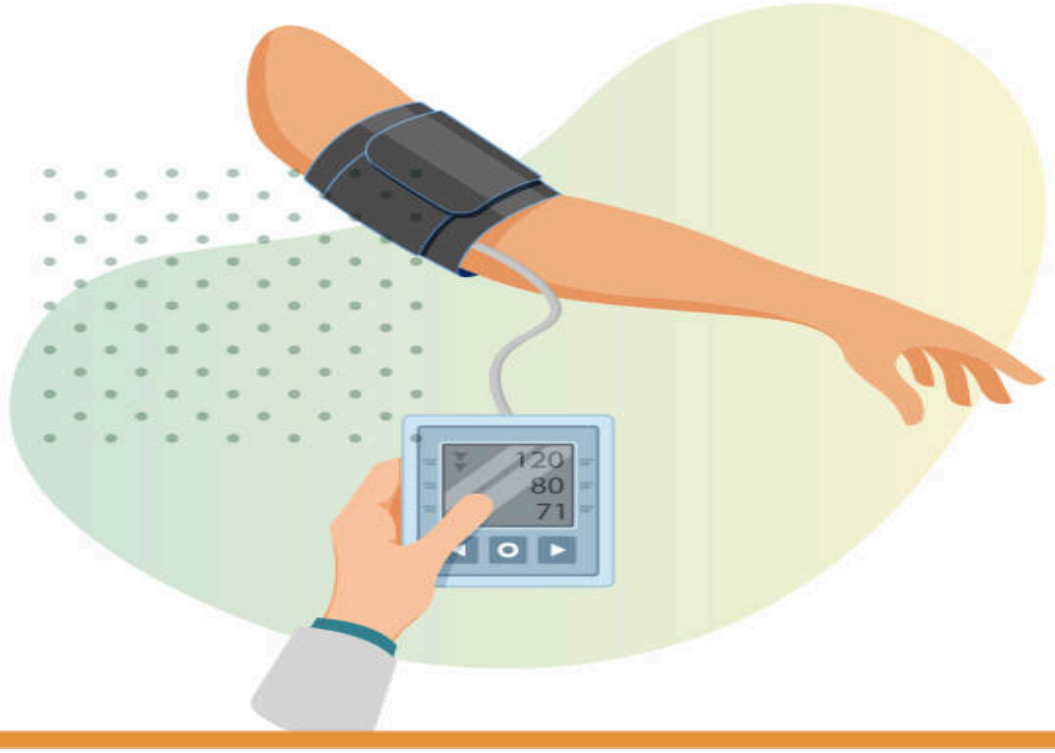
Rapport sur le système de surveillance des données de pression artérielle



Réalisé par :

- Jalakshana KANNAN
- Michelle TOBING

Table des matières :

Introduction

Ce document présente un système de surveillance qui permet de détecter les anomalies dans les données de pression artérielle des patients et de les visualiser en temps réel à l'aide de Kafka, Elasticsearch et Kibana.

2. Description du système

2.1. Architecture globale

Le système repose sur une architecture distribuée intégrant les composants suivants :

- **Génération de données** : Création de messages FHIR au format JSON décrivant les pressions systoliques et diastoliques des patients.
- **Transmission des données** : Utilisation de Kafka pour la publication et la consommation des messages entre différents services.
- **Détection des anomalies** : Analyse des données consommées pour détecter des anomalies (par exemple, hypertension ou hypotension).
- **Stockage et indexation** : Elasticsearch pour indexer les anomalies détectées.
- **Visualisation des données** : Tableau de bord Kibana pour surveiller les anomalies en temps réel.

2.2. Description détaillée des composants principaux

2.2.1. Génération des messages FHIR

- **Script : generate_fhir.py**
- **Description :** La bibliothèque Python **fhir.resources** est utilisée pour créer des objets FHIR conformes à la norme HL7, incluant des données sur les patients et leur pression artérielle (systolique et diastolique).
- **Exemple :** Les messages générés sont sauvegardés dans un fichier JSON (**blood_pressure.json**) pour être publiés ultérieurement sur Kafka.

2.2.2. Transmission via Kafka

- **Script : producer.py**
- **Description :** Publie des messages FHIR sur le topic Kafka **blood_pressure_readings**, contenant des données patient et des mesures vitales.
- **Détails techniques :**
    - Kafka est configuré avec un cluster local contenant un broker et un zookeeper.
    - Les messages sont sérialisés au format JSON avant d'être publiés.

2.2.3. Consommation et analyse des données

- **Script : consumer.py**
- **Description :** Consomme les messages du topic Kafka, détecte les anomalies, et indexe les résultats dans Elasticsearch.
- **Méthodologie :**
    - Les valeurs de pression artérielle sont analysées pour identifier des anomalies (hypertension ou hypotension).
    - Les anomalies détectées sont enrichies avec des métadonnées et indexées dans Elasticsearch pour le suivi.

2.2.4. Stockage dans Elasticsearch

- **Configuration :** Modèle d'index pour stocker les anomalies avec les champs suivants :
    - **patient_id**
    - **systolic_pressure**
    - **diastolic_pressure**
    - **anomaly_type**
    - **timestamp**
- **Création automatique :** Le script vérifie si l'index existe et le crée si nécessaire.

3. Déploiement du système

3.1. Prérequis

- Docker et Docker Compose installés.
- Bibliothèques Python nécessaires : **fhir.resources**, **faker**, **kafka-python**, **elasticsearch**.

3.2. Étapes de déploiement

3.2.1. Configuration de l'environnement

1. Création du fichier **docker-compose.yml**
    - Définit les services pour Zookeeper, Kafka, Elasticsearch, et Kibana.
    - Expose les ports nécessaires pour chaque composant (par exemple, 9093 pour Kafka, 9200 pour Elasticsearch, 5601 pour Kibana).
2. Lancement des conteneurs Docker à l'aide de **"docker-compose up -d"** dans le terminal.

3.2.2. Génération et publication des messages

1. Exécution du script **generate_fhir.py**
    - Génère un fichier JSON contenant les données FHIR des patients.
2. **Exécution du script producer.py**
    - Publie les messages sur le topic Kafka.

3.2.3. Consommation et analyse des messages

1. **Exécution du script consumer.py**

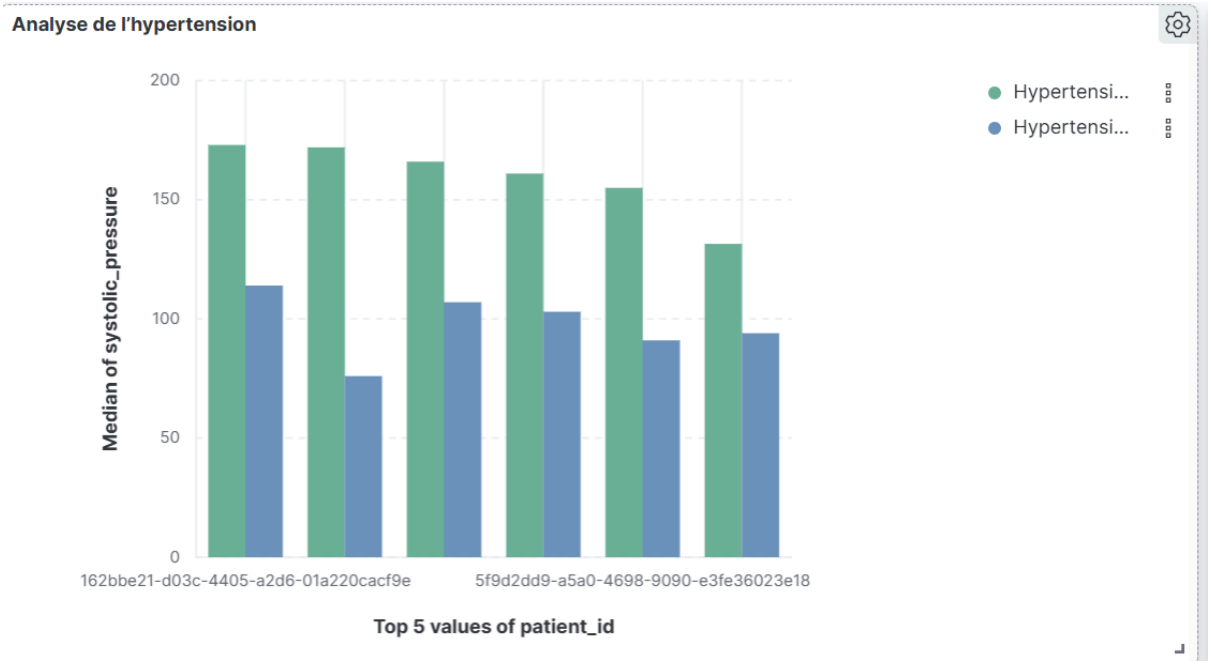
- Consomme les messages Kafka.
- Détecte les anomalies.
- Indexe les anomalies détectées dans Elasticsearch.

3.2.4. Configuration et accès à Kibana

1. **Accès au tableau de bord dans le navigateur**
   - URL : http://localhost:5601
2. **Création des visualisations**
   - Utilisation des outils Kibana pour créer des graphiques et des tableaux basés sur les données indexées.

4. Résultats du tableau de bord Kibana

1. **Analyse des hypertensions** ((Visualisation des pressions systoliques médianes pour les patients ayant présenté des anomalies d'hypertension) :

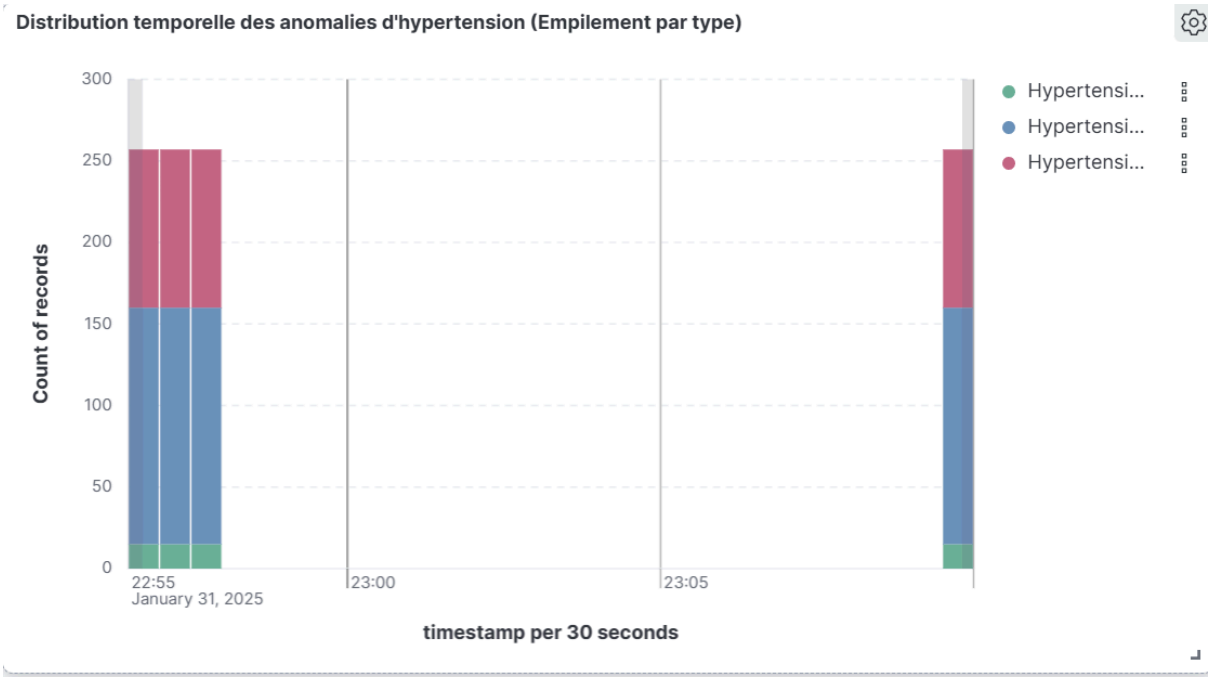


2. **Répartition des anomalies** (Tableau des anomalies détectées, regroupées par type et par patient) :

Répartition des anomalies d'hypertension par patient

Top 5 values of anomaly_type	Top 5 values of patient_id	Count of records
Hypertension	162bbe21-d03c-4405-a2d6-01a:	4
Hypertension	2a64ef1d-fa4d-4a11-bba4-af453	4
Hypertension	2cef7e9f-c249-4f74-b427-569f1	4
Hypertension	3910dd41-bb6e-4e87-8530-931	4
Hypertension	396fb588-7f01-4e6f-b4fd-33444	4
Hypertension	Other	40

3. **Distribution temporelle** (Histogramme montrant l'évolution temporelle des anomalies détectées) :



5. Annexes

1. Script Python : **generate_fhir.py**

```
from fhir.resources.observation import Observation
from faker import Faker
import json
```

```python
# Initialisation de Faker pour générer des données aléatoires
fake = Faker()

def generate_fhir_observation(patient_id, systolic, diastolic):
    """Génère une ressource FHIR Observation pour la pression artérielle."""
    observation = Observation(
        resourceType="Observation",
        id=fake.uuid4(),
        status="final",
        category=[{
            "coding": [{
                "system": "http://terminology.hl7.org/CodeSystem/observation-category",
                "code": "vital-signs",
                "display": "Vital Signs"
            }]
        }],
        code={
            "coding": [{
                "system": "http://loinc.org",
                "code": "85354-9",
                "display": "Blood pressure panel"
            }]
        },
        subject={"reference": f"Patient/{patient_id}"},
        component=[
            {
                "code": {"coding": [{"code": "8480-6", "display": "Systolic Blood Pressure"}]},
                "valueQuantity": {"value": systolic, "unit": "mmHg", "system": "http://unitsofmeasure.org"}
            },
            {
                "code": {"coding": [{"code": "8462-4", "display": "Diastolic Blood Pressure"}]},
                "valueQuantity": {"value": diastolic, "unit": "mmHg", "system": "http://unitsofmeasure.org"}
            }
        ]
    )
    return observation.dict()
```

```
# Exemple d'utilisation
if __name__ == "__main__":
    patient_id = "patient-123"
    systolic = 120  # Exemple de pression systolique
    diastolic = 80  # Exemple de pression diastolique
    observation = generate_fhir_observation(patient_id, systolic,
diastolic)

    # Sauvegarder le message dans un fichier JSON
    with open("blood_pressure.json", "w") as f:
        json.dump(observation, f, indent=2)

    print("Message FHIR généré et sauvegardé dans blood_pressure.json")
```

2. Exemple de message FHIR : **blood_pressure.json**

```
{
  "resourceType": "Observation",
  "id": "a6c8b192-3271-4107-ac4e-6200c23570a4",
  "status": "final",
  "category": [
    {
      "coding": [
        {
          "system":
"http://terminology.hl7.org/CodeSystem/observation-category",
          "code": "vital-signs",
          "display": "Vital Signs"
        }
      ]
    }
  ],
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "85354-9",
        "display": "Blood pressure panel"
      }
    ]
  },
  "subject": {
    "reference": "Patient/patient-123"
  },
```

```
  "component": [
    {
      "code": {
        "coding": [
          {
            "code": "8480-6",
            "display": "Systolic Blood Pressure"
          }
        ]
      },
      "valueQuantity": {
        "value": 120.0,
        "unit": "mmHg",
        "system": "http://unitsofmeasure.org"
      }
    },
    {
      "code": {
        "coding": [
          {
            "code": "8462-4",
            "display": "Diastolic Blood Pressure"
          }
        ]
      },
      "valueQuantity": {
        "value": 80.0,
        "unit": "mmHg",
        "system": "http://unitsofmeasure.org"
      }
    }
  ]
}
```

3. Extrait de code : **producer.py**

```
import logging
logging.basicConfig(level=logging.DEBUG)  # Activer les logs détaillés

from kafka import KafkaProducer
import json
import time
import random
from faker import Faker
```

```
from fhir.resources.observation import Observation
from fhir.resources.patient import Patient


# Initialisation du générateur de données fictives
fake = Faker()


# Configuration du producteur Kafka
producer = KafkaProducer(
    bootstrap_servers="localhost:9093",
    value_serializer=lambda v: json.dumps(v,
default=str).encode('utf-8')
)


# Génération d'une observation FHIR de pression artérielle
def generate_fhir_observation():
    patient_id = fake.uuid4()


    # Création d'un patient FHIR
    patient = Patient.parse_obj({
        "resourceType": "Patient",
        "id": patient_id,
        "name": [{"use": "official", "family": fake.last_name(),
"given": [fake.first_name()]}],
        "gender": random.choice(["male", "female"]),
        "birthDate": fake.date_of_birth(minimum_age=20,
maximum_age=80).isoformat(),
    })


    # Génération des valeurs de pression artérielle
    systolic = random.randint(90, 180)
    diastolic = random.randint(60, 120)


    # Création de l'observation FHIR
    observation = Observation.parse_obj({
        "resourceType": "Observation",
        "id": fake.uuid4(),
        "status": "final",
```

```
        "category": [{"coding": [{"system":
"http://terminology.hl7.org/CodeSystem/observation-category", "code":
"vital-signs"}]}],
        "code": {"coding": [{"system": "http://loinc.org", "code":
"85354-9", "display": "Blood pressure panel"}]},
        "subject": {"reference": f"Patient/{patient_id}"},
        "component": [
            {
                "code": {"coding": [{"system": "http://loinc.org",
"code": "8480-6", "display": "Systolic blood pressure"}]},
                "valueQuantity": {"value": systolic, "unit": "mmHg",
"system": "http://unitsofmeasure.org", "code": "mm[Hg]"}
            },
            {
                "code": {"coding": [{"system": "http://loinc.org",
"code": "8462-4", "display": "Diastolic blood pressure"}]},
                "valueQuantity": {"value": diastolic, "unit": "mmHg",
"system": "http://unitsofmeasure.org", "code": "mm[Hg]"}
            }
        ]
    })


    return {
        "patient": patient.dict(),
        "observation": observation.dict()
    }


# Envoi des messages à Kafka
topic = "blood_pressure_readings"


print(f"🔵 Envoi des messages sur Kafka (Topic: {topic})...")


for _ in range(10):  # Envoi de 10 messages pour tester
    data = generate_fhir_observation()
    producer.send(topic, value=data)
    print(f"✅ Message envoyé : {data['observation']['id']} (Systolic:
{data['observation']['component'][0]['valueQuantity']['value']} mmHg,
Diastolic:
{data['observation']['component'][1]['valueQuantity']['value']} mmHg)")
```

```
    time.sleep(1)


print("🔴 Fin de l'envoi des messages.")
producer.close()
```

4. Extrait de code : **consumer.py**

```
import json
from kafka import KafkaConsumer
from elasticsearch import Elasticsearch
from datetime import datetime  # ✅ Importation du module datetime
import os

# 🔌 Configuration Kafka
KAFKA_BROKER = "localhost:9093"  # Vérifie si ce port est correct
TOPIC_NAME = "blood_pressure_readings"

# 📊 Configuration Elasticsearch
ELASTICSEARCH_HOST = "http://host.docker.internal:9200"
INDEX_NAME = "blood_pressure_anomalies_v2"

# 🔗 Connexion à Elasticsearch
es = Elasticsearch([ELASTICSEARCH_HOST])

# 📌 Vérifier et créer l'index si nécessaire
if not es.indices.exists(index=INDEX_NAME):
    es.indices.create(index=INDEX_NAME, body={})

# 🛠️ Connexion au Kafka Consumer
consumer = KafkaConsumer(
    TOPIC_NAME,
    bootstrap_servers=KAFKA_BROKER,
    auto_offset_reset='earliest',
    enable_auto_commit=False,
    group_id="blood_pressure_group",
    value_deserializer=lambda x: json.loads(x.decode('utf-8'))
)

print("🔵 En attente de messages...")

# 🔄 Lecture des messages Kafka
for message in consumer:
```

```
    data = message.value

    patient_id = data["patient"]["id"]

    systolic = None
    diastolic = None

    # 🔍 Extraction des valeurs de pression artérielle
    for component in data["observation"]["component"]:
        code = component["code"]["coding"][0]["code"]
        value = component["valueQuantity"]["value"]

        if code == "8480-6":  # Systolic Blood Pressure
            systolic = value
        elif code == "8462-4":  # Diastolic Blood Pressure
            diastolic = value

    # ✅ Ajout du timestamp correct
    if "effectiveDateTime" in data["observation"]:
        timestamp = data["observation"]["effectiveDateTime"]  # ✅ Timestamp fourni dans le message
    else:
        timestamp = datetime.utcnow().isoformat()  # ✅ Génération d'un timestamp UTC

    print(f"📥 Message reçu - Patient {patient_id} : {systolic}/{diastolic} mmHg à {timestamp}")

    # 🔍 Détection des anomalies
    anomaly_type = None
    if systolic and diastolic:
        if systolic > 140 or diastolic > 90:
            anomaly_type = "Hypertension"
        elif systolic < 90 or diastolic < 60:
            anomaly_type = "Hypotension"

    # 🚨 Si une anomalie est détectée, l'indexer dans Elasticsearch
    if anomaly_type:
        print(f"⚠️ Anomalie détectée pour le patient {patient_id} ({anomaly_type}) !")

        anomaly_data = {
            "patient_id": patient_id,
```

```
            "systolic_pressure": systolic,
            "diastolic_pressure": diastolic,
            "anomaly_type": anomaly_type,
            "timestamp": timestamp  # ✅ Timestamp correct
        }

        es.index(index=INDEX_NAME, document=anomaly_data)
        print(f"✅ Anomalie indexée dans Elasticsearch :
{anomaly_data}")
```

5. Extrait du code : **docker-compose.yml**

```
version: '3.8'


services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.4.0
    container_name: blood_pressure_fhir-zookeeper-1
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      KAFKA_OPTS:
"-Dzookeeper.4lw.commands.whitelist=ruok,stat,conf,isro,mntr"
    volumes:
      - ./zookeeper.properties:/etc/kafka/zookeeper.properties


  kafka:
    image: confluentinc/cp-kafka:7.4.0
    container_name: blood_pressure_fhir-kafka-1
    ports:
      - "9092:9092"
      - "9093:9093"  # Ajout du port pour la connexion depuis l'hôte
local
    environment:
      KAFKA_ZOOKEEPER_CONNECT: blood_pressure_fhir-zookeeper-1:2181
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://blood_pressure_fhir-kafka-1:9092,PLAINTEXT_HOST://localhost
:9093
```

```
      KAFKA_LISTENERS:
PLAINTEXT://0.0.0.0:9092,PLAINTEXT_HOST://0.0.0.0:9093
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    depends_on:
      - zookeeper


  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.9.0
    container_name: blood_pressure_fhir-elasticsearch-1
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false  # 🔴 Désactive HTTPS et
l'authentification
    ports:
      - "9200:9200"


  kibana:
    image: docker.elastic.co/kibana/kibana:8.9.0
    container_name: blood_pressure_fhir-kibana-1
    depends_on:
      - elasticsearch
    ports:
      - "5601:5601"
    environment:
      -
ELASTICSEARCH_HOSTS=http://blood_pressure_fhir-elasticsearch-1:9200
```

6. Extrait du code : **zookeeper.properties**

```
dataDir=/var/lib/zookeeper/data
dataLogDir=/var/lib/zookeeper/log

clientPort=2181
```