**ЛАБОРАТОРНАЯ РАБОТА**

**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 заимодействия.

**Программное обеспечение:** Apache Kafka 3.0+; ZooKeeper 3.7+; Java 11+ / Python 3.8+; IntelliJ IDEA / VS Code; Docker (опционально)

**Необходимая теоретическая подготовка:** Основы распределённых систем; Принципы работы брокеров сообщений; API Kafka (Producer/Consumer); JSON-сериализация данных.

**Реализация:**

1. ZooKeeper и Kafka были установлены с использованием конфигурационного файла docker-compose.yml.:

version: '3'

services:

  zookeeper:

    image: confluentinc/cp-zookeeper:7.4.0

    environment:

      ZOOKEEPER\_CLIENT\_PORT: 2181

      ZOOKEEPER\_TICK\_TIME: 2000

    ports:

      - "2181:2181"

  kafka:

    image: confluentinc/cp-kafka:7.4.0

    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

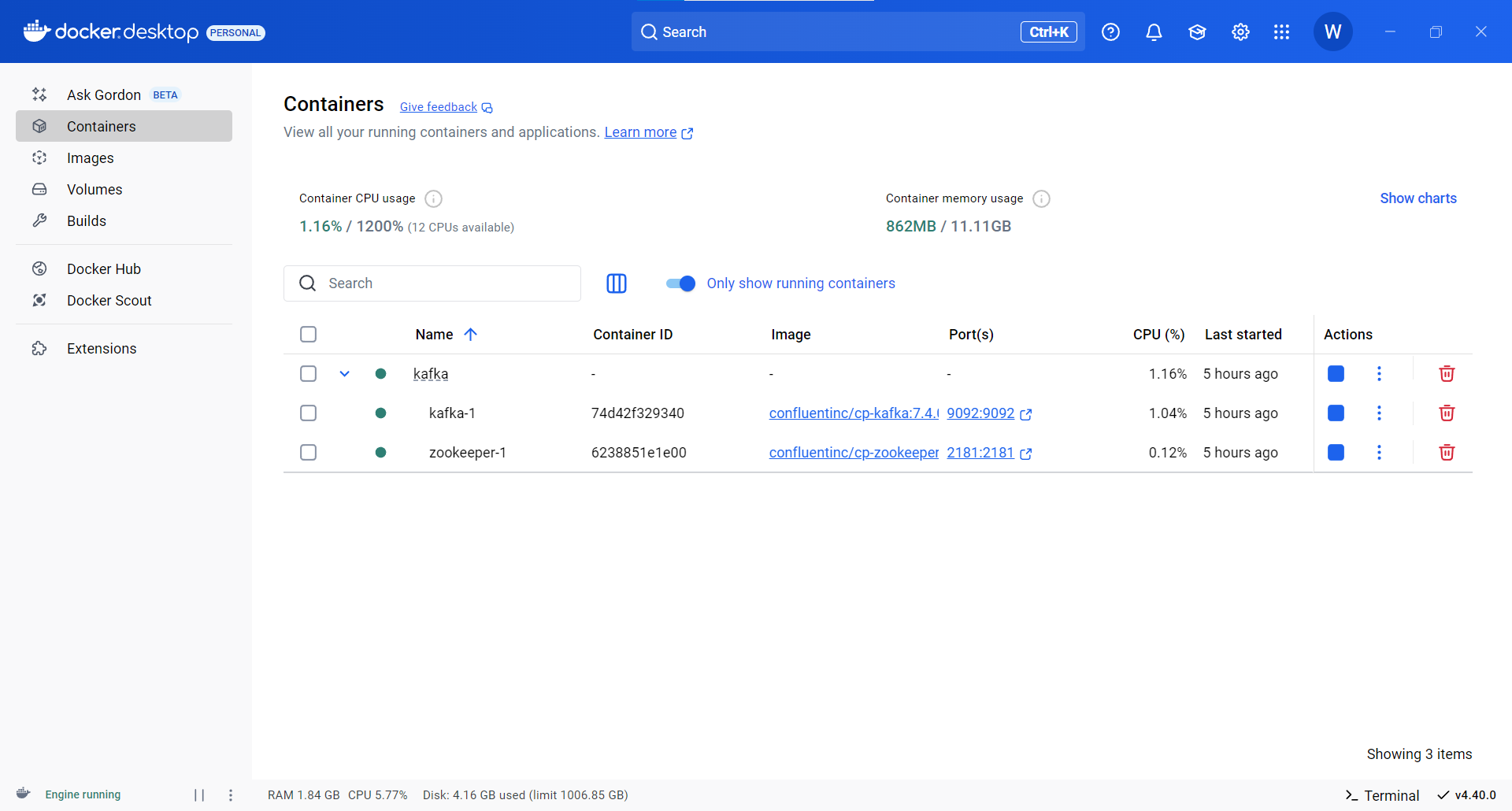


Рисунок 1 – **ZooKeeper и Kafka**

1. Топик user\_actions был успешно создан и содержит 3 партиции:

kafka-topics.bat --create --topic user\_actions --partitions 3 --replication-factor 1 --bootstrap-server localhost:9092

1. В папке были созданы файлы producer.py и consumer.py.
2. База данных была успешно создана вместе с таблицей:

CREATE DATABASE kafka\_events;

CREATE TABLE user\_actions (

id SERIAL PRIMARY KEY,

user\_id INTEGER NOT NULL,

action VARCHAR(50) NOT NULL,

timestamp TIMESTAMP NOT NULL,

processed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

1. Произведён запуск producer.py и двух экземпляров consumer.py.. Было отправлено два сообщения: юзер 101 и юзер 102:

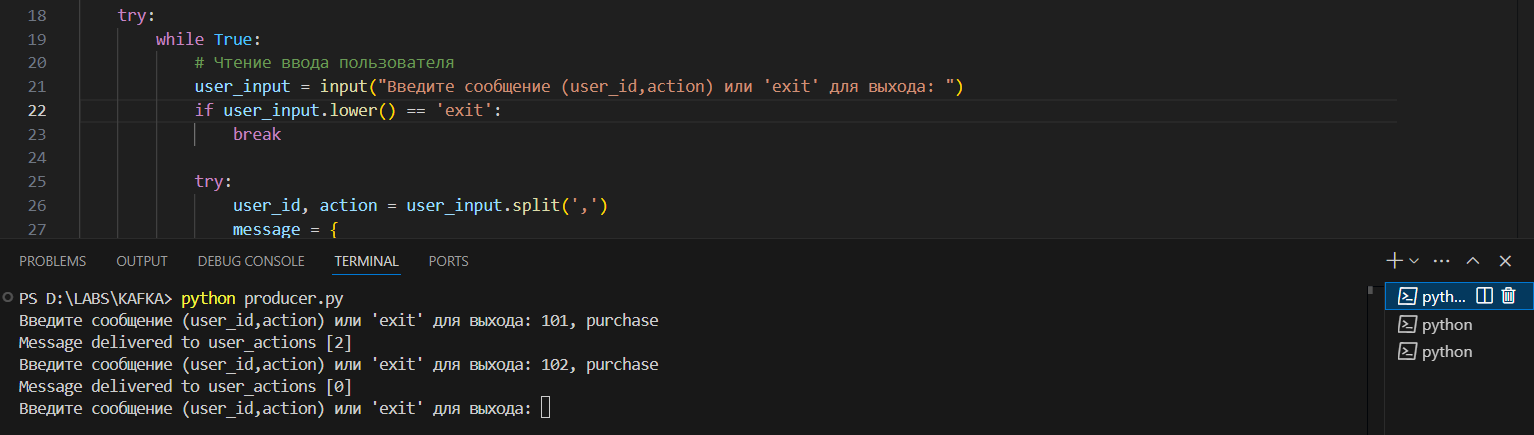


Рисунок 2 – **Запуск**

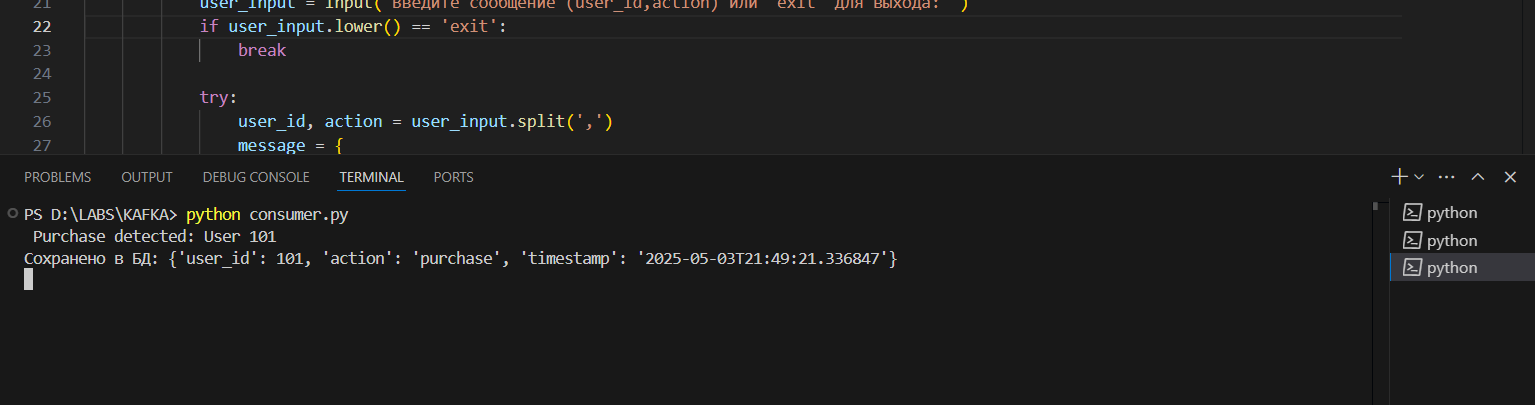


Рисунок 3 – **101 получил**



Рисунок 4 – **102 получил**

1. Данные успешно сохранены в базе данных:

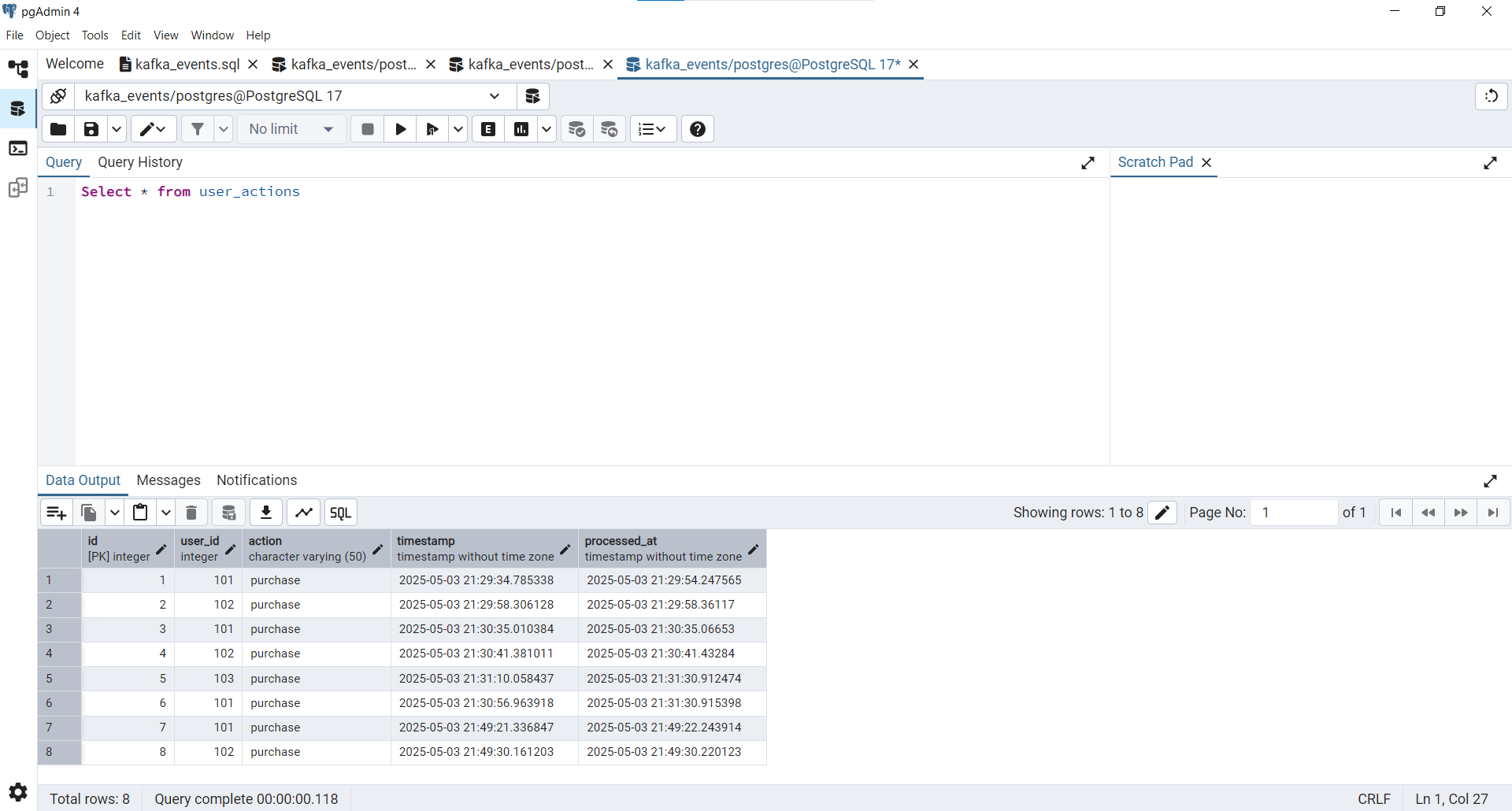


Рисунок 5 – **Успешное сохранение**

**Контрольные вопросы:**

1. Kafka использует Apache ZooKeeper для управления кластером. Он координирует брокеры, выбирает лидера для каждой партиции, хранит метаданные и помогает перераспределять нагрузку при сбоях. Это делает систему отказоустойчивой и надежной.
2. Чтобы обеспечить порядок сообщений в партиции, необходимо использовать одинаковый ключ партиционирования, чтобы все сообщения с этим ключом попадали в одну и ту же партицию. Также важно потреблять данные одним consumer в группе, иначе сообщения могут поступать в разном порядке. Дополнительно можно настроить параметр acks=all, чтобы сообщение записывалось всеми репликами перед подтверждением.
3. Параметр auto.offset.reset определяет, с какого места начинать чтение сообщений: **earliest** позволяет consumer'у обработать всю историю сообщений, начиная с первого доступного, а **latest** — получать только новые сообщения, игнорируя старые. Это важно при первом запуске consumer'а, когда у него нет сохраненного offset. Выбор параметра зависит от задачи: если нужно обработать исторические данные, стоит использовать earliest, если только новые — latest.