Mini- Projet

Application de Monitoring et d'Analyse de Logs avec Stack ELK, NoSQL et Interface Web

Mohamed Najeh ISSAOUI
E-mail : issaoui.mn@itbs.tn
IT Business School

Date de rendu : [À définir, un mail vous sera envoyé]
Date de soutenance : [À définir, un mail vous sera envoyé]

I. CONTEXTE GÉNÉRAL

I.1 Introduction

Dans le contexte actuel des systèmes d'information distribués, la production de logs est devenue exponentielle. Ces logs proviennent de multiples sources : applications web, microservices, bases de données, systèmes IoT, serveurs, API, etc. Cette masse de données constitue une mine d'informations cruciales pour plusieurs objectifs stratégiques :

- **Détection et diagnostic des incidents** : Identifier rapidement les anomalies et pannes système
- **Analyse des performances** : Optimiser les temps de réponse et l'expérience utilisateur
- **Sécurité et conformité** : Détecter les intrusions et respecter les réglementations (RGPD, PCI-DSS)
- **Business Intelligence** : Prendre des décisions basées sur l'analyse des comportements utilisateurs

Cependant, la gestion manuelle de millions de lignes de logs devient rapidement impossible. Il est donc nécessaire de mettre en place une infrastructure automatisée capable de collecter, indexer, rechercher et visualiser ces données en temps réel.

I.2 Problématique

Votre entreprise fait face aux défis suivants :

- Les logs sont dispersés sur différents serveurs et applications
- L'absence de centralisation rend le diagnostic des problèmes long et complexe
- Les équipes techniques perdent du temps à chercher manuellement dans les fichiers logs
- Il n'existe pas de système d'alerting automatique pour les événements critiques
- La visualisation des tendances et patterns est inexistante

I.3 Mission

En tant qu'ingénieurs informatiques, votre mission est de concevoir, développer et déployer une **plateforme complète de monitoring et d'analyse de logs** répondant aux besoins de l'entreprise. Cette plateforme devra :

1. Centraliser la collecte des logs de différentes sources
2. Indexer intelligemment ces données pour une recherche ultra-rapide
3. Proposer des visualisations pertinentes pour faciliter l'analyse
4. Offrir une interface web intuitive pour les équipes techniques et métiers
5. Être facilement déployable et scalable grâce à la conteneurisation

II. SCÉNARIOS MÉTIER

Pour contextualiser votre projet et le rendre réaliste, vous devez **choisir UN des quatre scénarios suivants**. Ce choix orientera la nature des logs traités, les visualisations créées, et les fonctionnalités prioritaires de votre application.

Scénario A : Plateforme E-Commerce

Contexte : Vous travaillez pour une plateforme de vente en ligne traitant des milliers de commandes quotidiennes.

Types de logs à traiter :

- Logs de transactions (paiements, commandes, remboursements)
- Logs d'erreurs applicatives (codes 404, 500, timeouts)
- Logs de comportement utilisateur (navigation, ajouts au panier, abandons)
- Logs de performance (temps de réponse API, latence base de données)
- Logs de fraude (tentatives de paiement suspects, bots)

Indicateurs clés (KPI) à suivre :

- Taux de conversion
- Taux d'abandon de panier
- Temps moyen de traitement des commandes
- Nombre d'erreurs par service
- Détection des pics de trafic

Cas d'usage prioritaires :

- Alerter l'équipe technique en cas de pic d'erreurs de paiement
- Analyser les parcours utilisateurs pour optimiser le tunnel de vente
- Identifier les produits générant le plus d'erreurs

Scénario B : Infrastructure IoT / Smart Building

Contexte : Vous gérez les systèmes de monitoring d'un bâtiment intelligent équipé de centaines de capteurs.

Types de logs à traiter :

- Logs de capteurs (température, humidité, luminosité, CO2)
- Logs d'alertes (anomalies détectées, seuils dépassés)
- Logs de maintenance préventive (usure équipements, pannes prédites)
- Logs de consommation énergétique (électricité, eau, chauffage)
- Logs d'occupation (présence dans les salles, taux d'utilisation)

Indicateurs clés (KPI) à suivre :

- Température moyenne par zone et par heure
- Nombre d'alertes critiques par jour
- Consommation énergétique en temps réel
- Taux d'occupation des espaces
- Prévisions de maintenance

Cas d'usage prioritaires :

- Déclencher une alerte si la température dépasse un seuil critique
- Optimiser la consommation énergétique en analysant les patterns
- Prédire les pannes équipements grâce à l'analyse historique

Scénario C : Sécurité et Authentification

Contexte : Vous êtes responsable de la sécurité informatique d'une entreprise et devez monitorer toutes les tentatives d'accès aux systèmes.

Types de logs à traiter :

- Logs d'authentification (connexions réussies/échouées)
- Logs de tentatives d'intrusion (brute force, injections SQL)
- Logs d'accès aux ressources sensibles (fichiers confidentiels, bases de données)
- Logs de pare-feu et IDS (Intrusion Detection System)
- Logs d'audit de conformité (actions administrateurs, modifications configurations)

Indicateurs clés (KPI) à suivre :

- Nombre de tentatives de connexion échouées par IP
- Géolocalisation des connexions suspectes
- Taux de réussite des authentifications
- Nombre d'alertes de sécurité par gravité
- Temps de réponse aux incidents

Cas d'usage prioritaires :

- Bloquer automatiquement une IP après 5 tentatives échouées
- Alerter le SOC en cas de connexion depuis un pays non autorisé
- Générer des rapports de conformité mensuels pour l'audit

Scénario D : Application Web SaaS

Contexte : Vous développez une application SaaS (Software as a Service) utilisée par des milliers d'entreprises clientes.

Types de logs à traiter :

- Logs serveur web (Apache/Nginx - accès, erreurs)
- Logs applicatifs (Django/Flask - exceptions, warnings, debug)
- Logs de base de données (requêtes lentes, deadlocks, erreurs)
- Logs de performances (temps de réponse, consommation mémoire/CPU)
- Logs d'utilisation métier (features utilisées, actions utilisateurs)

Indicateurs clés (KPI) à suivre :

- Temps de réponse moyen par endpoint
- Taux d'erreur 5xx par service
- Requêtes SQL les plus lentes
- Nombre d'utilisateurs actifs simultanés
- Features les plus/moins utilisées

Cas d'usage prioritaires :

- Alerter DevOps si le temps de réponse dépasse 3 secondes
- Identifier les requêtes SQL à optimiser
- Analyser l'adoption des nouvelles fonctionnalités

III. OBJECTIFS PÉDAGOGIQUES

À l'issue de ce mini-projet, vous serez capables de :

III.1. Compétences Techniques

❖ Big Data et NoSQL

- Concevoir et déployer une architecture Elasticsearch distribuée
- Créer des pipelines Logstash pour l'ingestion et la transformation de données
- Manipuler différentes bases de données NoSQL (Elasticsearch, MongoDB, Redis)
- Optimiser les performances d'indexation et de recherche

❖ Conteneurisation et DevOps

- Orchestrer une application multi-services avec Docker Compose
- Écrire des Dockerfiles optimisés et sécurisés
- Gérer les volumes, networks et variables d'environnement
- Comprendre les concepts de portabilité et scalabilité

❖ Développement Web Full-Stack

- Développer une API RESTful avec Flask ou Django
- Créer des interfaces utilisateur responsives et modernes

- Intégrer des bibliothèques de visualisation de données
- Implémenter l'authentification et la gestion des sessions

❖ Visualisation de Données

- Créer des dashboards interactifs avec Kibana
- Développer des graphiques personnalisés (Chart.js, Plotly)
- Concevoir des visualisations adaptées au métier

❖ Architecture Logicielle

- Modéliser une architecture technique complexe
- Appliquer les design patterns adaptés
- Gérer la communication entre microservices
- Implémenter un système d'alerting temps réel

III.2. Compétences Transversales

❖ Analyse et Conception

- Analyser un besoin métier et le traduire en spécifications techniques
- Modéliser des architectures
- Faire des choix techniques argumentés

❖ Documentation Professionnelle

- Rédiger une documentation technique claire et complète
- Créer des README exploitables par d'autres développeurs
- Documenter une API avec Swagger/OpenAPI

❖ Qualité Logicielle

- Écrire des tests unitaires et d'intégration
- Appliquer les bonnes pratiques de développement (PEP8, SOLID)
- Gérer les erreurs et la journalisation applicative

❖ Déploiement et Livraison

- Déployer une application en environnement conteneurisé
- Configurer des environnements de développement, test et production
- Automatiser les processus de build et déploiement

❖ Travail Collaboratif

- Travailler en équipe avec GitHub (partagez votre projet avec votre encadrant)
- Organiser le travail avec des méthodologies Agile
- Communiquer efficacement sur des sujets techniques

IV. ARCHITECTURE TECHNIQUE

IV.1 Stack Technologique Imposée

Votre projet devra obligatoirement intégrer les technologies suivantes :

Composants Core (Obligatoires)

Stack ELK :

- **Elasticsearch 8.x** : Moteur de recherche et d'analyse distribué pour l'indexation des logs
- **Logstash 8.x** : Pipeline d'ingestion pour collecter, transformer et envoyer les données
- **Kibana 8.x** : Interface de visualisation et d'exploration des données Elasticsearch

Conteneurisation :

- **Docker 24.x** : Plateforme de conteneurisation
- **Docker Compose 2.x** : Orchestration multi-conteneurs (obligatoire pour tous les services)

Framework de Développement Web (Au choix)

Option 1 : Flask 3.x

- Framework web Python léger et flexible
- Flask-RESTful pour l'API
- Flask-Login pour l'authentification (si fonctionnalité implémentée)
- Jinja2 pour les templates

Option 2 : Django 5.x

- Framework web Python complet (batteries included)
- Django REST Framework pour l'API
- Django Authentication System intégré
- Django Template Engine

Serveur d'application WSGI :

- Gunicorn ou uWSGI (pour la production)

Bases de Données NoSQL (Obligatoires)

MongoDB 7.x :

- Stockage des métadonnées des fichiers uploadés
- Stockage des configurations utilisateurs
- Stockage des dashboards personnalisés
- Historique des recherches effectuées

Redis 7.x :

- Cache des résultats de requêtes Elasticsearch
- Gestion des sessions utilisateurs
- File d'attente pour le traitement asynchrone
- Rate limiting des requêtes API

Frontend (Recommandations)

Technologies de base :

- HTML5 + CSS3 + JavaScript (Vanilla ou léger)

Frameworks CSS :

- Bootstrap 5 ou Tailwind CSS (au choix)
- Design responsive obligatoire

Bibliothèques de visualisation :

- Chart.js ou Plotly.js pour les graphiques personnalisés
- DataTables.js pour les tableaux interactifs
- FullCalendar.js pour les sélecteurs de dates (optionnel)

Outils Complémentaires (Optionnels mais recommandés)

- **Nginx** : Reverse proxy et load balancer
- **Filebeat 8.x** : Agent léger de collecte de logs (alternative à l'injection directe)
- **Certbot** : Certificats SSL/TLS gratuits (si déploiement HTTPS)

V. FONCTIONNALITÉS À IMPLÉMENTER

Votre projet sera évalué selon **trois niveaux de fonctionnalités** :

- **Niveau OBLIGATOIRE** : Note maximale de 12/20
- **Niveau INTERMÉDIAIRE** : Permet d'atteindre 16/20
- **Niveau AVANCÉ** : Permet d'atteindre 20/20

V.1 Niveau OBLIGATOIRE (12/20)

Ces fonctionnalités sont **indispensables** et constituent le socle minimum de votre application.

Module 1 : Gestion des Fichiers de Logs

Fonctionnalités :

❖ Upload de fichiers

- Interface web permettant l'upload de fichiers (CSV, JSON, TXT)
- Support du drag & drop
- Limite de taille configurable (ex : max 100 MB par fichier)

- Barre de progression pendant l'upload

❖ Validation des fichiers

- Vérification du format de fichier (extension, structure)
- Validation du contenu (nombre de colonnes pour CSV, syntaxe JSON)
- Messages d'erreur clairs en cas de fichier invalide
- Prévisualisation des 10 premières lignes avant injection

❖ Injection dans Logstash

- Sauvegarde du fichier dans un répertoire surveillé par Logstash
- Déclenchement automatique du pipeline d'ingestion
- Feedback en temps réel sur le statut de l'ingestion

❖ Liste des fichiers uploadés

- Tableau affichant tous les fichiers injectés
- Colonnes : Nom, Date d'upload, Taille, Statut, Nombre de logs
- Tri et pagination
- Filtres par statut et date

❖ Stockage des métadonnées

- Sauvegarde des informations de chaque fichier dans MongoDB
- Historique complet des uploads

Module 2 : Configuration de la Stack ELK

❖ Elasticsearch

- Création d'au moins un index template (ex : logs-*)
- Mapping adapté au scénario choisi
- Configuration des types de données corrects (date, keyword, text, ip, etc.)
- Index lifecycle management (optionnel mais recommandé)

❖ Logstash

- Au moins 2 pipelines de configuration :
    - Pipeline pour fichiers CSV
    - Pipeline pour fichiers JSON
- Utilisation de filtres appropriés (csv, json, date, mutate, grok)

- Gestion des erreurs de parsing
- Enrichissement des données (ajout de tags, géolocalisation si pertinent)
- Output vers Elasticsearch avec gestion des erreurs

❖ Kibana

- Configuration de la connexion à Elasticsearch
- Création d'au moins **3 visualisations pertinentes** pour le scénario choisi :

    - Exemple e-commerce : Courbe des transactions/heure, Top 10 erreurs, Répartition par type de paiement
    - Exemple IoT : Évolution température, Alertes par zone, Consommation énergétique
    - Exemple Sécurité : Tentatives de connexion par pays, Timeline des intrusions, Top IPs suspectes
    - Exemple SaaS : Temps de réponse par endpoint, Erreurs 5xx/jour, Utilisateurs actifs
- Création d'au moins **1 dashboard complet** intégrant les visualisations
- Export du dashboard pour sauvegarde et versioning

Module 3 : Interface Web de Base

❖ Page d'accueil (Dashboard principal)

- Statistiques globales en temps réel :
    - Nombre total de logs indexés
    - Nombre de logs aujourd'hui
    - Nombre de logs avec erreurs (niveau ERROR/CRITICAL)
    - Nombre de fichiers uploadés
- Graphiques de synthèse (utilisant Chart.js ou Plotly)
- Liens rapides vers les fonctionnalités principales
- Design professionnel et responsive

❖ Page d'upload de logs

- Formulaire d'upload intuitif avec drag & drop
- Sélection du type de fichier
- Validation côté client (taille, format)
- Affichage de la progression
- Messages de succès/erreur
- Liste des uploads récents

❖ Page de recherche

- Barre de recherche en texte libre
- Filtres essentiels :
    - Niveau de log (DEBUG, INFO, WARNING, ERROR, CRITICAL)
    - Service/Application
    - Plage de dates (date picker avec calendrier)
- Bouton de recherche et reset des filtres
- Sauvegarde des dernières recherches (dans MongoDB)

❖ Page de résultats

- Affichage des logs correspondants sous forme de tableau
- Colonnes affichées : Date/Heure, Niveau, Service, Message, Actions
- Pagination (50 logs par page)
- Tri par colonne (date, niveau, service)
- Indication du nombre total de résultats
- Bouton pour voir les détails d'un log
- Export des résultats en CSV (bouton)

❖ Design responsive

- Interface fonctionnelle sur desktop (1920x1080 minimum)
- Adaptation mobile et tablette (responsive design)
- Menu de navigation clair
- Footer avec informations du projet

Module 4 : Intégration MongoDB

❖ Stockage des métadonnées de fichiers

- Informations complètes sur chaque fichier uploadé
- Schema MongoDB défini et documenté

❖ Historique des recherches

- Sauvegarde automatique des requêtes effectuées
- Association avec l'utilisateur (si authentification)
- Page d'historique des recherches accessibles

❖ Configuration de la connexion

- Variables d'environnement pour les credentials
- Gestion des erreurs de connexion
- Tests de connectivité au démarrage

Module 5 : Déploiement Docker

Fichier docker-compose.yml fonctionnel

- Tous les services définis (elasticsearch, logstash, kibana, mongodb, redis, webapp)
- Configuration des ports exposés
- Définition des volumes pour persistance
- Réseau Docker bridge configuré
- Dépendances entre services (depends_on)
- Health checks pour les services critiques

❖ Dockerfile pour l'application web

- Image de base Python appropriée (python:3.11-slim ou alpine)
- Installation des dépendances (requirements.txt)
- Copie du code source
- Exposition du port 8000
- CMD ou ENTRYPOINT correctement défini
- Multi-stage build (bonus)

❖ Variables d'environnement

- Fichier .env pour la configuration locale
- Fichier .env.example versionné sur Git (sans secrets)
- Documentation des variables nécessaires

❖ **Documentation de déploiement**

- Instructions claires dans le README.md
- Commandes pour démarrer/arrêter les services
- Temps de démarrage estimé (avec téléchargement des images)
- Vérification que les services sont opérationnels

5.3 Niveau AVANCÉ (+4 points → 20/20)

Pour atteindre la note maximale de 20/20, vous devez implémenter **au moins 2 fonctionnalités** parmi les modules avancés suivants :

Module G : Système d'Alerting Intelligent

Définition de règles d'alertes

- Interface de création de règles :
    - Nom et description de l'alerte
    - Condition de déclenchement :
        - Seuil numérique (ex : > 100 erreurs/heure)
        - Pattern de texte (regex)
        - Absence de logs (silence alarm)
        - Anomalie détectée
    - Fenêtre temporelle (5min, 15min, 1h, 24h)
    - Gravité (Low, Medium, High, Critical)
- Éditeur de requête Elasticsearch pour la règle
- Test de la règle avant activation

Moteur d'alerting

- Worker séparé qui vérifie les règles périodiquement
- Exécution des requêtes contre Elasticsearch
- Évaluation des conditions
- Déclenchement des alertes si seuils dépassés
- Gestion du silencing (ne pas alerter plusieurs fois pour le même incident)
- Cooldown period configurable

Notifications multi-canaux

- **Email** :
    - Configuration SMTP
    - Template d'email personnalisable
    - Graphique du problème en pièce jointe
    - Lien vers la recherche concernée
- **Webhook** :
    - Support de Slack, Discord, Microsoft Teams
    - Payload personnalisable (JSON)
    - Retry automatique en cas d'échec

Dashboard des alertes

- Vue en temps réel des alertes actives
- Historique de toutes les alertes déclenchées

- Statistiques :
    - Nombre d'alertes par jour
    - Top 10 alertes les plus fréquentes
    - MTTR (Mean Time To Resolution)
- Actions possibles :
    - Acknowledge (marquer comme vu)
    - Resolve (résoudre)
    - Snooze (mettre en pause temporairement)
- Commentaires sur les incidents

Escalation automatique

- Définition de niveaux d'escalation :
    - Level 1 : Alerte normale
    - Level 2 : Si pas résolu après 15 min
    - Level 3 : Si pas résolu après 1h
- Notifications à différentes personnes selon le niveau
- Augmentation de la gravité automatique

Module H : Analyse et Visualisation Temps Réel

Dashboard temps réel avec WebSocket

- Connexion WebSocket entre navigateur et serveur
- Push automatique des nouveaux logs au client
- Mise à jour des graphiques en temps réel (pas de polling)
- Indicateur de connexion (connecté/déconnecté)
- Reconnexion automatique en cas de perte

Stream processing

- Pipeline Logstash en mode continu (pas uniquement fichiers)
- Ingestion depuis sources en streaming :
    - TCP/UDP sockets
    - HTTP webhook
    - Filebeat en mode tail
    - Redis Pub/Sub
- Traitement et indexation en temps réel (<1 seconde de latence)

Métriques live

- Compteurs temps réel :
    - Logs/seconde
    - Erreurs/minute
    - Requêtes actives
    - Users connectés
- Graphiques animés (animation smooth)
- Utilisation de bibliothèques temps réel (Socket.io, Chart.js streaming)

Live tail des logs

- Page "Live Logs" style tail -f
- Affichage en continu des nouveaux logs
- Scroll automatique

- Filtrage en temps réel (niveau, service)
- Pause/Resume du flux
- Limitation du nombre de logs affichés (buffer circulaire)
- Coloration syntaxique par niveau (ERROR en rouge, etc.)

Alertes visuelles

- Notifications desktop (HTML5 Notification API)
- Badge sur l'icône de l'onglet
- Son d'alerte (optionnel et désactivable)
- Flash de la barre de navigation

Module I : Machine Learning et Détection d'Anomalies

Intégration Elasticsearch ML

- Configuration des jobs Machine Learning dans Elasticsearch
- Création d'anomaly detection jobs :
    - Détection d'anomalies sur le volume de logs
    - Détection de patterns inhabituels
    - Détection de valeurs aberrantes (outliers)
- Configuration des buckets (5min, 15min, 1h)
- Définition des influences et split fields

Dashboard ML

- Visualisation des anomalies détectées
- Score d'anomalie (anomaly score)
- Timeline des anomalies
- Drill-down sur une anomalie :
    - Logs concernés
    - Contexte temporel
    - Causes probables suggérées

Alerting basé sur ML

- Création d'alertes déclenchées par les anomalies ML
- Seuil de score d'anomalie configurable
- Notification automatique à l'équipe

Détection de patterns avec NLP

- Utilisation de techniques NLP sur les messages de logs
- Clustering automatique des logs similaires :
    - Groupement par template de message
    - Identification des messages récurrents
    - Détection de nouveaux types de logs (novelty detection)
- Extraction d'entités (NER) :
    - IPs, emails, usernames, URLs
    - Indexation dans des champs séparés

Prédiction de tendances

- Modèle de prédiction des erreurs futures
- Graphique montrant :
  - Historique réel
  - Prédiction (avec intervalle de confiance)
- Alertes préventives si tendance préoccupante
- Utilisation de modèles simples (régression, ARIMA) ou complexes (LSTM si vous êtes ambitieux)

Auto-tagging intelligent

- Classification automatique des logs par catégorie
- Entraînement d'un modèle sur logs historiques
- Application automatique de tags pertinents
- Amélioration continue du modèle

Module J : Multi-tenancy et Isolation

Support de plusieurs organisations

- Notion d'organisation/tenant dans le système
- Page d'inscription pour créer une organisation
- Utilisateurs liés à une organisation
- Interface admin pour gérer les organisations

Isolation des données

- Chaque tenant a ses propres index Elasticsearch :
  - Exemple : logs-tenant1-*, logs-tenant2-*
- Collections MongoDB séparées ou field discriminator
- Clés Redis préfixées par tenant_id
- Impossible de voir les données d'un autre tenant

Quotas et limitations

- Définition de quotas par tenant :
  - Stockage maximum (GB)
  - Nombre de requêtes/jour
  - Nombre d'utilisateurs
  - Rétention des données (jours)
- Affichage de la consommation actuelle
- Alertes quand proche de la limite
- Blocage si quota dépassé

Facturation simulée

- Page de billing montrant :
  - Consommation du mois en cours
  - Montant simulé (ex : 0.10$/GB)
  - Historique des factures
- Export des factures en PDF
- Graphiques de consommation

Personnalisation par tenant

- Logo personnalisé
- Couleurs du thème (branding)
- Nom de domaine custom (optionnel)
- Configurations spécifiques (rétention, alertes, etc.)

Module K : CI/CD et Tests Automatisés

Tests unitaires

- Framework de tests : pytest (Python)
- Coverage de code > 70%
- Tests pour :
    - Fonctions de validation
    - Logique métier
    - API endpoints (mock des dépendances)
    - Utilitaires
- Rapport de coverage généré (HTML)

Tests d'intégration

- Tests de bout en bout (E2E)
- Scénarios réalistes :
    - Upload fichier → Vérif dans Elasticsearch
    - Recherche → Vérification résultats
    - Création utilisateur → Login
- Utilisation de fixtures pour les données de test
- Nettoyage après chaque test

Tests de performance

- Tests de charge avec Locust ou JMeter :
    - Simul ation de 100 utilisateurs concurrents
    - Mesure des temps de réponse
    - Identification des bottlenecks
- Tests de stress (jusqu'à la rupture)
- Rapport de performance

Pipeline CI/CD

- Configuration GitHub Actions ou GitLab CI
- Étapes du pipeline :
    1. Linting (flake8, pylint)
    2. Tests unitaires
    3. Tests d'intégration
    4. Build de l'image Docker
    5. Push vers registry
    6. Déploiement automatique (staging)
- Badges dans le README (build status, coverage)

Quality gates

- Intégration avec SonarQube (ou SonarCloud)
- Métriques de qualité :
    - Code smells
    - Bugs
    - Vulnerabilities
    - Code duplication
- Blocage du merge si quality gate échoué

Déploiement automatique

- Environnements multiples :
    - Development (local)
    - Staging (auto-deploy depuis main branch)
    - Production (deploy manuel ou tag-based)
- Scripts de déploiement
- Rollback automatique en cas d'échec
- Blue-Green deployment (optionnel)

Module L : Observabilité et Monitoring Avancé

Intégration Prometheus + Grafana

- Exposition de métriques applicatives au format Prometheus :
    - Nombre de requêtes HTTP (par endpoint, par status code)
    - Latence des requêtes (percentiles P50, P90, P99)
    - Taille des réponses
    - Nombre de connexions actives
    - Utilisation mémoire/CPU
- Scraping par Prometheus
- Dashboards Grafana prédéfinis :
    - Vue d'ensemble applicative
    - Performance Elasticsearch
    - Santé des services Docker

Métriques business

- Métriques spécifiques au scénario :
    - E-commerce : Transactions/min, panier moyen, taux conversion
    - IoT : Nombre de capteurs actifs, alertes/heure
    - Sécurité : Tentatives d'intrusion/jour, IPs bloquées
    - SaaS : MAU (Monthly Active Users), churn rate
- Tableaux de bord dédiés

Health checks complets

- Endpoint /health retournant :
    - Status global (healthy/degraded/down)
    - Status de chaque dépendance :
        - Elasticsearch (ping + cluster health)
        - MongoDB (connexion + latence)
        - Redis (ping + memory usage)
        - Logstash (API status)

    - Temps de réponse de chaque check
- Format JSON standardisé
- Utilisé par Docker healthcheck et monitoring externe

Logs structurés

- Logs applicatifs en JSON :

json

```
{
  "timestamp": "2024-01-15T10:30:00.000Z",
  "level": "ERROR",
  "logger": "webapp.api",
  "message": "Failed to query Elasticsearch",
  "context": {
    "user_id": "user_123",
    "query": "level:ERROR",
    "execution_time": 0.523,
    "error": "ConnectionTimeout"
  },
  "trace_id": "abc123def456"
}
```

- Facilite l'analyse automatique
- Corrélation avec les traces distribuées

Tracing distribué

- Intégration avec Jaeger ou Zipkin
- Instrumentation des requêtes à travers les composants
- Visualisation des spans :
    - Request → Flask → Elasticsearch → Response
    - Identification des lenteurs
- Trace ID propagé dans tous les logs

Alerting infrastructure

- Alertes Prometheus/Grafana sur :
    - CPU/RAM > 80%
    - Disk space < 10%
    - Service down
    - Latence > seuil
    - Error rate > seuil
- Notifications Slack/email depuis Grafana

VI. PHASES DE REALISATION & LIVRABLES

IV.1 Organisation du Projet

Analyse et Conception

Objectifs :

- Comprendre le contexte et choisir le scénario métier
- Définir les user stories et cas d'utilisation
- Concevoir l'architecture technique
- Créer les maquettes d'interface

Livrables :

- **Document d'analyse fonctionnelle** (5-8 pages) :
  - Contexte et scénario choisi (avec justification)
  - Acteurs du système
  - Diagramme de cas d'utilisation (UML Use Case)
  - User stories priorisées (format : En tant que [rôle], je veux [action], afin de [bénéfice])
  - Règles métier spécifiques
- **Architecture technique** (8-12 pages) :
  - Schéma d'architecture globale (style C4 Model - contexte et conteneurs)
  - Diagramme de déploiement Docker
  - Diagrammes de séquence (2-3 flux principaux)
  - Modèle de données Elasticsearch (mapping JSON)
  - Schémas MongoDB (collections et exemples de documents)
  - Architecture Redis (types de données et use cases)
- **Maquettes UI/UX** :
  - Wireframes des 5 pages principales minimum
  - Design system (couleurs, typographie, composants)
  - Flow utilisateur (user journey map)

Conseils :

- Privilégiez la clarté à la complexité
- Validez votre architecture avec l'enseignant dès que possible
- Utilisez des outils professionnels : Draw.io, Lucidchart, Figma, Miro
- Ne sous-estimez pas cette phase : une bonne conception évite des refactorisations coûteuses

Setup et Infrastructure

Objectifs :

- Mettre en place l'environnement de développement
- Configurer Docker Compose avec tous les services
- Initialiser la structure du projet
- Établir les premières connexions entre composants

Tâches prioritaires :

Git et Structure

- Créer le repository Git (GitHub/GitLab)
- Initialiser la structure de dossiers
- Créer le README.md de base
- Configurer .gitignore
- Premier commit et push

Docker et ELK

- Écrire le docker-compose.yml initial
- Démarrer Elasticsearch et vérifier l'accès (curl localhost:9200)
- Démarrer Kibana et créer le premier index pattern
- Démarrer Logstash avec un pipeline de test
- Configurer MongoDB et Redis
- Vérifier la communication entre services

Application Web

Initialiser le projet Flask/Django
Créer la structure des dossiers (models, routes, templates, static)

- Initialiser le projet Flask/Django
- Écrire le Dockerfile pour l'application
- Intégrer l'app dans docker-compose.yml
- Créer la page d'accueil basique (Hello World)
- Tester l'accès via http://localhost:8000

Tests et Documentation

- Générer des données de test (script Python avec Faker)
- Injecter les premières données dans Elasticsearch
- Documenter le processus de setup dans README.md
- Commit et push de la configuration complète

Livrables :

- Environnement Docker 100% fonctionnel
- README.md avec instructions de démarrage
- Script de génération de données de test
- Premières données visibles dans Kibana

Développement Backend Core

Objectifs :

- Développer les fonctionnalités obligatoires backend
- Implémenter l'API REST de base
- Configurer Logstash pour ingestion
- Intégrer MongoDB et Redis

Sprint Backend :

Module Upload :

- Route POST /upload pour recevoir les fichiers
- Validation du fichier (format, taille, structure)
- Sauvegarde temporaire sur le disque
- Extraction des métadonnées
- Enregistrement dans MongoDB
- Déclenchement de l'ingestion Logstash
- Gestion des erreurs et retours JSON

Module Logstash :

- Pipeline CSV avec filtres :
    - csv { } pour parser
    - date { } pour formater les timestamps
    - mutate { } pour transformation
    - grok { } si nécessaire pour patterns complexes
- Pipeline JSON avec validation
- Output vers Elasticsearch avec index dynamique
- Gestion des erreurs de parsing

Module Recherche :

- Route GET /search avec paramètres query string
- Construction de requêtes Elasticsearch (Query DSL)
- Pagination des résultats
- Tri et filtres
- Retour JSON des résultats
- Enregistrement de l'historique dans MongoDB

Module API REST :

- GET /api/v1/logs - Liste paginée
- GET /api/v1/logs/:id - Détail d'un log
- GET /api/v1/files - Liste des fichiers uploadés
- GET /api/v1/stats - Statistiques globales
- Documentation des endpoints dans README.md

Sprint Intégration :

- Connexion MongoDB (PyMongo)
- Connexion Redis (redis-py)
- Connexion Elasticsearch (elasticsearch-py)
- Tests unitaires des fonctions critiques
- Tests d'intégration (upload → Elasticsearch)
- Gestion des erreurs et logging applicatif
- Configuration via variables d'environnement

Livrables :

- API backend fonctionnelle
- Tests avec Postman/curl
- Code commenté et propre
- Collection Postman exportée (bonus)

Critères de validation :

- Upload d'un fichier CSV réussit
- Les logs apparaissent dans Elasticsearch après 10 secondes
- Une recherche retourne des résultats
- Les métadonnées sont dans MongoDB
- Aucune erreur dans les logs Docker

Développement Frontend et Visualisation

Objectifs :

- Créer les interfaces utilisateur
- Intégrer les appels API
- Développer les visualisations
- Configurer Kibana

Sprint Frontend :

Page d'accueil (Dashboard) :

- Layout responsive avec Bootstrap/Tailwind
- Cartes de statistiques (4 KPI minimum)
- Graphique d'évolution des logs (Chart.js)
- Tableau des derniers logs
- Appels AJAX vers l'API backend
- Rafraîchissement automatique (optionnel)

Page Upload :

- Formulaire avec drag & drop
- Validation côté client
- Barre de progression
- Messages de succès/erreur
- Liste des fichiers récemment uploadés
- Design professionnel

Page Recherche :

- Barre de recherche en texte libre
- Filtres (niveau, service, date)
- Date picker pour sélection de plage
- Bouton "Rechercher" et "Reset"
- Loading spinner pendant la recherche
- Affichage du nombre de résultats

Page Résultats :

- Tableau responsive avec DataTables.js
- Colonnes : Date, Niveau, Service, Message, Actions
- Pagination (50 résultats/page)
- Tri par colonne
- Bouton "Détails" pour chaque log
- Modal pour affichage des détails complets
- Bouton "Export CSV"

Navigation et Layout :

- Menu de navigation (navbar)
- Footer avec informations
- Breadcrumb pour la navigation
- Messages flash (success, error, warning)
- Page 404 personnalisée

Sprint Kibana:

Visualisations (minimum 3) :

Exemple E-Commerce :

1. Line chart : Transactions par heure (24h)
2. Pie chart : Répartition des erreurs par type
3. Bar chart : Top 10 produits avec erreurs

Exemple IoT :

1. Line chart : Température moyenne par heure
2. Heat map : Alertes par zone et par heure
3. Gauge : Consommation énergétique actuelle

Exemple Sécurité :

1. Timeline : Tentatives de connexion échouées
2. Map : Géolocalisation des connexions suspectes
3. Bar chart : Top 10 IPs avec plus d'échecs

Exemple SaaS :

1. Line chart : Temps de réponse moyen par endpoint
2. Area chart : Erreurs 5xx dans le temps
3. Metric : Nombre d'utilisateurs actifs (dernière heure)

Dashboard Kibana :

- Intégration des 3 visualisations
- Organisation cohérente
- Filtres temporels
- Refresh automatique
- Export du dashboard en JSON (sauvegarde)

Intégration Kibana dans l'app web :

- Page dédiée avec iframe vers Kibana
- Ou : Utilisation de l'API Kibana pour embed des visualisations
- Lien dans la navigation principale

Livrables :

- Interface web complète et fonctionnelle
- Design responsive testé (desktop, tablet, mobile)
- Visualisations Kibana créées et exportées
- Screenshots de l'application dans la documentation

Critères de validation :

- Toutes les pages sont accessibles et stylées
- Upload de fichier fonctionne via l'interface
- Recherche retourne des résultats visuels
- Dashboard Kibana affiche les données
- Pas d'erreurs JavaScript dans la console

Fonctionnalités Avancées

Objectifs :

- Implémenter les modules intermédiaires (3 minimum)
- Implémenter les modules avancés (2 minimum)
- Optimiser les performances
- Corriger les bugs identifiés

Stratégie recommandée :

Option A - Viser 16/20 : Choisir 3 modules intermédiaires parmi :

- Authentification + Rôles (impacte toute l'app, commencer tôt)
- Cache Redis (améliore les performances, relativement rapide)
- API RESTful + Swagger (valorise le backend)

Option B - Viser 20/20 : Choisir 3 modules intermédiaires + 2 modules avancés :

- Modules intermédiaires : Auth + Cache + Dashboards personnalisés
- Modules avancés : Alerting + Temps réel OU ML + CI/CD

Premier module intermédiaire

- Développement complet
- Tests manuels
- Documentation

Deuxième module intermédiaire

- Développement complet
- Intégration avec le reste de l'app
- Tests

Troisième module intermédiaire

- Développement
- Tests d'intégration globaux

Premier module avancé (si 20/20 visé)

- Développement de la fonctionnalité principale
- Tests basiques

Deuxième module avancé (si 20/20 visé)

- Développement
- Tests
- Refactoring et optimisations

Conseils :

- Ne pas chercher la perfection, viser le fonctionnel
- Documenter au fur et à mesure
- Faire des commits réguliers
- Prioriser les fonctionnalités les plus impactantes

Livrables :

- Modules supplémentaires opérationnels
- Code testé et documenté
- Mise à jour du README avec les nouvelles fonctionnalités

Tests, Documentation et Finalisation

Objectifs :

- Finaliser tous les développements
- Écrire la documentation complète
- Préparer la démo et la présentation
- Tests finaux et corrections

Planning détaillé :

Tests et Debug

- Tests de régression (vérifier que tout fonctionne)
- Tests sur différents navigateurs (Chrome, Firefox, Safari)
- Tests responsive (mobile, tablet)
- Correction des bugs critiques

- Tests de charge basiques (optionnel : Locust)
- Mesure des performances

Documentation Technique (Partie 1) Rédaction du document PDF (15-25 pages) :

- Page de garde professionnelle
- Table des matières
- Introduction (contexte, objectifs, scénario)
- Architecture globale (schémas + explications)
- Architecture détaillée par composant

Documentation Technique (Partie 2)

- Spécifications techniques (technologies, versions, justifications)
- Description des modules développés
- Extraits de code significatifs (max 2 pages de code)
- Configurations ELK (Logstash pipelines, mappings Elasticsearch)
- Modèles de données (MongoDB, Redis)

Documentation Technique (Partie 3)

- Guide d'installation et de déploiement
- Guide utilisateur avec screenshots
- Tests et validation (résultats, métriques)
- Difficultés rencontrées et solutions
- Perspectives d'amélioration
- Conclusion
- Annexes et références

Présentation Orale

- Création des slides PowerPoint/Google Slides (15-20 slides)
- Structure recommandée :
    - Slide 1 : Page de titre
    - Slide 2 : Plan de la présentation
    - Slides 3-4 : Contexte et problématique
    - Slide 5 : Scénario choisi
    - Slides 6-8 : Architecture technique (schémas clairs)
    - Slides 9-12 : Démonstration des fonctionnalités (screenshots)
    - Slide 13 : Technologies utilisées
    - Slide 14 : Difficultés et solutions
    - Slide 15 : Bilan et apprentissages
    - Slide 16 : Perspectives d'amélioration
    - Slide 17 : Questions
- Répétition de la présentation (chronomètre 20 minutes)
- Préparation des réponses aux questions fréquentes

Finalisation et Vérifications

- Checklist finale (voir section VII)
- Nettoyage du code (suppression du code commenté, debug prints)
- Vérification que `docker-compose up` fonctionne sur une machine vierge
- Mise à jour du README.md final

- Création de l'archive de livraison
- Test de l'archive sur une autre machine
- Soumission sur la plateforme (Moodle, email, etc.)

Livrables finaux :

- Code source complet (repository Git)
- Documentation technique PDF
- Vidéo de démo
- Slides de présentation
- README.md complet
- Fichier .env.example
- Script de génération de données de test
- Exports Kibana (dashboards, visualisations)