Système de Surveillance de Pression Artérielle

Architecture Big Data avec Kafka, Elasticsearch et Kibana

Mehdi

Projet Big Data

30 janvier 2026

Plan de la Présentation

Contexte du Projet

Objectif

Créer un système de surveillance en temps réel des mesures de pression artérielle pour détecter automatiquement les anomalies selon les standards médicaux AHA (American Heart Association).

- **Problématique** : Détection précoce des hypertensions et crises cardiaques
- **Solution** : Streaming temps réel avec analyse automatique
- **Technologies** : Kafka, Elasticsearch, Kibana, Python, FHIR

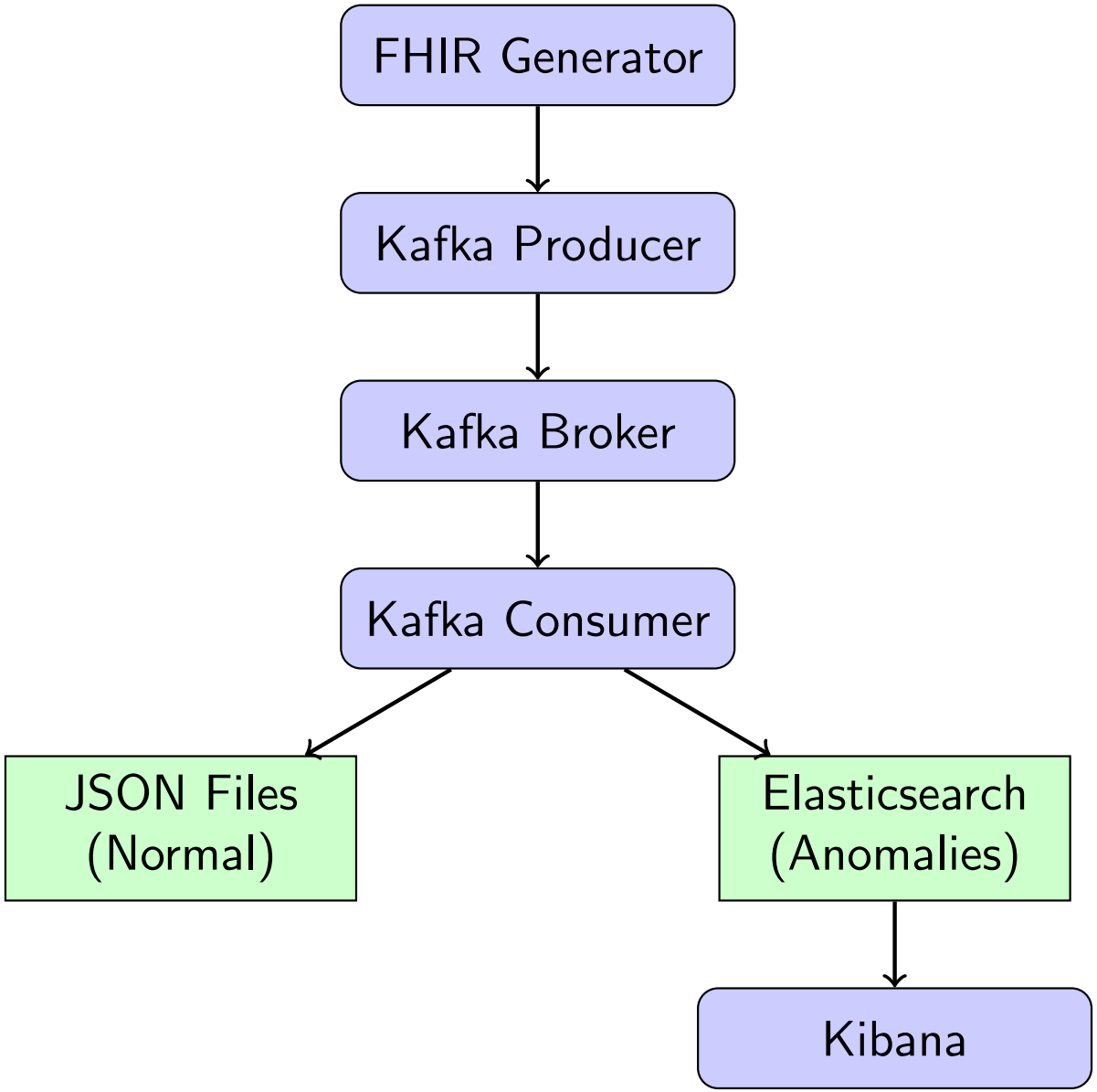
Standards Médicaux AHA

Catégorie	Systolique	Diastolique	Sévérité
Normal	< 120	ET < 80	-
Elevated	120-129	ET < 80	Modérée
Hypertension Stage 1	130-139	OU 80-89	Modérée
Hypertension Stage 2	≥ 140	OU ≥ 90	Haute
Hypertensive Crisis	> 180	OU > 120	Critique
Hypotension	< 90	OU < 60	Modérée

Urgence Médicale

Les crises hypertensives (>180/120) nécessitent une intervention immédiate !

Architecture Globale



Technologies Utilisées

Backend

- Python 3.13+
- Apache Kafka 7.5.0
- Elasticsearch 8.11.0
- Kibana 8.11.0
- Docker Compose

Standards & Librairies

- FHIR R4 (HL7)
- LOINC codes
- kafka-python-ng
- fhir.resources
- Faker (données)

Containerisation

Tout le système tourne dans Docker pour une reproductibilité complète

Docker Compose - Configuration

Listing 1 – docker-compose.yml

```
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS:
        PLAINTEXT://localhost:9092
```

Docker Compose - Services de Stockage

```
  elasticsearch:
    image: elasticsearch:8.11.0
    ports:
      - "9200:9200"
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    healthcheck:
      test: ["CMD-SHELL", "curl -f
        http://localhost:9200/_cluster/health"]

  kibana:
    image: kibana:8.11.0
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_HOSTS=
        http://elasticsearch:9200
```

Services Docker - Récapitulatif

Service	Port	Fonction
Zookeeper	2181	Coordination Kafka
Kafka	9092	Broker de messages
Elasticsearch	9200	Stockage anomalies
Kibana	5601	Visualisation

Commande de Démarrage

```
docker-compose up -d
```

Format FHIR (Fast Healthcare Interoperability Resources)

Qu'est-ce que FHIR ?

Standard HL7 pour l'échange de données de santé interopérables

Ressource Observation - Pression Artérielle

- **Codes LOINC** :
  - 85354-9 : Blood pressure panel
  - 8480-6 : Systolic blood pressure
  - 8462-4 : Diastolic blood pressure
- **Unité** : mmHg (millimètres de mercure)
- **Patient** : Identifiant + Nom (généré)
- **Timestamp** : Date/heure de la mesure

Code - Génération FHIR (1/2)

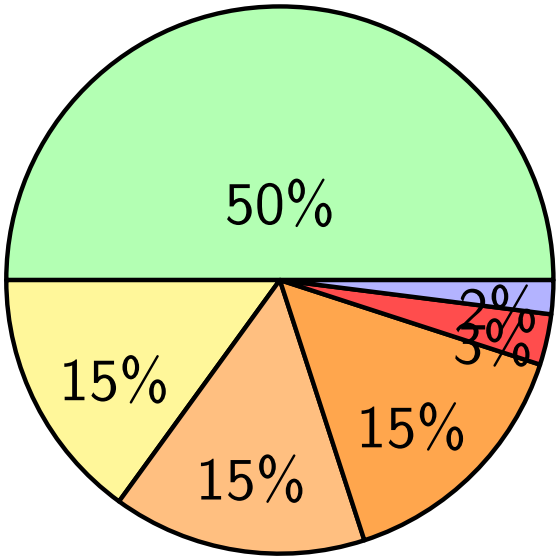
Listing 2 – fhir_generator.py

```python
from fhir.resources.observation import Observation
from faker import Faker
import random

class FHIRBloodPressureGenerator:
    def generate_blood_pressure_observation(self):
        # Distribution selon AHA
        category_choice = random.random()

        if category_choice < 0.50:  # 50% Normal
            systolic = random.randint(90, 119)
            diastolic = random.randint(60, 79)
        elif category_choice < 0.65:  # 15% Elevated
            systolic = random.randint(120, 129)
            diastolic = random.randint(60, 79)
        # ... autres categories
```

Code - Génération FHIR (2/2)

```
        # Creation ressource FHIR
        observation = Observation(
            resourceType="Observation",
            status="final",
            code={
                "coding": [{
                    "system": "http://loinc.org",
                    "code": "85354-9",
                    "display": "Blood pressure panel"
                }]
            },
            component=[
                {  # Systolic
                    "code": {"coding": [{
                        "system": "http://loinc.org",
                        "code": "8480-6"
                    }]},
                    "valueQuantity": {
                        "value": systolic,
                        "unit": "mmHg"
                    }
```

Distribution Réaliste des Données



- Normal (50%)
- Elevated (15%)
- Stage 1 (15%)
- Stage 2 (15%)
- Crisis (3%)
- Hypotension (2%)

Sur 1000 observations :

- 500 normales → JSON
- 500 anomalies → ES

Kafka Producer - Connexion

Listing 3 – kafka_producer.py

```python
from kafka import KafkaProducer
import json

class BloodPressureProducer:
    def __init__(self, kafka_server='localhost:9092'):
        self.producer = KafkaProducer(
            bootstrap_servers=kafka_server,
            value_serializer=lambda v:
                json.dumps(v).encode('utf-8'),
            acks='all',  # Attendre confirmation
            retries=3    # Retry en cas d'echec
        )
        self.topic = 'blood-pressure-observations'
        self.generator = FHIRBloodPressureGenerator()
```

Kafka Producer - Publication

```python
def send_observation(self):
    # Generer observation FHIR
    observation = self.generator \
        .generate_blood_pressure_observation()

    # Convertir en dict
    message = observation.dict()

    # Publier sur Kafka
    future = self.producer.send(
        self.topic,
        value=message
    )

    # Attendre confirmation
    record_metadata = future.get(timeout=10)

    return record_metadata
```

Kafka Producer - Utilisation CLI

Listing 4 – Ligne de commande

```
# Generer 1000 observations
python kafka_producer.py --count 1000 --interval 0.2

# Generation continue
python kafka_producer.py

# Avec serveur Kafka distant
python kafka_producer.py \
    --kafka-server kafka.example.com:9092
```

Débit

Avec `--interval 0.2`, on génère **5 messages/seconde**
1000 observations = **3 minutes 20 secondes**

Kafka Consumer - Connexion

Listing 5 – kafka_consumer.py

```python
from kafka import KafkaConsumer
from elasticsearch import Elasticsearch
import json

class BloodPressureConsumer:
    def __init__(self):
        # Connexion Kafka
        self.consumer = KafkaConsumer(
            'blood-pressure-observations',
            bootstrap_servers='localhost:9092',
            group_id='bp-consumer-group',
            auto_offset_reset='earliest',
            value_deserializer=lambda m:
                json.loads(m.decode('utf-8'))
        )

        # Connexion Elasticsearch
        self.es = Elasticsearch(['http://localhost:9200'])
```

Analyse des Anomalies - Seuils AHA

```python
class BloodPressureAnalyzer:
    # Seuils AHA
    SYSTOLIC_NORMAL_MAX = 120
    DIASTOLIC_NORMAL_MAX = 80
    SYSTOLIC_ELEVATED_MAX = 129
    SYSTOLIC_STAGE1_MAX = 139
    DIASTOLIC_STAGE1_MIN = 80
    DIASTOLIC_STAGE1_MAX = 89
    SYSTOLIC_STAGE2_MIN = 140
    DIASTOLIC_STAGE2_MIN = 90
    SYSTOLIC_CRISIS_MIN = 180
    DIASTOLIC_CRISIS_MIN = 120

    @classmethod
    def analyze_observation(cls, systolic, diastolic):
        anomalies = []
        category = 'unknown'
        severity = None
        # ... logique de classification
```

Analyse des Anomalies - Classification

```
    # HYPERTENSIVE CRISIS (Urgence!)
    if systolic > cls.SYSTOLIC_CRISIS_MIN or \
       diastolic > cls.DIASTOLIC_CRISIS_MIN:
        category = 'hypertensive_crisis'
        severity = 'critical'
        anomalies.append('hypertensive_crisis')

    # HYPERTENSION STAGE 2
    elif systolic >= cls.SYSTOLIC_STAGE2_MIN or \
         diastolic >= cls.DIASTOLIC_STAGE2_MIN:
        category = 'hypertension_stage2'
        severity = 'high'
        anomalies.append('hypertension_stage2')

    # ... autres categories

    return {
        'category': category,
        'severity': severity,
        'anomalies': anomalies
    }
```

Stockage Différencié

```python
def process_message(self, message):
    # Extraire pressions
    systolic = self.extract_systolic(message)
    diastolic = self.extract_diastolic(message)

    # Analyser
    analysis = BloodPressureAnalyzer \
        .analyze_observation(systolic, diastolic)

    if analysis['category'] == 'normal':
        # Sauvegarder en JSON local
        self.save_to_json(message)
    else:
        # Indexer dans Elasticsearch
        self.index_to_elasticsearch(
            message,
            analysis
        )
```

Modèle d'Index Elasticsearch

Listing 6 – elasticsearch_index_mapping.json

```
{
  "mappings": {
    "properties": {
      "observation_id": {"type": "keyword"},
      "patient_id": {"type": "keyword"},
      "patient_name": {"type": "text"},
      "systolic_pressure": {"type": "integer"},
      "diastolic_pressure": {"type": "integer"},
      "category": {"type": "keyword"},
      "severity": {"type": "keyword"},
      "anomalies": {"type": "keyword"},
      "effectiveDateTime": {"type": "date"},
      "processed_datetime": {"type": "date"},
      "fhir_resource": {
        "type": "object",
        "enabled": false
      }
    }
  }
```

Champs de l'Index - Détails

Champ	Type	Description
observation_id	keyword	ID unique FHIR
patient_id	keyword	ID du patient
patient_name	text	Nom (recherche)
systolic_pressure	integer	Pression systolique
diastolic_pressure	integer	Pression diastolique
category	keyword	Catégorie AHA
severity	keyword	Sévérité
anomalies	keyword	Anomalies détectées
effectiveDateTime	date	Date mesure
processed_datetime	date	Date traitement

Indexation dans Elasticsearch

```python
def index_to_elasticsearch(self, message, analysis):
    document = {
        'observation_id': message.get('id'),
        'patient_id': self.extract_patient_id(message),
        'patient_name': self.extract_patient_name(message),
        'systolic_pressure': systolic,
        'diastolic_pressure': diastolic,
        'category': analysis['category'],
        'severity': analysis['severity'],
        'anomalies': analysis['anomalies'],
        'effectiveDateTime': message.get('effectiveDateTime'
            ),
        'processed_datetime': datetime.now().isoformat(),
        'fhir_resource': message
    }

    self.es.index(
        index='blood-pressure-anomalies',
        body=document
    )
```

Requêtes Elasticsearch - Exemples (1/2)

Listing 7 – Compter les anomalies

```
curl -X GET "localhost:9200/blood-pressure-anomalies/_count"
```

Listing 8 – Distribution par categorie

```
curl -X GET "localhost:9200/blood-pressure-anomalies/_search
    " \
  -H 'Content-Type: application/json' -d '{
  "size": 0,
  "aggs": {
    "par_categorie": {
      "terms": {"field": "category"}
    }
  }
}'
```

Requêtes Elasticsearch - Exemples (2/2)

Listing 9 – Trouver les cas critiques

```
curl -X GET "localhost:9200/blood-pressure-anomalies/_search
    " \
  -H 'Content-Type: application/json' \
  -d '{"query": {"term": {"severity": "critical"}}}'
```

Listing 10 – Rechercher par patient

```
curl -X GET "localhost:9200/blood-pressure-anomalies/_search
    " \
  -H 'Content-Type: application/json' \
  -d '{"query": {"match": {"patient_name": "Dupont"}}}'
```

Dashboard Kibana - Vue d'Ensemble

Composants du Dashboard :

1. **Métriques principales**
   - Total anomalies détectées
   - Nombre de cas critiques
   - Pression artérielle moyenne
2. **Distribution par catégorie AHA** (Pie Chart)
3. **Distribution par sévérité** (Bar Chart)
4. **Top 10 cas critiques** (Data Table)
5. **Heat Map** pression par patient

Import du Dashboard

Méthode 1 : Interface Kibana

1. Ouvrir http ://localhost :5601
2. Menu → Management → Stack Management
3. Kibana → Saved Objects
4. Cliquer "Import"
5. Sélectionner `dashboard/DASHBOARD.ndjson`
6. Import terminé !

Méthode 2 : API

Commande curl

```
curl -X POST "localhost:5601/api/saved_objects/_import"
-H "kbn-xsrf: true" --form file=@DASHBOARD.ndjson
```

Visualisations - Détails

1. Distribution par Catégorie AHA (Pie Chart)

Configuration :

- Slice by : `category.keyword`
- Metric : Count
- Couleurs personnalisées par catégorie

2. Top 10 Cas Critiques (Data Table)

Configuration :

- Filter : `category: hypertensive_crisis`
- Rows : `patient_id.keyword`
- Metrics : Max systolic, Max diastolic, Count

Résultats sur 1000 Observations

Stockage :

- ∼500 normales
  → JSON local
- ∼500 anomalies
  → Elasticsearch

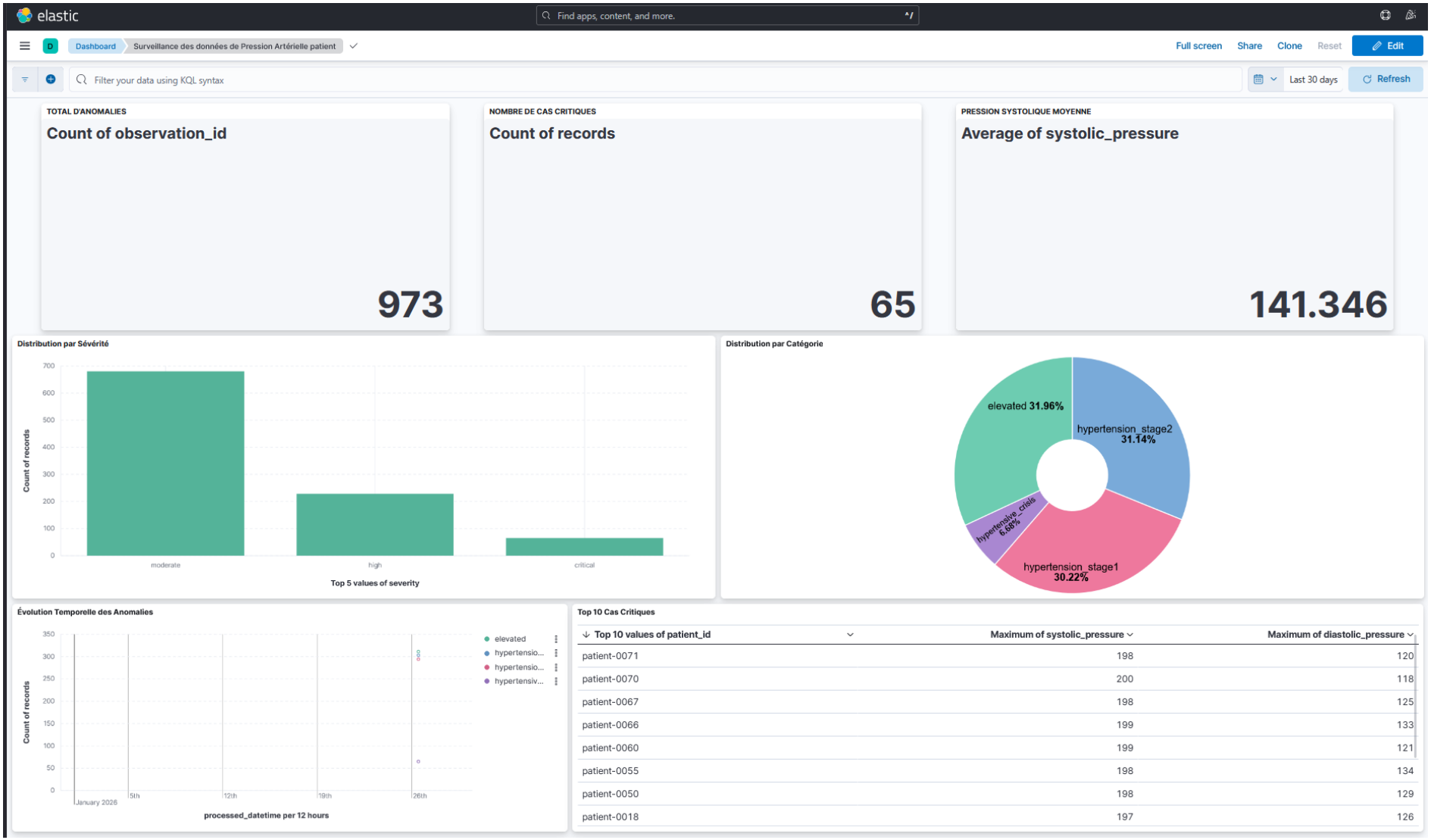
Distribution anomalies :

- 150 Elevated (15%)
- 150 Stage 1 (15%)
- 150 Stage 2 (15%)
- 30 Crisis (3%)
- 20 Hypotension (2%)

Temps de Traitement

1000 observations traitées en ∼**3 minutes**
(débit : 5 messages/seconde)

Dashboard Kibana



Démarrage du Système - Étape par Étape

1. Démarrer l'infrastructure Docker

```
docker-compose up -d
# Attendre 60 secondes
```

2. Lancer le Consumer (Terminal 1)

```
python scripts/kafka_consumer.py
```

3. Lancer le Producer (Terminal 2)

```
python scripts/kafka_producer.py --count 1000 --interval 0.2
```

4. Accéder à Kibana

```
# Navigateur
http://localhost:5601
```

Commandes de Vérification

Listing 11 – Verifier les services Docker

```
docker-compose ps
# Doit afficher 4 services "Up"
```

Listing 12 – Tester Elasticsearch

```
curl http://localhost:9200/_cluster/health
```

Listing 13 – Compter les anomalies

```
curl http://localhost:9200/blood-pressure-anomalies/_count
```

Listing 14 – Compter les observations normales

```
ls -1 data/normal_patients/ | wc -l
```

Tests Système Automatiques

```
python scripts/test_system.py
```

Tests effectués :

- ✓ Connexion Kafka
- ✓ Existence du topic `blood-pressure-observations`
- ✓ Connexion Elasticsearch
- ✓ Mapping de l'index
- ✓ Connexion Kibana
- ✓ Data Views Kibana
- ✓ Test end-to-end avec données réelles

Problèmes Techniques Rencontrés

1. **Incompatibilité kafka-python avec Python 3.13**
   - Erreur : ModuleNotFoundError : kafka.vendor.six.moves
   - Solution : Utilisation de `kafka-python-ng==2.2.2`

2. **Encodage Unicode sous Windows**
   - Erreur : UnicodeEncodeError avec caractères spéciaux
   - Solution : Remplacement par [OK], [ERREUR], [ALERTE]

3. **Bug de classification (catégorie "unknown")**
   - Problème : Systolic=120 classé en "unknown"
   - Solution : Changement de > à >= dans la condition

Solutions Apportées - Distribution Temporelle

Problème Initial

Toutes les données concentrées sur un seul jour
$\rightarrow$ Line Chart inutilisable pour voir l'évolution

Solution Implémentée

Génération de timestamps aléatoires sur **7 jours**

```
timestamp = datetime.now() - timedelta(
  days=random.randint(0, 7),
  hours=random.randint(0, 23),
  minutes=random.randint(0, 59)
)
```

Résultat : Visualisation temporelle exploitable dans Kibana

Points Forts de l'Architecture

Scalabilité

- Kafka : 3 partitions pour parallélisation
- Elasticsearch : Index optimisé avec sharding
- Docker : Déploiement reproductible

Conformité Standards

- FHIR R4 : Interopérabilité santé garantie
- LOINC : Codes standardisés reconnus mondialement
- AHA : Guidelines médicaux officiels

Optimisation Ressources

- Stockage différencié (JSON vs Elasticsearch)
- Seules les anomalies sont indexées
- Économie d'espace et de temps de requête

Innovations du Système

1. **Détection Intelligente**
   - 6 catégories AHA
   - 3 niveaux de sévérité (critical, high, moderate)
   - Classification automatique en temps réel
2. **Traitement Temps Réel**
   - Streaming continu avec Kafka
   - Analyse immédiate des observations
   - Alertes instantanées pour cas critiques
3. **Visualisation Interactive**
   - Dashboard Kibana complet
   - Filtres dynamiques
   - Drill-down par patient

Améliorations Futures - Court Terme

- **Alertes en temps réel**
  - Email/SMS pour cas critiques
  - Webhooks vers systèmes hospitaliers
  - Notifications push
- **API REST**
  - FastAPI pour exposition des données
  - Authentification JWT
  - Documentation Swagger
- **Interface Web**
  - Dashboard React/Vue.js
  - Gestion des patients
  - Export de rapports PDF

Améliorations Futures - Long Terme

- **Machine Learning**
  - Prédiction des risques cardiovasculaires
  - Détection d'anomalies par patterns
  - Recommandations personnalisées
- **Multi-hôpital**
  - Support de plusieurs établissements
  - Fédération de données
  - Anonymisation RGPD
- **Intégrations Supplémentaires**
  - HL7 v2 (standard historique)
  - DICOM (imagerie médicale)
  - Dossiers Patients Informatisés (DPI)

Récapitulatif du Projet

Objectif Atteint

Système de surveillance en temps réel fonctionnel, conforme aux standards médicaux et technologiques

Livrables :

- ✓ Scripts Python (5 fichiers)
- ✓ Infrastructure Docker complète
- ✓ Modèle d'index Elasticsearch
- ✓ Configuration Kafka
- ✓ Dashboard Kibana exportable
- ✓ Documentation technique
- ✓ Tests automatisés

Performances :

- 1000+ observations traitées en 3 minutes
- Détection d'anomalies en temps réel
- Visualisation interactive opérationnelle

Compétences Acquises

Technologies

- Apache Kafka
- Elasticsearch
- Kibana
- Docker
- Python avancé

Concepts

- Streaming temps réel
- Big Data
- Standards santé (FHIR)
- Architecture distribuée
- DevOps

Compétence Clé

Intégration de multiples technologies pour créer un système end-to-end fonctionnel

Démonstration du Système

http ://localhost :5601

- Génération de données en temps réel
- Visualisation dans Kibana
- Requêtes Elasticsearch
- Dashboard interactif

Annexe A - Structure des Livrables

```
LIVRABLES/
    README.md
    docker-compose.yml
    requirements.txt
    scripts/
        fhir_generator.py
        kafka_producer.py
        kafka_consumer.py
        setup_kibana.py
        test_system.py
    config/
        elasticsearch_index_mapping.json
        kafka_topic_config.json
    dashboard/
        DASHBOARD.ndjson
    exemples/
        exemple_observation_normale.json
        exemple_anomalie_critique.json
```

Annexe B - Dépendances Python

Listing 15 – requirements.txt

```
# Kafka (compatible Python 3.13+)
kafka-python-ng==2.2.2

# Elasticsearch
elasticsearch==8.11.0

# FHIR
fhir.resources==7.1.0

# Generation de donnees
faker==22.0.0

# Utilitaires
python-dateutil==2.8.2
```

Installation : `pip install -r requirements.txt`

Annexe C - Ressources Complémentaires

Documentation officielle :

- Kafka : https ://kafka.apache.org/documentation/
- Elasticsearch : https ://www.elastic.co/guide/
- FHIR : https ://www.hl7.org/fhir/
- LOINC : https ://loinc.org/

Standards médicaux :

- AHA Blood Pressure Guidelines
- HL7 FHIR Observation Resource
- LOINC Blood Pressure Codes