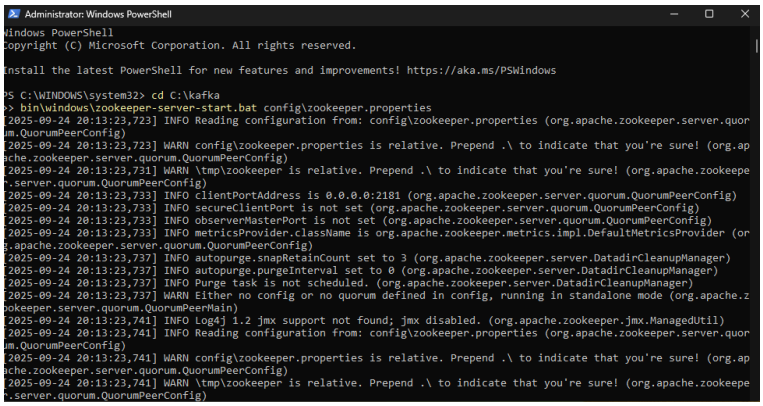
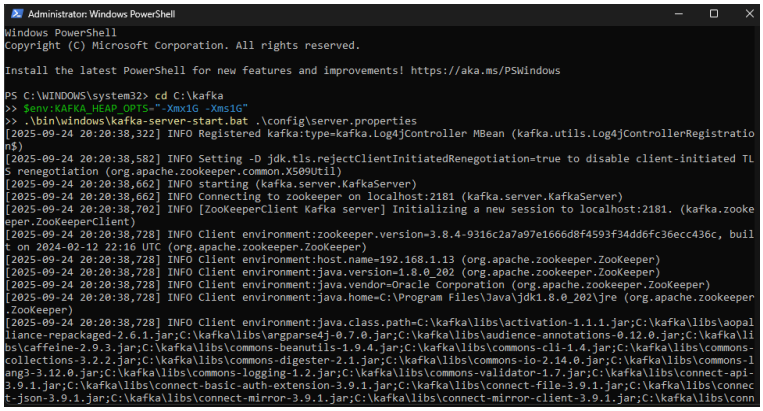
Actividad 5 – Módulo 8

1. **Instalación de Kafka en Windows**
   a. **Descargamos Kafka:** https://kafka.apache.org/downloads
   Para este ejercicio utilizamos la versión 3.9.1

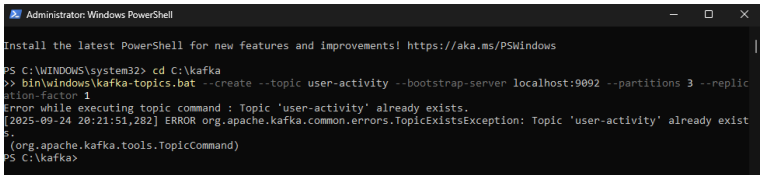
   b. **Iniciamos Zookeeper en una terminal de powershell**



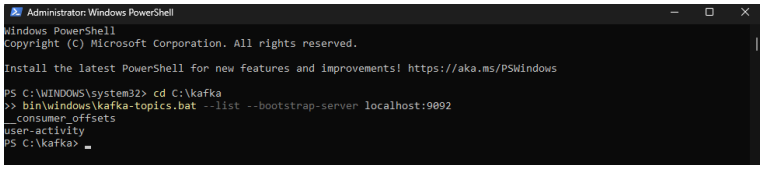
   c. **Iniciamos Kafka server en una terminal de powershell**



   d. **Creamos el Topic en una terminal de powershell**



   e. **Verificamos que Kafka este funcionando correctamente:**



2. Implementación

Se adjunta al final del informe los códigos

a. Productor de Eventos (producer.py)

El productor simula 10 tipos diferentes de eventos de usuario:

Características implementadas:

- Simulación de 5 usuarios únicos (IDs: 1-5)
- 10 productos diferentes (Gaming, Periféricos, Accesorios)
- 3 tipos de acciones: viewed_product, added_to_cart, purchased_product
- Timestamps automáticos en formato ISO
- Intervalos aleatorios entre eventos (1-3 segundos)

b. Consumidor de Eventos (consumer.py)

El consumidor procesa eventos en tiempo real y mantiene estadísticas:

Funcionalidades implementadas:

- Base de datos SQLite integrada (user_events.db)
- Estadísticas en tiempo real
- Sistema de alertas inteligente
- Cálculo de tasas de conversión
- Almacenamiento persistente de eventos
- Manejo de grupos de consumidores

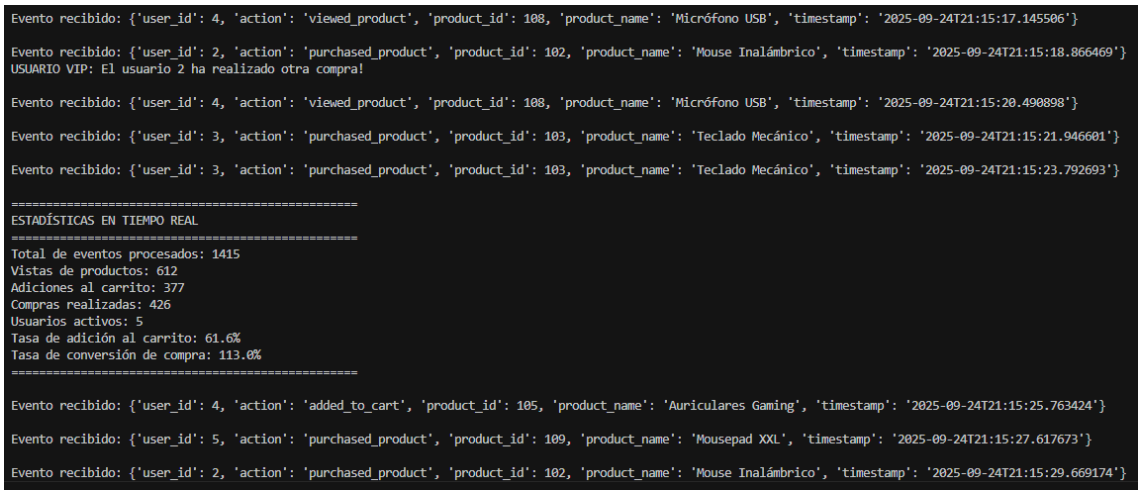
3. Verificación

Consola producer:

- Elementos enviados:

Consola consumer:

- Elementos recibidos:
- Alertas cliente vip
- Estadísticas en tiempo real



Los elementos son almacenados en la base de datos user_events.db utilizando la librería sqlite3.

producer.py

```python
from kafka import KafkaProducer
import json
import time
import random
from datetime import datetime

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

# Función para simular eventos de usuario
def simulate_user_activity():
    actions = [
        {"user_id": 1, "action": "viewed_product", "product_id": 101, "product_name": "Laptop Gaming"},
        {"user_id": 2, "action": "added_to_cart", "product_id": 102, "product_name": "Mouse Inalámbrico"},
        {"user_id": 3, "action": "purchased_product", "product_id": 103, "product_name": "Teclado Mecánico"},
        {"user_id": 1, "action": "viewed_product", "product_id": 104, "product_name": "Monitor 4K"},
        {"user_id": 4, "action": "added_to_cart", "product_id": 105, "product_name": "Auriculares Gaming"},
        {"user_id": 2, "action": "purchased_product", "product_id": 102, "product_name": "Mouse Inalámbrico"},
        {"user_id": 5, "action": "viewed_product", "product_id": 106, "product_name": "Silla Gaming"},
        {"user_id": 3, "action": "added_to_cart", "product_id": 107, "product_name": "Webcam HD"},
        {"user_id": 4, "action": "viewed_product", "product_id": 108, "product_name": "Micrófono USB"},
        {"user_id": 5, "action": "purchased_product", "product_id": 109, "product_name": "Mousepad XXL"}
    ]

    try:
        while True:
            # Seleccionar un evento aleatorio
            event = random.choice(actions)

            # Agregar timestamp
            event["timestamp"] = datetime.now().isoformat()

            # Enviar evento al tema 'user-activity'
            producer.send('user-activity', value=event)

            print(f"Evento enviado: {event}")

            # Esperar entre 1-3 segundos antes del siguiente evento
            time.sleep(random.uniform(1, 3))

    except KeyboardInterrupt:
        print("\nDeteniendo el productor...")
    finally:
        # Asegurar que todos los mensajes se envíen antes de cerrar
        producer.flush()
        producer.close()
        print("Productor cerrado correctamente.")

if __name__ == "__main__":
    print("Iniciando simulación de eventos de usuario...")
    print("Presiona Ctrl+C para detener")
    simulate_user_activity()
```

consumer.py

```
from kafka import KafkaConsumer
import json
from datetime import datetime
import sqlite3
import os

# Crear/conectar a una base de datos SQLite para almacenar eventos
def init_database():
    conn = sqlite3.connect('user_events.db')
    cursor = conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS user_events (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER,
            action TEXT,
            product_id INTEGER,
            product_name TEXT,
            timestamp TEXT,
            processed_at TEXT
        )
    ''')

    conn.commit()
    return conn

# Configurar consumidor Kafka
consumer = KafkaConsumer(
    'user-activity',  # Nombre del tema
    bootstrap_servers='localhost:9092',  # Dirección del servidor Kafka
    group_id='user-activity-consumer-group',  # Grupo de consumidores
    value_deserializer=lambda x: json.loads(x.decode('utf-8')),  # Deserializar JSON
    auto_offset_reset='latest'  # Comenzar desde los mensajes más recientes
)

# Estadísticas en tiempo real
stats = {
    'total_events': 0,
    'views': 0,
    'cart_additions': 0,
    'purchases': 0,
    'active_users': set()
}

def process_event(event, db_conn):
    """Procesa un evento individual y actualiza estadísticas"""
    global stats

    # Actualizar estadísticas
    stats['total_events'] += 1
    stats['active_users'].add(event['user_id'])

    if event['action'] == 'viewed_product':
        stats['views'] += 1
    elif event['action'] == 'added_to_cart':
        stats['cart_additions'] += 1
    elif event['action'] == 'purchased_product':
        stats['purchases'] += 1

    # Almacenar en base de datos
    cursor = db_conn.cursor()
    cursor.execute('''
```

```
        INSERT INTO user_events (user_id, action, product_id, product_name, timestamp, processed_at)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', (
        event['user_id'],
        event['action'],
        event['product_id'],
        event['product_name'],
        event['timestamp'],
        datetime.now().isoformat()
    ))
    db_conn.commit()

    return stats

def display_stats(stats):
    """Muestra estadísticas actualizadas"""
    print("\n" + "="*50)
    print("ESTADÍSTICAS EN TIEMPO REAL")
    print("="*50)
    print(f"Total de eventos procesados: {stats['total_events']}")
    print(f"Vistas de productos: {stats['views']}")
    print(f"Adiciones al carrito: {stats['cart_additions']}")
    print(f"Compras realizadas: {stats['purchases']}")
    print(f"Usuarios activos: {len(stats['active_users'])}")

    # Calcular tasas de conversión
    if stats['views'] > 0:
        cart_rate = (stats['cart_additions'] / stats['views']) * 100
        print(f"Tasa de adición al carrito: {cart_rate:.1f}%")

    if stats['cart_additions'] > 0:
        purchase_rate = (stats['purchases'] / stats['cart_additions']) * 100
        print(f"Tasa de conversión de compra: {purchase_rate:.1f}%")

    print("="*50)

def generate_alerts(event):
    """Genera alertas basadas en patrones de eventos"""
    # Alerta por compra de producto caro (simulación)
    if event['action'] == 'purchased_product' and event['product_id'] in [101, 104, 106]:
        print(f"ALERTA: Compra de producto premium por usuario {event['user_id']}: {event['product_name']}")

    # Alerta por usuario muy activo
    if event['user_id'] in [1, 2] and event['action'] == 'purchased_product':
        print(f"USUARIO VIP: El usuario {event['user_id']} ha realizado otra compra!")

def main():
    """Función principal del consumidor"""
    print("Iniciando consumidor de eventos de usuario...")
    print("Esperando eventos desde el tema 'user-activity'...")
    print("Presiona Ctrl+C para detener")

    # Inicializar base de datos
    db_conn = init_database()

    try:
        # Escuchar continuamente los eventos desde el tema
        for message in consumer:
            event = message.value

            # Procesar evento
            print(f"\nEvento recibido: {event}")

            # Actualizar estadísticas y almacenar en BD
```

```
            current_stats = process_event(event, db_conn)

            # Generar alertas si es necesario
            generate_alerts(event)

            # Mostrar estadisticas cada 5 eventos
            if current_stats['total_events'] % 5 == 0:
                display_stats(current_stats)

    except KeyboardInterrupt:
        print("\n\nDeteniendo el consumidor...")

    finally:
        # Cerrar conexiones
        consumer.close()
        db_conn.close()

        # Mostrar estadísticas finales
        print("\nESTADÍSTICAS FINALES:")
        display_stats(stats)
        print(f"\nEventos almacenados en: user_events.db")
        print("Consumidor cerrado correctamente.")

if __name__ == "__main__":
    main()
```