

# Лабораторная работа №7: Интеграция Kafka и Spark Streaming

---

1. Цель работы Освоить интеграцию Apache Kafka с Apache Spark Streaming для обработки потоковых данных в реальном времени. Получить практические навыки настройки прямого чтения данных из топиков Kafka и их обработки с помощью Structured Streaming.
2. Используемый стек технологий

Платформы: Apache Kafka Язык программирования: Python (PySpark) Библиотеки: kafka-python, pyspark ОС: Linux (Ubuntu) Инструменты: Jupyter Notebook, Kafka CLI tools

3. Теоретические сведения

**Интеграция Kafka-Spark** позволяет обрабатывать потоковые данные непосредственно из топиков Kafka с использованием мощностей Spark Streaming.

## Ключевые компоненты интеграции:

- **Kafka Direct API:** Прямое подключение к партициям топика
- **Offset management:** Управление позицией чтения сообщений
- **Exactly-once semantics:** Гарантия однократной обработки сообщений
- **Structured Streaming integration:** Нативная поддержка в Spark Structured Streaming

## Преимущества подхода:

- Высокая пропускная способность
- Отказоустойчивость
- Масштабируемость
- Поддержка различных форматов данных

4. Ход выполнения работы

## Часть 1: Подготовка Kafka-инфраструктуры

---

### 1.1 Создание топика для обработки

```
docker exec kafka kafka-topics.sh --create  
--topic real-time-transactions  
--bootstrap-server localhost:9092  
--partitions 3
```

```
--replication-factor 1  
--config retention.ms=86400000 # Хранение 24 часа
```

```
echo " Топик 'real-time-transactions' создан"
```

```
docker exec kafka kafka-topics.sh --create
```

```
--topic transaction-alerts
```

```
--bootstrap-server localhost:9092
```

```
--partitions 2
```

```
--replication-factor 1
```

```
docker exec kafka kafka-topics.sh --create
```

```
--topic user-statistics
```

```
--bootstrap-server localhost:9092
```

```
--partitions 2
```

```
--replication-factor 1
```

```
echo " Дополнительные топики созданы"
```

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

## 1.2 Producer для генерации тестовых данных

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

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

```
class TransactionProducer: def init(self, bootstrap_servers='localhost:9092'): """Инициализация Kafka  
Producer для транзакций""" 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', retries=3, compression_type='gzip', linger_ms=10, batch_size=16384 )  
logger.info(f" TransactionProducer подключен к {bootstrap_servers}")
```

```
def generate_transaction(self, transaction_id):  
    """Генерация тестовой транзакции"""  
    transaction_types = ['purchase', 'refund', 'transfer', 'withdrawal', 'deposit']  
  
    # Весовая вероятность для типов транзакций  
    weights = [0.6, 0.1, 0.15, 0.1, 0.05] # purchase, refund, transfer, withdrawal, deposit  
    transaction_type = random.choices(transaction_types, weights=weights)[0]  
  
    # Генерация суммы в зависимости от типа  
    if transaction_type == 'purchase':  
        amount_range = (10.0, 500.0)  
    elif transaction_type == 'refund':  
        amount_range = (5.0, 200.0)  
    elif transaction_type == 'transfer':  
        amount_range = (50.0, 1000.0)  
    elif transaction_type == 'withdrawal':
```

```

        amount_range = (20.0, 300.0)
else: # deposit
    amount_range = (100.0, 2000.0)

transaction = {
    'transaction_id': f'txn_{transaction_id:06d}',
    'user_id': f'user_{random.randint(1, 50)}', # 50 уникальных пользователей
    'amount': round(random.uniform(*amount_range), 2),
    'type': transaction_type,
    'timestamp': datetime.now().isoformat(),
    'location': random.choice(['online', 'store', 'atm', 'mobile_app']),
    'currency': random.choice(['USD', 'EUR', 'GBP']),
    'merchant_id': f'merchant_{random.randint(1, 20)}' if transaction_type in ['purchase', 'return']
    'status': random.choices(['success', 'failed', 'pending'], weights=[0.85, 0.1, 0.05]),
    'category': random.choice(['electronics', 'food', 'clothing', 'entertainment', 'utilities'])
        if transaction_type == 'purchase' else None,
    'fraud_score': round(random.uniform(0.0, 1.0), 3) # Симуляция скора мошенничества
}

return transaction

def send_transactions(self, num_transactions=1000, delay=0.5):
    """Отправка транзакций в Kafka"""
    logger.info(f" Отправка {num_transactions} транзакций...")

    metrics = {
        'sent': 0,
        'failed': 0,
        'by_type': {},
        'by_user': {}
    }

    for i in range(num_transactions):
        try:
            transaction = self.generate_transaction(i)
            user_id = transaction['user_id']

            # Отправка с ключом для партиционирования по user_id
            future = self.producer.send(
                topic='real-time-transactions',
                key=user_id,
                value=transaction
            )

            # Обновляем метрики
            metrics['sent'] += 1
            metrics['by_type'][transaction['type']] = metrics['by_type'].get(transaction['type'], 0) + 1
            metrics['by_user'][user_id] = metrics['by_user'].get(user_id, 0) + 1

            # Прогресс каждые 50 транзакций
            if (i + 1) % 50 == 0:
                logger.info(f"👉 Отправлено {i + 1}/{num_transactions} транзакций")
        except Exception as e:
            logger.error(f"Ошибка при отправке транзакции {i}: {e}")

```

```

time.sleep(delay)

except Exception as e:
    metrics['failed'] += 1
    logger.error(f" Ошибка при отправке транзакции {i}: {e}")

self.producer.flush()

# Вывод статистики
logger.info(f" Отправка завершена!")
logger.info(f" Успешно: {metrics['sent']} ")
logger.info(f" Ошибок: {metrics['failed']} ")

logger.info(f" Распределение по типам:")
for ttype, count in metrics['by_type'].items():
    logger.info(f" {ttype}: {count}")

return metrics

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

```

```
producer = TransactionProducer()
```

```
try: metrics = producer.send_transactions( num_transactions=200, # Уменьшено для демонстрации
delay=0.3 )
```

```

print(f" ИТОГОВАЯ СТАТИСТИКА:")
print(f" Всего транзакций: {metrics['sent'] + metrics['failed']} ")
print(f" Успешно отправлено: {metrics['sent']} ")
print(f" Ошибок отправки: {metrics['failed']} ")

```

```
except KeyboardInterrupt: print("\n\n Прервано пользователем") except Exception as e: print(f"\n Критическая ошибка: {e}") finally: producer.close()
```

## Часть 2: Настройка Spark Streaming для чтения из Kafka

### 2.1 Инициализация Spark Session с поддержкой Kafka

```

from pyspark.sql import SparkSession from pyspark.sql.functions import * from pyspark.sql.types import *
* from pyspark.sql.window import Window import warnings warnings.filterwarnings('ignore')

spark = SparkSession.builder
.appName("KafkaSparkIntegration")
.config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0")
.config("spark.sql.streaming.checkpointLocation", "/tmp/spark-checkpoints")
.config("spark.sql.shuffle.partitions", "4")

```

```
.config("spark.streaming.backpressure.enabled", "true")
.config("spark.streaming.kafka.maxRatePerPartition", "1000")
.master("local[*]")
.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

print(" Spark Session создана с поддержкой Kafka") print(f" Версия Spark: {spark.version}") print(f" Конфигурация:") print(f" - Checkpoint location: /tmp/spark-checkpoints") print(f" - Shuffle partitions: 4")
print(f" - Backpressure enabled: true")
```

## 2.2 Определение схемы для транзакций

```
transaction_schema = StructType([ StructField("transaction_id", StringType(), True),
StructField("user_id", StringType(), True), StructField("amount", DoubleType(), True), StructField("type",
StringType(), True), StructField("timestamp", TimestampType(), True), StructField("location",
StringType(), True), StructField("currency", StringType(), True), StructField("merchant_id", StringType(),
True), StructField("status", StringType(), True), StructField("category", StringType(), True),
StructField("fraud_score", DoubleType(), True) ])
```

```
print(" Схема транзакций определена:") transaction_schema
```

## 2.3 Чтение потоковых данных из Kafka

```
kafka_df = spark.readStream
.format("kafka")
.option("kafka.bootstrap.servers", "localhost:9092")
.option("subscribe", "real-time-transactions")
.option("startingOffsets", "earliest")
.option("maxOffsetsPerTrigger", "100")
.option("failOnDataLoss", "false")
.load()
```

```
print(" Подключение к Kafka установлено") print(f" Схема Kafka DataFrame:") kafka_df.printSchema()
```

```
print(f"\n Настройки Kafka consumer:") print(f" - Bootstrap servers: localhost:9092") print(f" - Topic: real-
time-transactions") print(f" - Starting offsets: earliest") print(f" - Max offsets per trigger: 100") print(f" - Is
streaming: {kafka_df.isStreaming}")
```

## 2.4 Парсинг и преобразование данных

```
def parse_kafka_data(df): # Парсим JSON из value колонки
parsed_df = df.select(
col("key").cast("string").alias("user_key"), from_json(col("value").cast("string"),
transaction_schema).alias("data"), col("timestamp").alias("kafka_timestamp"), col("partition"),
col("offset")).select( "user_key", "data.*", "kafka_timestamp", "partition", "offset" )

return parsed_df
```

```
parsed_transactions = parse_kafka_data(kafka_df)

print(" Данные успешно распарсены") print(f" Схема распарсенных данных:")
parsed_transactions.printSchema()
```

## 2.5 Базовая обработка данных

```
processed_df = parsed_transactions
.filter(col("status") == "success")
.withColumn("processing_timestamp", current_timestamp())
.withColumn("amount_usd", when(col("currency") == "EUR", col("amount") * 1.1) .when(col("currency")
== "GBP", col("amount") * 1.3) .otherwise(col("amount")) )
.withColumn("is_high_value", col("amount") > 500.0)
.withColumn("is_suspicious", col("fraud_score") > 0.8)
.select( "transaction_id", "user_id", "type", "amount", "currency", "amount_usd", "location", "category",
"status", "fraud_score", "is_high_value", "is_suspicious", "timestamp", "kafka_timestamp",
"processing_timestamp", "partition", "offset" )
```

```
print(" Данные обработаны") print(f" Схема обработанных данных:") processed_df.printSchema()
```