Вот пример кода для стриминговой системы подсчета типов автомобилей, проходящих через мост, реализованной на Python с использованием:

* **Apache Kafka** (для приема потока данных),
* **Apache Spark Structured Streaming** (для обработки данных в реальном времени),
* **PySpark** с **параллелизацией** и **группировкой по типу автомобиля**.

## **📦 Предварительные требования**

Установлен Kafka:  
  
 bash  
КопироватьРедактировать  
brew install kafka # или через docker

pip install kafka-python

Установлен PySpark:  
  
 bash  
КопироватьРедактировать  
pip install pyspark

1. Запущен Kafka-брокер и создан топик car\_stream.

## **🔧 Kafka Producer (эмуляция датчика)**

python

КопироватьРедактировать

# car\_producer.py

import json

import random

import time

from kafka import KafkaProducer

producer = KafkaProducer(bootstrap\_servers='localhost:9092',

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

car\_types = ['sedan', 'truck', 'bus', 'motorcycle', 'suv']

while True:

car = {

'type': random.choice(car\_types),

'timestamp': time.time()

}

producer.send('car\_stream', value=car)

time.sleep(random.uniform(0.1, 1)) # случайный интервал

## **⚙️ Spark Structured Streaming Consumer (подсчет и параллелизация)**

python

КопироватьРедактировать

# car\_counter\_stream.py

from pyspark.sql import SparkSession

from pyspark.sql.functions import from\_json, col, window, count

from pyspark.sql.types import StructType, StringType, DoubleType

# Создание SparkSession

spark = SparkSession.builder \

.appName("CarTypeStreamingCounter") \

.master("local[\*]") \

.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# Определение схемы входного JSON

schema = StructType() \

.add("type", StringType()) \

.add("timestamp", DoubleType())

# Подключение к Kafka

df\_raw = spark.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("subscribe", "car\_stream") \

.load()

# Преобразование value из байт в JSON

df\_parsed = df\_raw.select(from\_json(col("value").cast("string"), schema).alias("data")) \

.select("data.\*")

# Группировка по типу автомобиля и оконному интервалу

df\_grouped = df\_parsed.withColumn("event\_time", col("timestamp").cast("timestamp")) \

.groupBy(

window(col("event\_time"), "10 seconds"),

col("type")

).agg(count("\*").alias("count"))

# Вывод результата в консоль

query = df\_grouped.writeStream \

.outputMode("complete") \

.format("console") \

.option("truncate", False) \

.start()

query.awaitTermination()

## **🧪 Результат (пример вывода в консоль)**

pgsql

КопироватьРедактировать

+------------------------------------------+----------+-----+

|window |type |count|

+------------------------------------------+----------+-----+

|{2025-07-01 12:00:00, 2025-07-01 12:00:10}|truck |5 |

|{2025-07-01 12:00:00, 2025-07-01 12:00:10}|sedan |7 |

|{2025-07-01 12:00:00, 2025-07-01 12:00:10}|motorcycle|2 |

+------------------------------------------+----------+-----+

## **⚡ Пояснение**

* Kafka producer эмулирует поток машин, проезжающих через мост.
* Spark Streaming читает данные, преобразует их и группирует по типу.
* Обработка параллельна благодаря Spark, использующему все доступные CPU.
* Группировка по 10-секундному окну позволяет получать статистику в реальном времени.

## **🧩 Возможные улучшения**

* Сохранение результатов в базу данных или Data Lake.
* Расширенная схема: номер авто, скорость, направление и т.п.
* Алёрты при превышении порогов по типу транспорта (например, слишком много грузовиков).

Вот улучшенная версия стриминговой системы подсчёта автомобилей с:

1. **Сохранением в PostgreSQL (можно заменить на Data Lake / S3 / HDFS)**
2. **Расширенной схемой: номер, скорость, направление, тип**
3. **Алёртами при превышении порога по типу авто (например, грузовики > N)**

## **🔧 Обновлённый Kafka Producer (эмуляция расширенного потока)**

python

КопироватьРедактировать

# car\_producer\_v2.py

import json

import random

import time

from kafka import KafkaProducer

producer = KafkaProducer(bootstrap\_servers='localhost:9092',

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

car\_types = ['sedan', 'truck', 'bus', 'motorcycle', 'suv']

directions = ['north', 'south', 'east', 'west']

def generate\_plate():

return f"{random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ')}{random.randint(100,999)}{random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ')}"

while True:

car = {

'plate': generate\_plate(),

'type': random.choice(car\_types),

'speed': round(random.uniform(20, 120), 1), # км/ч

'direction': random.choice(directions),

'timestamp': time.time()

}

producer.send('car\_stream', value=car)

time.sleep(random.uniform(0.1, 1))

## **🧱 Расширенная схема + алёрты + запись в PostgreSQL**

python

КопироватьРедактировать

# car\_counter\_stream\_v2.py

from pyspark.sql import SparkSession

from pyspark.sql.functions import from\_json, col, window, count, avg, max as max\_, lit

from pyspark.sql.types import StructType, StringType, DoubleType, TimestampType

# Схема расширенного события

schema = StructType() \

.add("plate", StringType()) \

.add("type", StringType()) \

.add("speed", DoubleType()) \

.add("direction", StringType()) \

.add("timestamp", DoubleType())

# Порог для алёртов

ALERT\_THRESHOLD = 10 # например, если грузовиков > 10 в окне

spark = SparkSession.builder \

.appName("CarTypeExtendedCounter") \

.master("local[\*]") \

.config("spark.sql.shuffle.partitions", "4") \

.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# Чтение из Kafka

df\_raw = spark.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("subscribe", "car\_stream") \

.load()

# Преобразование

df = df\_raw.select(from\_json(col("value").cast("string"), schema).alias("data")) \

.selectExpr("data.\*") \

.withColumn("event\_time", col("timestamp").cast(TimestampType()))

# ===================

# Основная агрегация

# ===================

agg\_df = df.groupBy(

window(col("event\_time"), "10 seconds"),

col("type")

).agg(

count("\*").alias("vehicle\_count"),

avg("speed").alias("avg\_speed"),

max\_("speed").alias("max\_speed")

)

# ================

# Алёрты: много грузовиков

# ================

alerts\_df = agg\_df.filter((col("type") == "truck") & (col("vehicle\_count") > ALERT\_THRESHOLD)) \

.withColumn("alert\_level", lit("CRITICAL"))

# ====================

# Запись в PostgreSQL

# ====================

def write\_to\_postgres(df, epoch\_id):

df.write \

.format("jdbc") \

.option("url", "jdbc:postgresql://localhost:5432/traffic") \

.option("dbtable", "car\_statistics") \

.option("user", "postgres") \

.option("password", "password") \

.option("driver", "org.postgresql.Driver") \

.mode("append") \

.save()

agg\_query = agg\_df.writeStream \

.foreachBatch(write\_to\_postgres) \

.outputMode("update") \

.start()

# ====================

# Вывод алёртов в консоль

# ====================

alert\_query = alerts\_df.writeStream \

.outputMode("update") \

.format("console") \

.option("truncate", False) \

.start()

agg\_query.awaitTermination()

alert\_query.awaitTermination()

## **🗃️ SQL: Таблица PostgreSQL (car\_statistics)**

sql

КопироватьРедактировать

CREATE TABLE car\_statistics (

window\_start TIMESTAMP,

window\_end TIMESTAMP,

type VARCHAR,

vehicle\_count INT,

avg\_speed DOUBLE PRECISION,

max\_speed DOUBLE PRECISION

);

## **📈 Результат**

**Пример алёрта в консоли:**

pgsql

КопироватьРедактировать

+------------------------------------------+-----+-------------+------------------+---------+

|window |type |vehicle\_count|avg\_speed |alert\_level|

+------------------------------------------+-----+-------------+------------------+---------+

|{2025-07-01 12:00:00, 2025-07-01 12:00:10}|truck|15 |64.2 |CRITICAL |

+------------------------------------------+-----+-------------+------------------+---------+

## **💡 Возможные расширения**

Хранение в **Data Lake (например, S3, HDFS)** вместо PostgreSQL:  
  
 python  
КопироватьРедактировать  
df.write.parquet("s3://bucket/car\_stats/", mode="append")

* Отправка алёртов в Slack, Telegram, email и др.
* Построение дешборда (например, Apache Superset, Grafana, Redash) по данным из PostgreSQL.

В car\_counter\_stream.py при запуске возникает ошибка "pyspark.errors.exceptions.base.PySparkRuntimeError: [JAVA\_GATEWAY\_EXITED] Java gateway process exited before sending its port number.", а car\_producer.py возникает ошибка "kafka.errors.NoBrokersAvailable: NoBrokersAvailable"

## **🐍 Ошибка PySpark:**

### **❌ PySparkRuntimeError: [JAVA\_GATEWAY\_EXITED] Java gateway process exited before sending its port number.**

Это означает, что **PySpark не может запустить Java Gateway** — вероятно, из-за:

### **✅ Решение:**

**Проверь установлен ли Java** (Spark требует Java 8 или 11):  
  
 bash  
КопироватьРедактировать  
java -version

* + Если Java не установлен:

Установи OpenJDK 11:  
  
 bash  
КопироватьРедактировать  
sudo apt install openjdk-11-jdk # Ubuntu

brew install openjdk@11 # macOS

1. **Укажи переменную окружения JAVA\_HOME**:

Добавь в ~/.bashrc, ~/.zshrc, ~/.profile (или в текущую сессию):  
  
 bash  
КопироватьРедактировать  
export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64 # Linux

export JAVA\_HOME=/opt/homebrew/opt/openjdk@11 # macOS M1

Проверь:  
  
 bash  
КопироватьРедактировать  
echo $JAVA\_HOME

ls $JAVA\_HOME/bin/java

1. **Перезапусти терминал или IDE**.

## **🛰️ Ошибка Kafka:**

### **❌ kafka.errors.NoBrokersAvailable: NoBrokersAvailable**

Эта ошибка означает, что **Kafka не запущен**, либо ты обращаешься к **неправильному адресу**.

### **✅ Решение:**

#### **🔹 Если Kafka установлен локально:**

Убедись, что Kafka и Zookeeper **запущены**:  
  
 bash  
КопироватьРедактировать  
# Запуск ZooKeeper

bin/zookeeper-server-start.sh config/zookeeper.properties

# В другом окне — запуск Kafka

bin/kafka-server-start.sh config/server.properties

Проверь доступность брокера:  
  
 bash  
КопироватьРедактировать  
telnet localhost 9092

Создай топик car\_stream, если ещё нет:  
  
 bash  
КопироватьРедактировать  
bin/kafka-topics.sh --create --topic car\_stream --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

#### **🔹 Если Kafka работает в Docker:**

Создай docker-compose.yml:

yaml

КопироватьРедактировать

version: '3'

services:

zookeeper:

image: confluentinc/cp-zookeeper:7.0.1

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

kafka:

image: confluentinc/cp-kafka:7.0.1

ports:

- "9092:9092"

environment:

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

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

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

Запусти:

bash

КопироватьРедактировать

docker-compose up -d

## **📌 Быстрая проверка после исправлений**

### **Kafka:**

bash

КопироватьРедактировать

# Создание топика (если нужно)

kafka-topics.sh --create --topic car\_stream --bootstrap-server localhost:9092

# Просмотр сообщений

kafka-console-consumer.sh --topic car\_stream --bootstrap-server localhost:9092 --from-beginning

### **Java + PySpark:**

bash

КопироватьРедактировать

python -c "from pyspark.sql import SparkSession; SparkSession.builder.getOrCreate().range(5).show()"

## **🧠 Если нужно — могу помочь:**

* Настроить Kafka через Docker / Linux
* Установить Spark с Java в Windows / WSL
* Проверить соединение между Kafka и Spark

Хочешь, чтобы я сгенерировал готовый Docker Compose-пакет для Kafka + Spark + PostgreSQL для полной автономной системы?