**Architecture de la Solution**

L'objectif est de construire une architecture distribuée pour analyser les logs web en combinant des traitements batch et stream avec Apache Spark, Kafka, HDFS et MongoDB, le tout orchestré avec Docker.

**1. Architecture Globale**

L'architecture repose sur les composants suivants :

**1.1 Ingestion des logs**

Kafka : Collecte et transmet les logs en streaming.

Producteurs Kafka : Simule l'arrivée en continu des logs.

**1.2 Stockage des logs**

HDFS : Stockage distribué des logs pour traitement batch.

MongoDB : Sauvegarde des résultats des analyses.

**1.3 Traitement des données**

Apache Spark Batch : Analyse des logs stockés en HDFS.

Apache Spark Streaming : Consomme et analyse les logs en temps réel depuis Kafka.

**1.4 Visualisation et Sauvegarde**

MongoDB : Stocke les résultats pour consultation rapide.

Tableau de bord (optionnel) : Visualisation avec Power BI, Grafana ou Dash.

**2. Déploiement avec Docker Compose**

L'architecture est déployée avec Docker Compose, qui orchestre les services :

Kafka + Zookeeper : Pour la gestion des flux de données.

HDFS (Hadoop) : Pour le stockage des logs.

MongoDB : Pour stocker les résultats des analyses.

Spark Master & Workers : Pour exécuter les jobs d'analyse.

Python Producer : Simule un serveur web en envoyant des logs à Kafka.

**3. Utilisation des Codes**

Étape 1 : Lancer l’infrastructure avec Docker

Créer un fichier docker-compose.yml et y définir les services (Hadoop, Spark, Kafka, MongoDB).

Démarrer l’infrastructure avec :

docker-compose up -d

Étape 2 : Envoyer des logs en continu à Kafka

Lancer un script Python pour générer des logs web en temps réel et les envoyer à Kafka :

Étape 3 : Lancer l’analyse Batch avec Spark

Une fois des logs stockés en HDFS, exécuter un job Spark Batch :

spark-submit --master spark://spark-master:7077 batch\_processing.py

Étape 4 : Lancer l’analyse Streaming avec Spark

Consommer et analyser les logs en temps réel depuis Kafka :

spark-submit --master spark://spark-master:7077 stream\_processing.py

**4. Fonctionnalités et Statistiques Implémentées**

Traitement Batch (HDFS → Spark Batch)

🔹 Produits les plus consultés

🔹 Répartition des codes HTTP (200, 404, 500, etc.)

🔹 Top 10 des IPs les plus actives

🔹 Temps de réponse moyen par catégorie de produit

🔹 Analyse du trafic par heure (Peak Hour Analysis)

🔹 Détection de comportements suspects (attaques DDoS)

Traitement Streaming (Kafka → Spark Streaming)

🔹 Détection des erreurs en temps réel (pics de 404/500)

🔹 Produits les plus consultés en temps réel (tendances)

🔹 Surveillance des IPs suspectes (activités anormales)

**Architecture de la solution pour l’analyse des logs avec Spark**

L’architecture dépend des besoins :

* **Si on veut analyser les logs en différé** → **Batch Processing**
* **Si on veut surveiller les logs en temps réel** → **Streaming Processing**

Je vais détailler **les composants**, **les flux de données**, et **quand utiliser chaque approche**.

**🏗️ Architecture Globale**

**1️⃣ Composants Principaux**

| **Composant** | **Utilité** |
| --- | --- |
| **Web Server (Nginx, Apache)** | Génère les logs web |
| **Kafka** | Collecte et diffuse les logs en temps réel |
| **HDFS / S3** | Stocke les logs pour les analyses en batch |
| **Spark (Batch & Streaming)** | Effectue l’analyse des logs |
| **MongoDB** | Sauvegarde les résultats d’analyse |
| **Tableau de Bord (Grafana, Power BI)** | Visualise les métriques des logs |

**2️⃣ Quand utiliser Batch Processing vs Streaming Processing ?**

| **Cas d’usage** | **Approche** | **Pourquoi ?** |
| --- | --- | --- |
| Produits les plus consultés sur 24h | **Batch Processing** | On analyse les logs **historiques** stockés |
| Répartition des codes HTTP sur la journée | **Batch Processing** | On calcule des **statistiques rétrospectives** |
| Détection d’un pic d’erreurs HTTP 404/500 en direct | **Streaming Processing** | On surveille les logs **en temps réel** |
| Identification d’adresses IP suspectes (DDoS) | **Streaming Processing** | On détecte des **requêtes anormales en direct** |

**🏗️ Architecture pour le Batch Processing (Analyse des logs stockés)**

**📌 Description**

* Les logs sont générés par un **serveur web** et stockés dans **HDFS**.
* **Spark en mode Batch** va lire ces logs pour extraire des **statistiques globales**.
* Les résultats sont sauvegardés dans **MongoDB** et affichés sur un tableau de bord.

**🛠️ Stack Technique**

* **Stockage** → HDFS / S3
* **Traitement** → Apache Spark (Batch)
* **Sauvegarde** → MongoDB
* **Visualisation** → Grafana / Power BI

**📊 Flux des données**

1️⃣ **Web Server (Apache/Nginx)** écrit les logs.  
2️⃣ Un **script cron** ou **Fluentd** envoie les logs vers **HDFS**.  
3️⃣ **Spark Batch Job** est lancé toutes les X heures pour analyser les logs.  
4️⃣ Les résultats sont stockés dans **MongoDB**.  
5️⃣ Un **dashboard Grafana/Power BI** récupère les données de MongoDB.

💡 **Exemple de Job Spark pour Batch Processing :**

python

CopierModifier

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, count

# Initialisation Spark

spark = SparkSession.builder.appName("Batch Log Analysis").getOrCreate()

# Chargement des logs depuis HDFS

df = spark.read.option("header", True).csv("hdfs://namenode:9000/logs/\*.csv")

# 1️⃣ Calcul des produits les plus consultés

top\_products = df.filter(col("url").contains("/products/")).groupBy("url").count().orderBy(col("count").desc())

# 2️⃣ Analyse des codes HTTP

http\_status = df.groupBy("http\_code").count()

# 3️⃣ Adresses IP les plus actives

top\_ips = df.groupBy("ip\_address").count().orderBy(col("count").desc()).limit(10)

# Sauvegarde des résultats dans MongoDB

top\_products.write.format("mongo").option("uri", "mongodb://mongo:27017/logs.top\_products").mode("append").save()

http\_status.write.format("mongo").option("uri", "mongodb://mongo:27017/logs.http\_status").mode("append").save()

**🏗️ Architecture pour le Streaming Processing (Surveillance des logs en temps réel)**

**📌 Description**

* Les logs sont envoyés **en temps réel** via **Kafka**.
* **Spark Streaming** consomme les logs pour **détecter des anomalies (ex: pics de 404/500)**.
* On stocke les alertes dans **MongoDB** et on déclenche des notifications en cas de problème.

**🛠️ Stack Technique**

* **Ingestion Temps Réel** → Apache Kafka
* **Traitement Temps Réel** → Apache Spark Streaming
* **Stockage des Alertes** → MongoDB
* **Alertes & Monitoring** → Grafana, Slack, ELK Stack

**📊 Flux des données**

1️⃣ **Web Server** écrit les logs en temps réel.  
2️⃣ **Filebeat / Fluentd** envoie les logs vers **Kafka**.  
3️⃣ **Spark Streaming** consomme Kafka et traite les logs **en continu**.  
4️⃣ Les alertes sont envoyées dans **MongoDB**.  
5️⃣ Un **système de monitoring** (Grafana, Slack) déclenche une alerte si un seuil est dépassé.

💡 **Exemple de Spark Streaming pour détecter un pic de 404/500 :**

python

CopierModifier

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, window

# Initialisation Spark Streaming

spark = SparkSession.builder.appName("Streaming Log Analysis").getOrCreate()

# Lecture des logs depuis Kafka

logs\_df = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "kafka:9092").option("subscribe", "logs").load()

# Transformation des logs

logs\_parsed = logs\_df.selectExpr("CAST(value AS STRING) as log\_entry")

# Extraction des codes HTTP

logs\_filtered = logs\_parsed.filter(col("log\_entry").contains(" 404 ") | col("log\_entry").contains(" 500 "))

# Détection des pics d’erreurs (ex : plus de 10 erreurs en 5 minutes)

error\_counts = logs\_filtered.groupBy(window(col("timestamp"), "5 minutes")).count()

# Envoi des résultats à MongoDB

query = error\_counts.writeStream.outputMode("append").format("mongo").option("uri", "mongodb://mongo:27017/logs.alerts").start()

query.awaitTermination()

**🚀 Résumé : Quand utiliser Log Analysis avec Spark ?**

| **Besoins** | **Approche** | **Outils** |
| --- | --- | --- |
| **Analyse des logs historiques (statistiques sur 24h, top produits, erreurs)** | **Batch Processing** | Spark (Batch) + HDFS + MongoDB |
| **Surveillance en temps réel (pic d’erreurs 404/500, détection DDoS)** | **Streaming Processing** | Spark Streaming + Kafka + MongoDB |
| **Générer des tableaux de bord (KPI sur logs, monitoring du site)** | **Batch ou Streaming** | Power BI, Grafana, ELK Stack |

**📌 Conclusion**

* **Log Analysis avec Spark** peut être du **Batch Processing (logs historiques)** ou du **Streaming Processing (temps réel)**.
* **Batch Processing** est utile pour les **rapports et statistiques globales**.
* **Streaming Processing** est essentiel pour la **surveillance en temps réel** et la **détection d’anomalies**.
* L’architecture doit être adaptée en fonction du **besoin métier** !