МИНИСТЕРСТВО ТРАНСПОРТА И КОММУНИКАЦИЙ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Белорусский государственный университет транспорта»

Кафедра «Управление эксплуатационной работой и охрана труда»

Отчет  
по лабораторной работе №10

по дисциплине «Средства и технологии анализа и разработки информационных систем»

Выполнил Проверил

студент группы ГИ-31 зав. каф. УЭРиОТ

Сидорова А. А. Козлов В. Г.

Гомель, 2025

# Лабораторная работа №10

**РАЗРАБОТКА СИСТЕМЫ ОБРАБОТКИ СООБЩЕНИЙ С ИСПОЛЬЗОВАНИЕМ APACHE KAFKA**

**Цель работы:** Разработка системы асинхронной обработки сообщений с использованием Apache Kafka, включая создание продюсеров, консьюмеров и настройку их взаимодействия.

**Задание:**

1. **Установка и настройка Kafka**

• Запустите ZooKeeper и Kafka-брокер (локально или через Docker).

• Создайте топик user\_actions с 3 партициями.

1. **Разработка продюсера**

• Напишите приложение (Java/Python), отправляющее JSON-сообщения в топик. Пример сообщения: {"user\_id": 101, "action": "purchase", "timestamp": "2023-10-01T12:00:00"}

• Реализуйте ввод сообщений через консоль или чтение из файла.

**3. Разработка консьюмера**

• Напишите консьюмер, обрабатывающий сообщения из топика:

- Фильтрация по типу действия (например, "purchase").

- Вывод статистики в консоль (количество сообщений, частые действия).

**4. Обработка ошибок и масштабирование**

• Настройте Dead Letter Topic (DLT) для некорректных сообщений.

• Запустите 2 консьюмера в одной группе. Продемонстрируйте распределение сообщений между ними.

**5. Интеграция с внешней системой**

• Сохраняйте данные в PostgreSQL/MySQL.

***Задание 1.***

Исходный код docker-compose.yml:

version: '3.8'

services:

  zookeeper:

    image: confluentinc/cp-zookeeper:7.0.1

    container\_name: lab10\_zookeeper

    environment:

      ZOOKEEPER\_CLIENT\_PORT: 2181

      ZOOKEEPER\_TICK\_TIME: 2000

  kafka:

    image: confluentinc/cp-kafka:7.0.1

    container\_name: lab10\_kafka

    depends\_on:

      - zookeeper

    ports:

      - "9092:9092"

    environment:

      KAFKA\_BROKER\_ID: 1

      KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

      KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://localhost:9092

      KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

      KAFKA\_AUTO\_CREATE\_TOPICS\_ENABLE: "true"

  postgres:

    image: postgres:15

    container\_name: lab10\_postgres

    restart: always

    ports:

      - "5432:5432"

    environment:

      POSTGRES\_USER: postgres

      POSTGRES\_PASSWORD: 1234

      POSTGRES\_DB: test\_db

    volumes:

      - pg\_data:/var/lib/postgresql/data

      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

volumes:

  pg\_data:

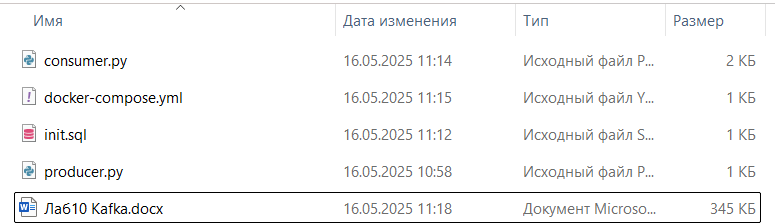


Рисунок 1.1 – **Структура проекта**

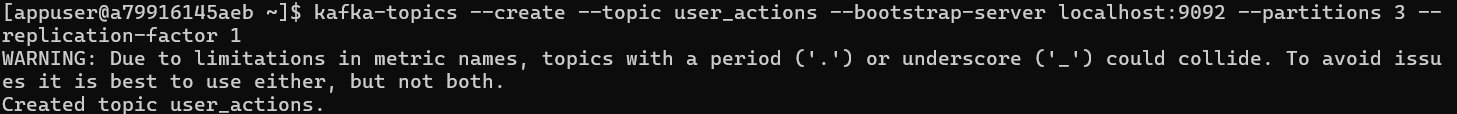


Рисунок 1.2 – **Создание user\_actions**

***Задание 2.***

Исходный код producer.py:

from kafka import KafkaProducer

import json

producer = KafkaProducer(

    bootstrap\_servers='localhost:9092',

    value\_serializer=lambda v: json.dumps(v).encode('utf-8')

)

while True:

    user\_id = input("Введите user\_id: ")

    action = input("Введите действие (например, purchase): ")

    message = {

        "user\_id": int(user\_id),

        "action": action,

        "timestamp": "2025-05-16T12:00:00"

    }

    producer.send('user\_actions', message)

    print("Отправлено:", message)

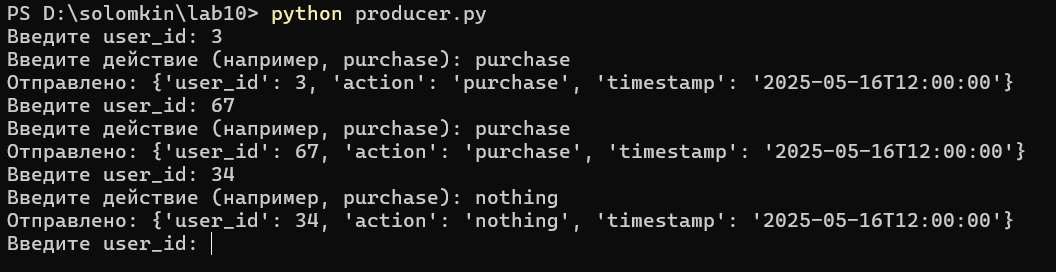


Рисунок 1.4 – **Работа программы producer.py**

***Задание 3.***

Исходный код consumer.py:

from kafka import KafkaConsumer, KafkaProducer

import json

from collections import Counter

import psycopg2

# Подключение к PostgreSQL

conn = psycopg2.connect(

    dbname="kafka\_messages",

    user="postgres",

    password="123456",  # замените на ваш пароль

    host="localhost",

    port="5432"

)

cur = conn.cursor()

# Kafka Consumer

consumer = KafkaConsumer(

    'user\_actions',

    bootstrap\_servers='localhost:9092',

    value\_deserializer=lambda m: json.loads(m.decode('utf-8')),

    group\_id='my\_consumer\_group',

    auto\_offset\_reset='earliest'

)

# Kafka Producer для DLT

dlt\_producer = KafkaProducer(

    bootstrap\_servers='localhost:9092',

    value\_serializer=lambda v: json.dumps(v).encode('utf-8')

)

action\_counter = Counter()

print("Консьюмер запущен. Ожидаем сообщения...")

for message in consumer:

    data = message.value

    try:

        # Проверка данных

        if not all(k in data for k in ("user\_id", "action", "timestamp")):

            raise ValueError("Некорректное сообщение: отсутствуют поля")

        # Фильтрация: только "purchase"

        if data['action'] == 'purchase':

            print("Покупка:", data)

            action\_counter[data['action']] += 1

            print("Статистика:", dict(action\_counter))

            # Сохраняем в PostgreSQL

            cur.execute(

                "INSERT INTO user\_actions (user\_id, action, timestamp) VALUES (%s, %s, %s)",

                (data["user\_id"], data["action"], data["timestamp"])

            )

            conn.commit()

        else:

            print("Пропущено (не purchase):", data)

    except Exception as e:

        print("Ошибка при обработке сообщения:", e)

        print("Отправка в DLT...")

        dlt\_producer.send('user\_actions\_dlt', data)

        dlt\_producer.flush()

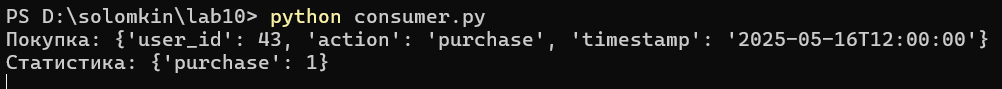


Рисунок 1.5 – **Работа программы consumer.py**

***Задание 4.***

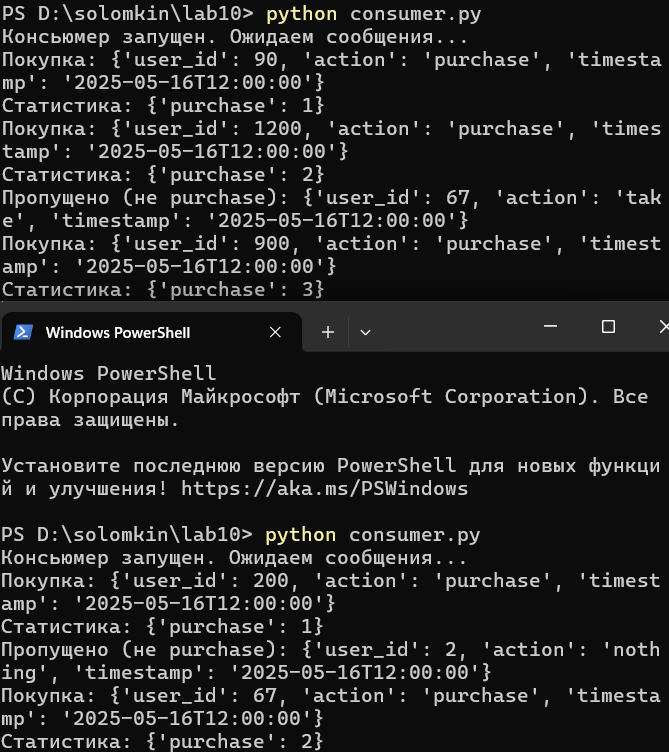


Рисунок 10.5 – **Результат выполнения задания 4**

***Задание 5.***

Исходный код init.sql:

CREATE TABLE IF NOT EXISTS user\_actions (

    id SERIAL PRIMARY KEY,

    user\_id INT NOT NULL,

    action TEXT NOT NULL,

    timestamp TEXT NOT NULL

);

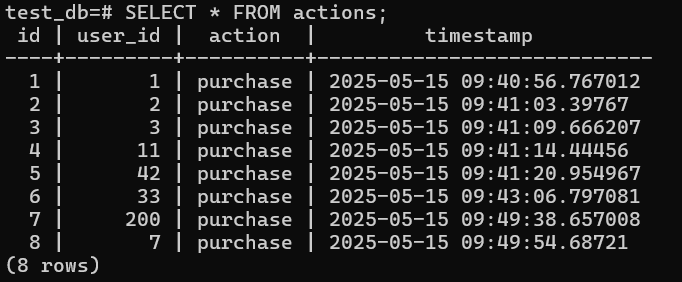


Рисунок 10.6 – **Результат заполнения базы данных**

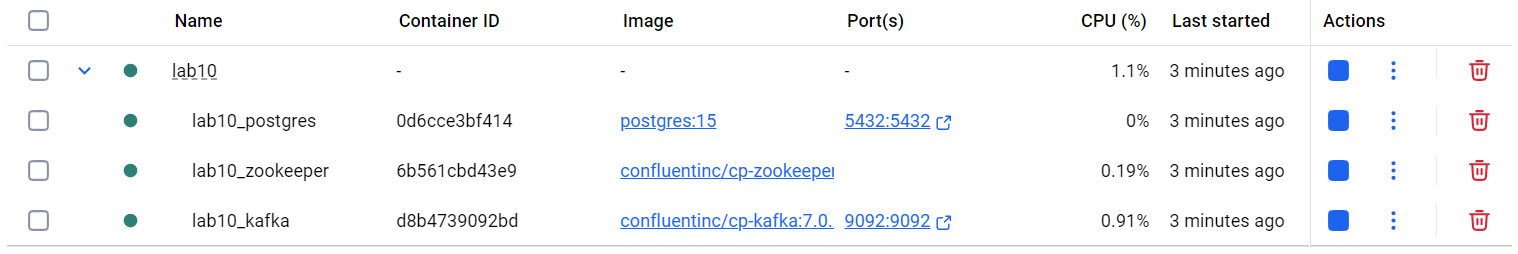


Рисунок 10.7 – **Структура Docker**

***Вывод:*** Apache Kafka – это распределённая платформа потоковой передачи сообщений, которая используется для построения масштабируемых и отказоустойчивых систем обработки данных в реальном времени. В рамках лабораторной работы была создана система, способная асинхронно принимать, обрабатывать и сохранять сообщения от внешних источников. Основные компоненты такой системы – это продюсер (отправитель сообщений), консьюмер (обработчик сообщений), брокер Kafka и база данных PostgreSQL для хранения результатов.

Во время выполнения работы возникли определённые трудности, связанные с многокомпонентной архитектурой Kafka. Для запуска требуются дополнительные сервисы, такие как ZooKeeper, а также понимание, как устроены топики, партиции и группы потребителей. Особые сложности вызывают вопросы настройки соединений, обработки ошибок и синхронизации компонентов между собой. Кроме того, работа с Kafka требует навыков работы с асинхронными системами, что для начинающих может быть непросто.

Тем не менее, проделанная работа позволяет понять основные принципы работы современных распределённых систем обмена сообщениями. Kafka показывает себя как мощный инструмент для организации потоковой передачи данных между независимыми компонентами, обеспечивая надёжность, масштабируемость и гибкость. Интеграция с внешними системами, такими как базы данных, демонстрирует, как Kafka может использоваться в реальных проектах – от логирования и аналитики до обработки пользовательских действий в реальном времени.

***Контрольные вопросы.***

1. Зачем Kafka использует ZooKeeper?

Kafka использует ZooKeeper для координации и управления состоянием кластера. С его помощью Kafka отслеживает, какие брокеры активны, управляет распределением топиков и партиций, а также выбирает лидера для каждой партиции. Без ZooKeeper Kafka не может правильно функционировать в распределённой среде, так как ей необходим механизм для синхронизации и управления кластером. Однако в более новых версиях Kafka планирует отказаться от ZooKeeper и перейти на собственный встроенный механизм управления (KRaft).

1. Как обеспечить порядок сообщений в партиции?

Порядок сообщений в Kafka обеспечивается внутри каждой партиции. Это значит, что если сообщения поступают в одну и ту же партицию, они будут читаться в точном порядке их отправки. Чтобы сохранить порядок логически связанных сообщений, например от одного пользователя, необходимо направлять их всегда в одну и ту же партицию, используя ключ маршрутизации. Важно также учитывать, что порядок между разными партициями Kafka не гарантирует.

1. В чём разница между auto.offset.reset=earliest и latest?

Разница между auto.offset.reset=earliest и latest заключается в том, с какого места консьюмер начнёт читать сообщения, если ранее он не сохранял смещение. При значении earliest консьюмер будет читать все сообщения с самого начала — это удобно для полной обработки накопленных данных. При значении latest консьюмер пропускает всё, что уже есть в топике, и читает только новые сообщения, которые поступают после его запуска.

1. Как Kafka обеспечивает отказоустойчивость?

Kafka обеспечивает отказоустойчивость за счёт репликации данных. Каждая партиция может иметь несколько реплик, которые хранятся на разных брокерах. Если один брокер выходит из строя, другой брокер с копией данных может продолжить работу. Kafka также следит за доступностью брокеров и автоматически переназначает лидеров партиций при сбоях. Это позволяет системе продолжать работать даже при потере отдельных узлов без потери данных.