

# **Proyecto Final: Sistema de Big Data**

**CE Inteligencia Artificial y Big Data  
Big Data Aplicado  
2024/2025**

**Daniel Marín López**

# Índice

<b>Introducción: El Big Data en la Sanidad.....</b>	<b>4</b>
<b>1. Propuesta: Sistema de Big Data.....</b>	<b>5</b>
1.1. Objetivo.....	5
1.2. Origen de los datos.....	5
1.3. ETL y Procesamiento.....	5
1.4. Visualización y BI.....	6
1.5. Infraestructura.....	6
1.6. Conocimiento que se espera obtener.....	6
1.7. Impacto del proyecto.....	6
<b>2. Obtención de los datos: Planteamiento de los datos.....</b>	<b>7</b>
2.1. Base de datos del proyecto.....	7
2.2. Plantilla de los datos.....	9
2.3. Código para crear los datos: Faker.....	10
<b>3. Ingesta de datos streaming: Kafka + Spark.....</b>	<b>12</b>
3.1. Planteamiento.....	12
3.2. Controller y brokers.....	12
3.3. Creación del topic y consumidor.....	15
3.4. Pasos para la ingestá streaming.....	17
1. Lanzamos el consumidor de Spark.....	17
2. Lanzamos nuestro generador de datos.....	17
3. Comprobamos que se consumen los datos con la API Consumer de Kafka.....	17
3.5. ETL de los datos streaming.....	18
3.5.1. Transformación.....	18
3.5.2. Carga.....	19
3.6. Ingesta a PostgreSQL y conexión a Grafana.....	21
3.6.1 Creación de la base de datos.....	21
3.6.2 Sistema Spark - Kafka Connect - PostgreSQL.....	22
3.6.3 Conectar PostgreSQL a Grafana.....	26
<b>4. Ingesta de datos batch: MySQL + Kafka + HDFS.....</b>	<b>27</b>
4.1. ¿Cómo abordar el problema?.....	27
4.2. Creación de las tablas de la base de datos.....	27
4.3. Ingesta de datos en batch.....	28
4.4. Creación de los Workers.....	29
4.5. Creación de los topics.....	29
4.6. Conectores de MySQL y HDFS.....	30
4.6.1. Conectores de las analíticas.....	30
4.6.2. Conectores de las antibióticos.....	30
4.6.3. Pasos para lanzar los conectores.....	30
4.7. Obtención de los datos para la visualización.....	33

4.8. Conectar MySQL a PostgreSQL para métricas combinadas.....	34
<b>5. Visualización.....</b>	<b>35</b>
5.1. Visualización en streaming (Grafana).....	35
5.1.1 Página 1: Resumen general de constantes vitales.....	36
5.1.2 Página 2: Consultas de los datos.....	36
5.1.3 Página 3: Métricas por pacientes.....	37
5.2. Visualización en batch (Power BI).....	37
5.2.1 Página 1: Resumen general de analíticas.....	38
5.2.2 Página 2: Resumen general de antibióticos.....	39
5.2.3 Página 3: Resistencia de la bacteria a los antibióticos.....	40
5.2.4 Página 4: Métricas de pacientes.....	41
5.2.5 Página 5: Tabla detallada de las analíticas.....	42
5.3. Visualización streaming + batch.....	42
5.3.1 Página 4: Constantes vitales por bacteria.....	42
5.3.2 Página 5: Consultas mixtas.....	43
<b>6. Business Intelligence.....</b>	<b>43</b>
6.1. BI de datos en streaming.....	43
6.2.1 Análisis General de Constantes Vitales.....	43
6.2.2 Análisis de las consultas.....	44
6.2.3 Análisis de constantes de un paciente (p006).....	45
6.2. BI de datos en batch.....	46
6.2.1 Análisis General de Antibiogramas y Bacterias.....	46
6.2.2 Relación entre Antibióticos, Bacterias y Tipo de Muestra.....	46
6.2.3 Niveles de Resistencia a Antibióticos.....	47
6.2.4 Información del Paciente y Procesamiento de Muestras.....	47
6.2.5 Oportunidades de Mejora y Estrategias.....	48
6.3. BI de datos en streaming + batch.....	48
6.3.1 Constantes vitales según bacterias.....	48
6.3.2 Algunas consultas interesantes.....	49
<b>7. Despliegue Completo.....</b>	<b>51</b>
7.1 Inicio del sistema.....	51
7.2 Inicio de Kafka.....	51
7.3 Ingesta en streaming.....	52
7.4 Ingesta en batch.....	53
<b>8. Posibles Mejoras.....</b>	<b>54</b>
<b>9. Conclusiones Finales.....</b>	<b>55</b>
<b>10. Enlaces de interés.....</b>	<b>55</b>

## Introducción: El Big Data en la Sanidad

En los últimos años, la aplicación de Big Data en el ámbito sanitario ha transformado la forma en que se gestionan y analizan los datos clínicos. Gracias a tecnologías como Kafka, Spark y Hadoop, los hospitales pueden hoy en día procesar grandes volúmenes de información para detectar patrones, predecir enfermedades, optimizar recursos y tomar decisiones médicas fundamentadas. Especialmente en el contexto hospitalario, el análisis continuo de datos permite identificar riesgos en tiempo real y mejorar significativamente la calidad asistencial. Una de las aplicaciones más críticas de estas tecnologías es la detección temprana de **infecciones nosocomiales**, especialmente aquellas causadas por **bacterias multirresistentes**.

Las infecciones nosocomiales son aquellas que los pacientes contraen durante su estancia en hospitales. Estas infecciones, en particular las provocadas por patógenos resistentes a múltiples antibióticos, representan un grave riesgo para la seguridad del paciente. Su detección temprana es esencial, ya que los brotes —agrupaciones de casos similares en tiempo y espacio— pueden expandirse rápidamente si no se actúa con **inmediatez**. Identificar estos brotes en las primeras 48-72 horas es clave para activar protocolos de aislamiento, modificar tratamientos y contener la propagación.

El sistema Big Data propuesto para este proyecto permite integrar y procesar dos tipos principales de datos:

- **En streaming (tiempo real)**: parámetros fisiológicos del paciente (frecuencia cardíaca, presión arterial, temperatura, saturación de oxígeno, glucosa, etc.) generados de forma sintética o simulados como si provinieran de sensores hospitalarios. Estos datos se ingieren mediante **Kafka**, se procesan con **Spark Streaming** y se visualizan en tiempo real mediante **Grafana**, permitiendo detectar rápidamente situaciones críticas como descompensaciones o signos de infección.
- **En batch (procesamiento periódico)**: resultados de microbiología, como la detección de bacterias en muestras clínicas (orina, sangre, esputo, etc.) y sus respectivos perfiles de resistencia antibiótica. Estos datos se generan en laboratorio (simulados para este proyecto), y se procesan de forma periódica con **Spark** para su análisis profundo, almacenándose en **HDFS** y consultándose a través de **Power BI**.

La combinación de ambos flujos de datos permite realizar un análisis completo del estado clínico del paciente y de los posibles brotes infecciosos. Por ejemplo, si varios pacientes de una misma unidad presentan simultáneamente fiebre elevada, frecuencia cardíaca acelerada y cultivos positivos para la misma bacteria resistente, el sistema puede generar automáticamente una **alerta de brote nosocomial**.

El análisis de brotes se estructura en distintos pasos: primero, se filtran los casos positivos con bacterias multirresistentes; después, se agrupan por microorganismo, ubicación (unidad hospitalaria) y fecha de detección. Si se identifican tres o más pacientes con la misma bacteria en una misma unidad dentro de un periodo de 72 horas, el sistema activa una alerta. Adicionalmente, se comparan los perfiles de resistencia para confirmar si los casos están relacionados epidemiológicamente o si son infecciones independientes.

A nivel técnico, Spark se utiliza tanto en su versión *streaming* como *batch*. En streaming, los datos se analizan en ventanas de tiempo para generar alertas inmediatas. En batch, se realizan análisis más complejos como la evolución temporal de los brotes, la frecuencia de aparición de bacterias resistentes o la evaluación de la efectividad de los tratamientos administrados. Los datos transformados se almacenan en **HDFS** en formato Parquet o JSON y se accede a ellos desde **Power BI** para crear paneles dinámicos y visualizaciones interactivas. Grafana complementa este análisis mostrando métricas en tiempo real, como la aparición de signos vitales críticos o la distribución geográfica de los pacientes.

## 1. Propuesta: Sistema de Big Data

Aquí se detalla un poco la propuesta de este proyecto para nuestro sistema de Big Data.

### 1.1. Objetivo

Desarrollar una plataforma de Big Data capaz de:

1. Signos vitales y parámetros clínicos de pacientes (streaming).
2. Resultados de análisis microbiológicos (batch).
3. Detección en tiempo real de **casos clínicos de riesgo** (hipertensión, hipoglucemia, infecciones multirresistentes).
4. Análisis histórico para evaluación de tratamientos y evolución epidemiológica.

### 1.2. Origen de los datos

Fuentes de datos (pueden ser reales o sintéticos combinados):

- Datos clínicos anonimizados (batch) de pacientes con infecciones hospitalarias.
- Datos en tiempo real (streaming) desde sensores o registros de laboratorios clínicos.
- Datos de antibiogramas de laboratorios (resistencia a antibióticos).
- Datos de ubicación hospitalaria (habitaciones, plantas, quirófanos).
- Registros de prescripción de antibióticos y procedimientos médicos.
- Datos del clima local del hospital (influye en ciertos patógenos).

Formatos:

- JSON
- CSV
- Avro (para streaming)

### 1.3. ETL y Procesamiento

**Ingesta:**

- ~~Kafka para datos en tiempo real desde laboratorios y sensores.~~

**Procesamiento (SPARK):**

- ~~Limpieza y normalización de datos.~~
- ~~Análisis de patrones temporales y espaciales de brotes.~~
- ~~Machine learning con MLlib para clasificación de riesgo de infección. (Opcional)~~
- ~~Agrupación de bacterias según resistencia para detección de cepas comunes. (Opcional)~~

**Almacenamiento:**

- ~~HDFS Hadoop para almacenar tanto los datos en batch como los datos en streaming procesados~~

**Justificación:**

- ~~Permite la integración de múltiples orígenes y tipos de datos, y la aplicación de modelos predictivos y descriptivos.~~

## 1.4. Visualización y BI

La visualización se limitará a:

- Power BI.** Conectado a archivos Parquet/CSV/JSON generados por Spark, para mostrar dashboards de resistencia, cantidad de casos, evolución temporal.
- Grafana (Opcional).** Conectado a Prometheus o a un exportador de métricas Spark, para ver rendimiento y métricas del sistema Big Data en sí.

## 1.5. Infraestructura

La infraestructura será la siguiente:

- **1 nodo master + 3 workers** para:
  - HDFS + YARN (Hadoop)
  - **Spark (modo cluster con 3 nodos ejecutores)**
  - **Kafka con Kraft (1 controller, 1-2 brokers, 2 workers)**
  - **Power BI** se conecta desde el host (no en clúster).
  - **Grafana** se ejecuta en el master node. (Opcional)

## 1.6. Conocimiento que se espera obtener

El conocimiento que se espera extraer es el siguiente:

- **Tendencias crónicas:** Visualizar evolución de enfermedades comunes por paciente.
- **Detección temprana:** Actuar ante patrones que anticipen crisis clínicas.
- **Brote bacteriano:** Detectar clústeres de resistencia y activar protocolos.
- **Evaluación de tratamientos:** Cruzar eficacia con parámetros reales.

## 1.7. Impacto del proyecto

El impacto que puede un proyecto de este calibre con mejores recursos sería el siguiente:

- Fuerte impacto sanitario.
- Justificación clara del procesamiento y su utilidad.
- Escenario híbrido (batch + streaming).
- Alto potencial para aplicar BI y modelos predictivos.
- Buena oportunidad de investigación adicional.

## 2. Obtención de los datos: Planteamiento de los datos

### 2.1. Base de datos del proyecto

Para nuestro proyecto necesitaremos una base de datos, la base de datos la haremos en MySQL y la llamaremos “MedicineDB”.

```
mysql> CREATE DATABASE MedicineDB;
Query OK, 1 row affected (0,06 sec)

mysql> SHOW DATABASES
    -> ;
+-----+
| Database |
+-----+
| MedicineDB |
| employees |
| information_schema |
| metastore |
| mysql |
| performance_schema |
| sys |
+-----+
7 rows in set (0,02 sec)
```

Dentro de la base de datos crearemos una tabla de pacientes que tendrán un ID, nombre, apellidos, edad, sexo y condiciones conocidas.

```
mysql> CREATE TABLE pacientes (
    -> patient_id VARCHAR(10) PRIMARY KEY,
    -> nombre VARCHAR(100),
    -> apellidos VARCHAR(500),
    -> edad INT,
    -> sex CHAR(1),
    -> condiciones_conocidas TEXT
    -> );
Query OK, 0 rows affected (0,23 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_MedicineDB |
+-----+
| pacientes |
+-----+
1 row in set (0,01 sec)
```

Y dentro de esta tabla es donde tendremos guardados una serie de pacientes que usaremos para generar nuestro datos tanto en streaming como en batch y que estén relacionados entre sí. Para ello usaremos este programa para insertar los pacientes en nuestra base de datos:

```
● ● ●

import mysql.connector
from faker import Faker
import random

# Configuración de conexión
conn = mysql.connector.connect(
    host='localhost',
    user='tu_usuario',
    password='tu_contraseña',
    database='tu_base_de_datos'
)

cursor = conn.cursor()
fake = Faker('es_ES')
random.seed(42)

condiciones = [
    "Hipertensión", "Diabetes tipo 2", "Insuficiencia renal",
    "Asma", "Colesterol alto", "EPOC", "Cáncer", "Obesidad"
]

def generar_condiciones():
    num = random.randint(0, 2)
    return ", ".join(random.sample(condiciones, num)) if num > 0 else ""

# Generar e insertar 100 pacientes
for i in range(100):
    patient_id = f"p{i+1:03d}"
    nombre = fake.first_name()
    apellidos = fake.last_name() + " " + fake.last_name()
    edad = random.randint(18, 90)
    sex = random.choice(["M", "F"])
    condiciones_conocidas = generar_condiciones()

    query = """
        INSERT INTO pacientes (patient_id, nombre, apellidos, edad, sex, condiciones_conocidas)
        VALUES (%s, %s, %s, %s, %s, %s)
    """
    values = (patient_id, nombre, apellidos, edad, sex, condiciones_conocidas)
    cursor.execute(query, values)

# Confirmar y cerrar
conn.commit()
cursor.close()
conn.close()

print("Pacientes insertados correctamente.")
```

Lo que generará pacientes falsos (datos sintéticos) que usaremos tanto para generar tanto nuestros datos en streaming como en batch.

p091	Lucio	Gálvez	Lobato	27	M	
Hipertensión,	Cáncer					
p092	Ascensión	Durán	Vicente	65	M	
Insuficiencia renal						
p093	Daniela	Anglada	Balaguer	23	F	
Hipertensión						
p094	Samuel	Murillo	Esparza	63	M	
Asma, EPOC						
p095	Silvestre	Badía	Carbó	31	F	
Cáncer, Colesterol alto						
p096	Carlota	Macias	Palomar	37	M	
p097	Carmela	Álvarez	Bonet	40	F	
p098	Juliana	Carbonell	Pozuelo	40	F	
Asma						
p099	Aristides	Boada	Alemán	52	M	
Diabetes tipo 2, Asma						
p100	Paula	Nadal	Villalonga	22	F	
+-----+-----+-----+-----+-----+-----+-----+						
100 rows in set (0,00 sec)						
mysql>						

## 2.2. Plantilla de los datos

A la hora de obtener un conjunto de datos acorde al problema que queremos abordar, es una propuesta muy difícil debido a que el campo de la microbiología es bastante amplio y se necesitaría de un estudio completo con ayuda de expertos a ser posible. Para este proyecto se han creado dos tipos de JSON que reflejan por un lado los datos que se monitorizan las constantes vitales (streaming) y los otros reflejan las analíticas que se realizan diariamente (batch).

```
● ● ●
{
  "timestamp": "2025-04-30T14:00:00Z",
  "patient_id": "12345",
  "heart_rate": 110,
  "blood_pressure": "155/100",
  "temperature": 38.3,
  "oxygen_saturation": 92,
  "blood_glucose": 160,
  "unit": "UCI"
}

● ● ●
{
  "patient_id": "12345",
  "sample_date": "2025-04-28",
  "sample_type": "esputo",
  "bacteria_detected": "Klebsiella pneumoniae",
  "antibiotic_resistance": {
    "ceftriaxona": "resistente",
    "imipenem": "intermedio",
    "colistina": "sensible"
  }
}
```

## 2.3. Código para crear los datos: Faker

La mejor forma de realizar la ingestión de datos tanto en streaming como en batch ya sea en Kafka o en Spark es usar Faker. Para los datos en streaming tenemos el siguiente código:

```
# streaming_generator.py
from faker import Faker
import random
import json
from datetime import datetime
from kafka import KafkaProducer
import time
import mysql.connector

fake = Faker()

# Conexión a MySQL para obtener patient_ids
conn = mysql.connector.connect(
    host="localhost",
    user="tu_usuario",
    password="tu_contraseña",
    database="tu_base_de_datos"
)
cursor = conn.cursor()
cursor.execute("SELECT patient_id FROM pacientes")
patient_ids = [row[0] for row in cursor.fetchall()]
cursor.close()
conn.close()

# Configuración del productor Kafka
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

# Función para crear datos sintéticos
def generate_vital_data():
    return {
        "timestamp": datetime.utcnow().isoformat(),
        "patient_id": random.choice(patient_ids),
        "heart_rate": random.randint(60, 140),
        "blood_pressure": f"{random.randint(100, 180)}/{random.randint(60, 120)}",
        "temperature": round(random.uniform(36.0, 40.5), 1),
        "oxygen_saturation": random.randint(85, 100),
        "blood_glucose": random.randint(70, 200),
        "unit": random.choice(["UCI", "Planta", "Urgencias"])
    }

# Bucle de envío continuo
while True:
    message = generate_vital_data()
    producer.send('vitales_pacientes', value=message)
    print("Enviado:", message)
    time.sleep(1.5) # Simula frecuencia de monitorización
```

Mientras que para generar los datos en batch tenemos este código:

```

● ● ●

# batch_generator.py
from faker import Faker
import random
import json
from datetime import datetime, timedelta
import mysql.connector
import subprocess
import os

fake = Faker('es_ES')

# Conexión a MySQL para obtener patient_ids
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="1234",
    database="MedicineDB"
)
cursor = conn.cursor()
cursor.execute("SELECT patient_id FROM pacientes")
patient_ids = [row[0] for row in cursor.fetchall()]

# Obtener el último analitica_id
cursor.execute("SELECT analitica_id FROM analiticas ORDER BY CAST(SUBSTRING(analitica_id, 3) AS UNSIGNED) DESC LIMIT 1;")
last_an = cursor.fetchone()
an_counter = int(last_an[0][2:]) if last_an else 0

# Obtener el último res_id
cursor.execute("SELECT res_id FROM antibiotic_res ORDER BY CAST(SUBSTRING(res_id, 4) AS UNSIGNED) DESC LIMIT 1;")
last_res = cursor.fetchone()
res_counter = int(last_res[0][3:]) if last_res else 0

# Datos posibles
bacterias = ["Klebsiella pneumoniae", "Escherichia coli", "Pseudomonas aeruginosa", "Staphylococcus aureus"]
antibioticos = ["ceftriaxona", "imipenem", "colistina", "ciprofloxacina"]

resistencias = ["resistente", "intermedio", "sensible"]
muestras = ["esputo", "sangre", "orina", "exudado nasal"]

# Generar lote de datos
resultados = []
for _ in range(200): # cantidad de muestras generadas
    an_counter += 1
    analitica_id = f'an{an_counter:03d}'
    patient_id = random.choice(patient_ids)
    sample_date = (datetime.today() - timedelta(days=random.randint(0, 10))).date().isoformat()
    sample_type = random.choice(muestras)
    bacteria_detected = random.choice(bacterias)

    query_an = """
        INSERT INTO analiticas (analitica_id, patient_id, sample_date, sample_type, bacteria_detected)
        VALUES (%s, %s, %s, %s, %s)
    """
    values_an = (analitica_id, patient_id, sample_date, sample_type, bacteria_detected)
    cursor.execute(query_an, values_an)

    for ab in antibioticos:
        res_counter += 1
        res_id = f'res{res_counter:03d}'
        antibiotic_name = random.choice(antibioticos)
        resistance_level = random.choice(resistencias)

        query_res = """
            INSERT INTO antibiotic_res (res_id, analitica_id, antibiotic_name, resistance_level)
            VALUES (%s, %s, %s, %s)
        """
        values_res = (res_id, analitica_id, antibiotic_name, resistance_level)
        cursor.execute(query_res, values_res)

    # Confirmar y cerrar
    conn.commit()
    cursor.close()
    conn.close()

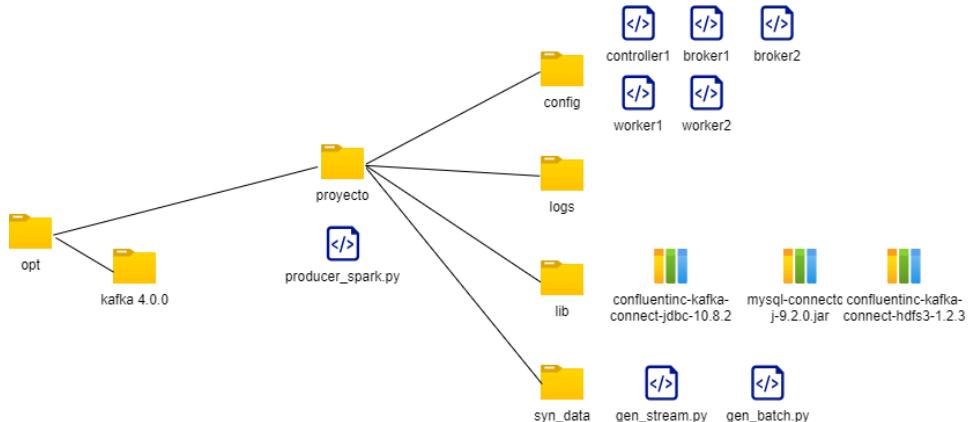
print("Analiticas insertadas correctamente.")

```

### 3. Ingesta de datos streaming: Kafka + Spark

#### 3.1. Planteamiento

El escenario de momento se presenta de esta manera:



Donde tenemos una carpeta para el proyecto, en esta hay 4 carpetas:

- **config:** Archivos de configuración de Kafka
- **logs:** Archivos de logs de Kafka
- **lib:** Plugins de Kafka
- **syn\_data:** Scripts para generar datos sintéticos

Y de momento solo tenemos el producer de spark.

#### 3.2. Controller y brokers

Como hemos indicado anteriormente, usaremos 2 Workers debido a la falta de recursos bastante notable. Nuestro controller y brokers tendrán esta estructura:

```

# Server Basics
process.roles=controller
node.id=1
controller.quorum.bootstrap.servers=localhost:9093
# Socket Server Settings
listeners=CONTROLLER://:9093
advertised.listeners=CONTROLLER://localhost:9093
controller.listener.names=CONTROLLER
# Log Basics
log.dirs=/opt/proyecto/logs/controller1
  
```

```

# Server Basics
process.roles=broker
node.id=2
controller.quorum.bootstrap.servers=localhost:9093
# Socket Server Settings
listeners=PLAINTEXT://localhost:9094
advertised.listeners=PLAINTEXT://localhost:9094
# Log Basics
log.dirs=/opt/proyecto/logs/broker1
  
```

```

# Server Basics
process.roles=broker
node.id=3
controller.quorum.bootstrap.servers=localhost:9093
# Socket Server Settings
listeners=PLAINTEXT://localhost:9095
advertised.listeners=PLAINTEXT://localhost:9095
# Log Basics
log.dirs=/opt/proyecto/logs/broker2
  
```

Para iniciar Kafka haremos los siguientes comandos:

```
● ● ●  
#Genera un cluster UUID  
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"  
echo $KAFKA_CLUSTER_ID  
  
#Formateamos los directorios de log  
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID --standalone -c /opt/proyecto/config/controller1.properties  
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c /opt/proyecto/config/broker1.properties  
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c /opt/proyecto/config/broker2.properties
```

```
hadoop@master:/opt/kafka_2.13-4.0.0$ KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"  
echo $KAFKA_CLUSTER_ID  
23JZj24TTp0-wsmupDfcXQ  
hadoop@master:/opt/kafka_2.13-4.0.0$ bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID --standalone -c /opt/proyecto/config/controller1.properties  
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c /opt/proyecto/config/broker1.properties  
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c /opt/proyecto/config/broker2.properties  
Formatting dynamic metadata voter directory /opt/proyecto/logs/controller1 with metadata.version 4.0-IV3.  
Formatting metadata directory /opt/proyecto/logs/broker1 with metadata.version 4.0-IV3.  
Formatting metadata directory /opt/proyecto/logs/broker2 with metadata.version 4.0-IV3.
```

Como es la primera vez, hay que generar un ID para el clúster y formatear el controller y los brokers. El ID se guardará en un archivo y lo usaremos directamente en vez de generararlo de nuevo.

Ahora iniciamos el controller y los brokers cada uno en una terminal distinta:

```
● ● ●
```

```
bin/kafka-server-start.sh /opt/proyecto/config/controller1.properties  
bin/kafka-server-start.sh /opt/proyecto/config/broker1.properties  
bin/kafka-server-start.sh /opt/proyecto/config/broker2.properties
```

```
Publisher with a snapshot at offset 7 (org.apache.kafka.image.loader.MetadataLoader)
[2025-05-09 16:37:02,683] INFO [MetadataLoader id=1] InitializeNewPublishers: initializing AclPublisher controller id=1 with a snapshot at offset 7 (org.apache.kafka.image.loader.MetadataLoader)
[2025-05-09 16:37:02,705] INFO [ControllerRegistrationManager id=1 incarnation=SMNyK81HSm2UbflKkjRQ4g] sendControllerRegistration: attempting to send ControllerRegistrationRequestData(controllerId=1, incarnationId=SMNyK81HSm2UbflKkjRQ4g, zkMigrationReady=false, listeners=[Listener(name='CONTROLLER', host='localhost', port=9093, securityProtocol=0)], features=[Feature(name='group.version', minSupportedVersion=0, maxSupportedVersion=1), Feature(name='transaction.version', minSupportedVersion=0, maxSupportedVersion=2), Feature(name='eligible.leader.replicas.version', minSupportedVersion=0, maxSupportedVersion=1), Feature(name='kraft.version', minSupportedVersion=0, maxSupportedVersion=1), Feature(name='metadata.version', minSupportedVersion=7, maxSupportedVersion=25)]) (kafka.server.ControllerRegistrationManager)
[2025-05-09 16:37:03,588] INFO [ControllerRegistrationManager id=1 incarnation=SMNyK81HSm2UbflKkjRQ4g] Our registration has been persisted to the metadata log. (kafka.server.ControllerRegistrationManager)
[2025-05-09 16:37:03,621] INFO [ControllerRegistrationManager id=1 incarnation=SMNyK81HSm2UbflKkjRQ4g] RegistrationResponseHandler: controller acknowledged ControllerRegistrationRequest. (kafka.server.ControllerRegistrationManager)
```

```
[2025-05-09 16:38:07,708] INFO [BrokerServer id=2] Finished waiting for all of the SocketServer Acceptors to be started (kafka.server.BrokerServer)
[2025-05-09 16:38:07,729] INFO [BrokerServer id=2] Transition from STARTING to STARTED (kafka.server.BrokerServer)
[2025-05-09 16:38:07,754] INFO Kafka version: 4.0.0 (org.apache.kafka.common.utils.AppInfoParser)
[2025-05-09 16:38:07,756] INFO Kafka commitId: 985bc99521d22bb (org.apache.kafka.common.utils.AppInfoParser)
[2025-05-09 16:38:07,757] INFO Kafka startTimeMs: 1746808687730 (org.apache.kafka.common.utils.AppInfoParser)
[2025-05-09 16:38:07,780] INFO [KafkaRaftServer nodeId=2] Kafka Server started (kafka.server.KafkaRaftServer)
```

```
[2025-05-09 16:39:12,224] INFO [BrokerServer id=3] Finished waiting for all of the SocketServer Acceptors to be started (kafka.server.BrokerServer)
[2025-05-09 16:39:12,226] INFO [BrokerServer id=3] Transition from STARTING to STARTED (kafka.server.BrokerServer)
[2025-05-09 16:39:12,228] INFO Kafka version: 4.0.0 (org.apache.kafka.common.utils.AppInfoParser)
[2025-05-09 16:39:12,232] INFO Kafka commitId: 985bc99521d22bb (org.apache.kafka.common.utils.AppInfoParser)
[2025-05-09 16:39:12,232] INFO Kafka startTimeMs: 1746808752226 (org.apache.kafka.common.utils.AppInfoParser)
[2025-05-09 16:39:12,253] INFO [KafkaRaftServer nodeId=3] Kafka Server started (kafka.server.KafkaRaftServer)
```

Comprobamos que se han levantado con éxito, por si acaso haremos este comando para ver si están levantados correctamente:



```
hadoop@master:~$ sudo ss -tunelp | grep 1100*
[sudo] password for hadoop:
tcp  LISTEN  0      3                               *:11003          *:*
    users:(("java",pid=10140,fd=115)) uid:1000  ino:43939 sk:8 cgroup:/user.slice/user-1000.slice/session-17.scope v6only:0 <->
tcp  LISTEN  0      3                               *:11002          *:*
    users:(("java",pid=9689,fd=115)) uid:1000  ino:44072 sk:9 cgroup:/user.slice/user-1000.slice/session-16.scope v6only:0 <->
tcp  LISTEN  0      3                               *:11001          *:*
    users:(("java",pid=9270,fd=115)) uid:1000  ino:43487 sk:a cgroup:/user.slice/user-1000.slice/session-15.scope v6only:0 <->
```

Vemos que no hay ningún problema.

### 3.3. Creación del topic y consumidor

Lo siguiente será diseñar nuestro topic, que tendrá 2 particiones y un factor de replicación de 2 también.

```
bin/kafka-topics.sh \
--create \
--bootstrap-server localhost:9094 \
--replication-factor 2 \
--partitions 2 \
--topic vitales_pacientes
```

Una vez creado, revisamos que se ha creado.

```
bin/kafka-topics.sh --describe --topic vitales_pacientes --bootstrap-server localhost:9094
```

```
hadoop@master:/opt/kafka_2.13-4.0.0$ bin/kafka-topics.sh \
--create \
--bootstrap-server localhost:9094 \
--replication-factor 2 \
--partitions 2 \
--topic vitales_pacientes
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore('_') could collide. To avoid issues it is best to use either, but not both.
Created topic vitales_pacientes.
hadoop@master:/opt/kafka_2.13-4.0.0$ bin/kafka-topics.sh --describe \
--topic vitales_pacientes --bootstrap-server localhost:9094
Topic: vitales_pacientes          TopicId: ghJS3x3-SHKIqaqlAyv7wA Par
titionCount: 2  ReplicationFactor: 2  Configs: segment.bytes=1073
741824
      Topic: vitales_pacientes      Partition: 0      Leader: 2 R
Replicas: 2,3    Isr: 2,3        Elr:      LastKnownElr:
      Topic: vitales_pacientes      Partition: 1      Leader: 3 R
Replicas: 3,2    Isr: 3,2        Elr:      LastKnownElr:
```

Lo siguiente será diseñar nuestro consumidor que será en Spark. Para ello, tenemos el archivo `streaming_spark.py` que será nuestro consumidor de Spark:

```
# spark_streaming_ingest.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import *

# 1. Crear sesión Spark
spark = SparkSession.builder \
    .appName("KafkaToSparkStreaming") \
    .getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

# 2. Definir el esquema del JSON entrante
schema = StructType([
    StructField("timestamp", StringType(), True),
    StructField("patient_id", StringType(), True),
    StructField("heart_rate", IntegerType(), True),
    StructField("blood_pressure", StringType(), True),
    StructField("temperature", FloatType(), True),
    StructField("oxygen_saturation", IntegerType(), True),
    StructField("blood_glucose", IntegerType(), True),
    StructField("unit", StringType(), True),
])
]

# 3. Leer desde Kafka
df_kafka = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "192.168.18.8:9094") \
    .option("subscribe", "vitales_pacientes") \
    .option("startingOffsets", "earliest") \
    .load()

# 4. Convertir el value (bytes) a JSON
df_parsed = df_kafka.selectExpr("CAST(value AS STRING) as json_str") \
    .select(from_json(col("json_str"), schema).alias("data")) \
    .select("data.*")

# 5. Procesamiento opcional (por ejemplo: filtrar signos críticos)
df_filtered = df_parsed.filter(col("heart_rate") > 100)

# 6. Salida en consola (para desarrollo)
query = df_filtered.writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", False) \
    .start()

query.awaitTermination()
```

### 3.4. Pasos para la ingesta streaming

Los pasos para la ingesta de Kafka + Spark son los siguientes:

#### 1. Lanzamos el consumidor de Spark



```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.4 --master spark://192.168.18.8:7077 /opt/proyecto/streaming_spark.py
```

#### 2. Lanzamos nuestro generador de datos



```
python3 /opt/proyecto/streaming_generator.py
```

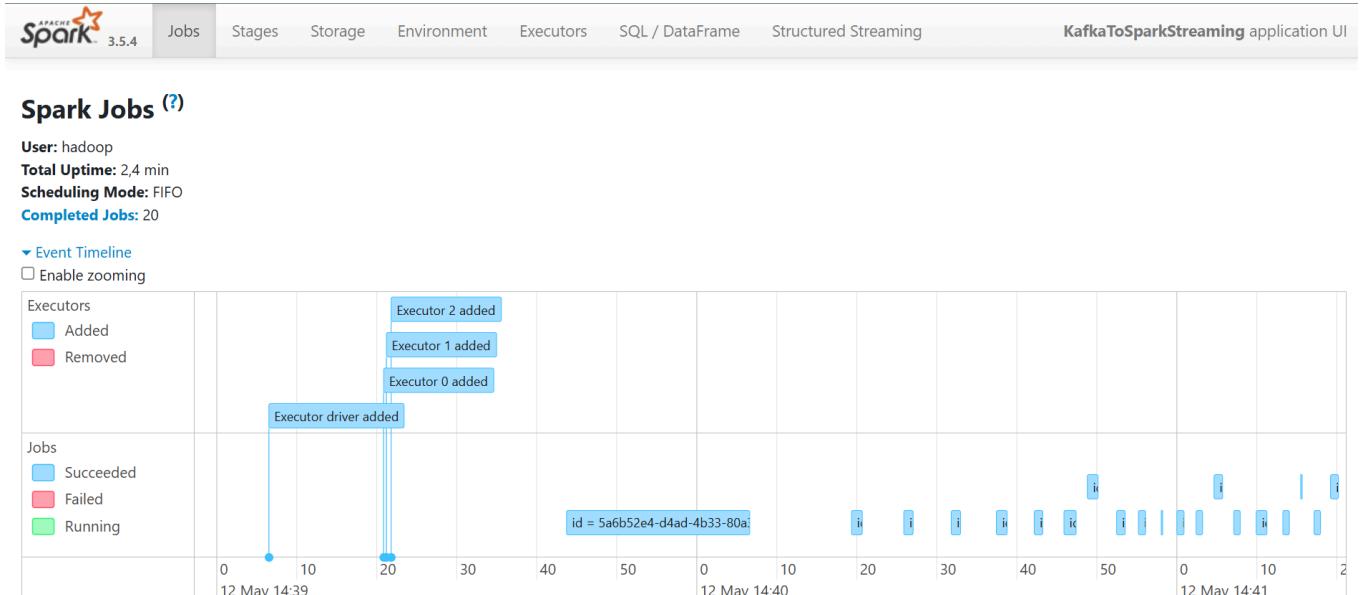
#### 3. Comprobamos que se consumen los datos con la API Consumer de Kafka



```
bin/kafka-console-consumer.sh --topic player-position --from-beginning --bootstrap-server 192.168.18.8:9094
```

```
G
25/05/12 14:39:06 INFO StandaloneAppClient$ClientEndpoint:
  Executor updated: app-20250512143906-0001/2 is now RUNNING
G
25/05/12 14:39:08 INFO StandaloneSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached
minRegisteredResourcesRatio: 0.0
-----
Batch: 0
-----
+-----+-----+-----+-----+
-----+-----+-----+-----+
--+ |timestamp |patient_id|heart_rate|blood_pr
essure|temperature|oxygen_saturation|blood_glucose|unit
|-----+-----+-----+-----+
-----+-----+-----+-----+
--+ |2025-05-10T18:54:16.861182|p074 |112 |166/74
|36.0 |90 |73 |Urgenci
as|
|2025-05-10T18:54:22.013889|p100 |124 |140/78
|36.6 |88 |171 |UCI
|
|2025-05-10T18:54:23.537993|p026 |133 |126/106
|38.5 |98 |138 |UCI
|
|2025-05-10T18:54:28.170462|p086 |133 |128/60
|36.4 |99 |77 |Urgenci
```

```
Enviado: {'timestamp': '2025-05-12T14:41:33.021032', 'patient_id': 'p056', 'heart_rate': 105, 'blood_pressure': '113/88', 'temperature': 36.2, 'oxygen_saturation': 97, 'blood_glucose': 160, 'unit': 'Urgencias'}
Enviado: {'timestamp': '2025-05-12T14:41:34.529466', 'patient_id': 'p007', 'heart_rate': 116, 'blood_pressure': '150/106', 'temperature': 40.1, 'oxygen_saturation': 91, 'blood_glucose': 151, 'unit': 'Urgencias'}
Enviado: {'timestamp': '2025-05-12T14:41:36.047871', 'patient_id': 'p003', 'heart_rate': 136, 'blood_pressure': '117/68', 'temperature': 36.2, 'oxygen_saturation': 90, 'blood_glucose': 74, 'unit': 'UCI'}
Enviado: {'timestamp': '2025-05-12T14:41:37.719680', 'patient_id': 'p039', 'heart_rate': 82, 'blood_pressure': '110/65', 'temperature': 39.6, 'oxygen_saturation': 94, 'blood_glucose': 78, 'unit': 'Urgencias'}
{"timestamp": "2025-05-12T14:40:35.114856", "patient_id": "p024", "heart_rate": 139, "blood_pressure": "169/77", "temperature": 37.2, "oxygen_saturation": 100, "blood_glucose": 131, "unit": "Planta"}
{"timestamp": "2025-05-12T14:40:39.812712", "patient_id": "p033", "heart_rate": 111, "blood_pressure": "173/82", "temperature": 36.5, "oxygen_saturation": 86, "blood_glucose": 156, "unit": "UCI"}
{"timestamp": "2025-05-12T14:40:44.584060", "patient_id": "p100", "heart_rate": 96, "blood_pressure": "100/114", "temperature": 37.9, "oxygen_saturation": 89, "blood_glucose": 86, "unit": "Urgencias"}
```



### 3.5. ETL de los datos streaming

Bien, una vez tenemos los datos siendo recibidos por Spark desde Kafka. Debemos pensar cuál será el ETL que realizaremos con nuestros datos, unos pasos que podemos seguir son los siguientes:

1. Lectura de datos desde Kafka
2. Deserialización del JSON
3. Conversión a columnas útiles (cast, split, filtrado)
4. Cálculo de indicadores (alertas, medias, etc.)
5. Escritura en HDFS o Parquet

De lo cual, ya tenemos tanto la lectura de Kafka como la deserialización en JSON, lo que corresponde con la **extracción de los datos**. Solamente cambiaríamos el código de la media que está simplemente de ejemplo.

#### 3.5.1. Transformación

Para la transformación de los datos se ha planteado los siguientes puntos que podemos tratar:

- La presión sanguínea contiene dos tipos de presiones sanguíneas que se pueden separar por tipo. Una vez tenemos las presiones separadas por tipo, podemos eliminar la columna de la presión sanguínea.
- La columna de la unidad hospitalaria se puede mapear para que los valores sean representados por valores numéricos.
- Realizar filtros para datos erróneos
- Y, por último, agregaremos una nueva columna para indicar qué pacientes pueden estar en un estado de alerta médica en base a sus contantes vitales.

El código resultante es el siguiente:

```
● ● ●

# Separar presión arterial
df = df_json.withColumn("systolic_pressure", split(col("blood_pressure"), "/").getItem(0).cast("int")) \
    .withColumn("diastolic_pressure", split(col("blood_pressure"), "/").getItem(1).cast("int")) \
    .drop("blood_pressure")

# Mapeo de unidades hospitalarias
df = df.withColumn("unit_index",
    when(col("unit") == "Urgencias", 0)
    .when(col("unit") == "UCI", 1)
    .when(col("unit") == "Planta", 2)
    .otherwise(-1))

# Filtro de datos erróneos
df = df.filter((col("temperature") >= 35) & (col("temperature") <= 42))

# Generación de alertas médicas
df = df.withColumn("alert",
    when((col("systolic_pressure") > 140) | (col("diastolic_pressure") > 90), "HIPERTENSION")
    .when((col("oxygen_saturation") < 90), "HIPOXIA")
    .when((col("blood_glucose") > 150), "HIPERGLUCEMIA")
    .otherwise("OK"))
```

### 3.5.2. Carga

Los datos se guardarán como un histórico en HDFS en formato parquet con el siguiente código:

```
● ● ●

# Escritura en consola o en parquet (HDFS)
query = df.writeStream \
    .outputMode("append") \
    .format("parquet") \
    .option("path", "bda/proyecto/streaming/parquet") \
    .option("checkpointLocation", "bda/proyecto/streaming/checkpoints") \
    .start()

query.awaitTermination()
```

En donde guardaremos los archivos en la carpeta “parquet” y los checkpoints en la carpeta “checkpoints”.

Realizamos la nueva ingesta con el ETL aplicado:

```
with 2 core(s), 1024.0 MiB RAM
25/05/22 17:24:23 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20250522172423-0001/2 on worker-20250522170819-19
2.168.18.10-390001 (192.168.18.10:39001) with 2 core(s)
25/05/22 17:24:23 INFO StandaloneSchedulerBackend: Granted executor ID app-20250522172423-0001/2 on hostPort 192.168.18.10:39001
with 2 core(s), 1024.0 MiB RAM
25/05/22 17:24:24 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 40621.
25/05/22 17:24:24 INFO NettyBlockTransferService: Server created on cluster-bda:40621
25/05/22 17:24:24 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
25/05/22 17:24:24 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, cluster-bda, 40621, None)
25/05/22 17:24:24 INFO BlockManagerMasterEndpoint: Registering block manager cluster-bda:40621 with 434.4 MiB RAM, BlockManagerId(driver, cluster-bda, 40621, None)
25/05/22 17:24:24 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, cluster-bda, 40621, None)
25/05/22 17:24:24 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, cluster-bda, 40621, None)
25/05/22 17:24:24 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20250522172423-0001/0 is now RUNNING
25/05/22 17:24:24 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20250522172423-0001/2 is now RUNNING
25/05/22 17:24:24 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20250522172423-0001/1 is now RUNNING
25/05/22 17:24:25 INFO StandaloneSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
```

```
': 'p003', 'heart_rate': 136, 'blood_pressure': '107/79', 'temperature': 39.8, 'oxygen_saturation': 93, 'blood_glucose': 96, 'unit': 'Planta'}
Enviado: {'timestamp': '2025-05-22T17:25:12.098135', 'patient_id': 'p075', 'heart_rate': 90, 'blood_pressure': '155/118', 'temperature': 39.6, 'oxygen_saturation': 90, 'blood_glucose': 122, 'unit': 'UCI'}
Enviado: {'timestamp': '2025-05-22T17:25:13.603239', 'patient_id': 'p085', 'heart_rate': 135, 'blood_pressure': '170/78', 'temperature': 39.7, 'oxygen_saturation': 96, 'blood_glucose': 144, 'unit': 'UCI'}
Enviado: {'timestamp': '2025-05-22T17:25:15.110726', 'patient_id': 'p044', 'heart_rate': 131, 'blood_pressure': '160/112', 'temperature': 38.7, 'oxygen_saturation': 100, 'blood_glucose': 165, 'unit': 'Planta'}
|
{
    "heart_rate": 136, "blood_pressure": "107/79", "temperature": 39.8, "oxygen_saturation": 93, "blood_glucose": 96, "unit": "Planta"
}
{"timestamp": "2025-05-22T17:25:12.098135", "patient_id": "p075", "heart_rate": 90, "blood_pressure": "155/118", "temperature": 39.6, "oxygen_saturation": 90, "blood_glucose": 122, "unit": "UCI"}
{"timestamp": "2025-05-22T17:25:13.603239", "patient_id": "p085", "heart_rate": 135, "blood_pressure": "170/78", "temperature": 39.7, "oxygen_saturation": 96, "blood_glucose": 144, "unit": "UCI"}
{"timestamp": "2025-05-22T17:25:15.110726", "patient_id": "p044", "heart_rate": 131, "blood_pressure": "160/112", "temperature": 38.7, "oxygen_saturation": 100, "blood_glucose": 165, "unit": "Planta"}
```

Ahora los datos ya no salen por consola ya que los datos se están guardando en parquet, pero vemos que el consumo sigue realizándose igual que antes.



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	May 22 19:29	0	0 B	_spark_metadata
-rw-r--r--	hadoop	supergroup	3.03 KB	May 22 19:26	1	128 MB	part-00000-03bd14ee-4a45-48d2-8e61-a946547a6282-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	3.12 KB	May 22 19:27	1	128 MB	part-00000-0631b141-3729-4e72-81a1-e0228fc8a74a-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	3.14 KB	May 22 19:26	1	128 MB	part-00000-0659b7ee-d628-4f07-befc-2022efebf28e-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	3.24 KB	May 22 19:26	1	128 MB	part-00000-086bab6c-55fe-4f97-9e61-c0442bda606b-c000.snappy.parquet

### 3.6. Ingesta a PostgreSQL y conexión a Grafana

Para la visualización en Grafana hemos optado por usar una base de datos como PostgreSQL para almacenar los datos en streaming y luego hacer una conexión a Grafana. PostgreSQL es mucho más estable que MySQL a la hora de manejar cantidades masivas de datos. Primero iremos a la [Página de PostgreSQL](#) y descargamos en nuestra máquina la base de datos con el siguiente comando:



```
sudo apt install postgresql
```

#### 3.6.1 Creación de la base de datos

Lo siguiente será crear la base de datos para almacenar nuestros datos en streaming, para ello crearemos no solo la base de datos sino también un usuario con el que nuestro sistema pueda insertar los nuevos datos. Lo haremos de esta forma:

```
postgres=# CREATE DATABASE health_data;
CREATE DATABASE
postgres=# CREATE USER sparkuser WITH PASSWORD 'sparkpass';
CREATE ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE health_data TO sparkuser;
GRANT
postgres=# \q
```

Una vez tenemos nuestra base de datos, crearemos la tabla correspondiente a los datos en streaming. Muy parecido a MySQL lo hacemos de esta manera:

```
health_data=> CREATE TABLE vitales_pacientes (
    id SERIAL PRIMARY KEY,
    timestamp TIMESTAMP,
    patient_id TEXT,
    heart_rate INTEGER,
    systolic_pressure INTEGER,
    diastolic_pressure INTEGER,
    temperature DOUBLE PRECISION,
    oxygen_saturation INTEGER,
    blood_glucose INTEGER,
    unit TEXT,
    unit_index INTEGER,
    alert TEXT
);
CREATE TABLE
```

En esta tabla podemos apreciar algo peculiar, `id SERIAL PRIMARY KEY`. Esto hará que la base de datos generé un ID incremental y no tengamos que generarla manualmente como hicimos anteriormente. Bastante útil sobre todo teniendo en cuenta que Spark puede no procesar las peticiones en el momento y las vuelve a intentar más tarde, con esto no tendremos problemas con los duplicados.

### 3.6.2 Sistema Spark - Kafka Connect - PostgreSQL

Lo siguiente será crear ahora un sistema donde Spark le mandé a Kafka de vuelta un JSON con los datos procesados y que este mediante un conector los inserte en PostgreSQL. El inconveniente principal es que Spark manda JSON sin esquema, lo cual no tolera Kafka. Por ello, debemos preparar un esquema y una estructura para nuestro JSON que Kafka toleré. El resultado final es el siguiente:

```
# 6. Construcción schema+payload
schema_json = {
    "type": "struct",
    "fields": [
        {"field": "timestamp", "type": "int64", "optional": True},
        {"field": "patient_id", "type": "string", "optional": True},
        {"field": "heart_rate", "type": "int32", "optional": True},
        {"field": "temperature", "type": "float", "optional": True},
        {"field": "oxygen_saturation", "type": "int32", "optional": True},
        {"field": "blood_glucose", "type": "int32", "optional": True},
        {"field": "unit", "type": "string", "optional": True},
        {"field": "systolic_pressure", "type": "int32", "optional": True},
        {"field": "diastolic_pressure", "type": "int32", "optional": True},
        {"field": "unit_index", "type": "int32", "optional": True},
        {"field": "alert", "type": "string", "optional": True}
    ],
    "optional": False,
    "name": "VitalSigns"
}

schema_str = json.dumps(schema_json)
```

```

payload_struct = struct(
    col("timestamp"),
    col("patient_id"),
    col("heart_rate"),
    col("temperature"),
    col("oxygen_saturation"),
    col("blood_glucose"),
    col("unit"),
    col("systolic_pressure"),
    col("diastolic_pressure"),
    col("unit_index"),
    col("alert")
)

df_out = df.select(
    concat(lit(''), col("patient_id"), lit('')).alias("key"),
    to_json(payload_struct).alias("payload_json")
).select(
    col("key"),
    concat(
        lit('{\"schema\": ') , lit(schema_str), lit(', \"payload\": '), col("payload_json"), lit('}')
    ).alias("value")
)
)

```

De esta forma nuestro JSON tendrá un esquema y una estructura que Kafka podrá aceptar sin problemas. También en el ETL se ha agregado una nueva transformación: Transformar la fecha a milisegundos UNIX para que PostgreSQL no reciba una cadena. Para ello hacemos lo siguiente:

```

# Convertir timestamp a milisegundos UNIX y reemplazar la columna
df = df.withColumn("timestamp_trimmed", expr("substring(timestamp, 1, 19)")) \
    .withColumn("timestamp_ms", (unix_timestamp("timestamp_trimmed", "yyyy-MM-dd'T'HH:mm:ss") * 1000).cast("long")) \
    .drop("timestamp") \
    .withColumnRenamed("timestamp_ms", "timestamp")

```

Primero redondeamos y luego hacemos la transformación. Si no redondeamos obtendremos un error relacionado con un tema de incompatibilidad entre versiones. Lo último es enviar los datos a Kafka en el `writeStream`:

```

# 7. Escribir al topic Kafka de salida
query = df_out.writeStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "192.168.18.8:9094") \
    .option("topic", "alertas_pacientes") \
    .option("checkpointLocation", "/bda/proyecto/streaming/checkpoints") \
    .start()

query.awaitTermination()

```

Los mensajes los recibirá Kafka a través del topic llamado “alertas\_pacientes”. Ahora debemos crear el conector de PostgreSQL para que Kafka pueda enviar sin problemas los datos y se inserten en la tabla de la base de datos. Hay que destacar que hemos tenido problemas con la versión 10.8.2 del conector JDBC de confluent por lo que se ha actualizado a la 10.8.4.

El conector quedará de la siguiente forma:

```

{
  "name": "postgres-alertas-sink",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
    "tasks.max": "1",

    "topics": "alertas_pacientes",
    "connection.url": "jdbc:postgresql://localhost:5432/health_data",
    "connection.user": "sparkuser",
    "connection.password": "sparkpass",

    "table.name.format": "vitales_pacientes",
    "insert.mode": "insert",
    "pk.mode": "record_value",
    "pk.fields": "patient_id",

    "auto.create": "true",
    "auto.evolve": "true",

    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "true",

    "transforms": "convertTS",
    "transforms.convertTS.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value",
    "transforms.convertTS.field": "timestamp",
    "transforms.convertTS.target.type": "Timestamp"
  }
}

```

De esta manera le decimos a Kafka que inserte los datos en la base de datos que tendrán de clave el ID del paciente, que le llega en forma de JSON y que la fecha la transforme para que sea en formato TIMESTAMP.

Con esto hecho, el proceso sería igual que antes pero los datos llegarían ahora a Kafka de nuevo procesados y los guarda en la base de datos de PostgreSQL.

```

tor ID app-20250606102227-0026/1 on hostPort 192.168.18.9:41165
with 2 core(s), 1024.0 MiB RAM
25/06/06 10:22:27 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20250606102227-0026/2 on worker-20250606070830-19
2.168.18.10-40599 (192.168.18.10:40599) with 2 core(s)
25/06/06 10:22:27 INFO StandaloneSchedulerBackend: Granted executor ID app-20250606102227-0026/2 on hostPort 192.168.18.10:40599
with 2 core(s), 1024.0 MiB RAM
25/06/06 10:22:27 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 37533.
25/06/06 10:22:27 INFO NettyBlockTransferService: Server created on cluster-bda:37533
25/06/06 10:22:27 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
25/06/06 10:22:27 INFO BlockManagerMaster: Registering BlockManagerId(driver, cluster-bda, 37533, None)
25/06/06 10:22:27 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20250606102227-0026/1 is now RUNNING
25/06/06 10:22:27 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20250606102227-0026/0 is now RUNNING
25/06/06 10:22:27 INFO StandaloneAppClient$ClientEndpoint: Executor updated: app-20250606102227-0026/2 is now RUNNING
25/06/06 10:22:27 INFO BlockManagerMasterEndpoint: Registering b
lock manager cluster-bda:37533 with 434.4 MiB RAM, BlockManagerId(driver, cluster-bda, 37533, None)
25/06/06 10:22:27 INFO BlockManagerMaster: Registered BlockManagerId(driver, cluster-bda, 37533, None)
25/06/06 10:22:27 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, cluster-bda, 37533, None)
25/06/06 10:22:29 INFO StandaloneSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisters dResourcesRatio: 0.0

```

Enviado: {'timestamp': '2025-06-06T10:24:50.590487', 'patient\_id': 'p035', 'heart\_rate': 98, 'blood\_pressure': '104/85', 'temperature': 40.1, 'oxygen\_saturation': 92, 'blood\_glucose': 199, 'unit': 'UCI'}

Enviado: {'timestamp': '2025-06-06T10:24:52.106713', 'patient\_id': 'p034', 'heart\_rate': 83, 'blood\_pressure': '134/63', 'temperature': 39.1, 'oxygen\_saturation': 93, 'blood\_glucose': 78, 'unit': 'Urgencias'}

Enviado: {'timestamp': '2025-06-06T10:24:53.609925', 'patient\_id': 'p060', 'heart\_rate': 125, 'blood\_pressure': '124/110', 'temperature': 39.9, 'oxygen\_saturation': 90, 'blood\_glucose': 113, 'unit': 'Urgencias'}

Enviado: {'timestamp': '2025-06-06T10:24:55.114048', 'patient\_id': 'p026', 'heart\_rate': 101, 'blood\_pressure': '126/120', 'temperature': 39.4, 'oxygen\_saturation': 86, 'blood\_glucose': 88, 'unit': 'Urgencias'}

e": "string", "optional": true}, {"field": "heart\_rate", "type": "int32", "optional": true}, {"field": "temperature", "type": "float", "optional": true}, {"field": "oxygen\_saturation", "type": "int32", "optional": true}, {"field": "blood\_glucose", "type": "int32", "optional": true}, {"field": "unit", "type": "string", "optional": true}, {"field": "systolic\_pressure", "type": "int32", "optional": true}, {"field": "diastolic\_pressure", "type": "int32", "optional": true}, {"field": "unit\_index", "type": "int32", "optional": true}, {"field": "alert", "type": "string", "optional": true}], "optional": false, "name": "VitalSigns"}, "payload": {"timestamp": 1749205495000, "patient\_id": "p026", "heart\_rate": 101, "temperature": 39.4, "oxygen\_saturation": 86, "blood\_glucose": 88, "unit": "Urgencias", "systolic\_pressure": 126, "diastolic\_pressure": 88, "unit\_index": 0, "alert": "HIPERTENSION"}}, {"schema": [{"type": "struct", "fields": [{"field": "timestamp", "type": "int64", "optional": true}, {"field": "patient\_id", "typ

```

sumer-postgres-alertas-sink-0-afb62a3d-22b0-4c96-9465-02426ffe9b
6e', protocol='range'} (org.apache.kafka.clients.consumer.intern
als.ConsumerCoordinator:666)
[2025-06-06 10:23:48,801] INFO [postgres-alertas-sink|task-0] [C
onsumer clientId=connector-consumer-postgres-alertas-sink-0, gro
upId=connect-postgres-alertas-sink] Finished assignment for grou
p at generation 5: {connector-consumer-postgres-alertas-sink-0-a
-fb62a3d-22b0-4c96-9465-02426ffe9b6e=Assignment(partitions=[alert
as_pacientes-0, alertas_pacientes-1])} (org.apache.kafka.clients
.consumer.internals.ConsumerCoordinator:664)
[2025-06-06 10:23:48,823] INFO [postgres-alertas-sink|task-0] [C
onsumer clientId=connector-consumer-postgres-alertas-sink-0, gro
upId=connect-postgres-alertas-sink] Successfully synced group in
generation Generation{generationId=5, memberId=connector-consu
mer-postgres-alertas-sink-0-afb62a3d-22b0-4c96-9465-02426ffe9b6e
', protocol='range'} (org.apache.kafka.clients.consumer.internal
s.ConsumerCoordinator:843)
[2025-06-06 10:23:48,828] INFO [postgres-alertas-sink|task-0] [C
onsumer clientId=connector-consumer-postgres-alertas-sink-0, gro
upId=connect-postgres-alertas-sink] Notifying assignor about the
new Assignment(partitions=[alertas_pacientes-0, alertas_pacient
es-1]) (org.apache.kafka.clients.consumer.internals.ConsumerCoor
dinator:324)
[2025-06-06 10:23:48,831] INFO [postgres-alertas-sink|task-0] [C
onsumer clientId=connector-consumer-postgres-alertas-sink-0, gro
upId=connect-postgres-alertas-sink] Adding newly assigned partit
ions: alertas_pacientes-0, alertas_pacientes-1 (org.apache.kafka
.clients.consumer.internals.ConsumerRebalanceListenerInvoker:58)
[2025-06-06 10:23:48,837] INFO [postgres-alertas-sink|task-0] [C
onsumer clientId=connector-consumer-postgres-alertas-sink-0, gro
upId=connect-postgres-alertas-sink] Found no committed offset fo
r partition alertas_pacientes-0 (org.apache.kafka.clients.consum
er.internals.ConsumerCoordinator:1508)

s": "alertas_pacientes", "connection.url": "jdbc:postgresql://local
host:5432/health_data", "connection.user": "sparkuser", "connection
.password": "sparkpass", "table.name.format": "viales_pacientes", "
insert.mode": "insert", "pk.mode": "record_value", "pk.fields": "pati
ent_id", "auto.create": "true", "auto.evolve": "true", "errors.tolera
nce": "all", "errors.log.enable": "true", "errors.deadletterqueue.to
pic.name": "errores_postgres", "errors.deadletterqueue.context.hea
ders.enable": "true", "errors.deadletterqueue.topic.replication.fa
ctor": "1", "value.converter": "org.apache.kafka.connect.json.JsonC
onverter", "value.converter.schemas.enable": "true", "transforms": "
convertTS", "transforms.convertTS.type": "org.apache.kafka.connect
.transforms.TimestampConverter$Value", "transforms.convertTS.fiel
d": "timestamp", "transforms.convertTS.target.type": "Timestamp", "n
ame": "postgres-alertas-sink"}, "tasks": [], "type": "sink"}hadoop@ma
ster:~$  

hadoop@master:~$
```

```

e --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --
partitions 2 --topic alertas_pacientes
WARNING: Due to limitations in metric names, topics with a perio
d ('.') or underscore ('_') could collide. To avoid issues it is
best to use either, but not both.
Created topic alertas_pacientes.
hadoop@master:/opt/kafka_2.13-4.0.0$ bin/kafka-topics.sh --delet
e --topic alertas_pacientes --bootstrap-server 192.168.18.8:9094
hadoop@master:/opt/kafka_2.13-4.0.0$ bin/kafka-topics.sh --creat
e --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --
partitions 2 --topic alertas_pacientes
WARNING: Due to limitations in metric names, topics with a perio
d ('.') or underscore ('_') could collide. To avoid issues it is
best to use either, but not both.
Created topic alertas_pacientes.
hadoop@master:/opt/kafka_2.13-4.0.0$ |
```

	id	timestamp	patient_id	heart_rate	systolic_pressure	diast olic_pressure	temperature	oxygen_saturation	blood_glucose	unit	unit_index	alert
	1	2025-06-06 10:19:12	p089		98	148						
		91	38.599998474121094		100	196						Planta
			2	HIPERTENSION								
	2	2025-06-06 10:19:15	p013		73	123						
		60		38	88	102						Planta
			2	HIPOXIA								
	3	2025-06-06 10:22:21	p014		139	178						
		113	39.70000076293945		89	126						Urgenci as
			0	HIPERTENSION								
	4	2025-06-06 10:22:26	p041		74	168						
		100	38.900001525878906		100	97						UCI
			1	HIPERTENSION								
	5	2025-06-06 10:22:30	p092		83	170						
		81	36.79999923706055		99	73						Planta
			2	HIPERTENSION								
	6	2025-06-06 10:22:32	p079		102	113						
		77	36.900001525878906		95	154						Planta
:												

### 3.6.3 Conectar PostgreSQL a Grafana

Esta es la parte más fácil de la conexión en streaming ya que simplemente encendemos Prometheus (el cual fue configurado como en el ejemplo 2 de clase) y posteriormente Grafana. Dentro de Grafana buscamos el conector de PostgreSQL y rellenamos los campos.

The screenshot shows the Grafana interface for managing data sources. The top navigation bar includes 'Home', 'Connections', 'Data sources', and the specific entry 'grafana-postgresql-datasource'. The search bar at the top right contains the placeholder 'Search or jump to...'. Below the search bar are tabs for 'Type' (PostgreSQL), 'Alerting' (Supported), 'Explore data', and 'Build a dashboard'. The main content area is titled 'grafana-postgresql-datasource' and specifies 'Type: PostgreSQL'. A 'Name' field is set to 'grafana-postgresql-datasource' with a 'Default' button and a toggle switch. A note below states: 'Before you can use the Postgres data source, you must configure it below or in the config file. For detailed instructions, [view the documentation](#)'. A note below that reads: 'Fields marked with \* are required'. A section titled 'User Permissions' contains a note: 'The database user should only be granted SELECT permissions on the specified database & tables you want to query. Grafana does not validate that queries are safe so queries can contain any SQL statement. For example, statements like `DELETE FROM user;` and `DROP TABLE user;` would be executed.' Another note says: 'To protect against this we Highly recommend you create a specific PostgreSQL user with restricted permissions. Check out the docs for more information.'

The screenshot displays two panels for configuring the PostgreSQL data source. The left panel, titled 'Connection', contains fields for 'Host URL \*' (set to 'http://localhost:5432'), 'Database name \*' (set to 'health\_data'), and 'Username \*' (set to 'sparkuser'). It also includes sections for 'Password \*' (redacted), 'TLS/SSL Mode \*' (set to 'require'), and 'TLS/SSL Method \*' (set to 'File system path'). The right panel, titled 'Additional settings', contains 'PostgreSQL Options' with 'Version' set to '14'. It includes sections for 'Min time interval \*' (set to '1m'), 'TimescaleDB' (disabled), 'Connection limits' with 'Max open' set to '100' and 'Auto max idle' checked, and 'Max idle' set to '100'. It also includes 'Max lifetime' set to '14400'.

Tras la configuración, guardamos la conexión y ya estaría PostgreSQL conectado a Grafana.



## 4. Ingesta de datos batch: MySQL + Kafka + HDFS

Una vez hemos obtenido nuestros datos en streaming, el siguiente paso es obtener nuestros datos en batch que se irán actualizando en un intervalo concreto. En el caso de las analíticas, se pueden actualizar diariamente cada noche.

### 4.1. ¿Cómo abordar el problema?

La idea para los datos en batch es la siguiente:

1. Los datos en batch se irán subiendo a la base de datos.
2. A través del conector de MySQL, Kafka recopilará toda la información almacenada en dicha base de datos.
3. Al final, Kafka guardará los datos en Hadoop usando el conector de HDFS en archivos parquet.
4. Descargaremos a nuestra máquina la data almacenada en HDFS.
5. Cargaremos y visualizaremos la data en Power BI.

### 4.2. Creación de las tablas de la base de datos

Primero crearemos las tablas para recopilar los datos en batch. Como las analíticas tienen los antibióticos de manera anidada, crearemos una tabla para los antibióticos que esté relacionada con las analíticas. Para ello tenemos este código:

```
CREATE TABLE analiticas (
    analitica_id INT PRIMARY KEY,
    patient_id VARCHAR(10) NOT NULL,
    sample_date DATE NOT NULL,
    sample_type VARCHAR(50),
    bacteria_detected VARCHAR(100)
);
```

```
CREATE TABLE antibiotic_res (
    res_id INT PRIMARY KEY,
    analitica_id INT NOT NULL,
    antibiotic_name VARCHAR(100),
    resistance_level VARCHAR(50),
    FOREIGN KEY (analitica_id) REFERENCES analiticas(analitica_id) ON DELETE CASCADE
);
```

Comprobamos que las tablas se han creado.

```
mysql> CREATE TABLE analiticas (
    -> analitica_id INT PRIMARY KEY,
    -> patient_id VARCHAR(10) NOT NULL,
    -> sample_date DATE NOT NULL,
    -> sample_type VARCHAR(50),
    -> bacteria_detected VARCHAR(100)
    -> );
Query OK, 0 rows affected (0,26 sec)
```

```
mysql> CREATE TABLE antibiotic_res (
    -> res_id INT PRIMARY KEY,
    -> analitica_id INT NOT NULL,
    -> antibiotic_name VARCHAR(100),
    -> resistance_level VARCHAR(50),
    -> FOREIGN KEY (analitica_id) REFERENCES analiticas(analitica_id) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0,28 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_MedicineDB |
+-----+
| analiticas           |
| antibiotic_res        |
| pacientes             |
+-----+
```

### 4.3. Ingesta de datos en batch

Lo siguiente será ejecutar nuestro generador de datos batch para que los inserte dentro de nuestra base de datos.

Ejecutaremos el comando:



```
python3 gen_batch.py
```

Tras ejecutarlo, se insertarán 200 datos dentro de la base de datos. Esto simulará como los datos son actualizados por el sistema del hospital que cargará las nuevas analíticas durante la noche.

Comprobamos que las tablas se han actualizado.

```
mysql> select * from analiticas
-> ;
+-----+-----+-----+-----+-----+
| analitica_id | patient_id | sample_date | sample_type | bacteria_detected |
+-----+-----+-----+-----+-----+
| 1 | p042 | 2025-05-18 | exudado nasal | Escherichia coli |
| 2 | p063 | 2025-05-14 | esputo | Pseudomonas aeruginosa |
| 3 | p019 | 2025-05-15 | sangre | Klebsiella pneumoniae |
| 4 | p077 | 2025-05-09 | orina | Staphylococcus aureus |
| 5 | p011 | 2025-05-14 | esputo | Escherichia coli |
| 6 | p019 | 2025-05-18 | esputo | Pseudomonas aeruginosa |
| 7 | p058 | 2025-05-10 | sangre | Pseudomonas aeruginosa |
| 8 | p048 | 2025-05-14 | sangre | Staphylococcus aureus |
| 9 | p025 | 2025-05-08 | orina | Klebsiella pneumoniae |
| 10 | p077 | 2025-05-16 | exudado nasal | Escherichia coli |
| 11 | p090 | 2025-05-13 | orina | Staphylococcus aureus |
| 12 | p042 | 2025-05-10 | esputo | Staphylococcus aureus |
| 13 | p010 | 2025-05-13 | esputo | Klebsiella pneumoniae |
| 14 | p046 | 2025-05-09 | sangre | Staphylococcus aureus |
| 15 | p059 | 2025-05-09 | esputo | Staphylococcus aureus |
| 16 | p080 | 2025-05-18 | orina | Pseudomonas aeruginosa |
| 17 | p050 | 2025-05-13 | esputo | Staphylococcus aureus |
| 18 | p039 | 2025-05-16 | exudado nasal | Pseudomonas aeruginosa |
| 19 | p036 | 2025-05-12 | exudado nasal | Klebsiella pneumoniae |
| 20 | p049 | 2025-05-17 | esputo | Escherichia coli |
| 21 | p037 | 2025-05-16 | sangre | Escherichia coli |
| 22 | p016 | 2025-05-10 | sangre | Klebsiella pneumoniae |
| 23 | p058 | 2025-05-11 | esputo | Escherichia coli |
| 24 | p081 | 2025-05-13 | exudado nasal | Klebsiella pneumoniae |
| 25 | p034 | 2025-05-11 | esputo | Klebsiella pneumoniae |
| 26 | p047 | 2025-05-08 | esputo | Klebsiella pneumoniae |
| 27 | p004 | 2025-05-17 | orina | Klebsiella pneumoniae |
| 28 | p001 | 2025-05-08 | exudado nasal | Klebsiella pneumoniae |
| 29 | p068 | 2025-05-10 | exudado nasal | Staphylococcus aureus |
| 30 | p018 | 2025-05-11 | orina | Klebsiella pneumoniae |
| 31 | p094 | 2025-05-18 | sangre | Klebsiella pneumoniae |
| 32 | p085 | 2025-05-10 | sangre | Escherichia coli |
| 33 | p034 | 2025-05-15 | esputo | Klebsiella pneumoniae |
| 34 | p093 | 2025-05-11 | orina | Escherichia coli |
| 35 | p059 | 2025-05-10 | orina | Escherichia coli |
+-----+
```

```
mysql> select * from antibiotic_res;
+-----+-----+-----+-----+
| res_id | analitica_id | antibiotic_name | resistance_level |
+-----+-----+-----+-----+
| 1 | 1 | colistina | intermedio |
| 2 | 1 | ceftriaxona | resistente |
| 3 | 1 | imipenem | intermedio |
| 4 | 1 | ceftriaxona | sensible |
| 5 | 2 | colistina | intermedio |
| 6 | 2 | ceftriaxona | sensible |
| 7 | 2 | ciprofloxacina | intermedio |
| 8 | 2 | ceftriaxona | sensible |
| 9 | 3 | imipenem | intermedio |
| 10 | 3 | imipenem | sensible |
| 11 | 3 | ceftriaxona | sensible |
| 12 | 3 | colistina | resistente |
| 13 | 4 | ciprofloxacina | resistente |
| 14 | 4 | imipenem | intermedio |
| 15 | 4 | ceftriaxona | resistente |
| 16 | 4 | ciprofloxacina | sensible |
| 17 | 5 | colistina | sensible |
| 18 | 5 | colistina | sensible |
| 19 | 5 | colistina | intermedio |
| 20 | 5 | imipenem | intermedio |
| 21 | 6 | imipenem | sensible |
| 22 | 6 | ciprofloxacina | intermedio |
| 23 | 6 | ciprofloxacina | intermedio |
| 24 | 6 | colistina | intermedio |
| 25 | 7 | imipenem | resistente |
| 26 | 7 | ciprofloxacina | sensible |
| 27 | 7 | colistina | resistente |
| 28 | 7 | imipenem | intermedio |
| 29 | 8 | colistina | sensible |
| 30 | 8 | ciprofloxacina | sensible |
| 31 | 8 | ciprofloxacina | resistente |
| 32 | 8 | imipenem | resistente |
| 33 | 9 | ciprofloxacina | sensible |
| 34 | 9 | ceftriaxona | sensible |
+-----+
```

Hemos generado un total de 800 analíticas para nuestro proyecto.

## 4.4. Creación de los Workers

Lo siguiente será definir nuestros workers que se encargarán de ejecutar los plugins de nuestro clúster. Debido a la falta de recurso evidente, usaremos solo 2 Workers en el proyecto. Las configuraciones son las siguientes:

```
bootstrap.servers=localhost:9094,localhost:9095
group.id=connect-cluster
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=true
offset.storage.topic=connect-offsets
offset.storage.replication.factor=2
config.storage.topic=connect-configs
config.storage.replication.factor=2
status.storage.topic=connect-status
status.storage.replication.factor=2
plugin.path=/opt/proyecto/libs
listeners=http://localhost:8083
```

```
bootstrap.servers=localhost:9094,localhost:9095
group.id=connect-cluster
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=true
offset.storage.topic=connect-offsets
offset.storage.replication.factor=2
config.storage.topic=connect-configs
config.storage.replication.factor=2
status.storage.topic=connect-status
status.storage.replication.factor=2
plugin.path=/opt/proyecto/libs
listeners=http://localhost:8084
```

Para iniciar los workers haremos los siguientes comandos:



```
bin/connect-distributed.sh /opt/proyecto/config/worker1.properties
bin/connect-distributed.sh /opt/proyecto/config/worker2.properties
```

## 4.5. Creación de los topics

Crearemos 2 topics para cada tabla correspondiente. Ambos tendrán 2 particiones y un factor de replicación de 2, los nombres empezarán con “micro-” seguido de los nombres de las tablas.

```
bin/kafka-topics.sh --create --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --partitions 2 --topic micro-analiticas
```

```
bin/kafka-topics.sh --create --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --partitions 2 --topic micro-antibiotic_res
```

Revisamos que se han creado correctamente:

```
hadoop@master:/opt/kafka_2.13-4.0.0$ bin/kafka-topics.sh --describe --topic micro-analiticas --bootstrap-server 192.168.18.8:9094
Topic: micro-analiticas TopicId: KheEa1lBQ9GMOQRNeF7WXQ PartitionCount: 2 ReplicationFactor: 2 Configs: segment.bytes=1073741824
        Topic: micro-analiticas Partition: 0 Leader: 2R replicas: 2,3 Isr: 2,3 Elr: LastKnownElr:
        Topic: micro-analiticas Partition: 1 Leader: 3R replicas: 3,2 Isr: 3,2 Elr: LastKnownElr:
```

```
hadoop@master:/opt/kafka_2.13-4.0.0$ bin/kafka-topics.sh --create --topic micro-antibiotic_res --bootstrap-server 192.168.18.8:9094
Topic: micro-antibiotic_res TopicId: iZDVScIwTc-KxHfugfahOA PartitionCount: 2 ReplicationFactor: 2 Configs: segment.bytes=1073741824
        Topic: micro-antibiotic_res Partition: 0 Leader: 2R replicas: 2,3 Isr: 2,3 Elr: LastKnownElr:
        Topic: micro-antibiotic_res Partition: 1 Leader: 3R replicas: 3,2 Isr: 3,2 Elr: LastKnownElr:
```

## 4.6. Conectores de MySQL y HDFS

El siguiente paso es crear los conectores de MySQL para acceder a los datos de la base de datos y guardarlos en HDFS. El problema es que, como hemos indicado anteriormente, las tablas analíticas y antibiotic\_res están relacionadas y el conector sólo puede trabajar con 1 tabla. Tendremos que crear 2 conectores de MySQL y HDFS por tabla.

### 4.6.1. Conectores de las analíticas

```
{
  "name": "mysql-analiticas-source",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:mysql://localhost:3306/MedicineDB",
    "connection.user": "root",
    "connection.password": "1234",
    "table.whitelist": "analiticas",
    "mode": "incrementing",
    "incrementing.column.name": "analitica_id",
    "topic.prefix": "micro-",
    "poll.interval.ms": "5000"
  }
}

{
  "name": "hdfs3-analiticas-sink-connector",
  "config": {
    "connector.class": "io.confluent.connect.hdfs3.Hdfs3SinkConnector",
    "tasks.max": "3",
    "confluent.topic.bootstrap.servers": "192.168.18.8:9094",
    "topics": "micro-analiticas",
    "store.url": "hdfs://cluster-bda:9000/bda/proyecto/batch/analiticas",
    "logs.dir": "logs/hdfs3sink",
    "format.class": "io.confluent.connect.hdfs3.json.JsonFormat",
    "path.format": "'year'=YYYY/'month'=MM/'day'=dd",
    "flush.size": "3",
    "hadoop.conf.dir": "/opt/hadoop-3.4.1/etc/hadoop/",
    "hadoop.home": "/opt/hadoop-3.4.1/"
  }
}
```

### 4.6.2. Conectores de las antibióticos

```
{
  "name": "mysql-antibioticos-source",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:mysql://localhost:3306/MedicineDB",
    "connection.user": "root",
    "connection.password": "1234",
    "table.whitelist": "antibiotic_res",
    "mode": "incrementing",
    "incrementing.column.name": "res_id",
    "topic.prefix": "micro-",
    "poll.interval.ms": "5000"
  }
}

{
  "name": "hdfs3-antibioticos-sink-connector",
  "config": {
    "connector.class": "io.confluent.connect.hdfs3.Hdfs3SinkConnector",
    "tasks.max": "3",
    "confluent.topic.bootstrap.servers": "192.168.18.8:9094",
    "topics": "micro-antibiotic_res",
    "store.url": "hdfs://cluster-bda:9000/bda/proyecto/batch/antibioticos",
    "logs.dir": "logs/hdfs3sink",
    "format.class": "io.confluent.connect.hdfs3.json.JsonFormat",
    "path.format": "'year'=YYYY/'month'=MM/'day'=dd",
    "flush.size": "3",
    "hadoop.conf.dir": "/opt/hadoop-3.4.1/etc/hadoop/",
    "hadoop.home": "/opt/hadoop-3.4.1/"
  }
}
```

### 4.6.3. Pasos para lanzar los conectores

Los pasos a seguir para realizar bien la ingestión de cada tabla es la siguiente:

1. Lanzar los workers
2. Lanzar el conector de MySQL Source. Donde {conector-mysql} es el conector de una tabla.

```
curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/{conector-mysql} http://192.168.18.8:8083/connectors
```

3. Verificar que hay mensajes en el topic de la tabla correspondiente.

```
bin/kafka-console-consumer.sh --topic {topic} --from-beginning --bootstrap-server 192.168.18.8:9094
```

4. Lanzar el conector de HDFS Sink. Donde {conector-hdfs} es el conector de una tabla.

```
curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/{conector-hdfs} http://192.168.18.8:8083/connectors
```

5. Luego miramos en HDFS para comprobar que se han guardado los datos.

### Ingesta de las analíticas:

```
Active ESM Apps para recibir futuras actualizaciones de seguridad adicionales.  
Vea https://ubuntu.com/esm o ejecute «sudo pro status»
```

```
The list of available updates is more than a week old.  
To check for new updates run: sudo apt update  
New release '24.04.2 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.
```

```
Last login: Sun May 18 08:40:58 2025 from 192.168.165.1  
hadoop@master:~$ curl http://192.168.18.8:8083/connectors  
[]  
hadoop@master:~$  
hadoop@master:~$ curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/mysql-analiticas-source-connector.json http://192.168.18.8:8083/connectors  
{"name": "mysql-analiticas-source", "config": {"connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector", "tasks.max": "1", "connection.url": "jdbc:mysql://localhost:3306/MedicineDB", "connection.user": "root", "connection.password": "1234", "table.whitelist": "analiticas", "mode": "incrementing", "incrementing.column.name": "analitica_id", "topic.prefix": "micro-", "poll.interval.ms": "5000", "name": "mysql-analiticas-source"}, "tasks": [], "type": "source"}  
hadoop@master:~$  
hadoop@master:~$ |
```

```
onal":false,"field":"analitica_id"}, {"type":"string","optional":false,"field":"patient_id"}, {"type": "int32", "optional": false, "name": "org.apache.kafka.connect.data.Date", "version": 1, "field": "sample_date"}, {"type": "string", "optional": true, "field": "sample_type"}, {"type": "string", "optional": true, "field": "bacteria_detected"}, {"optional": false, "name": "Escherichia coli"} } {"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "analitica_id"}, {"type": "string", "optional": false, "field": "patient_id"}, {"type": "int32", "optional": false, "name": "org.apache.kafka.connect.data.Date", "version": 1, "field": "sample_date"}, {"type": "string", "optional": true, "field": "sample_type"}, {"type": "string", "optional": true, "field": "bacteria_detected"}, {"optional": false, "name": "analiticas"}, "payload": {"analitica_id": 798, "patient_id": "p060", "sample_date": 20224, "sample_type": "esputo", "bacteria_detected": "Escherichia coli"} } {"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "analitica_id"}, {"type": "string", "optional": false, "field": "patient_id"}, {"type": "int32", "optional": false, "name": "org.apache.kafka.connect.data.Date", "version": 1, "field": "sample_date"}, {"type": "string", "optional": true, "field": "sample_type"}, {"type": "string", "optional": true, "field": "bacteria_detected"}, {"optional": false, "name": "analiticas"}, "payload": {"analitica_id": 799, "patient_id": "p028", "sample_date": 20222, "sample_type": "orina", "bacteria_detected": "Staphylococcus aureus"} } {"schema": {"type": "struct", "fields": [{"type": "int32", "optional": false, "field": "analitica_id"}, {"type": "string", "optional": false, "field": "patient_id"}, {"type": "int32", "optional": false, "name": "org.apache.kafka.connect.data.Date", "version": 1, "field": "sample_date"}, {"type": "string", "optional": true, "field": "sample_type"}, {"type": "string", "optional": true, "field": "bacteria_detected"}, {"optional": false, "name": "analiticas"}, "payload": {"analitica_id": 800, "patient_id": "p051", "sample_date": 20222, "sample_type": "esputo", "bacteria_detected": "Staphylococcus aureus"} } }
```

```
eListenerInvoker:58)  
[2025-05-18 10:03:09,500] INFO [mysql-analiticas-source|task-0] Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)  
[2025-05-18 10:03:09,513] INFO [mysql-analiticas-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)  
[2025-05-18 10:03:14,516] INFO [mysql-analiticas-source|task-0] Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)  
[2025-05-18 10:03:14,519] INFO [mysql-analiticas-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)  
[2025-05-18 10:03:19,522] INFO [mysql-analiticas-source|task-0] Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)  
[2025-05-18 10:03:19,526] INFO [mysql-analiticas-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)  
[2025-05-18 10:03:24,528] INFO [mysql-analiticas-source|task-0] Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)  
[2025-05-18 10:03:24,530] INFO [mysql-analiticas-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)
```

```
iticas-0 (io.confluent.connect.hdfs3.TopicPartitionWriter:822)  
[2025-05-18 10:03:25,335] INFO [hdfs3-analiticas-sink-connector|task-0] Starting commit and rotation for topic partition micro-analiticas-0 with start offsets {partition=0=252} and end offsets {partition=0=254} (io.confluent.connect.hdfs3.TopicPartitionWriter:398)  
[2025-05-18 10:03:25,412] INFO [hdfs3-analiticas-sink-connector|task-0] Committed hdfs://cluster-bda:9000/bda/proyecto/batch/analiticas/topics/micro-analiticas/partition=0/micro-analiticas-0+0000000252+0000000254.json for micro-analiticas-0 (io.confluent.connect.hdfs3.TopicPartitionWriter:822)  
[2025-05-18 10:03:25,419] INFO [hdfs3-analiticas-sink-connector|task-0] Starting commit and rotation for topic partition micro-analiticas-0 with start offsets {partition=0=255} and end offsets {partition=0=257} (io.confluent.connect.hdfs3.TopicPartitionWriter:398)  
[2025-05-18 10:03:25,481] INFO [hdfs3-analiticas-sink-connector|task-0] Committed hdfs://cluster-bda:9000/bda/proyecto/batch/analiticas/topics/micro-analiticas/partition=0/micro-analiticas-0+0000000255+0000000257.json for micro-analiticas-0 (io.confluent.connect.hdfs3.TopicPartitionWriter:822)  
[2025-05-18 10:03:25,491] INFO [hdfs3-analiticas-sink-connector|task-0] Starting commit and rotation for topic partition micro-analiticas-0 with start offsets {partition=0=258} and end offsets {partition=0=260} (io.confluent.connect.hdfs3.TopicPartitionWriter:398)
```

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	hadoop	supergroup	390 B	May 18 12:03	1	128 MB	micro-analiticas+0+0000000000+0000000002.json
□	-rw-r--r--	hadoop	supergroup	373 B	May 18 12:03	1	128 MB	micro-analiticas+0+0000000003+0000000005.json
□	-rw-r--r--	hadoop	supergroup	380 B	May 18 12:03	1	128 MB	micro-analiticas+0+0000000006+0000000008.json
□	-rw-r--r--	hadoop	supergroup	390 B	May 18 12:03	1	128 MB	micro-analiticas+0+0000000009+0000000011.json
□	-rw-r--r--	hadoop	supergroup	387 B	May 18 12:03	1	128 MB	micro-analiticas+0+0000000012+0000000014.json
□	-rw-r--r--	hadoop	supergroup	374 B	May 18 12:03	1	128 MB	micro-analiticas+0+0000000015+0000000017.json
□	-rw-r--r--	hadoop	supergroup	386 B	May 18 12:03	1	128 MB	micro-analiticas+0+0000000018+0000000020.json

### Ingesta de los antibióticos:

```

ctor.class":"io.confluent.connect.hdfs3.Hdfs3SinkConnector
","tasks.max":3,"confluent.topic.bootstrap.servers":"192
.168.18.8:9094","topics":"micro-analiticas","store.url":"
hdfs://cluster-bda:9000/bda/proyecto/batch/analiticas","log
s.dir":"logs/hdfs3sink","format.class":"io.confluent.conne
ct.hdfs3.json.JsonFormat","path.format":"'year'=YYYY/'mont
h'=MM/'day'=dd","flush.size":1000,"hadoop.conf.dir":"/op
t/hadoop-3.4.1/etc/hadoop/","hadoop.home":"/opt/hadoop-3.4
.1/","name":"hdfs3-analiticas-sink-connector"},"tasks":[],"
type":"sink"}hadoop@master:$ curl -X POST -H "Content-Ty
DELETE http://192.168.18.8:8083/conn
ectors/hdfs3-analiticas-sink-connector
hadoop@master:$ curl -X DELETE http://192.168.18.8:8083/c
onnectors/mysql-analiticas-source
hadoop@master:$ curl http://192.168.18.8:8083/connectors
[]hadoop@master:$
hadoop@master:$
hadoop@master:$ ^C
hadoop@master:$
hadoop@master:$ curl -X POST -H "Content-Type: applicatio
n/json" --data @/opt/proyecto/config/mysql-antibioticos-so
urce-connector.json http://192.168.18.8:8083/connectors
{"name":"mysql-antibioticos-source","config":{"connector.c
lass":"io.confluent.connect.jdbc.JdbcSourceConnector","tas
ks.max":1,"connection.url":"jdbc:mysql://localhost:3306/
MedicineDB","connection.user":"root","connection.password"
:"1234","table.whitelist":"antibiotic_res","mode":"increm
enting","incrementing.column.name":"res_id","topic.prefix":
"micro-", "poll.interval.ms": "5000", "name": "mysql-antibioti
cos-source"}, "tasks":[], "type": "source"}hadoop@master:$

```

```

rue,"field":"antibiotic_name"}, {"type": "string", "optional"
: true, "field": "resistance_level"}], "optional": false, "name"
: "antibiotic_res"}, "payload": {"res_id": 1582, "analitica_id"
: 396, "antibiotic_name": "colistina", "resistance_level": "int
ermedio"}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "opti
onal": false, "field": "res_id"}, {"type": "int32", "optional": f
alse, "field": "analitica_id"}, {"type": "string", "optional": t
rue, "field": "antibiotic_name"}, {"type": "string", "optional"
: true, "field": "resistance_level"}], "optional": false, "name"
: "antibiotic_res"}, "payload": {"res_id": 1583, "analitica_id"
: 396, "antibiotic_name": "ceftriaxona", "resistance_level": "r
esistente"}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "opti
onal": false, "field": "res_id"}, {"type": "int32", "optional": f
alse, "field": "analitica_id"}, {"type": "string", "optional": t
rue, "field": "antibiotic_name"}, {"type": "string", "optional"
: true, "field": "resistance_level"}], "optional": false, "name"
: "antibiotic_res"}, "payload": {"res_id": 1584, "analitica_id"
: 396, "antibiotic_name": "colistina", "resistance_level": "se
nsible"}}
{"schema": {"type": "struct", "fields": [{"type": "int32", "opti
onal": false, "field": "res_id"}, {"type": "int32", "optional": f
alse, "field": "analitica_id"}, {"type": "string", "optional": t
rue, "field": "antibiotic_name"}, {"type": "string", "optional"
: true, "field": "resistance_level"}], "optional": false, "name"
: "antibiotic_res"}, "payload": {"res_id": 1585, "analitica_id"
: 397, "antibiotic_name": "colistina", "resistance_level": "int
ermedio"}}
|
```

```

nect.jdbc.util.CachedConnectionProvider:64)
[2025-05-18 09:41:28,026] INFO [mysql-antibioticos-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)
[2025-05-18 09:41:28,121] INFO Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)
[2025-05-18 09:41:33,090] INFO [mysql-antibioticos-source|task-0] Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)
[2025-05-18 09:41:33,119] INFO [mysql-antibioticos-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)
[2025-05-18 09:41:38,122] INFO [mysql-antibioticos-source|task-0] Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)
[2025-05-18 09:41:38,132] INFO [mysql-antibioticos-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)
[2025-05-18 09:41:43,137] INFO [mysql-antibioticos-source|task-0] Database connection established. (io.confluent.connect.jdbc.util.CachedConnectionProvider:64)
[2025-05-18 09:41:43,148] INFO [mysql-antibioticos-source|task-0] Current Result is null. Executing query. (io.confluent.connect.jdbc.source.TimestampIncrementingTableQuerier:174)

```

connector|task-0] committing files after waiting for rotateIntervalMs time but less than flush.size records available. (io.confluent.connect.hdfs3.TopicPartitionWriter:448)

[2025-05-18 09:40:19,549] INFO [hdfs3-antibioticos-sink-connector|task-1] committing files after waiting for rotateIntervalMs time but less than flush.size records available. (io.confluent.connect.hdfs3.TopicPartitionWriter:448)

[2025-05-18 09:40:20,222] INFO [hdfs3-antibioticos-sink-connector|task-1] Successfully acquired lease for hdfs://cluster-bda:9000/bda/proyecto/batch/antibioticos/logs/hdfs3sink/micro-antibiotic\_res/1/log (io.confluent.connect.hdfs3.wal.FSWAL:67)

[2025-05-18 09:40:20,259] INFO [hdfs3-antibioticos-sink-connector|task-0] Successfully acquired lease for hdfs://cluster-bda:9000/bda/proyecto/batch/antibioticos/logs/hdfs3sink/micro-antibiotic\_res/0/log (io.confluent.connect.hdfs3.wal.FSWAL:67)

[2025-05-18 09:40:21,072] INFO [hdfs3-antibioticos-sink-connector|task-0] Committed hdfs://cluster-bda:9000/bda/proyecto/batch/antibioticos/topics/micro-antibiotic\_res/partition=0/micro-antibiotic\_res+0+000000000+000000099.json for micro-antibiotic\_res-0 (io.confluent.connect.hdfs3.TopicPartitionWriter:822)

[2025-05-18 09:40:21,073] INFO [hdfs3-antibioticos-sink-connector|task-1] Committed hdfs://cluster-bda:9000/bda/proyecto/batch/antibioticos/topics/micro-antibiotic\_res/partition=1/micro-antibiotic\_res+1+000000000+000000099.json for micro-antibiotic\_res-1 (io.confluent.connect.hdfs3.TopicPartitionWriter:822)

/bda/proyecto/batch/antibioticos/topics/micro-antibiotic_res/partition=0								Go!				
Show	25	entries								Search:		
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name				
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	277 B	May 18 12:09	<a href="#">1</a>	128 MB	micro-antibiotic_res+0+000000000+000000002.json				
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	276 B	May 18 12:09	<a href="#">1</a>	128 MB	micro-antibiotic_res+0+000000003+000000005.json				
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	280 B	May 18 12:09	<a href="#">1</a>	128 MB	micro-antibiotic_res+0+000000006+000000008.json				
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	276 B	May 18 12:09	<a href="#">1</a>	128 MB	micro-antibiotic_res+0+000000009+000000011.json				
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	285 B	May 18 12:09	<a href="#">1</a>	128 MB	micro-antibiotic_res+0+000000012+000000014.json				
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	278 B	May 18 12:09	<a href="#">1</a>	128 MB	micro-antibiotic_res+0+000000015+000000017.json				
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	275 B	May 18 12:09	<a href="#">1</a>	128 MB	micro-antibiotic_res+0+000000018+000000020.json				

## 4.7. Obtención de los datos para la visualización

El siguiente paso es el tema de la visualización, para ello ya hemos indicado que usaremos Power BI para la visualización de los datos en batch ya que este no permite visualizar los datos en streaming de manera nativa. Power BI puede abrir los datos en HDFS, pero al intentarlo nos salta un error de que no puede acceder a ciertos nodos ya que los datos se distribuyen entre los nodos y solo se puede acceder al master. La solución a esto es descargar los datos a local y luego a nuestra máquina anfitriona, no es lo más adecuado pero es la única solución más rentable por el momento.

Descargamos los datos con scp a nuestra máquina anfitriona.

```
PS C:\Users\-----> scp -r hadoop@192.168.165.8:/home/hadoop/data C:\Users\-----\Desktop\CURSO_IA\BDA\ proyecto\ proyecto-final-bda-2024-25-Dansarasix-DML
hadoop@192.168.165.8's password:
micro-antibiotic_res+0+0000001365+0000001367.json                                         100% 291   31.6KB/s  00:00
micro-antibiotic_res+0+0000000159+0000000161.json                                         100% 285   30.9KB/s  00:00
micro-antibiotic_res+0+0000001437+0000001439.json                                         100% 292   40.7KB/s  00:00
micro-antibiotic_res+0+0000001230+0000001232.json                                         100% 299   58.4KB/s  00:00
micro-antibiotic_res+0+0000000339+0000000341.json                                         100% 286   34.9KB/s  00:00
micro-antibiotic_res+0+0000000621+0000000623.json                                         100% 300   41.1KB/s  00:00
micro-antibiotic_res+0+0000000042+0000000044.json                                         100% 287   31.1KB/s  00:00
micro-antibiotic_res+0+0000000119+00000001121.json                                         100% 287   46.7KB/s  00:00
micro-antibiotic_res+0+0000000180+0000000182.json                                         100% 279   38.9KB/s  00:00
micro-antibiotic_res+0+0000000864+0000000866.json                                         100% 288   46.9KB/s  00:00
micro-antibiotic_res+0+0000000921+0000000923.json                                         100% 294   35.9KB/s  00:00
micro-antibiotic_res+0+0000000654+0000000656.json                                         100% 295   41.2KB/s  00:00
micro-antibiotic_res+0+0000000741+0000000743.json                                         100% 292   40.7KB/s  00:00
micro-antibiotic_res+0+0000000102+0000000104.json                                         100% 281   39.2KB/s  00:00
micro-antibiotic_res+0+00000001122+00000001124.json                                         100% 293   40.9KB/s  00:00
micro-antibiotic_res+0+0000000783+0000000785.json                                         100% 289   40.3KB/s  00:00
micro-antibiotic_res+0+0000000879+0000000881.json                                         100% 292   25.9KB/s  00:00
micro-antibiotic_res+0+0000000249+0000000251.json                                         100% 286   31.0KB/s  00:00
micro-antibiotic_res+0+0000001023+0000001025.json                                         100% 298   26.5KB/s  00:00
micro-antibiotic_res+0+0000001272+0000001274.json                                         100% 290   25.8KB/s  00:00
micro-antibiotic_res+0+0000001245+0000001247.json                                         100% 293   28.6KB/s  00:00
micro-antibiotic_res+0+0000000942+0000000944.json                                         100% 297   36.3KB/s  00:00
micro-antibiotic_res+0+0000001398+0000001400.json                                         100% 298   22.4KB/s  00:00
micro-antibiotic_res+0+0000000354+0000000356.json                                         100% 287   25.5KB/s  00:00
micro-antibiotic_res+0+0000000222+0000000224.json                                         100% 289   31.4KB/s  00:00
```

Los datos estarán disponibles en el repositorio de github también.

## 4.8. Conectar MySQL a PostgreSQL para métricas combinadas

Para poder intentar realizar visualizaciones combinando datos en streaming y en batch en Grafana debemos conectar MySQL a PostgreSQL. Lo haremos a través de una extensión FDW (Foreign Data Wrapper) donde nuestro usuario tendrá acceso mediante el usuario de MySQL a la base de datos de la misma.

1. Primero instalamos la extensión y la agregamos a nuestra base de datos.

```
hadoop@master:~$ sudo apt install postgresql-14-mysql-fdw
[sudo] password for hadoop:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
 default-libmysqlclient-dev libmysqlclient-dev libmysqlclient21 libssl-dev libssl3 libzstd-dev
Paquetes sugeridos:
 libssl-doc
Se instalarán los siguientes paquetes NUEVOS:
 default-libmysqlclient-dev libmysqlclient-dev libmysqlclient21 libssl-dev libzstd-dev postgresql-14-mysql-fdw
Se actualizarán los siguientes paquetes:
 libssl3
1 actualizados, 6 nuevos se instalarán, 0 para eliminar y 146 no actualizados.
Se necesita descargar 5.915 kB/7.820 kB de archivos.
Se utilizarán 30,2 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] S
Des:1 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libmysqlclient21 amd64 8.0.42-0ubuntu0.22.04.1 [1.308 kB]
Des:2 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libssl-dev amd64 3.0.2-0ubuntu1.19 [2.376 kB]
Des:3 http://archive.ubuntu.com/ubuntu jammy/main amd64 libzstd-dev amd64 1.4.8+dfsg-3build1 [401 kB]
Des:4 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libmysqlclient-dev amd64 8.0.42-0ubuntu0.22.04.1 [1.702 kB]
Des:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 default-libmysqlclient-dev amd64 1.0.8 [3.586 B]
```



```
CREATE EXTENSION IF NOT EXISTS mysql_fdw;
```

2. Creamos el servidor de MySQL en nuestra base de datos.



```
CREATE SERVER mysql_server
FOREIGN DATA WRAPPER mysql_fdw
OPTIONS (host 'localhost', port '3306');
```

3. Creamos el usuario que mapeará la base de datos de MySQL.

```
CREATE USER MAPPING FOR sparkuser
SERVER mysql_server
OPTIONS (username 'root', password '1234');
```

4. Crear las tablas foráneas de las tablas de MySQL.

```
CREATE FOREIGN TABLE public.bacterias (
    id INT,
    patient_id TEXT,
    bacteria_detected TEXT,
    sample_date TIMESTAMP,
    sample_type TEXT,
) SERVER mysql_server
OPTIONS (dbname 'MedicineDB', table_name 'bacterias');
```

```
CREATE FOREIGN TABLE public.antibiotic_res (
    res_id INT,
    analitica_id INT,
    antibiotic_name TEXT,
    resistance_level TEXT
) SERVER mysql_server
OPTIONS (dbname 'MedicineDB', table_name 'antibiotic_res');
```

5. Damos permisos a nuestro usuario.

```
GRANT SELECT ON TABLE analiticas TO sparkuser;
GRANT SELECT ON TABLE antibiotic_res TO sparkuser;
```

Y con esto, Grafana con el conector de PostgreSQL podrá acceder a las tablas de MySQL y hacer consultas sin problemas como si hubieran estado en PostgreSQL desde el principio.

## 5. Visualización

El siguiente paso, tras la obtención, procesado y almacenamiento de los datos es la visualización de los mismos para identificar patrones que nos ayuden a realizar nuestra lógica de negocio (Business Intelligence) y así, realizar una toma de decisiones acorde a la información obtenida. Para ello, se han usado herramientas como Grafana o Power BI.

### 5.1. Visualización en streaming (Grafana)

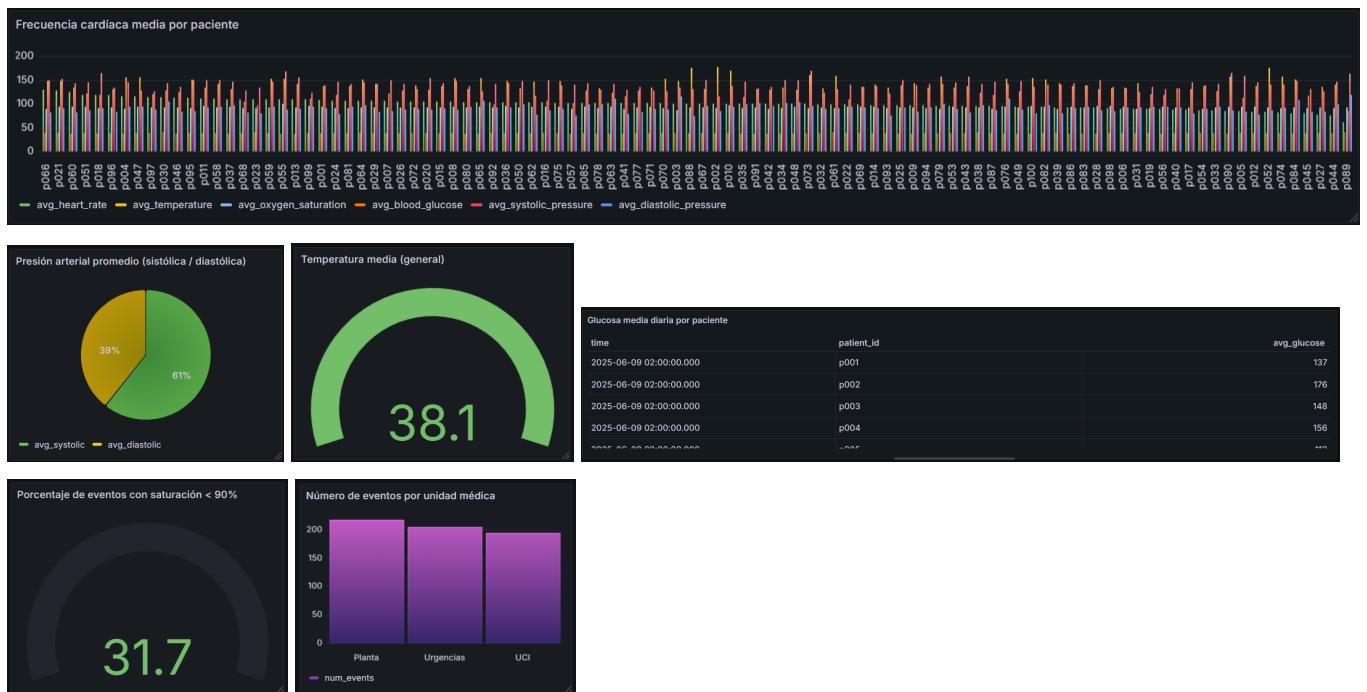
Como hemos hecho previamente, para la visualización en Grafana hemos optado por usar una base de datos como PostgreSQL. Por lo que solamente realizaremos las consultas necesarias para nuestros paneles en Grafana y ajustar el tiempo de actualización cuando estemos en streaming.

### 5.1.1 Página 1: Resumen general de constantes vitales



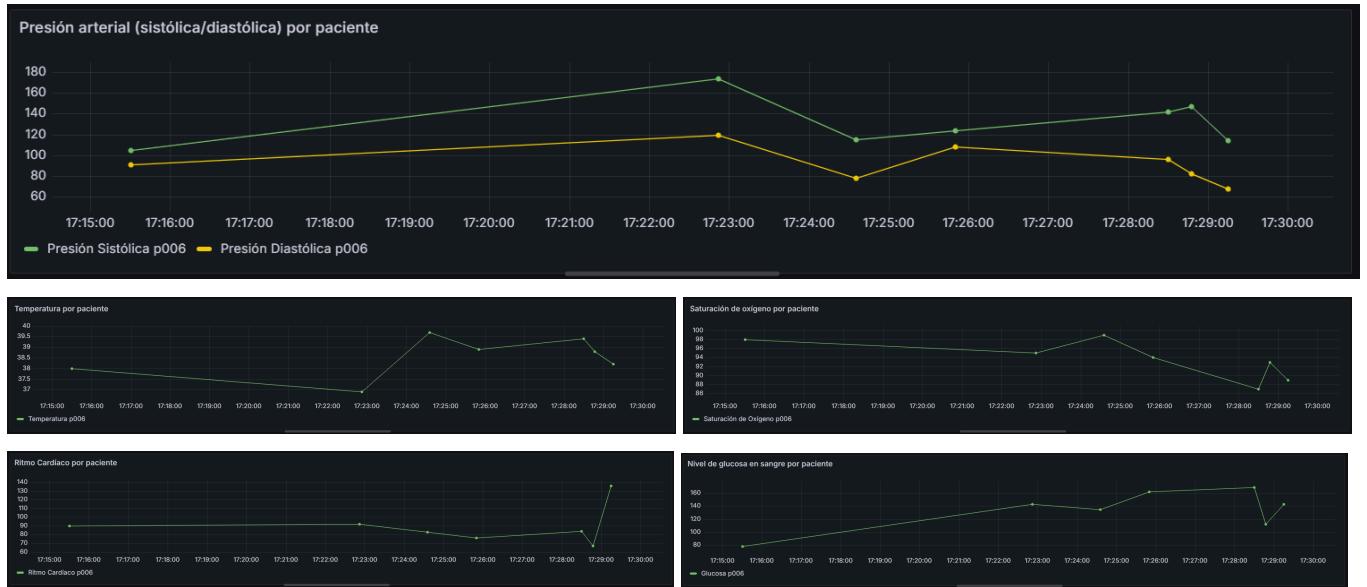
En la primera página se tiene una vista general de las constantes vitales en donde se ven como las distintas constantes vitales van oscilando a lo largo del tiempo. También hay una tabla que tiene las 20 últimas alertas recogidas. Esto nos da una vista general de cómo se están distribuyendo los datos y nos puede dar información de si en un punto se pueden disparar los valores en cierto punto, como en brotes de infecciones nosocomiales o en caso de epidemia/pandemia.

### 5.1.2 Página 2: Consultas de los datos



Aquí se realizan distintas consultas sobre los datos, se realiza una media de las constantes vitales a todos los pacientes, también tenemos la media de la presión sanguínea, temperatura, nivel de glucosa media diaria por paciente, el porcentaje de aquellos con saturación de oxígeno superior al 90% y el conteo de los datos por unidad médica.

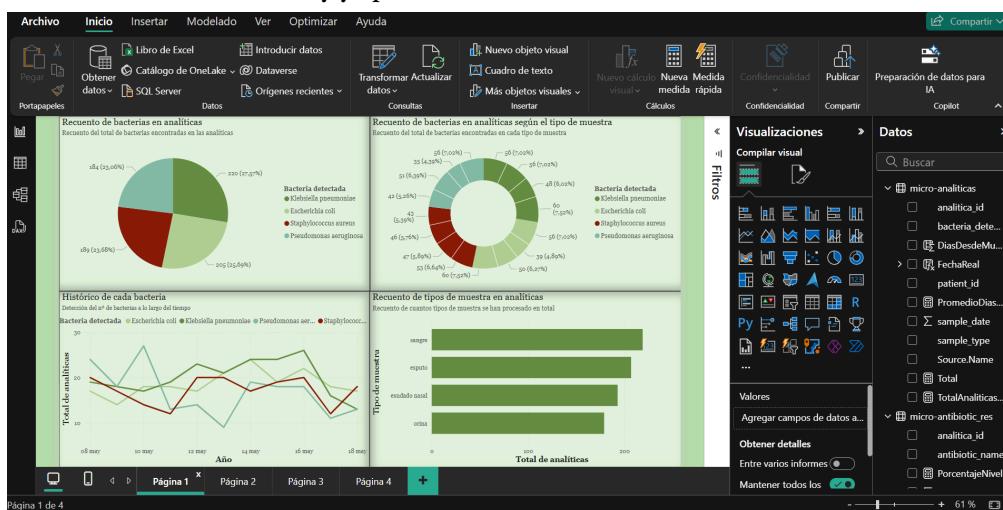
### 5.1.3 Página 3: Métricas por pacientes



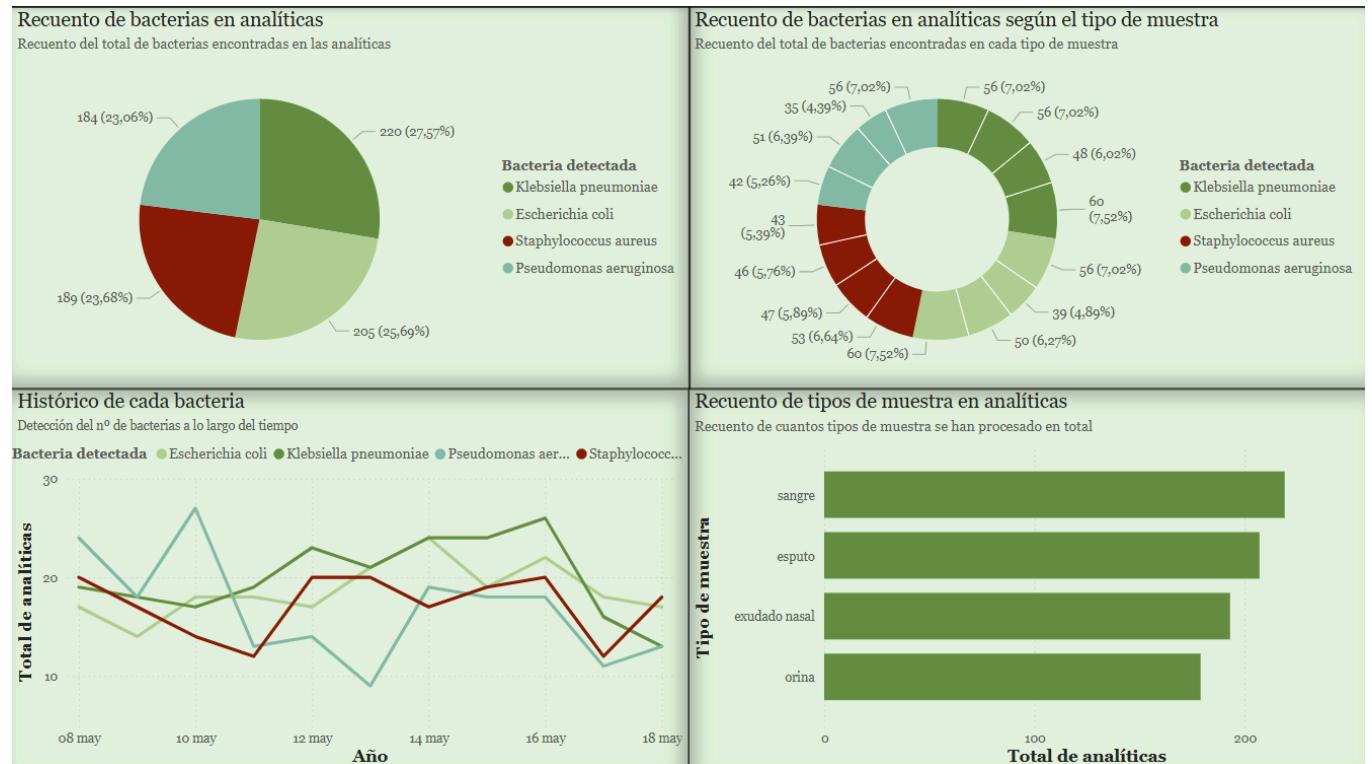
En esta página tratamos las constantes vitales por cada paciente que se puede escoger en la variable **patient\_id**. Lo cual nos ayuda a monitorizar los pacientes uno por uno e identificar si alguno tiene las constantes alteradas y su vida puede correr peligro.

## 5.2. Visualización en batch (Power BI)

Para los datos en batch usaremos el programa Power BI para la visualización, primero cargamos los datos de las analíticas y luego los datos de los antibióticos para evitar posibles conflictos. Usaremos la combinación y transformación de los datos que nos ofrece Power BI. Una vez que los datos están cargados, estos aparecerán a la derecha en la sección Datos y ya podremos montar nuestro dashboard.



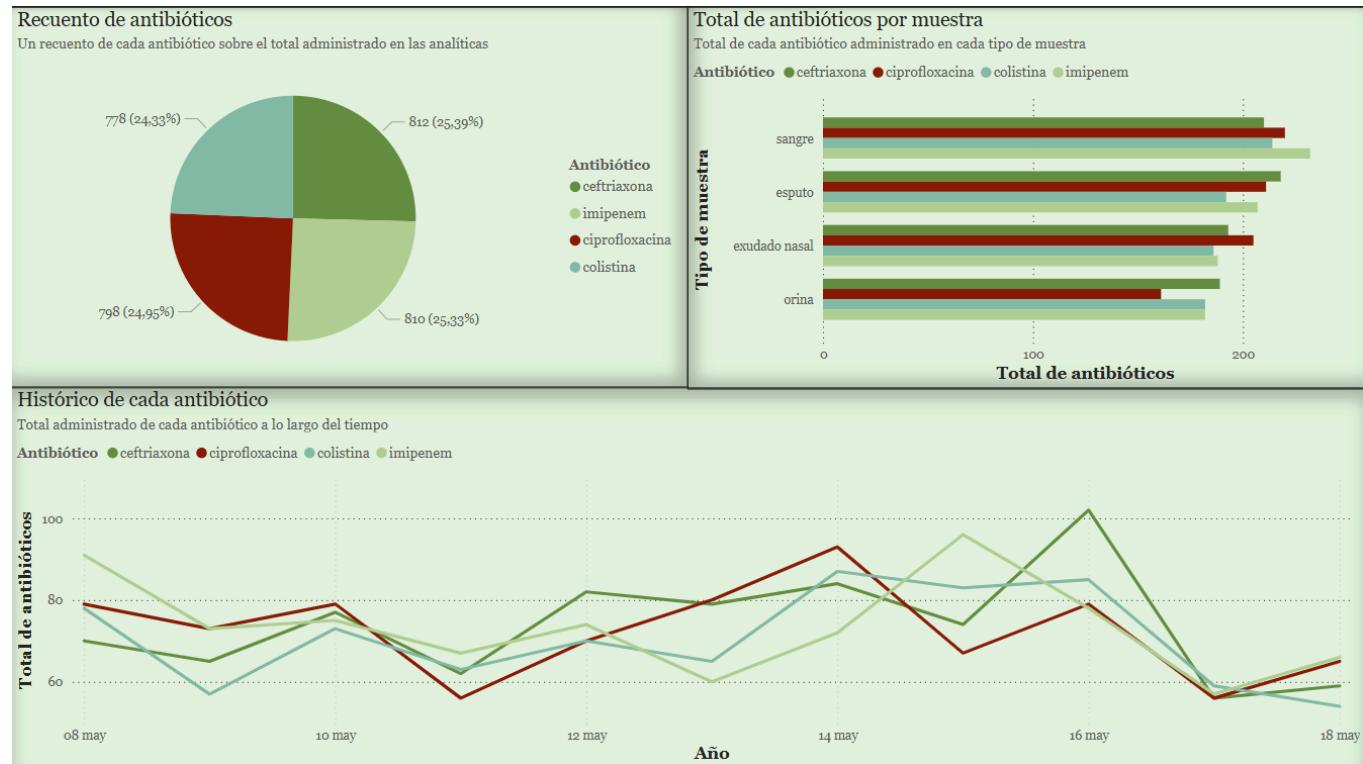
### 5.2.1 Página 1: Resumen general de analíticas



En esta primera página tenemos un resumen de las analíticas como se han distribuido las bacterias en ellas. Las métricas utilizadas son las siguientes:

- **Recuento de bacterias en analíticas:** Se realiza un recuento de las bacterias sobre el total de analíticas, lo que nos puede ayudar a identificar qué número de bacterias se han detectado en las analíticas. Se aprecia que hay un equilibrio entre todas las bacterias, siendo la Klebsiella pneumoniae la que más abunda con un 27.57% en total.
- **Recuento de bacterias en analíticas según el tipo de muestra:** Muy parecido al anterior pero ahora por tipo de muestra, lo que ayuda no solo a saber qué bacterias son más abundantes, sino también en qué tipo de muestra han aparecido con mayor regularidad. Aquí se muestra que la Klebsiella se ha detectado más en sangre, pero la Staphylococcus aureus se ha detectado más en esputo.
- **Histórico de cada bacteria:** Un histórico que recoge el número de bacterias detectadas a lo largo del tiempo, esto nos puede servir para ver en qué días ha habido un mayor número de detecciones de una bacteria en las analíticas. Tenemos que la Klebsiella pegó un pico el 16 de mayo con un total de 26 detecciones ese día, por otro lado, la Pseudomonas aeruginosa pegó un pico el 10 con 27 detecciones en total ese día.
- **Recuento de tipos de muestra en analíticas:** Aquí se cuentan cuántos tipos de muestra se han realizado en total, lo que nos indica que tipo de muestra se ha realizado más veces en las analíticas. En este caso, las muestras en sangre son el número mayoritario en las analíticas.

### 5.2.2 Página 2: Resumen general de antibióticos



La segunda página haría lo mismo que la primera pero tratando los antibióticos esta vez. Las métricas recogidas son las siguientes:

- **Recuento de antibióticos:** Es un recuento de cada antibiótico sobre el total suministrado en las analíticas. Vemos que las proporciones son casi de un 25% en todos, siendo la ceftriaxona el que más se ha suministrado.
- **Total de antibióticos por muestra:** Lo mismo que el anterior pero para cada tipo de muestra, el gráfico de barras arrojaba mejor la información que el circular por lo que se ha optado por este. Aquí vemos que antibióticos son más utilizados dependiendo del tipo de muestra tomada, como es el caso del imipenem que aparece más en muestras cuyo tipo es sangre.
- **Histórico de cada antibiótico:** Un histórico que indica cuántos antibióticos se han suministrado a lo largo del tiempo, lo que nos puede indicar que días se ha suministrado más un antibiótico. Cómo la ceftriaxona que tuvo un pico el 16 de mayo.

### 5.2.3 Página 3: Resistencia de la bacteria a los antibióticos



En la página 3 solo hay una métrica, pero es posiblemente la más importante de todas con diferencia. Consiste en el porcentaje de cada nivel de resistencia de cada bacteria para cada antibiótico según el tipo de muestra. Es muy importante ya que el nivel de resistencia a los antibióticos determina cuál de ellos se aplicará para luchar contra esa bacteria en concreto, y no es lo mismo que sea en una infección de orina que en sangre. Es por ello que, usando un segmentador, el médico podrá identificar correctamente qué bacterias presentan una resistencia mucho menor (sensible) a un antibiótico y mandarlo. En nuestro caso, tenemos 3 niveles de resistencia (habría que ajustar los niveles a la realidad):

- **sensible:** La bacteria es muy débil al antibiótico.
- **intermedio:** La bacteria está en un punto intermedio, no es ni muy débil ni muy resistente.
- **resistente:** La bacteria es muy fuerte contra el antibiótico.

Por ejemplo: Vemos que la bacteria *Staphylococcus aureus* es resistente un 28.33% a la ceftriaxona en las infecciones de orina, en estos casos la ceftriaxona no se recetaría a los pacientes ya que la bacteria no sufriría daño alguno.

### 5.2.4 Página 4: Métricas de pacientes



En esta página se han realizado una serie de métricas más enfocadas a los pacientes, las métricas en cuestión son las siguientes:

- Promedio de días desde la muestra:** Una gráfica de barras que nos indica el promedio de días que han pasado entre la muestra y la fecha actual, también separadas por tipo de muestra y bacteria detectada. Nos puede servir para saber si existe algún tipo de retraso en el diagnóstico de las analíticas en cuestión.
- Dispersión de muestras por paciente:** Una gráfica de dispersión que recoge de cada paciente todas las analíticas que se le han realizado y qué bacterias han sido detectadas, de las cuales se indican cuáles han resultado resistentes a determinados antibióticos. Esto nos sirve para visualizar patrones de resistencia a nivel individual, identificar posibles infecciones recurrentes o persistentes, evaluar la evolución de la resistencia bacteriana en cada paciente a lo largo del tiempo y facilitar la toma de decisiones clínicas más informadas respecto al tratamiento antibiótico más adecuado.
- Total de analíticas por paciente:** Una gráfica donde se ve cuántas analíticas se han realizado a cada paciente, distinguiendo por el tipo de muestra. Esto nos puede ayudar a hacer un historial clínico de cada paciente, identificando patrones en la frecuencia de análisis, evaluando la carga asistencial por paciente y detectando si ha habido un seguimiento intensivo o no. Además, podemos ver qué tipos de muestras son más frecuentes en los pacientes, lo que ayuda a identificar condiciones clínicas específicas o necesidades particulares de monitoreo.

## 5.2.5 Página 5: Tabla detallada de las analíticas

Detalles de las analíticas								
Tabla detallada de las analíticas realizadas en total								
analitica_id	bacteria_detected	patient_id	sample_type	Año	Mes	Día	antibiotic_name	resistance_level
1	Escherichia coli	p042	exudado nasal	2025	mayo	18	ceftriaxona	resistente
1	Escherichia coli	p042	exudado nasal	2025	mayo	18	ceftriaxona	sensible
1	Escherichia coli	p042	exudado nasal	2025	mayo	18	colistina	intermedio
1	Escherichia coli	p042	exudado nasal	2025	mayo	18	imipenem	intermedio
2	Pseudomonas aeruginosa	p063	espulo	2025	mayo	14	ceftriaxona	sensible
2	Pseudomonas aeruginosa	p063	espulo	2025	mayo	14	ciprofloxacina	intermedio
2	Pseudomonas aeruginosa	p063	espulo	2025	mayo	14	colistina	intermedio
3	Klebsiella pneumoniae	p019	sangre	2025	mayo	15	ceftriaxona	sensible
3	Klebsiella pneumoniae	p019	sangre	2025	mayo	15	colistina	resistente
3	Klebsiella pneumoniae	p019	sangre	2025	mayo	15	imipenem	intermedio
3	Klebsiella pneumoniae	p019	sangre	2025	mayo	15	imipenem	sensible
4	Staphylococcus aureus	p047	orina	2025	mayo	14	ceftriaxona	resistente
4	Staphylococcus aureus	p047	orina	2025	mayo	14	ciprofloxacina	resistente
4	Staphylococcus aureus	p047	orina	2025	mayo	14	ciprofloxacina	sensible
5	Escherichia coli	p011	espulo	2025	mayo	14	colistina	intermedio
5	Escherichia coli	p011	espulo	2025	mayo	14	colistina	sensible
5	Escherichia coli	p011	espulo	2025	mayo	14	imipenem	intermedio
6	Pseudomonas aeruginosa	p019	espulo	2025	mayo	18	ciprofloxacina	intermedio
6	Pseudomonas aeruginosa	p019	espulo	2025	mayo	18	colistina	intermedio
6	Pseudomonas aeruginosa	p019	espulo	2025	mayo	18	imipenem	sensible
7	Pseudomonas aeruginosa	p058	sangre	2025	mayo	10	ciprofloxacina	sensible
7	Pseudomonas aeruginosa	p058	sangre	2025	mayo	10	colistina	resistente
7	Pseudomonas aeruginosa	p058	sangre	2025	mayo	10	imipenem	intermedio
7	Pseudomonas aeruginosa	p058	sangre	2025	mayo	10	imipenem	resistente
8	Staphylococcus aureus	p048	sangre	2025	mayo	14	ciprofloxacina	resistente
8	Staphylococcus aureus	p048	sangre	2025	mayo	14	ciprofloxacina	sensible
8	Staphylococcus aureus	p048	sangre	2025	mayo	14	colistina	sensible
8	Staphylococcus aureus	p048	sangre	2025	mayo	14	imipenem	resistente
9	Klebsiella pneumoniae	p025	orina	2025	mayo	8	ceftriaxona	sensible
9	Klebsiella pneumoniae	p025	orina	2025	mayo	8	ciprofloxacina	intermedio
9	Klebsiella pneumoniae	p025	orina	2025	mayo	8	ciprofloxacina	sensible
9	Klebsiella pneumoniae	p025	orina	2025	mayo	8	colistina	sensible
10	Escherichia coli	p077	exudado nasal	2025	mayo	16	ceftriaxona	resistente
10	Escherichia coli	p077	exudado nasal	2025	mayo	16	ciprofloxacina	intermedio
10	Escherichia coli	p077	exudado nasal	2025	mayo	16	ciprofloxacina	sensible

Por último, se ha realizado una tabla detallada de todas las analíticas que recogen la bacteria detectada en la analítica, el paciente al que se le ha realizado la analítica, el tipo de muestra de la analítica, la fecha en la que se realizó, el antibiótico administrado y la resistencia de la bacteria a dicho antibiótico. En el momento de la captura no se habían puesto segmentadores de datos, ahora la tabla se puede segmentar por bacteria detectada, tipo de muestra, antibiótico administrado y resistencia de la bacteria.

## 5.3. Visualización streaming + batch

### 5.3.1 Página 4: Constantes vitales por bacteria

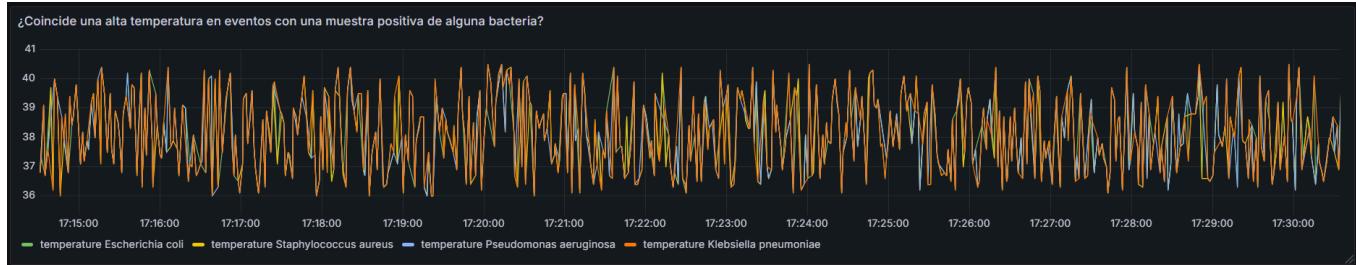


Esta página es un versionado de la primera pero se puede filtrar según las bacterias de las analíticas, esto nos puede ayudar a ver si las bacterias afectan significativamente las constantes vitales de los pacientes que están infectados y así poder identificar síntomas posibles que causen dichas bacterias.

### 5.3.2 Página 5: Consultas mixtas

Pacientes con glucosa elevada en eventos recientes con bacterias resistentes						
patient_id	timestamp		blood_glucose	bacteria_detected	antibiotic_name	resistance_level
p082	2025-06-09 17:31:00.000		190	Klebsiella pneumoniae	imipenem	resistente
p074	2025-06-09 17:30:58.000		154	Klebsiella pneumoniae	colistina	resistente
p074	2025-06-09 17:30:58.000		154	Klebsiella pneumoniae	ciprofloxacina	resistente
p025	2025-06-09 17:30:53.000		154	Klebsiella pneumoniae	imipenem	resistente
p020	2025-06-09 17:30:52.000		146	Klebsiella pneumoniae	imipenem	resistente
p020	2025-06-09 17:30:52.000		146	Klebsiella pneumoniae	ciprofloxacina	resistente

Pacientes que tienen signos vitales alterados y han recibido diagnósticos microbiológicos complejos								
patient_id	heart_rate	temperature	oxygen_saturation	systolic_pressure	diastolic_pressure	blood_glucose	recuento_bacterias	
p080	135	36.8	90	162	76	144	8	
p027	96	40	99	126	83	100	9	
p006	136	38.2	89	114	68	143	13	
p059	64	37.4	99	138	93	188	6	
p032	66	38.4	93	119	85	136	13	
p080	75	37.8	92	135	75	146	8	



Aquí se realizaron consultas con ambos tipos de datos, como si el nivel de glucosa en sangre se relaciona con bacterias resistentes, pacientes con signos vitales alterados recibieron analíticas complejas o si alta temperatura es signo de padecer una bacteria mediante si la muestra dió positivo. Esto nos ayuda a ver si existe realmente una relación entre datos en streaming y batch de manera sencilla.

## 6. Business Intelligence

A continuación, se realizará el Business Intelligence sobre las métricas obtenidas para realizar una toma de decisiones adecuada para, en este caso, el hospital que tenga estos datos. Para ello, el BI se hará en un formato de preguntas y respuestas sobre las métricas obtenidas y el conocimiento que podemos extraer de él.

### 6.1. BI de datos en streaming

En este apartado trataremos el BI de los datos obtenidos en streaming cuya visualización ha sido realizada anteriormente en Grafana.

#### 6.2.1 Análisis General de Constantes Vitales

##### 1. ¿Qué tipo de alertas médicas son las más frecuentes y en qué unidades se están generando principalmente?

- Las alertas de **HIPERTENSIÓN** son las más frecuentes en el panel de alertas recientes. Se están generando principalmente en la **UCI** y **Urgencias**, seguidas por Planta.

2. ¿Existen patrones de volatilidad significativos en alguna de las métricas vitales (ritmo cardíaco, presión arterial, glucosa, saturación de oxígeno, temperatura) que puedan indicar un riesgo o una necesidad de atención especial?
  - Sí, los **niveles de glucosa en sangre** y la **temperatura corporal** muestran una **gran volatilidad**, con fluctuaciones amplias que incluyen picos altos (posible hiperglucemia, fiebre) y valles bajos. El **ritmo cardíaco** y la **presión arterial** también son dinámicos, mientras que la **saturación de oxígeno** es relativamente más estable, aunque sus caídas son clínicamente importantes.
3. ¿Podríamos identificar a pacientes con métricas que consistentemente se encuentren fuera de rangos saludables o que muestren tendencias preocupantes, incluso antes de que se genere una alerta explícita?
  - Sí, aunque las alertas se activan por umbrales, el análisis de las tendencias en los gráficos (por ejemplo, una presión sistólica que, aunque no alcanza el umbral de alerta, se mantiene en el límite superior, o la glucosa con variaciones muy abruptas) podría permitir la identificación proactiva de pacientes en riesgo. Sin embargo, para esto se requeriría un análisis más sofisticado de los datos en streaming.
4. ¿Cómo podemos utilizar la combinación de estas métricas en streaming para mejorar la eficiencia y la calidad de la atención hospitalaria?
  - La combinación de estas métricas permite una **monitorización integral** y en **tiempo real** de los pacientes. Esto posibilita la **detección temprana** de deterioros, la **intervención médica oportuna** antes de que las condiciones se agraven, y la **optimización en la asignación de recursos** al identificar las unidades o pacientes con mayor necesidad de atención (por ejemplo, aquellos con alertas frecuentes o métricas muy volátiles). A largo plazo, estos datos podrían alimentar **sistemas predictivos** para anticipar complicaciones.

### 6.2.2 Análisis de las consultas

1. ¿Cuál es la temperatura media general de los pacientes, y qué porcentaje de eventos muestran una saturación de oxígeno baja?
  - La temperatura media general es de 38.1 grados Celsius. Además, un 31.7% de los eventos registrados presentan una saturación de oxígeno por debajo del 90%, lo cual es un dato relevante para la atención médica.
2. ¿Cómo se distribuye la presión arterial promedio entre sistólica y diastólica en todos los eventos monitoreados?
  - En promedio, la presión sistólica representa el 61% y la presión diastólica el 39% de la lectura total de la presión arterial.
3. ¿Cuál es la unidad médica con mayor actividad o número de eventos registrados, y cómo se compara con las otras unidades?
  - La unidad de "Planta" (Hospitalización) registra el mayor número de eventos, seguida de cerca por "Urgencias" y finalmente "UCI". Esto indica que "Planta" es la unidad más activa en cuanto a la monitorización de signos vitales.

4. ¿Podemos identificar patrones en las métricas vitales (frecuencia cardíaca, temperatura, saturación de oxígeno, glucosa, presión arterial) por paciente, y es posible ver la glucosa diaria promedio de cada uno?
- Sí, es posible observar las métricas vitales promedio de cada paciente individualmente. La gráfica muestra que algunos pacientes, especialmente aquellos hacia el final del listado (como P080 al P088), tienden a tener una frecuencia cardíaca promedio más alta. Adicionalmente, se puede consultar una tabla que detalla la glucosa promedio diaria para cada paciente, permitiendo un seguimiento específico de sus niveles de glucosa a lo largo del tiempo.

### 6.2.3 Análisis de constantes de un paciente (p006)

1. ¿Ha habido alguna tendencia preocupante o cambio agudo en la presión arterial del paciente p006 durante el periodo de tiempo analizado?
  - Sí. La presión sistólica del paciente p006 mostró una tendencia general al aumento desde alrededor de 105 mmHg a las 17:15 hasta un pico de casi 170 mmHg a las 17:23, para luego descender bruscamente a las 17:25 y volver a subir. La presión diastólica siguió un patrón similar. La caída significativa de ambas presiones alrededor de las 17:25, seguida de una nueva elevación y una caída brusca cerca de las 17:29, sugiere episodios de alta variabilidad que podrían requerir atención médica inmediata.
2. ¿Cómo ha evolucionado la temperatura corporal del paciente p006, y hay indicios de fiebre o hipotermia?
  - La temperatura corporal del paciente p006 comenzó alrededor de 38°C, descendió a un mínimo de 37°C alrededor de las 17:22:30, y luego experimentó un aumento significativo y rápido a casi 40°C alrededor de las 17:24:30. Posteriormente, se mantuvo por encima de los 38°C con fluctuaciones, descendiendo a 38.2°C al final del período. Este patrón indica un episodio de fiebre aguda alrededor de las 17:24:30 que luego se estabilizó a valores febriles.
3. ¿Existe alguna preocupación sobre la saturación de oxígeno del paciente p006?
  - Sí, hay un patrón preocupante en la saturación de oxígeno. Aunque comenzó alrededor del 98%, mostró una tendencia a la baja después de las 17:24:30, cayendo significativamente por debajo del 90% (a 87%) alrededor de las 17:28:30. Aunque se recuperó ligeramente al final del período, esta caída por debajo de los niveles óptimos (generalmente 90-95% es un umbral de preocupación) indica un episodio de hipoxia o dificultad respiratoria, lo cual es una señal crítica.
4. ¿Cómo se correlacionan las métricas de Ritmo Cardíaco y Glucosa del paciente p006 con los cambios observados en la presión arterial, saturación de oxígeno y temperatura?
  - **Ritmo Cardíaco:** El ritmo cardíaco se mantuvo relativamente estable alrededor de 90 lpm hasta las 17:24:30. Sin embargo, en el mismo período en que la temperatura subió y la saturación de oxígeno comenzó a bajar (alrededor de las 17:28:30), el ritmo cardíaco experimentó una caída brusca a 65 lpm y luego un aumento explosivo a casi 135 lpm cerca de las 17:29:30. Esto sugiere una posible respuesta compensatoria a los cambios en la temperatura y la oxigenación, indicando un posible estrés fisiológico.
  - **Glucosa:** Los niveles de glucosa mostraron una tendencia general al alza desde 80 mg/dL a las 17:15 hasta un pico de alrededor de 165 mg/dL a las 17:28:30. Esta elevación sostenida, combinada con la caída de la saturación de oxígeno y el aumento de la temperatura, podría indicar un estado de estrés metabólico o una respuesta fisiológica compleja. La caída abrupta de la glucosa justo después de las 17:28:30 y su posterior recuperación también es un punto de interés.

## 6.2. BI de datos en batch

En este apartado trataremos el BI de los datos obtenidos en batch cuya visualización ha sido realizada anteriormente en Power BI.

### 6.2.1 Análisis General de Antibiogramas y Bacterias

1. **¿Cuáles son los antibióticos más recetados y las bacterias más comúnmente detectadas?**
  - Según el gráfico "Recuento de antibióticos", los antibióticos se recetan de manera bastante equitativa, con Cefriaxona e Imipenem ligeramente por encima del 25% cada uno (aproximadamente 25.39% y 25.33% respectivamente). Ciprofloxacina y Colistina representan alrededor del 24.95% y 24.33%. En cuanto a las bacterias, "Recuento de bacterias en analíticas" muestra que Escherichia coli es la más frecuente (27.57%), seguida de Klebsiella pneumoniae (25.69%), Staphylococcus aureus (23.68%) y Pseudomonas aeruginosa (23.06%).
2. **¿Existe alguna tendencia en el uso de antibióticos o la detección de bacterias a lo largo del tiempo?**
  - El gráfico "Histórico de cada antibiótico" muestra fluctuaciones diarias en el total de antibióticos administrados. No hay una tendencia clara y sostenida de aumento o disminución para ningún antibiótico en particular en el período mostrado (del 8 al 18 de mayo). Lo mismo aplica para el "Histórico de cada bacteria", donde la detección de bacterias también varía día a día sin una tendencia definida. Esto podría indicar que las infecciones son esporádicas o que los patrones de detección son sensibles a factores externos diarios.

### 6.2.2 Relación entre Antibióticos, Bacterias y Tipo de Muestra

1. **¿Qué antibióticos se administran con mayor frecuencia para cada tipo de muestra (sangre, esputo, exudado nasal, orina)?**
  - Observando el gráfico "Total de antibióticos por muestra", se puede apreciar que para todas las muestras (sangre, esputo, exudado nasal, orina), la administración de los cuatro antibióticos (Cefriaxona, Ciprofloxacina, Colistina, Imipenem) es relativamente similar, aunque con ligeras variaciones. No hay un antibiótico que domine claramente en un solo tipo de muestra, lo que sugiere un enfoque de amplio espectro o la necesidad de ajustar el tratamiento según el antibiograma.
2. **¿Qué bacterias se detectan con mayor frecuencia en cada tipo de muestra?**
  - El gráfico "Recuento de bacterias en analíticas según el tipo de muestra" proporciona esta información. Por ejemplo, en sangre, Klebsiella pneumoniae, Escherichia coli y Pseudomonas aeruginosa son las más detectadas. En esputo, Staphylococcus aureus. Estas distribuciones pueden guiar el tratamiento empírico antes de tener resultados de antibiogramas.

### 6.2.3 Niveles de Resistencia a Antibióticos

#### 1. ¿Cuál es el nivel de resistencia de cada bacteria a los diferentes antibióticos, desglosado por tipo de muestra?

- Los gráficos de "Nivel de resistencia de bacterias en cada tipo de muestra" son cruciales aquí. Permiten identificar combinaciones específicas de bacteria-antibiótico con altos niveles de resistencia. Por ejemplo, en "orina", Escherichia coli muestra una resistencia considerable a Ciprofloxacina (barra roja más alta) y también a Ceftriaxona. En "exudado nasal", Klebsiella pneumoniae presenta alta resistencia a Ciprofloxacina y Colistina. Esta información es vital para la toma de decisiones clínicas y la formulación de guías de tratamiento.

#### 2. ¿Cuáles son las combinaciones bacteria-antibiótico con mayor resistencia?

- De los gráficos de resistencia, es evidente que la resistencia varía significativamente. Por ejemplo, se observa que:
  - **Escherichia coli:** Muestra una resistencia alta a la ciprofloxacina en esputo y orina. También presenta resistencia intermedia a la ceftriaxona en sangre y exudado nasal.
  - **Klebsiella pneumoniae:** Alta resistencia a la ciprofloxacina y colistina en exudado nasal, a la ciprofloxacina en orina y al imipenem en sangre.
  - **Pseudomonas aeruginosa:** Presenta una resistencia notable a la colistina y ceftriaxona en exudado nasal, orina y sangre.
  - **Staphylococcus aureus:** Tiene resistencia significativa a la ciprofloxacina en todas las muestras, y también a la colistina en esputo.
- Identificar estas combinaciones es fundamental para priorizar el desarrollo de nuevos antibióticos, la vigilancia de la resistencia y la optimización de los protocolos de tratamiento.

### 6.2.4 Información del Paciente y Procesamiento de Muestras

#### 1. ¿Cuál es el tiempo promedio transcurrido desde la toma de la muestra hasta la fecha actual, por tipo de muestra y bacteria detectada?

- El gráfico "Promedio de días desde la muestra" indica que, en promedio, el tiempo desde la toma de la muestra hasta la fecha actual es similar para todas las bacterias y tipos de muestra, rondando los 10-12 días. Esto puede ser relevante para evaluar la rapidez de los procesos de laboratorio.

#### 2. ¿Cómo se distribuyen las analíticas por paciente y tipo de muestra?

- El gráfico "Total de analíticas por paciente" muestra la cantidad y tipo de muestras procesadas para cada paciente. Algunos pacientes (ej. p099, p064, p093) tienen un mayor número de analíticas, con una mezcla de tipos de muestra. Otros pacientes tienen menos analíticas o se centran en un solo tipo de muestra. Esta información puede ayudar a identificar pacientes con infecciones recurrentes o más complejas.

#### 3. ¿Hay pacientes con múltiples muestras y resistencia detectada?

- El gráfico "Dispersión de muestras por paciente" muestra la cantidad de analíticas y las bacterias que presentan resistencia. Los puntos de mayor tamaño (que indican más analíticas) y los colores que corresponden a bacterias resistentes, pueden ayudar a identificar pacientes con infecciones persistentes o que desarrollan resistencia. Por ejemplo, si el paciente p002 tiene puntos grandes y de colores asociados a bacterias resistentes, podría ser un caso de interés.

### 6.2.5 Oportunidades de Mejora y Estrategias

1. ¿Qué estrategias se podrían implementar para combatir la resistencia a los antibióticos basándose en estos datos?

- **Guías de tratamiento actualizadas:** Basadas en los patrones de resistencia, se pueden crear guías que recomiendan antibióticos de primera línea con menor resistencia.
- **Programas de stewardship de antibióticos:** Fomentar el uso racional de antibióticos, evitando el uso de aquellos a los que se ha demostrado una alta resistencia en bacterias comunes.
- **Vigilancia activa:** Continuar monitoreando los patrones de resistencia para detectar nuevas cepas resistentes o el aumento de la resistencia en cepas existentes.
- **Investigación y desarrollo:** Los datos pueden señalar la necesidad de nuevos antibióticos para bacterias con altos niveles de resistencia a los tratamientos actuales.
- **Mejora de la higiene y prevención de infecciones:** Reducir la propagación de bacterias resistentes a través de medidas de control de infecciones.

2. ¿Qué información adicional sería útil para un análisis más profundo de la inteligencia de negocio?

- **Datos demográficos de los pacientes:** Edad, sexo, comorbilidades.
- **Origen de la infección:** Comunitaria o nosocomial.
- **Resultados clínicos:** Éxito del tratamiento, duración de la estancia hospitalaria.
- **Costos asociados:** Costo de los antibióticos, costo de la atención hospitalaria para pacientes con resistencia.
- **Datos de consumo de antibióticos:** Cantidades exactas administradas para cada antibiótico.
- **Historial de tratamientos previos:** Para entender la exposición previa a antibióticos.

## 6.3. BI de datos en streaming + batch

En este apartado trataremos el BI de los datos obtenidos en streaming y batch juntos cuya visualización ha sido realizada anteriormente en Grafana.

### 6.3.1 Constantes vitales según bacterias

1. ¿Qué patrones de salud podemos identificar al correlacionar las métricas vitales con las bacterias detectadas?

- Podemos identificar patrones de deterioro o mejora. Por ejemplo, en presencia de bacterias como *Klebsiella pneumoniae*, observamos fluctuaciones significativas en glucosa (picos altos), ritmo cardíaco (taquicardia frecuente), y temperatura (episodios de fiebre), mientras que la saturación de oxígeno puede tener caídas ocasionales y la presión sanguínea muestra amplia variabilidad. Estos patrones son indicativos de la respuesta del cuerpo a la infección.

2. ¿Cómo pueden estos datos en streaming apoyar la toma de decisiones clínicas y la asignación de recursos?

- Los datos permiten una monitorización continua y la detección temprana de anomalías en los signos vitales, lo que es crucial para intervenir rápidamente en casos de infecciones. Al identificar las unidades o pacientes con mayor incidencia de alertas bacterianas (como la unidad de Urgencias y el paciente p013), se pueden optimizar la asignación de personal, equipos y camas, mejorando la eficiencia y la calidad de la atención.

**3. ¿Podemos desarrollar un sistema de alerta temprana basado en estas métricas?**

- Sí, al establecer umbrales específicos para cada métrica vital (temperatura, ritmo cardíaco, saturación de oxígeno, etc.), se pueden generar alertas automáticas cuando se superan estos límites. La combinación de múltiples signos vitales alterados podría aumentar la precisión de estas alertas, permitiendo una intervención proactiva antes de que la condición del paciente se deteriore gravemente debido a una infección.

**4. ¿Qué oportunidades de mejora en la gestión de infecciones ofrecen estos datos?**

- Al analizar las bacterias más comunes y su impacto específico en los signos vitales, se pueden refinar los protocolos de tratamiento antibiótico y las estrategias de control de infecciones. Además, la capacidad de predecir la probabilidad de desarrollar una infección o el empeoramiento de una condición mediante el análisis de tendencias en los signos vitales abre la puerta a intervenciones preventivas y personalizadas.

**6.3.2 Algunas consultas interesantes****1. ¿Qué información crucial obtenemos de los pacientes con glucosa elevada y bacterias resistentes?****a. ¿Cuál es la tendencia de glucosa en pacientes con bacterias resistentes?**

- La tabla muestra que varios pacientes tienen niveles de glucosa elevados (superiores a 140), algunos alcanzando hasta 190. Esto sugiere que la presencia de bacterias resistentes podría estar asociada con desregulaciones de la glucosa, lo cual es crítico en el manejo de infecciones y sepsis, donde la hiperglucemía puede empeorar los resultados.

**b. ¿Qué bacterias resistentes y antibióticos están más frecuentemente involucrados?**

- La bacteria *Klebsiella pneumoniae* es consistentemente detectada como resistente a antibióticos como Imipenem y Ciprofloxacina, y también a Colistina. Esto es una preocupación importante, ya que *Klebsiella pneumoniae* resistente a estos antibióticos de amplio espectro representa un desafío terapéutico significativo.

**c. ¿Existe algún patrón temporal en la detección de estas resistencias y glucosa elevada?**

- Todos los registros mostrados son del 9 de junio de 2025, concentrados en un lapso de minutos (17:30-17:31). Esto podría indicar una situación de alerta puntual o un proceso de carga de datos específico. Es fundamental monitorear si estos eventos son aislados o si representan una tendencia creciente de resistencia bacteriana.

**2. ¿Cómo podemos identificar y caracterizar a los pacientes con signos vitales alterados y diagnósticos microbiológicos complejos?****a. ¿Qué signos vitales están más consistentemente alterados en pacientes con múltiples detecciones bacterianas?**

- Observamos pacientes con ritmos cardíacos elevados (ej., p080 con 135 bpm, p006 con 136 bpm), temperaturas elevadas (ej., p027 con 40°C, p006 con 38.2°C), saturación de oxígeno comprometida (ej., p080 con 90%, p006 con 89%), y niveles de glucosa altos (ej., p059 con 188, p080 con 144). La combinación de estos factores indica un estado de salud comprometido.

- b. ¿Existe una correlación entre el número de bacterias detectadas y la severidad de las alteraciones en los signos vitales?
- Aunque el "recuento\_bacterias" es de 6 a 13 en los ejemplos, no hay una correlación lineal obvia inmediata con la severidad de las alteraciones. Por ejemplo, p027 tiene 9 bacterias y una temperatura de 40°C, mientras que p006 tiene 13 bacterias y una temperatura de 38.2°C. Se necesitaría un análisis estadístico más profundo para determinar si un mayor recuento bacteriano se asocia con alteraciones más graves en los signos vitales.
- c. ¿Qué implicaciones tiene el diagnóstico de múltiples bacterias para el tratamiento de estos pacientes?
- La detección de múltiples bacterias sugiere una coinfección o una infección polimicrobiana, lo que complica significativamente el tratamiento. Podría requerir el uso de antibióticos de amplio espectro, una vigilancia más intensiva y un enfoque multidisciplinario en la atención al paciente.
3. ¿Coinciden las altas temperaturas con los eventos de muestras positivas de alguna bacteria, y qué bacterias están más frecuentemente asociadas con la fiebre?
- a. ¿Qué rango de temperatura se observa en pacientes con muestras bacterianas positivas?
- El gráfico de temperatura muestra fluctuaciones constantes entre aproximadamente 36°C y 40.5°C en la presencia de *Escherichia coli*, *Staphylococcus aureus*, *Pseudomonas aeruginosa*, y *Klebsiella pneumoniae*. Esto confirma que las temperaturas elevadas (fiebre) son una respuesta común a la infección por estas bacterias.
- b. ¿Existen diferencias notables en los patrones de temperatura entre las distintas bacterias?
- A simple vista, los patrones de temperatura para las diferentes bacterias son muy similares y se superponen. Esto sugiere que, para estas bacterias, la respuesta febril es una característica general de la infección, y no hay una fluctuación de temperatura distintiva que las separe claramente basándose únicamente en este gráfico.
4. ¿Qué valor de negocio o clínico adicional se puede extraer de la combinación de estos conjuntos de datos?
- a. ¿Podemos predecir la resistencia a antibióticos basándonos en la evolución de los signos vitales y los niveles de glucosa?
- Si se observa un patrón de signos vitales persistentemente alterados (ej. fiebre continua, glucosa elevada) a pesar de la administración de antibióticos específicos, y esto se correlaciona con la posterior detección de resistencia, se podría desarrollar un modelo predictivo temprano de resistencia. Esto permitiría ajustar el tratamiento de manera más rápida y efectiva.
- b. ¿Cómo podemos usar esta información para mejorar los protocolos de vigilancia epidemiológica y control de infecciones?
- Al rastrear la incidencia de bacterias resistentes y su impacto en los signos vitales de los pacientes, los hospitales pueden identificar brotes, evaluar la efectividad de las medidas de control de infecciones y adaptar sus políticas de uso de antibióticos para reducir la propagación de la resistencia.

- c. ¿Qué implicaciones tiene la alta incidencia de pacientes con múltiples bacterias y signos vitales alterados para la gestión hospitalaria?
- Indica la necesidad de protocolos robustos para el diagnóstico rápido y preciso de infecciones complejas. También subraya la importancia de tener acceso rápido a resultados de antibiogramas para guiar el tratamiento, y posiblemente la necesidad de unidades especializadas o equipos multidisciplinares para manejar estos casos complejos de manera eficiente.

## 7. Despliegue Completo

A continuación se realizará un resumen del despliegue completo de nuestro sistema de big data para que cualquiera no tenga dudas al momento de desplegarlo.

### 7.1 Inicio del sistema

Lo primero será lanzar los servicios de HDFS Hadoop, YARN y Apache Spark. Abriremos una ventana en la terminal accediendo con SSH y ejecutaremos estos comandos:

```
● ● ●

start-dfs.sh
start-yarn.sh
./sbin/start-master.sh (en $SPARK_HOME)
./sbin/start-workers.sh (en $SPARK_HOME)
```

Para ahorrar el proceso, se ha realizado un `init_system.sh` y un `stop_system.sh` que realizará estos pasos automáticamente.

### 7.2 Inicio de Kafka

Lo siguiente será iniciar Kafka, para ello abrimos otra ventana y la dividimos en 3 partes y ejecutamos los siguientes comandos:

```
● ● ●

bin/kafka-server-start.sh /opt/proyecto/config/controller1.properties
bin/kafka-server-start.sh /opt/proyecto/config/broker1.properties
bin/kafka-server-start.sh /opt/proyecto/config/broker2.properties
```

Si es la primera vez, debemos formatear el clúster. Para ello usaremos estos comandos:

```
● ● ●

#Genera un cluster UUID y controller UUID
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
CONTROLLER_1_UUID="$(bin/kafka-storage.sh random-uuid)"
echo $KAFKA_CLUSTER_ID

#Formateamos los directorios de log
bin/kafka-storage.sh format --cluster-id ${KAFKA_CLUSTER_ID} --initial-controllers "192.168.18.8:9093:${CONTROLLER_1_UUID}" --config /opt/proyecto/config/controller1.properties
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c /opt/proyecto/config/broker1.properties
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c /opt/proyecto/config/broker2.properties
```

En caso de no tener los topics creados, aquí están los comandos para crear todos los topics usados en el proyecto. Se debe hacer en la carpeta de Kafka.

```
bin/kafka-topics.sh --create --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --partitions 2 --topic vitales_pacientes
bin/kafka-topics.sh --create --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --partitions 2 --topic micro-analiticas
bin/kafka-topics.sh --create --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --partitions 2 --topic micro-antibiotic_res
bin/kafka-topics.sh --create --bootstrap-server 192.168.18.8:9094 --replication-factor 2 --partitions 2 --topic alertas_pacientes
```

## 7.3 Ingesta en streaming

Primero empezaremos con la ingestá en streaming. Primero ejecutaremos el tercer worker creado que se usa solamente para esta ingestá.

```
bin/connect-distributed.sh /opt/proyecto/config/worker3.properties
```

Y esperamos a que se carguen los plugins, una vez se han cargado todos ejecutaremos primero el consumer de Spark y luego el producer.

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.4 --master spark://192.168.18.8:7077 /opt/proyecto/streaming_spark.py
python3 /opt/proyecto/syn_data/gen_stream.py
```

Podemos comprobar que lo que le llega al topic con el siguiente comando:

```
bin/kafka-console-consumer.sh --topic vitales_pacientes --from-beginning --bootstrap-server 192.168.18.8:9094
```

Si se desea borrar un topic, se puede hacer con el siguiente comando:

```
bin/kafka-topics.sh --delete --topic alertas_pacientes --bootstrap-server 192.168.18.8:9094
```

Una vez vemos que Spark está recibiendo correctamente los datos, agregamos nuestro conector de PostgreSQL:

```
curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/postgresql-sink-connector.json http://192.168.18.8:8085/connectors
```

Si necesitamos borrar un conector, podemos hacerlo con este comando:

```
curl -X DELETE http://192.168.18.8:8085/connectors/postgres-alertas-sink
```

Tras ver que Kafka empieza con la inserta de los datos en PostgreSQL, podemos acceder a la base de datos a comprobar que todo funciona sin problemas.

```
psql -U sparkuser -d health_data
SELECT * FROM vitales_pacientes;
```

Si se desea vaciar la tabla, podemos hacerlo con TRUNCATE:



```
TRUNCATE vitales_pacientes;
```

Tras ver que tenemos datos, lanzamos Prometheus (para el dashboard de Grafana de métricas de Kafka) y Grafana.



```
./prometheus --config.file=prometheus.yml (En la carpeta de Prometheus)
sudo systemctl start grafana-server
```

Y tras eso, accedemos a los dashboards de Grafana. Ajustamos el tiempo de actualización (5 segundos está bien) y observamos como se van actualizando las métricas automáticamente.

Para las métricas mixtas deberá haber información en MySQL para que funcione. Para ello revise el punto 4 de la Wiki. Ingesta en batch

## 7.4 Ingesta en batch

Para la ingestá en batch podemos cerrar todos los procesos de la ingestá en streaming. Incluso si se desea se puede apagar Spark, podemos hacerlo con estos comandos:



```
./sbin/stop-master.sh (en $SPARK_HOME)
./sbin/stop-workers.sh (en $SPARK_HOME)
```

Si no tenemos datos en MySQL. Ejecutamos los producers para generar los pacientes y las analíticas.



```
python3 inser_patients.py
python3 /opt/proyecto/syn_data/gen_batch.py
```

Tras esto, iniciamos los worker 1 y 2.



```
bin/connect-distributed.sh /opt/proyecto/config/worker1.properties
bin/connect-distributed.sh /opt/proyecto/config/worker2.properties
```

Una vez se han cargado, primero guardaremos las analíticas.



```
curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/mysql-analiticas-source-connector.json http://192.168.18.8:8083/connectors
bin/kafka-console-consumer.sh --topic micro-analiticas --from-beginning --bootstrap-server 192.168.18.8:9094
curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/hdfs3-analiticas-sink-connector.json http://192.168.18.8:8083/connectors
```

Si en un punto necesitáramos comprobar los conectores, tenemos el siguiente comando:



```
curl http://192.168.18.8:8083/connectors
```

Una vez se han guardado las analíticas, hacemos lo mismo con los antibióticos.

```
curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/mysql-antibioticos-source-connector.json http://192.168.18.8:8083/connectors bin/kafka-console-consumer.sh --topic micro-antibiotic_res --from-beginning --bootstrap-server 192.168.18.8:9094  
curl -X POST -H "Content-Type: application/json" --data @/opt/proyecto/config/hdfs3-antibioticos-sink-connector.json http://192.168.18.8:8083/connectors
```

Una vez tenemos los datos en HDFS podemos conectarnos con Power BI.

En caso de que Power BI nos saltará con un error, la mejor alternativa es descargar los datos a nuestra máquina anfitriona. Para ello, haremos estos comandos:

```
hdfs dfs -get /bda/proyecto/batch/analiticas/topics/micro-analiticas/*.parket /home/hadoop/data  
hdfs dfs -get /bda/proyecto/batch/antibioticos/topics/micro-antibiotic_res/*.parket /home/hadoop/data
```

Y luego hacemos scp en nuestra máquina anfitriona:

```
scp -r hadoop@192.168.165.8:/home/hadoop/data C:\ruta_destino
```

Donde la ruta\_destino la cargaremos en Power BI.

Con esto, ya tendríamos las métricas por lo que se podría detener el sistema entero. Podemos parar Hadoop con estos comandos:

```
stop-yarn.sh  
stop-dfs.sh
```

## 8. Posibles Mejoras

Tras la realización del proyecto, se plantean posibles mejoras que se podrían realizar para enriquecer el proyecto y tuviera un mejor desempeño:

- **Datos sintéticos VS Datos reales:** Aunque el proyecto haya funcionado a través de datos sintéticos, hay que recordar que estos no reflejan la realidad real. Obviamente estos datos son demasiados sensibles como para exponerlos de manera pública, pero los datos sintéticos no podrán equipararse con los reales. Además, las métricas de las constantes vitales a nivel individual no se reflejaban bien porque los datos se generaban uno por uno y de manera aleatoria lo que hacía que algunos pacientes no se pudieran monitorear correctamente.
- **Usar una base de datos solamente:** En el proyecto se han usado 2 tipos de bases de datos cuando lo recomendable hubiera sido usar solo una. PostgreSQL es más estable que MySQL a la hora de manejar grandes cantidades de datos, aunque sería mejor optar por bases de datos como MongoDB, ClickHouse o Apache Cassandra que se centran en el Big Data. O incluso guardar los datos en HDFS en PARQUET como hacíamos antes y cargar los datos desde HDFS en Power BI.
- **Mejores recursos:** Este proyecto se ha realizado en un entorno muy reducido, algo que afecta la calidad del mismo. Es por ello que para que este proyecto tenga un mejor rendimiento tenga una infraestructura mucho mejor y que cubra sin problemas la falta de recursos como falta de RAM o almacenamiento.

- **Métricas de Spark en Grafana:** Prometheus se ha usado para graficar las métricas de Kafka, lo que nos lleva a pensar que estaría bien sacar las métricas de Apache Spark y ver su rendimiento. Por falta de tiempo no se ha podido llevar a cabo pero sería interesante ver cómo trabajan los distintos nodos para repartirse el trabajo o cuánta RAM consume cada uno de ellos, algo que con Prometheus y Grafana se puede visualizar de la misma forma que hicimos con Kafka.
- **Machine Learning:** Como indicamos al principio, se intentaría realizar de manera opcional algo de Machine Learning para un sistema de alertas. Algo que estaría muy bien para avisar posibles brotes que se den en el hospital o para dar la alarma de pacientes en situación crítica. De nuevo, la falta de tiempo no nos ha podido implementar esta característica, pero que sería muy útil para el personal sanitario disponer de un sistema de alarmas que les ayude a realizar su trabajo con mucha más eficiencia.

## 9. Conclusiones Finales

Este proyecto nos ha mostrado que en ámbitos tan delicados y esenciales como el ámbito de la salud, el Big Data desempeña un papel crucial. La capacidad de recolectar, examinar y relacionar vastas cantidades de datos médicos en tiempo real proporciona una visión más amplia y precisa sobre la salud de los pacientes. Crear un sistema que nos permita vigilar de manera continua a los pacientes para mantener su estabilidad y detectar tendencias de enfermedades mediante el análisis de sus signos vitales, expedientes clínicos y resultados de pruebas puede ser decisivo entre una respuesta médica reactiva y una intervención anticipada y adecuada. Esta habilidad no solo aumenta considerablemente las probabilidades de prevenir enfermedades o complicaciones de manera efectiva, sino que también mejora la eficiencia de los recursos en el sistema de salud, permitiendo a los profesionales atender con mayor rapidez y eficacia. Ante eventuales brotes de infecciones en el hospital, por ejemplo, el análisis de información puede facilitar la localización de áreas de contagio y tomar medidas inmediatas para impedir su propagación. Así, no solo se protege el bienestar de los pacientes, sino que también se aligera la carga del personal médico y se optimiza la gestión de camas, medicamentos y tiempos de atención.

## 10. Enlaces de interés

Por último se dejan una serie de enlaces de interés que han proporcionado información al proyecto.

- Página de Hadoop: <https://hadoop.apache.org/>
- Página de Apache Spark: <https://spark.apache.org/>
- Página de Kafka: <https://kafka.apache.org/>
- Página de PostgreSQL: <https://www.postgresql.org/>
- Página de MySQL: <https://www.mysql.com/>
- Página de Grafana: <https://grafana.com/>
- Página de Power BI: <https://www.microsoft.com/es-es/power-platform/products/power-bi>
- Instalar TimescaleDB (se intentó usar antes de PostgreSQL):  
<https://docs.timescale.com/self-hosted/latest/install/installation-linux/>
- Página de ClickHouse (se intentó usar antes de PostgreSQL): <https://clickhouse.com/>