

Лабораторная работа №6: Работа с Apache Kafka

1. Цель работы Освоить основы работы с Apache Kafka как платформой для обмена сообщениями. Получить практические навыки создания топиков, написания производителей (producers) и потребителей (consumers) на Python, а также понимания архитектуры Kafka.
2. Используемый стек технологий

Платформа: Apache Kafka

Язык программирования: Python

Библиотека: kafka-python

ОС: Linux (Ubuntu)

Инструменты: Kafka CLI tools, Python IDE

3. Теоретические сведения

Apache Kafka — распределённая платформа для обмена сообщениями и потоковой обработки данных.

Ключевые понятия:

- **Топик (Topic):** Категория или имя потока сообщений
- **Партиция (Partition):** Топики разделяются на партиции для параллельной обработки
- **Производитель (Producer):** Приложение, отправляющее сообщения в топик
- **Потребитель (Consumer):** Приложение, читающее сообщения из топика
- **Consumer Group:** Группа потребителей, совместно обрабатывающих сообщения
- **Брокер (Broker):** Сервер Kafka в кластере
- **Zookeeper:** Сервис для координации и управления кластером

4. Ход выполнения работы

Часть 1: Установка и настройка Kafka

1.1 Установка Kafka с использованием Docker

```
cat > docker-compose.yml << 'EOF'
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.4.0
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"
    networks:
      - kafka-net
```

```
kafka: image: confluentinc/cp-kafka:7.4.0 container_name: 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_TRANSACTION_STATE_LOG_MIN_ISR: 1 KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1 networks: - kafka-net networks: kafka-net: driver: bridge EOF
```

```
echo "Запуск Kafka и Zookeeper через Docker..." docker-compose up -d
```

```
echo "Проверка статуса контейнеров..." docker ps
```

Часть 2: Работа с топиками через CLI

2.1 Создание топика

```
echo "Создание топика kafka-logs..." docker exec kafka kafka-topics.sh --create --topic kafka-logs --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1 --config retention.ms=604800000 # Хранение 7 дней
```

```
echo "Топик kafka-logs создан!"
```

2.2 Проверка списка топиков

```
echo "Список всех топиков:" docker exec kafka kafka-topics.sh --list --bootstrap-server localhost:9092
```

2.3 Просмотр информации о топике

```
echo "Детальная информация о топике kafka-logs:" docker exec kafka kafka-topics.sh --describe --topic kafka-logs --bootstrap-server localhost:9092
```

2.4 Дополнительные операции с топиками

```
for topic in "user-events" "system-metrics" "payment-transactions"; do docker exec kafka kafka-topics.sh --create --topic $topic --bootstrap-server localhost:9092 --partitions 2 --replication-factor 1 echo "Топик $topic создан" done
```

Часть 3: Написание Producer на Python

3.1 Установка зависимостей

```
!pip install kafka-python
```

3.2 Базовый Producer

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

Настройка логирования

```
logging.basicConfig(level=logging.INFO) logger = logging.getLogger(name)

class KafkaLogProducer: def __init__(self, bootstrap_servers='localhost:9092'): """Инициализация Kafka Producer""" self.producer = KafkaProducer( bootstrap_servers=bootstrap_servers, value_serializer=lambda v: json.dumps(v).encode('utf-8'), key_serializer=lambda v: str(v).encode('utf-8')) if v else None, acks='all', # Гарантия доставки до всех реплик retries=3, # Количество попыток повтора max_in_flight_requests_per_connection=1, # Для упорядоченности compression_type='gzip', # Сжатие для уменьшения трафика linger_ms=5, # Задержка перед отправкой батча batch_size=16384 # Размер батча в байтах ) logger.info(f" Producer подключен к Kafka на {bootstrap_servers}")
```

```
def generate_log_entry(self, log_id):
    """Генерация тестовой записи лога"""
    log_types = ['INFO', 'WARNING', 'ERROR', 'DEBUG']
    services = ['web-server', 'auth-service', 'database', 'api-gateway', 'payment-service']
    messages = [
        'Database connection established',
        'User authentication successful',
        'Database connection failed',
        'API request processed',
        'Payment transaction completed',
        'Memory usage high',
        'Disk space running low',
        'Network latency detected',
        'Cache hit ratio optimal',
        'Failed to send email notification'
    ]
    # Распределяем логи по разным сервисам
    service = services[log_id % len(services)]
    # Весовая вероятность для уровней логирования
    weights = [0.6, 0.2, 0.15, 0.05] # INFO, WARNING, ERROR, DEBUG
    level = random.choices(log_types, weights=weights)[0]
    log_entry = {
        'timestamp': datetime.now().isoformat(),
        'timestamp_unix': time.time(),
```

```
'level': level,
'message': f'{random.choice(messages)} - ID: {log_id}',
'service': service,
'ip': f'192.168.1.{random.randint(1, 254)}',
'user_id': f'user_{random.randint(1000, 9999)}' if random.random() > 0.3 else None,
'response_time_ms': random.randint(10, 5000) if service in ['web-server', 'api-gatewa',
'status_code': random.choice([200, 201, 400, 401, 403, 404, 500]) if 'service' in ser
}

# Добавляем специфичные поля для ERROR логов
if level == 'ERROR':
    log_entry['error_code'] = f'ERR{random.randint(100, 999)}'
    log_entry['stack_trace'] = '... stack trace details ...' if random.random() > 0.5 els

return log_entry

def send_logs(self, topic='kafka-logs', num_logs=100, delay=0.1):
    """Отправка логов в Kafka топик"""
    logger.info(f"👉 Отправка {num_logs} логов в топик '{topic}'...")

    success_count = 0
    error_count = 0

    for i in range(num_logs):
        try:
            # Генерируем запись лога
            log_entry = self.generate_log_entry(i)

            # Определяем ключ дляpartitionирования (по сервису)
            key = log_entry['service']

            # Отправляем сообщение
            future = self.producer.send(
                topic=topic,
                key=key, # Partitionирование по сервису
                value=log_entry
            )

            # Обработка результата отправки (опционально, для надежности)
            # result = future.get(timeout=10)
            # logger.debug(f"Сообщение {i} отправлено в партицию {result.partition}")

            success_count += 1

            # Логируем прогресс
            if (i + 1) % 10 == 0:
                logger.info(f"Отправлено {i + 1}/{num_logs} сообщений...")

            # Задержка между отправками
            time.sleep(delay)

        except Exception as e:
            error_count += 1
```

```
logger.error(f" Ошибка при отправке сообщения {i}: {e}")

# Ожидаем отправки всех сообщений
self.producer.flush()

logger.info(f" Отправка завершена!")
logger.info(f" Успешно: {success_count}")
logger.info(f" С ошибками: {error_count}")

return success_count, error_count

def close(self):
    """Закрытие producer"""
    self.producer.close()
    logger.info(" Producer закрыт")
```