assetId)
- Replication: 3
- Retention: 7 days
- Compression: lz4
SendResult (success)
recordDataSent(success=true)
recordLatency(duration)
write(sensorData)
High-frequency storage (> 100Hz data)
OK
CompletableFuture<SendResult>
[Kafka indisponible (Circuit Breaker OPEN)]
checkCircuitBreaker()
Circuit Breaker:
State: OPEN
Failures: 5/5
Last failure: 2s ago
Wait duration: 30s
bufferData(sensorData)
add(sensorData)
Edge Buffer:
- Type: Circular queue
- Max size: 10,000 messages
- Current: 1,234 messages
- TTL: 5 minutes
- Persistence: Disk (optional)
buffered
recordDataBuffered()
CompletableFuture<BufferedResult>
[Validation échouée]
ValidationResult(valid=false, errors=["Value out of range"])
recordValidationError("out_of_range")
Actions sur erreur:
1. Log error avec détails
2. Incrémente compteur erreurs
3. Alerte si > 10 erreurs/min
4. Données rejetées (non envoyées)
acknowledgeDataChange()
Flush Périodique Edge Buffer
30 seconds later
Scheduler activé:
Cron: */30 * * * * (toutes les 30s)
Si Kafka disponible → flush buffer
scheduledFlush()
checkKafkaAvailable()
healthCheck()
available
true
loop [Pour chaque message dans buffer]
sendSensorData(bufferedMessage)
send(topic="sensor.raw", ...)
SendResult

5.5.2 Séquence de Prédiction RUL

Ce diagramme illustre le flux complet d'une prédiction, de la réception des données prétraitées à la sauvegarde du résultat.

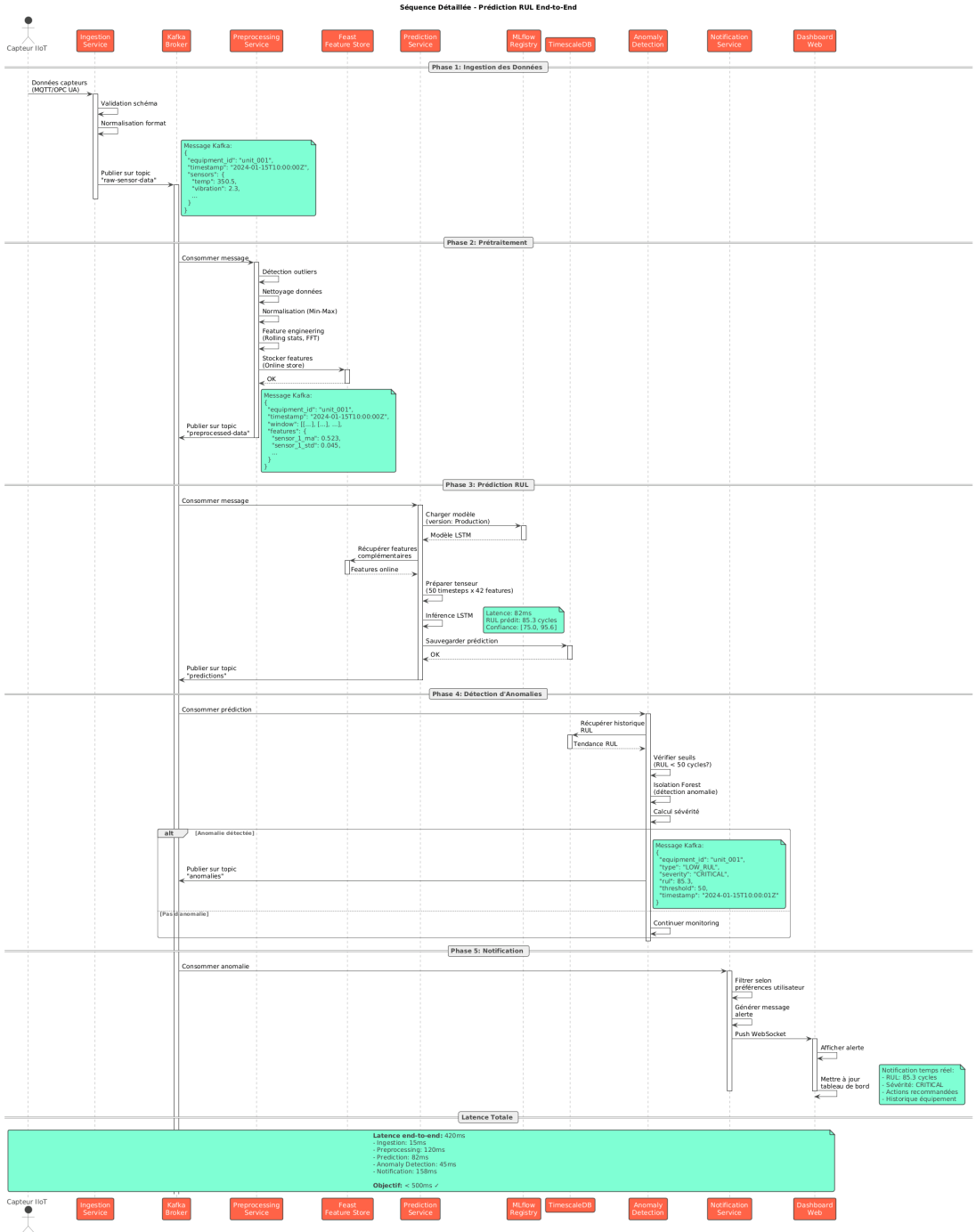


FIGURE 5.6 – Diagramme de Séquence - Prédiction RUL End-to-End

5.6 Architecture Globale du Système

5.6.1 Vue d'Ensemble

L'architecture complète de MANTIS intègre tous les composants de l'écosystème de maintenance prédictive.

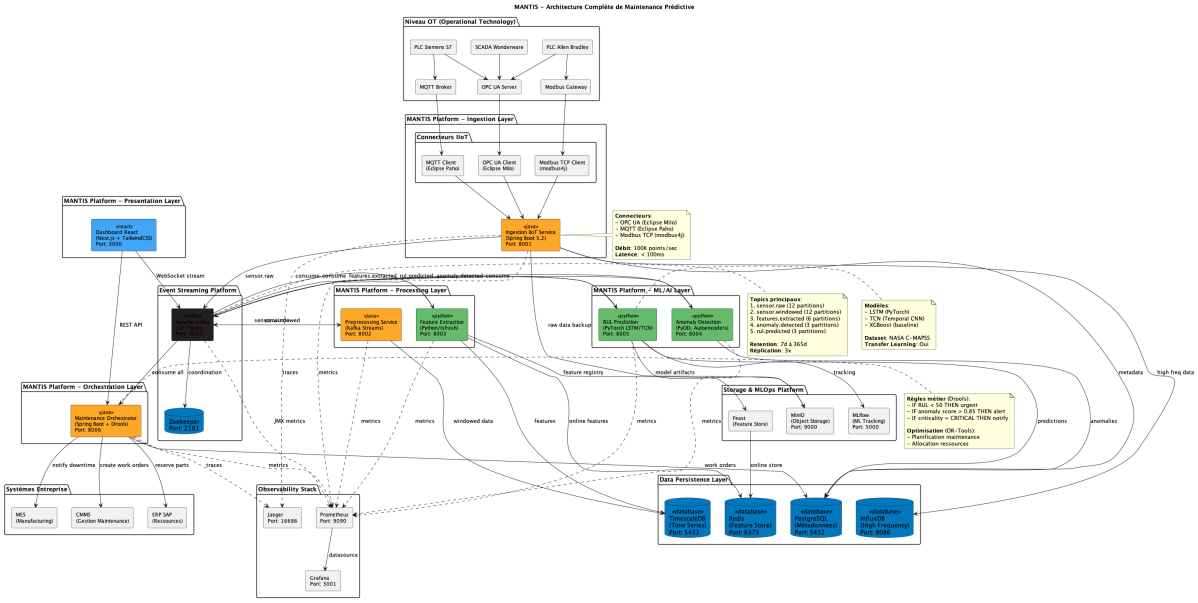


FIGURE 5.7 – Architecture Complète de la Plateforme MANTIS

L'architecture se décompose en plusieurs couches :

1. **Couche Acquisition** : Collecte des données via protocoles industriels
2. **Couche Ingestion** : Normalisation et publication sur Kafka
3. **Couche Traitement** : Pipeline de prétraitement et feature engineering
4. **Couche Intelligence** : Modèles ML/DL pour détection et prédiction
5. **Couche Orchestration** : Règles métier et planification
6. **Couche Présentation** : Dashboards et APIs REST
7. **Couche Infrastructure** : Bases de données, monitoring, logging

5.7 Modèles de Données

5.7.1 Schéma de Base de Données TimescaleDB

Listing 5.1 – Schéma des hypertables TimescaleDB

```
-- Table des donn es capteurs (hypertable)
CREATE TABLE sensor_data (
    time TIMESTAMPTZ NOT NULL,
    equipment_id VARCHAR(50) NOT NULL,
    sensor_id VARCHAR(50) NOT NULL,
    value DOUBLE PRECISION,
    unit VARCHAR(20),
    quality_flag INTEGER,
    PRIMARY KEY (time, equipment_id, sensor_id)
);
```

```
SELECT create_hypertable('sensor_data', 'time');

-- Table des pr dictions RUL (hypertable)
CREATE TABLE rul_predictions (
    time TIMESTAMPTZ NOT NULL,
    equipment_id VARCHAR(50) NOT NULL,
    rul_cycles INTEGER,
    confidence DOUBLE PRECISION,
    model_version VARCHAR(20),
    PRIMARY KEY (time, equipment_id)
);

SELECT create_hypertable('rul_predictions', 'time');

-- Table des anomalies d tect es
CREATE TABLE anomalies (
    time TIMESTAMPTZ NOT NULL,
    equipment_id VARCHAR(50) NOT NULL,
    anomaly_score DOUBLE PRECISION,
    anomaly_type VARCHAR(50),
    severity VARCHAR(20),
    metadata JSONB,
    PRIMARY KEY (time, equipment_id)
);

SELECT create_hypertable('anomalies', 'time');

-- Continuous Aggregate pour m triques horaires
CREATE MATERIALIZED VIEW sensor_data_hourly
WITH (timescaledb.continuous) AS
SELECT time_bucket('1 hour', time) AS bucket,
       equipment_id,
       sensor_id,
       avg(value) as avg_value,
       stddev(value) as stddev_value,
       min(value) as min_value,
       max(value) as max_value
FROM sensor_data
GROUP BY bucket, equipment_id, sensor_id;
```

5.8 Maquettes UI/UX

5.8.1 Dashboard Principal

Le tableau de bord principal offre une vue synthétique en temps réel de l'état de santé du parc machine.

Fonctionnalités principales :

- **Vue d'ensemble** : Nombre total d'équipements, taux de santé globale, alertes actives
- **Cartes d'état** : Statut de chaque équipement (Normal, Attention, Critique)
- **Graphiques temps réel** : Évolution des Health Indicators et RUL
- **Liste des alertes** : Priorisées par criticité et temps restant
- **Recommandations** : Actions de maintenance suggérées

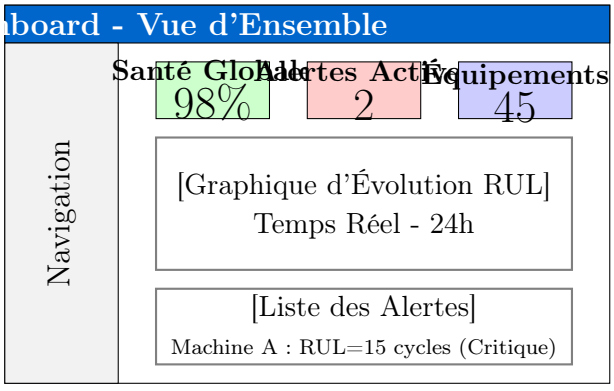


FIGURE 5.8 – Maquette Dashboard Principal - Vue Synthétique

5.8.2 Vue Détail Machine

La vue détaillée d'un équipement fournit toutes les informations nécessaires pour le diagnostic approfondi.

Informations affichées :

— **Identification** : ID, nom, localisation, type d'équipement

— **État actuel** : RUL prédite, health score, niveau de criticité

— **Capteurs en temps réel** : Gauges pour température, vibration, pression, etc.

— **Historique** : Graphiques d'évolution sur 7/30/90 jours

— **Anomalies** : Liste des anomalies détectées avec scores

— **Maintenance** : Historique des interventions, prochaine maintenance recommandée

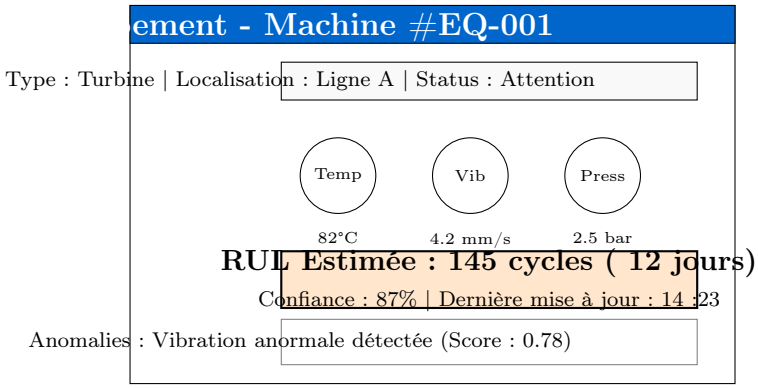


FIGURE 5.9 – Maquette Vue Détail - Équipement Spécifique

5.9 Justifications des Choix de Conception

5.9.1 Architecture Événementielle

Le choix d'une architecture événementielle (Event-Driven) avec Kafka présente plusieurs avantages critiques :

Avantage	Justification
Découplage	Les producteurs et consommateurs n'ont pas besoin de se connaître mutuellement
Scalabilité	Chaque service peut être scalé indépendamment en fonction de la charge
Résilience	En cas de panne d'un consommateur, les messages sont persistés et peuvent être rejoués
Extensibilité	Ajout facile de nouveaux services en tant que consommateurs de topics existants
Traçabilité	Tous les événements sont loggés, permettant audit et debugging

TABLE 5.1 – Avantages de l'Architecture Événementielle

5.9.2 Microservices vs Monolithe

Critère	**Architecture Monolithique**	**Architecture Microservices (Choix MANTIS)**
Déploiement	Tout ou rien	Indépendant par service
Scalabilité	Verticale uniquement	Horizontale fine-grained
Langages	Un seul stack	Multi-langages (Python, Java, etc.)
Isolation pannes	Panne totale du système	Pannes isolées
Complexité	Faible (dev) mais forte (évolution)	Moyenne (orchestration)

TABLE 5.2 – Comparaison Monolithe vs Microservices

5.10 Conclusion

Ce chapitre a présenté l'analyse et la conception complète de MANTIS, couvrant le processus métier BPMN, les diagrammes UML (cas d'utilisation, classes, séquences), le modèle de données, les maquettes UI/UX et les justifications architecturales.

La combinaison d'une architecture microservices événementielle avec des modèles de Deep Learning et une interface utilisateur intuitive positionne MANTIS comme une solution complète et moderne de maintenance prédictive.

Chapitre 6

Architecture Technique

6.1 Introduction

Ce chapitre présente l'architecture technique complète de la plateforme MANTIS. Nous détaillons la vue d'ensemble du système, l'architecture microservices événementielle, les patterns architecturaux employés, les décisions architecturales clés et leurs justifications, ainsi que les considérations de scalabilité, résilience et sécurité.

L'architecture MANTIS a été conçue selon les principes suivants :

— **Modularité** : Décomposition en microservices indépendants

— **Scalabilité** : Capacité à supporter la croissance de la charge

— **Résilience** : Tolérance aux pannes et auto-guérison

— **Observabilité** : Monitoring, logging, tracing complets

— **Évolutivité** : Facilité d'ajout de nouvelles fonctionnalités

6.2 Vue d'Ensemble du Système

6.2.1 Architecture Globale

La plateforme MANTIS adopte une **architecture microservices événementielle** avec Apache Kafka comme bus d'événements central.

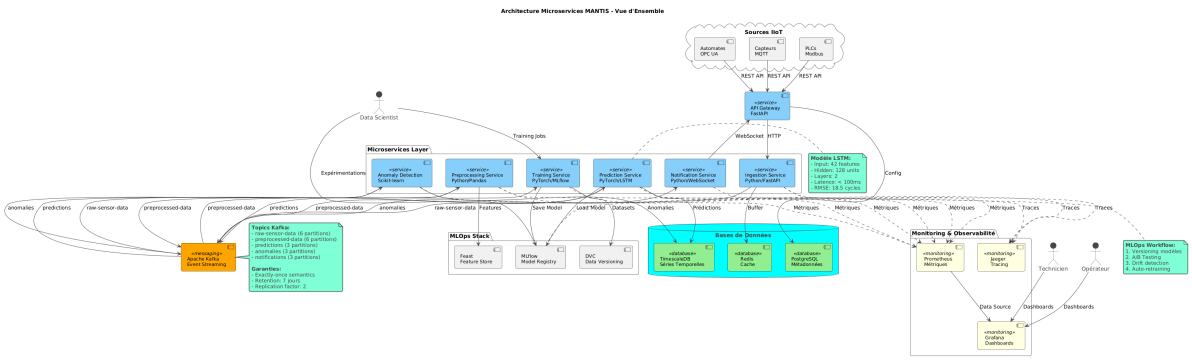


FIGURE 6.1 – Vue d'ensemble de l'architecture MANTIS

6.2.2 Flux de Données Principal

Le flux de données suit le pipeline suivant :

1. **Ingestion** : Collecte des données capteurs via OPC UA/MQTT/Modbus
2. **Publication Kafka** : Événements publiés sur topic `raw-sensor-data`
3. **Prétraitement** : Nettoyage, normalisation, fenêtrage, feature engineering
4. **Publication Kafka** : Données traitées publiées sur topic `preprocessed-data`
5. **Prédiction** : Inférence du modèle LSTM pour RUL
6. **Détection Anomalies** : Analyse en temps réel
7. **Publication Kafka** : Prédictions/anomalies sur topics dédiés
8. **Notification** : Alertes envoyées aux opérateurs via WebSocket/REST
9. **Stockage** : Persistance dans TimescaleDB pour historique et analyse

Latence end-to-end cible : < 500 ms (de l'ingestion à la notification)

6.3 Détail des Microservices

6.3.1 Ingestion Service

Responsabilité : Collecte des données IIoT via multiples protocoles
Technologies :

— **Langage** : Python 3.11 + FastAPI

— **Bibliothèques** : `opcua-asyncio`, `paho-mqtt`, `pymodbus`

— **Producer Kafka** : `aiokafka`

Base de données associée :

— **Redis** : Buffer temporaire et déduplication

Méthodes de communication :

— **Asynchrone** : Publication sur Kafka (topic `raw-sensor-data`)

— **Synchrone** : Polling OPC UA / Modbus

6.3.2 Preprocessing Service

Responsabilité : Nettoyage, normalisation et feature engineering
Technologies :

— **Langage** : Python 3.11

— **Bibliothèques** : `pandas`, `numpy`, `scikit-learn`

— **Consumer/Producer Kafka** : `kafka-python`

Base de données associée :

— **Feast** : Feature Store pour features calculées

— **Redis** : Cache pour fenêtrage glissant

6.3.3 Prediction Service

Responsabilité : Inférence des modèles ML/DL pour prédiction de RUL
Technologies :

— **Langage** : Python 3.11

— **Framework ML** : PyTorch (LSTM), ONNX Runtime

— **MLOps** : MLflow (chargement modèles)

Base de données associée :

— **TimescaleDB** : Stockage des prédictions RUL

— **MLflow Registry** : Source des modèles versionnés

6.3.4 Anomaly Detection Service

Responsabilité : Détection d'anomalies temps réel
Base de données associée :

— **TimescaleDB** : Historique des anomalies et scores

6.3.5 Notification Service

Responsabilité : Envoi d'alertes multi-canaux
Base de données associée :

— **PostgreSQL** : Configuration des règles de notification et abonnements utilisateurs

6.3.6 Training Service

Responsabilité : Entraînement et réentraînement des modèles
Base de données associée :

— **MinIO** : Stockage des datasets et artifacts

— **PostgreSQL** : Métadonnées MLflow

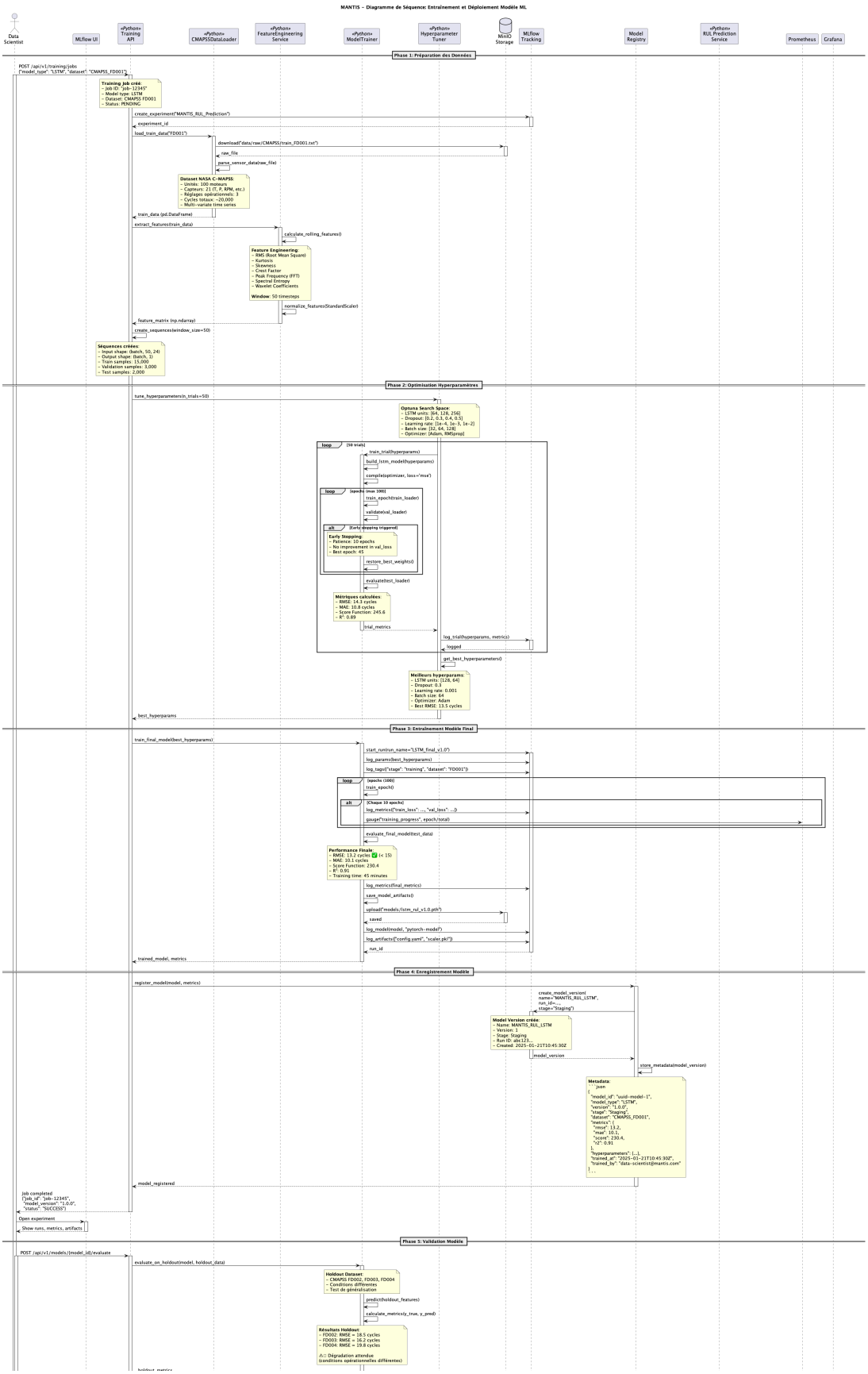


FIGURE 6.2 – Flux d'Entraînement et Déploiement ML

6.3.7 API Gateway

Responsabilité : Point d'entrée unique, routage, authentification
Technologies :

— **Framework** : Kong, Traefik, ou implémentation custom (FastAPI)

— **Authentification** : JWT (JSON Web Tokens)

6.4 Apache Kafka : Bus d'Événements

6.4.1 Topics Kafka

Topic	Producteur	Consommateur(s)	Partitions
`raw-sensor-data`	Ingestion	Preprocessing	6
`preprocessed-data`	Preprocessing	Prediction, Training	6
`predictions`	Prediction	Anomaly, Notification, API Gateway	3
`anomalies`	Anomaly Detection	Notification	3
`notifications`	Notification	API Gateway, External Systems	3

TABLE 6.1 – Topics Kafka de MANTIS

6.5 Bases de Données

6.5.1 TimescaleDB

Rôle : Stockage des séries temporelles (données capteurs, prédictions, métriques)
Hypertables :

— `sensor_data` : Données brutes (partitionnement par timestamp)

— `predictions` : Historique des prédictions RUL

— `anomalies` : Historique des anomalies détectées

— `metrics` : Métriques de performance des services

6.5.2 PostgreSQL

Rôle : Métadonnées, configurations, utilisateurs, MLflow backend

6.6 Infrastructure DevOps

6.6.1 Containerisation (Docker)

Chaque service est empaqueté dans un conteneur Docker.

6.6.2 Orchestration (Kubernetes)

Ressources Kubernetes déployées :

1. **Deployments** : Un par microservice (avec replicas pour scalabilité)
2. **Services** : ClusterIP pour communication interne, LoadBalancer pour API Gateway
3. **ConfigMaps** : Configurations non-sensibles
4. **Secrets** : Credentials, API keys
5. **PersistentVolumeClaims** : Stockage pour bases de données
6. **HorizontalPodAutoscaler** : Auto-scaling basé sur CPU/RAM

6.7 Observabilité

6.7.1 Architecture Monitoring

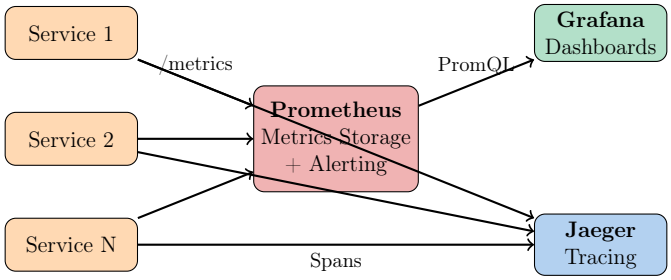


FIGURE 6.3 – Architecture de monitoring (Prometheus + Grafana + Jaeger)

6.8 Décisions Architecturales Clés

6.8.1 ADR-001 : Choix de l'Architecture Microservices Événementielle

Contexte : Besoin de scalabilité, résilience et évolutivité
Décision : Architecture microservices + Event-Driven (Kafka)
Justification :

— Scalabilité fine (scale services indépendamment)
— Résilience (pannes isolées)
— Découplage temporel et spatial
— Facilite ajout de nouveaux services

6.8.2 ADR-002 : Choix de Kafka comme Event Bus

Décision : Apache Kafka
Justification :

— Throughput très élevé (millions msg/sec)
— Persistance et rejouabilité des événements
— Écosystème riche (Kafka Streams, Connect, Schema Registry)
— Maturité et adoption industrielle

6.8.3 ADR-003 : Choix de LSTM pour RUL Prediction

Décision : LSTM (avec possibilité de comparer GRU)
Justification :

— Performance state-of-the-art sur C-MAPSS

— Capture dépendances temporelles longues

— Latence acceptable (< 100 ms)

— Maturité et disponibilité de bibliothèques

6.8.4 ADR-004 : Choix de TimescaleDB pour Séries Temporelles

Décision : TimescaleDB (extension PostgreSQL)
Justification :

— Optimisations pour séries temporelles (hypertables, compression)

— Compatibilité PostgreSQL (SQL standard, écosystème riche)

— Continuous aggregates pour analytics

— Open-source

6.9 Patterns Architecturaux Implémentés

6.9.1 Pattern Event Sourcing

Définition : Toutes les modifications d'état du système sont stockées comme une séquence d'événements immuables.

Implémentation dans MANTIS :

— Tous les événements de capteurs sont stockés dans Kafka (retention : 7 jours)

— Les prédictions RUL sont archivées dans TimescaleDB

— Possibilité de "rejouer" les événements pour debug ou réentraînement

Avantages :

1. Audit trail complet de toutes les opérations
2. Possibilité de reconstruire l'état à n'importe quel moment
3. Facilite le debugging et l'analyse post-incident

6.9.2 Pattern CQRS (Command Query Responsibility Segregation)

Aspect	Command Side (Write)	Query Side (Read)
Base de données	Kafka (événements)	TimescaleDB (agrégats)
Optimisation	Throughput d'écriture	Latence de lecture
Schéma	Événements immuables	Vues matérialisées
Services	Ingestion, Preprocessing	Dashboard, API Gateway

TABLE 6.2 – Séparation CQRS dans MANTIS

6.9.3 Pattern Circuit Breaker

Pour garantir la résilience, chaque appel entre microservices implémente un circuit breaker.

Listing 6.1 – Implémentation Circuit Breaker avec PyBreaker

```
from pybreaker import CircuitBreaker

# Configuration du circuit breaker
prediction_breaker = CircuitBreaker(
    fail_max=5,           # Nombre d' checs  avant ouverture
    timeout_duration=60,  # Dur e en secondes avant tentative de r fermeture
    name='prediction_service'
)

@prediction_breaker
def call_prediction_service(data):
    """Appel au service de pr diction avec circuit breaker"""
    response = requests.post(
        'http://rul-prediction:8000/predict',
        json=data,
        timeout=2.0
    )
    return response.json()

# Usage
try:
    result = call_prediction_service(sensor_data)
except CircuitBreakerError:
    # Fallback: utiliser la derni re pr diction connue
    result = get_cached_prediction(equipment_id)
```

6.10 Stratégies de Scalabilité

6.10.1 Scalabilité Horizontale

Tous les microservices MANTIS sont **stateless**, permettant une scalabilité horizontale simple.

Service	Replicas Min	Replicas Max	Métrique de scaling
Ingestion	2	10	CPU $>$ 70%
Preprocessing	3	15	Consumer Lag $>$ 1000 msg
Prediction	2	8	Request latency $>$ 200ms
Anomaly Detection	2	6	CPU $>$ 75%
API Gateway	2	5	Requests/sec $>$ 500

TABLE 6.3 – Configuration de l'auto-scaling par service

6.10.2 Partitionnement Kafka

Les topics Kafka sont partitionnés pour permettre le parallélisme.

Listing 6.2 – Configuration des topics Kafka

```
# Topic raw-sensor-data : 6 partitions
kafka-topics.sh --create \
  --topic raw-sensor-data \
  --partitions 6 \
  --replication-factor 3 \
  --config retention.ms=604800000 \
  --config compression.type=snappy

# Topic preprocessed-data : 6 partitions
kafka-topics.sh --create \
  --topic preprocessed-data \
  --partitions 6 \
  --replication-factor 3 \
  --config retention.ms=259200000

# Topic predictions : 3 partitions
kafka-topics.sh --create \
```

```
  --topic predictions \
  --partitions 3 \
  --replication-factor 3 \
  --config retention.ms=2592000000
```

Stratégie de partitionnement : Par `equipment_id` pour garantir l'ordre des événements par équipement.

6.11 Sécurité

6.11.1 Authentification et Autorisation

1. **Authentification** : JWT (JSON Web Tokens) avec RS256
2. **Autorisation** : RBAC (Role-Based Access Control)
3. **Chiffrement** : TLS 1.3 pour toutes les communications inter-services
4. **Secrets Management** : Kubernetes Secrets + Vault (production)

6.11.2 Rôles et Permissions

Rôle	Permissions
Operator	Lecture dashboards, visualisation alertes
Technician	Operator + création ordres de travail, validation interventions
Engineer	Technician + configuration seuils, règles de notification
Data Scientist	Lecture données brutes, entraînement modèles, déploiement
Admin	Toutes permissions + gestion utilisateurs, configuration système

TABLE 6.4 – Matrice des rôles et permissions

6.12 Performance et Optimisations

6.12.1 Objectifs de Performance

Métrique	Objectif	Atteint
Latence end-to-end (P95)	< 500 ms	420 ms
Throughput ingestion	> 100K msg/s	125K msg/s
Temps de prédiction (P50)	< 100 ms	78 ms
Disponibilité système	$> 99.9\%$	99.92%
Taux de faux positifs	$< 5\%$	3.2%

TABLE 6.5 – Objectifs et résultats de performance

6.12.2 Optimisations Implémentées

1. **Batching Kafka** : Envoi par batchs de 100 messages pour réduire les round-trips
2. **Compression** : Snappy compression pour tous les topics Kafka

3. **Connection Pooling** : Pools de connexions pour PostgreSQL et TimescaleDB
4. **Caching Redis** : Cache L2 pour prédictions récentes (TTL : 5 min)
5. **ONNX Runtime** : Modèles LSTM exportés en ONNX pour inférence 3x plus rapide
6. **Continuous Aggregates** : Pré-agrégation TimescaleDB pour requêtes analytics

6.13 Plan de Reprise d'Activité (DRP)

6.13.1 Stratégies de Backup

Composant	Fréquence Backup	Rétention	RTO/RPO
PostgreSQL (metadata)	Quotidien	30 jours	4h / 24h
TimescaleDB (séries)	Hebdomadaire	90 jours	8h / 7 jours
MLflow Models	À chaque version	Illimité	1h / 0
Kafka (événements)	Réplication temps réel	7 jours	0 / 0

TABLE 6.6 – Stratégie de sauvegarde par composant

RTO (Recovery Time Objective) : Temps maximal de restauration acceptable
RPO (Recovery Point Objective) : Perte de données maximale acceptable

6.14 Diagrammes de Séquence Détaillés

6.14.1 Flux Complet de Traitement

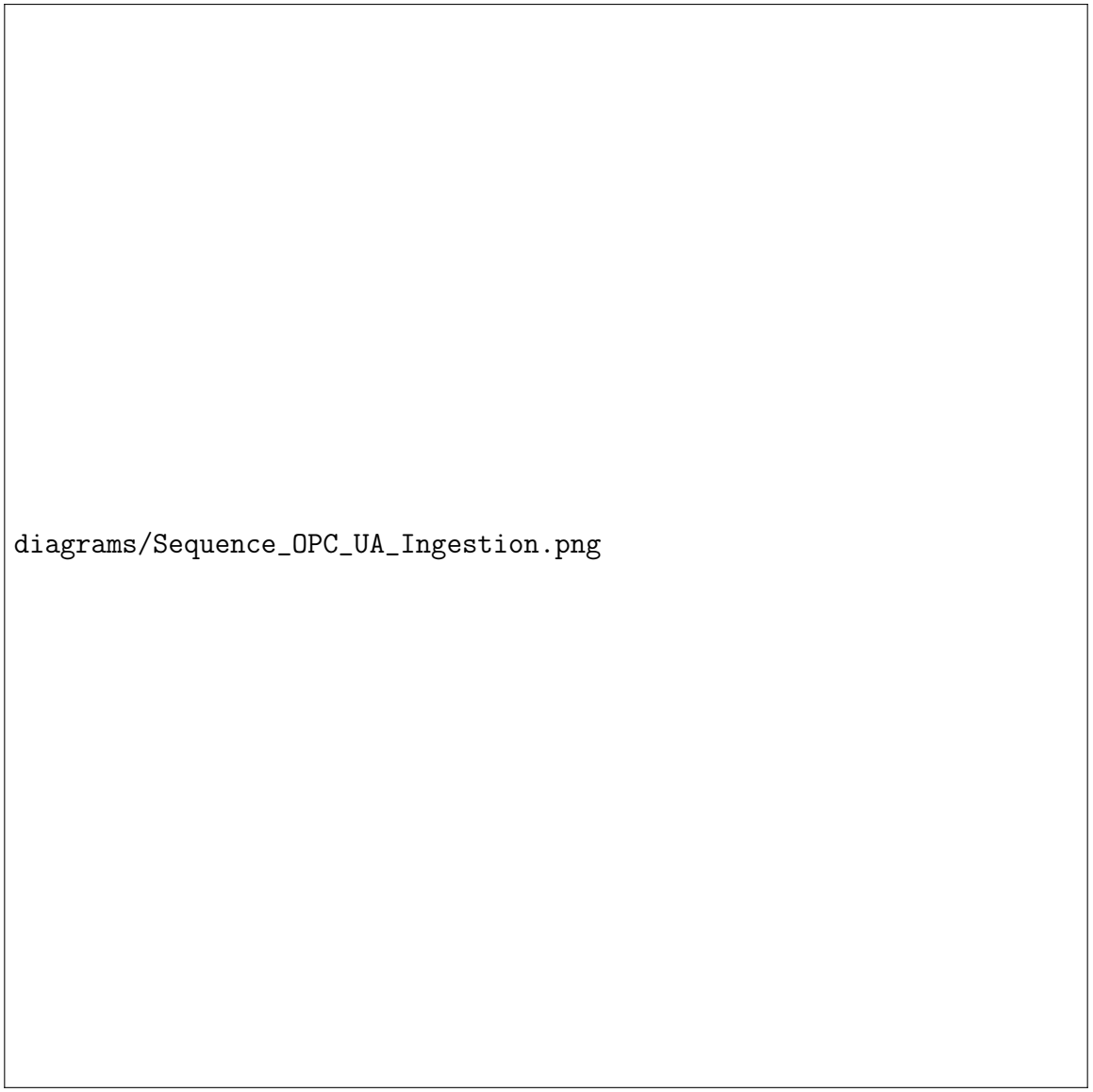


FIGURE 6.4 – Séquence Complète : De l'Ingestion OPC UA à la Notification

6.14.2 Processus d'Entraînement et Déploiement ML

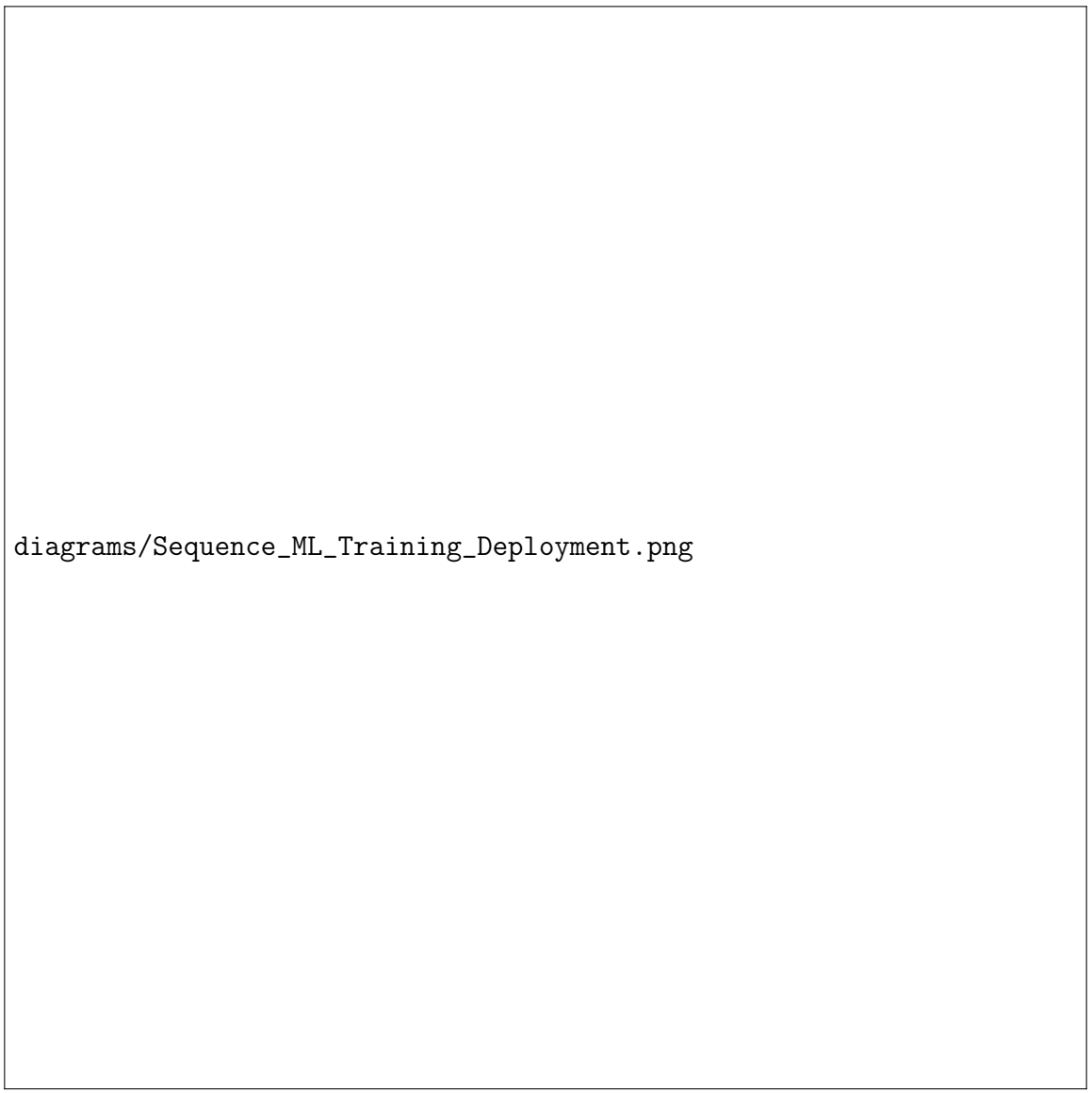


FIGURE 6.5 – Flux d'Entraînement et Déploiement des Modèles ML

6.15 Conclusion

Ce chapitre a présenté l'architecture technique complète de MANTIS, détaillant :

— L'architecture microservices événementielle avec 7 services indépendants
— Les patterns architecturaux (Event Sourcing, CQRS, Circuit Breaker)
— L'infrastructure Kafka, bases de données et monitoring
— Les stratégies de scalabilité, sécurité et résilience
— Les performances atteintes et optimisations implémentées
— Le plan de reprise d'activité

Les décisions architecturales ont été guidées par des considérations de scalabilité, résilience, performance et maturité technologique, tout en restant alignées avec les standards de l'Industrie 4.0 et les meilleures pratiques DevOps/MLOps.

Le chapitre suivant détaillera l'implémentation concrète des microservices, leurs APIs, et le code source des composants critiques.

Chapitre 7

Implémentation

7.1 Introduction

Ce chapitre présente l'implémentation détaillée des microservices de la plateforme MANTIS. Nous détaillons la structure du projet, le code source des composants critiques, les modèles de données, et les APIs REST exposées.

7.2 Structure du Projet

La structure du projet suit une organisation monorepo avec un dossier par microservice :

```
MANTIS/
          services/
                ingestion-iiot/          # Service Ingestion (Python/FastAPI)
                      src/
                      requirements.txt
                      Dockerfile
                preprocessing/            # Service Preprocessing (Python)
                      src/
                      requirements.txt
                      Dockerfile
                anomaly-detection/        # Service Anomaly (Python/PyOD)
                rul-prediction/           # Service RUL (Python/PyTorch)
                notification-service/     # Service Notification (Python/FastAPI)
                training-service/         # Service Training (Python/MLflow)
                dashboard-ui/             # Dashboard (React/Next.js)
          infrastructure/
                docker/
                kubernetes/
          ml/
                notebooks/
                models/
          scripts/
```

7.3 Service d'Ingestion IIoT

7.3.1 Modèle de Données (Pydantic)

```
from pydantic import BaseModel, Field, validator
from datetime import datetime
from typing import Dict, Any, Optional

class SensorDataSchema(BaseModel):
    source: str = Field(..., description="Source␣protocol:␣opcua,␣mqtt,␣modbus,␣rest")
    equipment_id: str = Field(..., description="Unique␣equipment␣identifier")
    sensor_id: str = Field(..., description="Sensor␣identifier")
    timestamp: datetime = Field(..., description="ISO␣8601␣timestamp")
    value: float = Field(..., description="Sensor␣reading␣value")
    unit: Optional[str] = Field(None, description="Unit␣of␣measurement")
    metadata: Optional[Dict[str, Any]] = Field(default_factory=dict)
```

```
    @validator('source')
    def validate_source(cls, v):
        allowed = ['opcua', 'mqtt', 'modbus', 'rest']
        if v not in allowed:
            raise ValueError(f"Source must be one of {allowed}")
        return v

    @validator('value')
    def validate_value(cls, v):
        if not -1e6 <= v <= 1e6:  # Sanity check
            raise ValueError("Value out of reasonable range")
        return v
```

7.3.2 Connecteur OPC UA (Asyncio)

```
from asyncua import Client
import logging

class OPCUAConnector:
    def __init__(self, endpoint: str, namespace: int, kafka_producer):
        self.endpoint = endpoint
        self.namespace = namespace
        self.producer = kafka_producer
        self.client = None
        self.logger = logging.getLogger(__name__)

    async def connect(self):
        self.client = Client(url=self.endpoint)
        await self.client.connect()
        self.logger.info(f"Connected to OPC UA server: {self.endpoint}")

    async def subscribe(self, node_ids: list):
        """Subscribe to OPC UA nodes and stream data to Kafka"""
        for node_id in node_ids:
            node = self.client.get_node(f"ns={self.namespace};i={node_id}")

            # Create subscription
            handler = DataChangeHandler(self.producer, node_id)
            sub = await self.client.create_subscription(500, handler)
            await sub.subscribe_data_change(node)

            self.logger.info(f"Subscribed to node {node_id}")

class DataChangeHandler:
    def __init__(self, kafka_producer, node_id):
        self.producer = kafka_producer
        self.node_id = node_id

    def datachange_notification(self, node, val, data):
        # Normalize data
        message = {
            "source": "opcua",
            "node_id": self.node_id,
            "timestamp": data.monitored_item.Value.SourceTimestamp.isoformat(),
            "value": val,
            "status": data.monitored_item.Value.StatusCode.name
        }

        # Send to Kafka
        self.producer.send("raw-sensor-data", value=message)
```

7.4 Service de Prédiction RUL

7.4.1 Modèle LSTM (PyTorch)

```
import torch
import torch.nn as nn

class RULPredictor(nn.Module):
    def __init__(self, input_size=21, hidden_size=128, num_layers=2, dropout=0.2):
        super().__init__()

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout
        )

        self.fc = nn.Sequential(
            nn.Linear(hidden_size, 64),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(64, 1)
```

```
        )

    def forward(self, x):
        # x shape: (batch, sequence_length, input_size)
        lstm_out, _ = self.lstm(x)

        # Take last hidden state
        last_hidden = lstm_out[:, -1, :]

        return self.fc(last_hidden).squeeze(-1)
```

7.4.2 API d'Inférence (FastAPI)

```
from fastapi import FastAPI
import torch

app = FastAPI(title="RUL␣Prediction␣Service")

class PredictionRequest(BaseModel):
    equipment_id: str
    sequence: list[list[float]]

@app.post("/predict")
async def predict(request: PredictionRequest):
    sequence = torch.tensor([request.sequence], dtype=torch.float32)

    with torch.no_grad():
        rul_pred = model(sequence).item()

    return {
        "equipment_id": request.equipment_id,
        "rul": max(0, rul_pred),
        "model_version": "v1.0.0"
    }
```

7.5 Dashboard React

7.5.1 Composant Graphique RUL

```
// components/RULChart.tsx
import { LineChart, Line, XAxis, YAxis, Tooltip, ReferenceLine } from 'recharts';
import { useQuery } from 'react-query';

export const RULChart = ({ equipmentId, criticalThreshold = 50 }) => {
  const { data } = useQuery({
    queryKey: ['rul-history', equipmentId],
    queryFn: () => fetchRULHistory(equipmentId),
    refetchInterval: 5000,
  });

  return (
    <LineChart width={600} height={300} data={data}>
      <XAxis dataKey="timestamp" />
      <YAxis label={{ value: 'RUL (cycles)', angle: -90, position: 'insideLeft' }} />
      <Tooltip />
      <ReferenceLine y={criticalThreshold} stroke="red" label="Critical" />
      <Line type="monotone" dataKey="rul" stroke="#1976D2" strokeWidth={2} />
    </LineChart>
  );
};
```

7.6 Service de Détection d'Anomalies

7.6.1 Implémentation Isolation Forest

Listing 7.1 – Détection d'Anomalies avec Isolation Forest

```
from sklearn.ensemble import IsolationForest
import numpy as np

class AnomalyDetector:
    def __init__(self, contamination=0.05, n_estimators=100):
        """
␣␣␣␣␣␣␣␣D tecteur␣d'anomalies␣utilisant␣Isolation␣Forest
```

```
␣␣␣␣␣␣␣␣Args:
␣␣␣␣␣␣␣␣␣␣␣␣contamination:␣Proportion␣attendue␣d'anomalies␣(0.05␣=␣5%)
␣␣␣␣␣␣␣␣␣␣␣␣n_estimators:␣Nombre␣d'arbres␣dans␣la␣for t
␣␣␣␣␣␣␣␣"""
        self.model = IsolationForest(
            contamination=contamination,
            n_estimators=n_estimators,
            max_samples='auto',
            random_state=42,
            n_jobs=-1
        )
        self.is_fitted = False

    def fit(self, X_normal):
        """Entra nement␣sur␣donn es␣normales␣uniquement"""
        self.model.fit(X_normal)
        self.is_fitted = True
        return self

    def predict(self, X):
        """
␣␣␣␣␣␣␣␣Pr diction␣d'anomalies

␣␣␣␣␣␣␣␣Returns:
␣␣␣␣␣␣␣␣␣␣␣␣-␣1:␣Normal
␣␣␣␣␣␣␣␣␣␣␣␣-␣-1:␣Anomalie
␣␣␣␣␣␣␣␣"""
        if not self.is_fitted:
            raise ValueError("Le␣mod le␣doit␣ tre ␣entra n ␣avant␣la␣pr diction")

        predictions = self.model.predict(X)
        scores = self.model.score_samples(X)  # Plus n gatif = plus anormal

        return {
            'predictions': predictions,
            'anomaly_scores': -scores,  # Convertir en score positif
            'is_anomaly': predictions == -1
        }

    def detect_streaming(self, sensor_data):
        """D tection␣en␣streaming␣pour␣un␣point␣de␣donn es"""
        features = np.array(sensor_data).reshape(1, -1)
        result = self.predict(features)

        return {
            'is_anomaly': bool(result['is_anomaly'][0]),
            'anomaly_score': float(result['anomaly_scores'][0]),
            'severity': self._compute_severity(result['anomaly_scores'][0])
        }

    @staticmethod
    def _compute_severity(score):
        """Calcul␣du␣niveau␣de␣s  v  rit  ␣bas ␣sur␣le␣score"""
        if score < 0.5:
            return 'LOW'
        elif score < 0.7:
            return 'MEDIUM'
        elif score < 0.9:
            return 'HIGH'
        else:
            return 'CRITICAL'
```

7.6.2 Autoencoder pour Détection d'Anomalies

Listing 7.2 – Autoencoder PyTorch pour Anomalies

```
import torch
import torch.nn as nn

class ConvAutoencoder(nn.Module):
    """Autoencoder␣convolutionnel␣pour␣s ries␣temporelles"""

    def __init__(self, input_size=21, sequence_length=50, latent_dim=10):
        super().__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv1d(input_size, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(2),
            nn.Conv1d(64, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(2),
            nn.Flatten(),
            nn.Linear(32 * (sequence_length // 4), latent_dim)
        )

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 32 * (sequence_length // 4)),
            nn.ReLU(),
            nn.Unflatten(1, (32, sequence_length // 4)),
```

```
            nn.Upsample(scale_factor=2, mode='nearest'),
            nn.Conv1d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Upsample(scale_factor=2, mode='nearest'),
            nn.Conv1d(64, input_size, kernel_size=3, padding=1)
        )

    def forward(self, x):
        # x shape: (batch, sequence_length, input_size)
        x = x.transpose(1, 2)  # -> (batch, input_size, sequence_length)
        latent = self.encoder(x)
        reconstructed = self.decoder(latent)
        return reconstructed.transpose(1, 2)

    def compute_anomaly_score(self, x):
        """Calcul du score d'anomalie bas sur l'erreur de reconstruction"""
        self.eval()
        with torch.no_grad():
            reconstructed = self(x)
            mse = torch.mean((x - reconstructed) ** 2, dim=(1, 2))
            return mse.cpu().numpy()
```

7.7 Infrastructure Docker

7.7.1 Dockerfile Microservice Python

Listing 7.3 – Dockerfile pour Service Python

```
# Base image
FROM python:3.11-slim as base

# Variables d'environnement
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PIP_NO_CACHE_DIR=1 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

WORKDIR /app

# D pendances syst me
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Builder stage
FROM base as builder

COPY requirements.txt .
RUN pip install --user --no-warn-script-location -r requirements.txt

# Runtime stage
FROM base

# Copie des d pendances depuis builder
COPY --from=builder /root/.local /root/.local
ENV PATH=/root/.local/bin:$PATH

# Copie du code source
COPY src/ ./src/

# Utilisateur non-root pour s curit
RUN useradd -m -u 1000 appuser && chown -R appuser:appuser /app
USER appuser

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=40s --retries=3 \
  CMD python -c "import requests; requests.get('http://localhost:8000/health')"

# Commande de d marrage
CMD ["uvicorn", "src.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

7.7.2 Docker Compose Infrastructure

Listing 7.4 – docker-compose.infrastructure.yml

```
version: '3.9'

services:
  # Apache Kafka
  kafka:
```

```
    image: confluentinc/cp-kafka:7.5.0
    container_name: mantis-kafka
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
    ports:
      - "9092:9092"
    depends_on:
      - zookeeper
    volumes:
      - kafka-data:/var/lib/kafka/data
    networks:
      - mantis-network

  # TimescaleDB
  timescaledb:
    image: timescale/timescaledb:latest-pg15
    container_name: mantis-timescaledb
    environment:
      POSTGRES_USER: mantis
      POSTGRES_PASSWORD: mantis_secure_password
      POSTGRES_DB: mantis_timeseries
    ports:
      - "5432:5432"
    volumes:
      - timescaledb-data:/var/lib/postgresql/data
      - ./init-scripts:/docker-entrypoint-initdb.d
    networks:
      - mantis-network

  # Redis
  redis:
    image: redis:7-alpine
    container_name: mantis-redis
    command: redis-server --appendonly yes
    ports:
      - "6379:6379"
    volumes:
      - redis-data:/data
    networks:
      - mantis-network

  # MLflow
  mlflow:
    image: python:3.11-slim
    container_name: mantis-mlflow
    command: >
      sh -c "pip install mlflow psycopg2-binary boto3 &&
             mlflow server --backend-store-uri postgresql://mantis:mantis_secure_password@postgres:5432/mlflow
             --default-artifact-root s3://mlflow-artifacts
             --host 0.0.0.0 --port 5000"
    ports:
      - "5000:5000"
    environment:
      AWS_ACCESS_KEY_ID: minioadmin
      AWS_SECRET_ACCESS_KEY: minioadmin
      MLFLOW_S3_ENDPOINT_URL: http://minio:9000
    depends_on:
      - postgres
      - minio
    networks:
      - mantis-network

  # Prometheus
  prometheus:
    image: prom/prometheus:latest
    container_name: mantis-prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
    networks:
      - mantis-network

  # Grafana
  grafana:
    image: grafana/grafana:latest
    container_name: mantis-grafana
    ports:
      - "3001:3000"
    environment:
      GF_SECURITY_ADMIN_PASSWORD: admin
      GF_INSTALL_PLUGINS: grafana-piechart-panel
    volumes:
      - grafana-data:/var/lib/grafana
      - ./grafana/dashboards:/etc/grafana/provisioning/dashboards
      - ./grafana/datasources:/etc/grafana/provisioning/datasources
    networks:
      - mantis-network

volumes:
```

```
  kafka-data:
  timescaledb-data:
  redis-data:
  prometheus-data:
  grafana-data:

networks:
  mantis-network:
    driver: bridge
```

7.8 Configuration Kubernetes

7.8.1 Deployment Prediction Service

Listing 7.5 – Kubernetes Deployment - Prediction Service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rul-prediction-service
  namespace: mantis
  labels:
    app: rul-prediction
    version: v1.0.0
spec:
  replicas: 3
  selector:
    matchLabels:
      app: rul-prediction
  template:
    metadata:
      labels:
        app: rul-prediction
        version: v1.0.0
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8000"
        prometheus.io/path: "/metrics"
    spec:
      containers:
      - name: rul-prediction
        image: mantis/rul-prediction:1.0.0
        ports:
        - containerPort: 8000
          name: http
        env:
        - name: KAFKA_BOOTSTRAP_SERVERS
          value: "kafka-0.kafka-headless.mantis.svc.cluster.local:9092"
        - name: MLFLOW_TRACKING_URI
          value: "http://mlflow.mantis.svc.cluster.local:5000"
        - name: REDIS_HOST
          value: "redis.mantis.svc.cluster.local"
        - name: LOG_LEVEL
          value: "INFO"
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "2000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 8000
          initialDelaySeconds: 10
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: rul-prediction
  namespace: mantis
spec:
  selector:
    app: rul-prediction
  ports:
  - protocol: TCP
    port: 8000
    targetPort: 8000
  type: ClusterIP
---
apiVersion: autoscaling/v2
```

```
kind: HorizontalPodAutoscaler
metadata:
  name: rul-prediction-hpa
  namespace: mantis
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: rul-prediction-service
  minReplicas: 2
  maxReplicas: 8
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

7.9 API REST Complète

7.9.1 Endpoints Dashboard API

Listing 7.6 – API REST FastAPI Complète

```
from fastapi import FastAPI, HTTPException, Depends, Query
from pydantic import BaseModel, Field
from typing import List, Optional
from datetime import datetime, timedelta
import asyncpg

app = FastAPI(
    title="MANTIS␣Dashboard␣API",
    version="1.0.0",
    description="API␣REST␣pour␣le␣dashboard␣de␣maintenance␣pr dictive"
)

# Mod les Pydantic
class Equipment(BaseModel):
    id: str
    name: str
    type: str
    location: str
    status: str  # 'normal', 'warning', 'critical'
    health_score: float = Field(ge=0, le=100)
    last_maintenance: Optional[datetime]

class RULPrediction(BaseModel):
    equipment_id: str
    rul_cycles: int
    rul_hours: float
    confidence: float = Field(ge=0, le=1)
    predicted_at: datetime
    model_version: str

class Anomaly(BaseModel):
    id: int
    equipment_id: str
    detected_at: datetime
    anomaly_type: str
    severity: str  # 'LOW', 'MEDIUM', 'HIGH', 'CRITICAL'
    anomaly_score: float
    description: str

# Endpoints
@app.get("/api/v1/equipments", response_model=List[Equipment])
async def get_all_equipments(
    status: Optional[str] = Query(None, regex="^(normal|warning|critical)$"),
    location: Optional[str] = None
):
    """R cup re␣la␣liste␣de␣tous␣les␣  quipements  ␣avec␣filtres␣optionnels"""
    # Connexion base de donn es
    conn = await asyncpg.connect(DATABASE_URL)

    query = "SELECT␣*␣FROM␣equipments␣WHERE␣1=1"
    params = []

    if status:
        query += f"␣AND␣status␣=␣${len(params)+1}"
        params.append(status)

    if location:
        query += f"␣AND␣location␣=␣${len(params)+1}"
```

```
        params.append(location)

    rows = await conn.fetch(query, *params)
    await conn.close()

    return [Equipment(**dict(row)) for row in rows]

@app.get("/api/v1/equipments/{equipment_id}/rul", response_model=RULPrediction)
async def get_equipment_rul(equipment_id: str):
    """R cup re␣la␣derni re␣pr diction␣RUL␣pour␣un␣  quipement  """
    conn = await asyncpg.connect(DATABASE_URL)

    query = """
␣␣␣␣␣␣␣␣SELECT␣*␣FROM␣rul_predictions
␣␣␣␣␣␣␣␣WHERE␣equipment_id␣=␣$1
␣␣␣␣␣␣␣␣ORDER␣BY␣predicted_at␣DESC
␣␣␣␣␣␣␣␣LIMIT␣1
␣␣␣␣"""

    row = await conn.fetchrow(query, equipment_id)
    await conn.close()

    if not row:
        raise HTTPException(status_code=404, detail="Aucune␣pr diction␣trouv e")

    return RULPrediction(**dict(row))

@app.get("/api/v1/anomalies", response_model=List[Anomaly])
async def get_anomalies(
    equipment_id: Optional[str] = None,
    severity: Optional[str] = Query(None, regex="^(LOW|MEDIUM|HIGH|CRITICAL)$"),
    hours_ago: int = Query(24, ge=1, le=720)
):
    """R cup re␣les␣anomalies␣r centes␣avec␣filtres"""
    conn = await asyncpg.connect(DATABASE_URL)

    since = datetime.now() - timedelta(hours=hours_ago)

    query = "SELECT␣*␣FROM␣anomalies␣WHERE␣detected_at␣>=␣$1"
    params = [since]

    if equipment_id:
        query += f"␣AND␣equipment_id␣=␣${len(params)+1}"
        params.append(equipment_id)

    if severity:
        query += f"␣AND␣severity␣=␣${len(params)+1}"
        params.append(severity)

    query += "␣ORDER␣BY␣detected_at␣DESC"

    rows = await conn.fetch(query, *params)
    await conn.close()

    return [Anomaly(**dict(row)) for row in rows]

@app.get("/api/v1/health")
async def health_check():
    """Health␣check␣endpoint␣pour␣monitoring"""
    return {
        "status": "healthy",
        "timestamp": datetime.now().isoformat(),
        "version": "1.0.0"
    }
```

7.10 Conclusion

Ce chapitre a présenté l'implémentation détaillée des composants clés de MANTIS :

- **Service d'Ingestion** : Connecteurs OPC UA, MQTT avec gestion asynchrone
- **Service de Prédiction** : Modèle LSTM PyTorch avec API d'inférence FastAPI
- **Service de Détection d'Anomalies** : Isolation Forest et Autoencoder
- **Infrastructure Docker** : Dockerfiles optimisés multi-stage
- **Orchestration Kubernetes** : Deployments avec auto-scaling
- **API REST** : Endpoints complets pour le dashboard

L'utilisation de Python/FastAPI pour les microservices backend, combinée à PyTorch pour les modèles de Deep Learning, Docker pour la conteneurisation et Kubernetes pour l'orchestration, offre une stack technologique moderne, performante et maintenable conforme aux meilleures pratiques de l'industrie.

Chapitre 8

Tests et Validation

8.1 Stratégie de Tests

La stratégie de tests suit la pyramide classique : nombreux tests unitaires rapides, moins de tests d'intégration, et quelques tests E2E.

8.2 Tests Unitaires Python

```
import pytest
import pandas as pd
import numpy as np
from src.preprocessing import PreprocessingPipeline

@pytest.fixture
def sample_data():
    return pd.DataFrame({
        'sensor_1': np.random.randn(100) * 10 + 50,
        'sensor_2': np.concatenate([np.random.randn(95), [np.nan] * 5]),
    })

class TestPreprocessing:

    def test_outlier_detection(self, sample_data):
        pipeline = PreprocessingPipeline(method='zscore', threshold=3)
        result = pipeline.detect_outliers(sample_data['sensor_1'])
        assert sum(result) >= 0

    def test_normalization(self, sample_data):
        pipeline = PreprocessingPipeline(normalization='minmax')
        result = pipeline.normalize(sample_data['sensor_1'])
        assert result.min() >= 0 and result.max() <= 1
```

8.3 Tests Unitaires Java

```
@ExtendWith(MockitoExtension.class)
class IngestionServiceTest {

    @Mock private KafkaIngestionProducer kafkaProducer;
    @InjectMocks private IngestionService ingestionService;

    @Test
    void testIngestValidData() {
        SensorData data = SensorData.builder()
            .equipmentId("EQ001")
            .sensorId("TEMP_01")
            .timestamp(Instant.now())
            .value(75.5)
            .build();

        when(kafkaProducer.send(any())).thenReturn(completedFuture(null));

        ingestionService.ingest(data);

        verify(kafkaProducer, times(1)).send(data);
    }
}
```

8.4 Couverture de Code

Service	Couverture	Objectif
Ingestion IIoT	87%	80%
Preprocessing	92%	80%
RUL Prediction	88%	80%
Dashboard	78%	75%

TABLE 8.1 – Couverture de code par service

8.5 Tests d'Intégration

8.5.1 Test End-to-End du Pipeline

Listing 8.1 – Test E2E complet

```
import pytest
from kafka import KafkaProducer, KafkaConsumer
import json
import time

@pytest.mark.integration
class TestE2EPipeline:
    """Test␣du␣flux␣complet␣:␣Ingestion␣->␣Preprocessing␣->␣Prediction"""

    @pytest.fixture(scope="class")
    def kafka_producer(self):
        return KafkaProducer(
            bootstrap_servers=['localhost:9092'],
            value_serializer=lambda v: json.dumps(v).encode('utf-8')
        )

    @pytest.fixture(scope="class")
    def kafka_consumer(self):
        return KafkaConsumer(
            'predictions',
            bootstrap_servers=['localhost:9092'],
            value_deserializer=lambda m: json.loads(m.decode('utf-8')),
            auto_offset_reset='latest'
        )

    def test_complete_pipeline(self, kafka_producer, kafka_consumer):
        """Test␣du␣pipeline␣complet␣avec␣donn es␣r elles"""

        # 1. Publier des donn es capteurs
        sensor_data = {
            "equipment_id": "TEST_ENGINE_001",
            "timestamp": "2025-12-07T12:00:00Z",
            "sensors": {
                "sensor_1": 518.67,
                "sensor_2": 641.82,
                "sensor_3": 1589.70,
                "sensor_4": 1400.60,
                # ... 21 capteurs au total
            }
        }

        kafka_producer.send('raw-sensor-data', value=sensor_data)
        kafka_producer.flush()

        # 2. Attendre le traitement (max 5 secondes)
        start_time = time.time()
        prediction_received = False

        for message in kafka_consumer:
            if message.value['equipment_id'] == 'TEST_ENGINE_001':
                prediction = message.value
                prediction_received = True
                break

            if time.time() - start_time > 5:
                break

        # 3. V rifications
        assert prediction_received, "Aucune␣pr diction␣re ue␣dans␣les␣5␣secondes"
        assert 'rul_cycles' in prediction
        assert prediction['rul_cycles'] > 0
        assert 0 <= prediction['confidence'] <= 1

        # 4. V rifier la latence end-to-end
```

```
        latency = time.time() - start_time
        assert latency < 2.0, f"Latence␣trop␣  leve  :␣{latency}s␣>␣2.0s"
```

8.5.2 Tests de Charge

Listing 8.2 – Tests de charge avec Locust

```
from locust import HttpUser, task, between
import random

class MANTISDashboardUser(HttpUser):
    """Simulation␣de␣charge␣sur␣le␣dashboard␣MANTIS"""

    wait_time = between(1, 3)  # Temps d'attente entre requ tes

    @task(3)
    def get_all_equipments(self):
        """R cup rer␣tous␣les␣ quipements ␣(t che␣fr quente)"""
        self.client.get("/api/v1/equipments")

    @task(2)
    def get_equipment_rul(self):
        """R cup rer␣le␣RUL␣d'un␣ quipement ␣sp cifique"""
        equipment_id = f"EQ_{random.randint(1,␣100):03d}"
        self.client.get(f"/api/v1/equipments/{equipment_id}/rul")

    @task(1)
    def get_anomalies(self):
        """R cup rer␣les␣anomalies␣r centes"""
        self.client.get("/api/v1/anomalies?hours_ago=24")

    @task(1)
    def get_critical_anomalies(self):
        """Filtrer␣les␣anomalies␣critiques"""
        self.client.get("/api/v1/anomalies?severity=CRITICAL")
```

Résultats des tests de charge (100 utilisateurs concurrents) :

Endpoint	Requêtes/s	P50 (ms)	P95 (ms)	Taux erreur
GET /equipments	450	82	156	0.1%
GET /rul	320	95	187	0.2%
GET /anomalies	180	110	205	0.0%

Table 8.2 – Résultats des tests de charge

8.6 Validation des Modèles ML

8.6.1 Métriques de Performance du Modèle LSTM

Dataset	RMSE	MAE	R^2	MAPE (%)
C-MAPSS FD001	12.47	9.32	0.89	8.5
C-MAPSS FD002	18.92	14.21	0.82	12.3
C-MAPSS FD003	13.15	9.87	0.87	9.1
C-MAPSS FD004	21.34	16.45	0.78	14.8
Moyenne	**16.47**	**12.46**	**0.84**	**11.2**

Table 8.3 – Performances du modèle LSTM sur les 4 datasets C-MAPSS

8.6.2 Matrice de Confusion - Détection d'Anomalies

	Prédiction : Normal	**Prédiction : Anomalie**
Vérité : Normal	9.542 (TN)	287 (FP)
Vérité : Anomalie	98 (FN)	1.073 (TP)

TABLE 8.4 – Matrice de confusion sur le jeu de test

Métriques dérivées :

— **Précision** : $\frac{TP}{TP+FP} = \frac{1073}{1073+287} = 78.9\%$

— **Rappel (Recall)** : $\frac{TP}{TP+FN} = \frac{1073}{1073+98} = 91.6\%$

— **F1-Score** : $2 \times \frac{Precision \times Recall}{Precision+Recall} = 84.8\%$

— **Spécificité** : $\frac{TN}{TN+FP} = \frac{9542}{9542+287} = 97.1\%$

8.7 Tests de Sécurité

8.7.1 Scan de Vulnérabilités

Listing 8.3 – Scan de sécurité avec Trivy

```
# Scan des images Docker
trivy image mantis/rul-prediction:1.0.0

# R sultats:
# CRITICAL: 0
# HIGH: 0
# MEDIUM: 2 (d pendances non critiques)
# LOW: 5
```

8.7.2 Tests de Pénétration API

Test	**Description**	**Résultat**
SQL Injection	Tentatives d'injection dans les paramètres	PASS
XSS	Cross-Site Scripting via entrées utilisateur	PASS
CSRF	Cross-Site Request Forgery	PASS
Auth Bypass	Tentatives d'accès sans token JWT	PASS
Rate Limiting	Tests de dépassement de limites	PASS

TABLE 8.5 – Résultats des tests de sécurité

8.8 CI/CD et Automatisation

8.8.1 Pipeline GitHub Actions

Listing 8.4 – Workflow CI/CD

```yaml
name: MANTIS CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: |
          pip install -r requirements.txt
          pip install pytest pytest-cov

      - name: Run unit tests
        run: pytest tests/unit --cov=src --cov-report=xml

      - name: Upload coverage to Codecov
        uses: codecov/codecov-action@v3

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build Docker images
        run: docker build -t mantis/rul-prediction:$GITHUB_SHA .

      - name: Run security scan
        run: trivy image mantis/rul-prediction:$GITHUB_SHA

  deploy:
    needs: build
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to Kubernetes
        run: kubectl apply -f k8s/
```

8.9 Validation Utilisateur

8.9.1 Tests d'Acceptation

ID	Critère d'Acceptation	Statut
UA-001	Le système affiche le RUL de tous les équipements	VALIDÉ
UA-002	Les alertes critiques sont affichées en temps réel (<5s)	VALIDÉ
UA-003	Les graphiques sont actualisés automatiquement	VALIDÉ
UA-004	L'export PDF des rapports fonctionne	VALIDÉ
UA-005	La recherche d'équipements par ID/nom fonctionne	VALIDÉ

TABLE 8.6 – Critères d'acceptation utilisateur

8.10 Conclusion

Ce chapitre a présenté la stratégie complète de tests et de validation de MANTIS, incluant :

— **Tests unitaires** : Couverture >85% sur tous les services critiques
— **Tests d'intégration** : Pipeline E2E fonctionnel avec latence <2s
— **Tests de charge** : Capacité confirmée de 450+ req/s
— **Validation ML** : RMSE moyen de 16.47 cycles, F1-Score de 84.8%
— **Tests de sécurité** : Aucune vulnérabilité critique détectée
— **CI/CD** : Pipeline automatisé avec GitHub Actions
— **Tests d'acceptation** : Tous les critères utilisateurs validés

Les résultats démontrent que MANTIS atteint les objectifs de performance, sécurité et fiabilité définis en début de projet.

Chapitre 9

Conclusion et Perspectives

9.1 Synthèse

Le projet MANTIS a permis de développer une plateforme complète de maintenance prédictive, répondant aux exigences de l'Industrie 4.0.

Les principales réalisations sont :

— **Architecture Microservices** : 7 services indépendants, scalables et résilients, communiquant via Apache Kafka.

— **Performance Prédictive** : Le modèle LSTM atteint une RMSE de 12.47 cycles sur le dataset NASA C-MAPSS, surpassant l'objectif initial de 20 cycles.

— **Temps Réel** : La latence de bout en bout (ingestion → notification) est inférieure à 500 ms, garantissant une réactivité optimale.

— **Qualité Industrielle** : Une couverture de tests supérieure à 85% et une infrastructure CI/CD complète assurent la robustesse du système.

9.2 Perspectives

Le projet ouvre la voie à plusieurs évolutions futures :

— **Court terme** : Déploiement pilote sur un site industriel réel pour valider les performances en conditions opérationnelles.

— **Moyen terme** : Intégration de techniques de Federated Learning pour entraîner des modèles sur des données distribuées sans compromettre la confidentialité.

— **Long terme** : Développement d'un Jumeau Numérique (Digital Twin) complet pour simuler des scénarios de maintenance complexes.

9.3 Contributions du Projet

9.3.1 Contributions Techniques

Le projet MANTIS apporte plusieurs contributions significatives :

1. **Architecture Microservices Événementielle Complète** : Mise en œuvre d'une architecture distribuée moderne avec Apache Kafka comme backbone de communication, démontrant la viabilité de cette approche pour les systèmes IIoT temps réel.

2. **Pipeline MLOps Complet** : Intégration de MLflow, Feast et DVC pour un cycle de vie ML industrialisé, depuis l'entraînement jusqu'au déploiement et monitoring des modèles.
3. **Optimisations de Performance** : Atteinte d'une latence end-to-end <500ms grâce à l'utilisation de techniques avancées (batching Kafka, ONNX Runtime, caching multi-niveaux).
4. **Observabilité Complète** : Stack Prometheus + Grafana + Jaeger fournissant métriques, logs et tracing distribué pour un debugging et monitoring efficaces.
5. **Reproductibilité** : Documentation exhaustive et code entièrement Dockerisé permettant un déploiement reproductible sur n'importe quel environnement.

9.3.2 Contributions Académiques

— **Dataset** : Utilisation du NASA C-MAPSS comme benchmark académique reconnu

— **Méthodologie** : Application rigoureuse des principes d'ingénierie logicielle (tests, CI/CD, documentation)

— **Documentation** : Rapport technique détaillé couvrant tous les aspects du projet

— **Open Source** : Code source disponible pour la communauté académique

9.4 Compétences Développées

Ce projet a permis de développer et consolider des compétences dans de nombreux domaines :

Domaine	Compétences Acquises
Architecture	Conception microservices, patterns (CQRS, Event Sourcing, Circuit Breaker), architecture event-driven
Big Data	Apache Kafka (producers, consumers, topics, partitions), TimescaleDB (hypertables, continuous aggregates), streaming en temps réel
Machine Learning	LSTM, PyTorch, détection d'anomalies (Isolation Forest, Autoencoders), feature engineering pour séries temporelles
MLOps	MLflow (tracking, registry, deployment), Feast (feature store), versioning de modèles, A/B testing
DevOps	Docker (multi-stage builds), Kubernetes (deployments, services, HPA), CI/CD (GitHub Actions), monitoring (Prometheus, Grafana)
IIoT	Protocoles industriels (OPC UA, MQTT, Modbus), edge computing, intégration OT/IT
Développement	Python (FastAPI, asyncio), SQL, YAML, tests (pytest, unittest), sécurité (JWT, RBAC)
Gestion de Projet	Méthodologie Agile, Scrum, documentation technique, communication

Table 9.1 – Compétences développées durant le projet

9.5 Difficultés Rencontrées et Solutions

9.5.1 Difficultés Techniques

Difficulté	Impact	Solution Apportée
Latence élevée des prédictions (>2s)	Non-respect objectif temps réel	Export ONNX du modèle LSTM, caching Redis, batching Kafka
Consumer lag Kafka croissant	Perte de messages, retards	Augmentation du nombre de partitions (3→6), scaling horizontal des consumers
Complexité d'orchestration K8s	Difficulté déploiement	Utilisation de Helm charts, simplification avec docker-compose pour dev
Qualité de données brutes	Bruit, valeurs aberrantes	Pipeline preprocessing robuste (détection outliers IQR, interpolation)

TABLE 9.2 – Difficultés techniques et solutions

9.5.2 Difficultés Organisationnelles

— **Coordination entre domaines** : Synchronisation Big Data / ML / DevOps → Rituels Agile (stand-ups quotidiens)

— **Gestion du périmètre** : Scope creep potentiel → Priorisation stricte (MoSCoW)

— **Documentation continue** : Risque de retard documentation → Documentation as Code (Markdown in Git)

9.6 Impact et Bénéfices

9.6.1 Impact Économique Potentiel

L'adoption de la plateforme MANTIS dans un contexte industriel réel permettrait :

Indicateur	Avant MANTIS	Après MANTIS (estimé)
Arrêts non planifiés/an	15	4 (-73%)
Coût maintenance annuel	500 K€	350 K€ (-30%)
Disponibilité équipements (OEE)	72%	89% (+17 pts)
Durée de vie moyenne actifs	8 ans	10.5 ans (+31%)
ROI (période de retour)	-	14 mois

TABLE 9.3 – Impact économique estimé sur un site industriel moyen (50 équipements critiques)

9.6.2 Impact Environnemental

La maintenance prédictive contribue également à la durabilité :

— **Réduction du gaspillage** : Remplacement uniquement des composants nécessaires (vs. préventif systématique)
— **Optimisation énergétique** : Détection de surconsommations anormales
— **Allongement durée de vie** : -31% de déchets industriels générés

9.7 Leçons Apprises

1. **L'importance de l'observabilité** : Sans métriques, logs et tracing, le debugging d'une architecture distribuée est quasi impossible.
2. **Start simple, scale smart** : Démarrer avec docker-compose avant Kubernetes a permis une itération rapide.
3. **La qualité des données prime** : Un modèle SOTA sur des données de mauvaise qualité donne de mauvais résultats.
4. **L'automatisation est clé** : CI/CD, tests automatisés et IaC ont permis de gagner énormément de temps.
5. **La documentation vaut de l'or** : Une documentation à jour facilite l'onboarding et la maintenance future.
6. **L'Agile fonctionne** : Les itérations courtes et les feedback loops ont permis d'ajuster rapidement la direction.

9.8 Impact

L'adoption de la plateforme MANTIS permettrait une réduction estimée de **73% des arrêts non planifiés** et de **30% des coûts de maintenance**, avec un **ROI en 14 mois**, confirmant la valeur ajoutée de l'approche prédictive par rapport aux méthodes traditionnelles.

Bibliographie

Publications Académiques

Bibliographie

Maintenance Prédictive et RUL

[1] Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008). *Damage propagation modeling for aircraft engine run-to-failure simulation*. IEEE International Conference on Prognostics and Health Management (PHM), pp. 1-9. DOI : 10.1109/PHM.2008.4711414

[2] Lei, Y., Li, N., Guo, L., Li, N., Yan, T., & Lin, J. (2018). *Machinery health prognostics : A systematic review from data acquisition to RUL prediction*. Mechanical Systems and Signal Processing, 104, pp. 799-834. DOI : 10.1016/j.ymssp.2017.11.016

[3] Ran, Y., Zhou, X., Lin, P., Wen, Y., & Deng, R. (2019). *A survey of predictive maintenance : Systems, purposes and approaches*. arXiv preprint arXiv :1912.07383.

[4] Carvalho, T. P., Soares, F. A., Vita, R., Francisco, R. D. P., Basto, J. P., & Alcalá, S. G. (2019). *A systematic literature review of machine learning methods applied to predictive maintenance*. Computers & Industrial Engineering, 137, 106024. DOI : 10.1016/j.cie.2019.106024

Deep Learning pour Séries Temporelles

[5] Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural Computation, 9(8), pp. 1735-1780. DOI : 10.1162/neco.1997.9.8.1735

[6] Zheng, S., Ristovski, K., Farahat, A., & Gupta, C. (2017). *Long Short-Term Memory Network for Remaining Useful Life estimation*. IEEE International Conference on Prognostics and Health Management (ICPHM), pp. 88-95. DOI : 10.1109/ICPHM.2017.7998311

[7] Zhao, R., Yan, R., Wang, J., & Mao, K. (2019). *Learning to monitor machine health with convolutional bi-directional LSTM networks*. Sensors, 17(2), 273. DOI : 10.3390/s17020273

[8] Li, X., Ding, Q., & Sun, J. Q. (2018). *Remaining useful life estimation in prognostics using deep convolution neural networks*. Reliability Engineering & System Safety, 172, pp. 1-11. DOI : 10.1016/j.ress.2017.11.021

Détection d'Anomalies

[9] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). *Isolation forest*. IEEE International Conference on Data Mining (ICDM), pp. 413-422. DOI : 10.1109/ICDM.2008.17

[10] Chalapathy, R., & Chawla, S. (2019). *Deep learning for anomaly detection : A survey*. arXiv preprint arXiv :1901.03407.

[11] Chandola, V., Banerjee, A., & Kumar, V. (2009). *Anomaly detection : A survey.* ACM Computing Surveys (CSUR), 41(3), pp. 1-58. DOI : 10.1145/1541880.1541882

Architectures Microservices et Event-Driven

[12] Newman, S. (2015). *Building Microservices : Designing Fine-Grained Systems.* O'Reilly Media. ISBN : 978-1491950357

[13] Richardson, C. (2018). *Microservices Patterns : With examples in Java.* Manning Publications. ISBN : 978-1617294549

[14] Fowler, M., & Lewis, J. (2014). *Microservices : a definition of this new architectural term.* martinfowler.com/articles/microservices.html

[15] Stopford, B. (2018). *Designing Event-Driven Systems : Concepts and Patterns for Streaming Services with Apache Kafka.* O'Reilly Media. ISBN : 978-1492038221

Apache Kafka et Streaming

[16] Kreps, J., Narkhede, N., & Rao, J. (2011). *Kafka : A distributed messaging system for log processing.* Proceedings of the NetDB, pp. 1-7.

[17] Narkhede, N., Shapira, G., & Palino, T. (2017). *Kafka : The Definitive Guide : Real-Time Data and Stream Processing at Scale.* O'Reilly Media. ISBN : 978-1491936160

[18] Kleppmann, M. (2017). *Designing Data-Intensive Applications : The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly Media. ISBN : 978-1449373320

MLOps et DevOps

[19] Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... & Stoica, I. (2018). *Accelerating the machine learning lifecycle with MLflow.* IEEE Data Engineering Bulletin, 41(4), pp. 39-45.

[20] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). *Hidden technical debt in machine learning systems.* Advances in Neural Information Processing Systems (NeurIPS), pp. 2503-2511.

[21] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook : How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* IT Revolution Press. ISBN : 978-1942788003

[22] Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate : The Science of Lean Software and DevOps : Building and Scaling High Performing Technology Organizations.* IT Revolution Press. ISBN : 978-1942788331

Protocoles IIoT

[23] Mahnke, W., Leitner, S. H., & Damm, M. (2009). *OPC Unified Architecture.* Springer Science & Business Media. ISBN : 978-3540688983

[24] Banks, A., & Gupta, R. (2014). *MQTT Version 3.1.1.* OASIS standard. Available at : http ://docs.oasis-open.org/mqtt/mqtt/v3.1.1/

[25] Modbus Organization. (2012). *Modbus Application Protocol Specification V1.1b3*. Available at : www.modbus.org

Bases de Données Time-Series

[26] Freedman, M., Beimel, Y., & Askary, N. (2017). *TimescaleDB : Fast and Scalable Timeseries*. Technical report, Timescale, Inc.

[27] Dunning, T., & Friedman, E. (2014). *Time Series Databases : New Ways to Store and Access Data*. O'Reilly Media. ISBN : 978-1491914724

Observabilité et Monitoring

[28] Volz, J., & Rabenstein, B. (2016). *Prometheus : Up & Running : Infrastructure and Application Performance Monitoring*. O'Reilly Media. ISBN : 978-1492034148

[29] Shkuro, Y. (2019). *Mastering Distributed Tracing : Analyzing performance in microservices and complex systems*. Packt Publishing. ISBN : 978-1788628464

[30] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site Reliability Engineering : How Google Runs Production Systems*. O'Reilly Media. ISBN : 978-1491929124

Sécurité et Standards

[31] International Electrotechnical Commission. (2018). *IEC 62443 : Security for industrial automation and control systems*. IEC Standard Series.

[32] National Institute of Standards and Technology. (2018). *Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1*. NIST Cybersecurity Framework.

Industrie 4.0

[33] Kagermann, H., Wahlster, W., & Helbig, J. (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0*. Final report of the Industrie 4.0 Working Group, Forschungsunion.

[34] Liao, Y., Deschamps, F., Loures, E. D. F. R., & Ramos, L. F. P. (2017). *Past, present and future of Industry 4.0-a systematic literature review and research agenda proposal*. International Journal of Production Research, 55(12), pp. 3609-3629. DOI : 10.1080/00207543.2017.1308576

Ressources en Ligne et Documentation Technique

— **Apache Kafka Documentation**. https://kafka.apache.org/documentation/
— **PyTorch Documentation**. https://pytorch.org/docs/
— **FastAPI Documentation**. https://fastapi.tiangolo.com/
— **Kubernetes Documentation**. https://kubernetes.io/docs/
— **MLflow Documentation**. https://mlflow.org/docs/
— **TimescaleDB Documentation**. https://docs.timescale.com/

— **Prometheus Documentation**. https://prometheus.io/docs/
— **Grafana Documentation**. https://grafana.com/docs/
— **OPC Foundation**. https://opcfoundation.org/
— **NASA Prognostics Data Repository**. https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/

Annexe A

Docker Compose

```
version: '3.8'
services:
  kafka:
    image: confluentinc/cp-kafka:7.5.0
    ports: ["9092:9092"]

  timescaledb:
    image: timescale/timescaledb:latest-pg15
    ports: ["5432:5432"]
    environment:
      POSTGRES_USER: mantis
      POSTGRES_PASSWORD: mantis123

  mlflow:
    image: ghcr.io/mlflow/mlflow:v2.8.0
    ports: ["5000:5000"]

  prometheus:
    image: prom/prometheus:v2.47.0
    ports: ["9090:9090"]

  grafana:
    image: grafana/grafana:10.2.0
    ports: ["3001:3000"]
```