Apache Flink para processamento em tempo real de dados IoT na nuvem

1st Luiza Martins de Freitas Cintra
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Goiás
cintraluiza@discente.ufg.br

2nd Matheus Yosimura Lima
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Goiás
matheusyosimura@discente.ufg.br

3rd Mateus Eduardo Silva Ribeiro
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Goiás
mateus.eduardo@discente.ufg.br

4th Humberto Vittório Almeida Guadagnini
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Goiás
hvittorioguadagnini@discente.ufg.br

5th Isabela de Queiroz Rodrigues
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Goiás
Isabela.queiroz@discente.ufg.br

***Abstract*—O presente estudo visa promover uma resolução de problema relacionado ao processamento de dados em tempo real coletados por um dispositivo IoT. O paradigma envolve a captação de dados de temperatura em tempo real através da utilização de sensores. Por meio destes, haverá a emissão de um aviso alertando sobre picos de temperatura.**

***Index Terms*—IoT, sensores, Sistema Operacional, Cloud**

I. INTRODUÇÃO E REVISÃO BIBLIOGRÁFICA

A "Internet das Coisas" surgiu recentemente como um novo conceito de rede, que abrange comunicações e processamento dos mais diversos equipamentos [1]. IoT, como é popularmente conhecida, é uma nova visão para a internet, em que ela passa a abranger não só computadores, como também, objetos do dia a dia [2].

As aplicações mais vislumbradas pela IoT, atualmente, é a telemetria, coleta de dados, atuação direta com os mais diversos objetos, relacionamentos em rede, entre muitos outros casos. Ela também aparece relacionada ao conceito de "Big Data", muito difundido e estudado na contemporaneidade, cuja ideia está relacionada a uma expansão inimaginável de dados. Esses dados são gerados e coletados continuamente por objetos e computadores, indicando um enorme volume de memorização e processamento. A possibilidade de existência da IoT ocorre com um avanço específico do protocolo da internet, em que cada equipamento passa a ter o seu "endereço IP" próprio.

O Apache Flink é um mecanismo distribuído de código aberto para processamento com estado em conjunto de dados ilimitados (fluxos) e limitados (lotes) [3]. As aplicações de processamento são projetadas para serem executadas continuamente, tendo um mínimo de tempo de inatividade, e para que haja o processamento de dados à medida em que são ingeridos. O Apache Flink foi pensado para processamento de baixa latência, execução de cálculos na memória, alta disponibilidade, remoção de pontos únicos de falhas e aumento da escala na horizontal.

Os atributos do Apache Flink incluem gerenciamento avançado de estado com garantias de consistência exatamente uma vez, semântica de processamento de tempo de evento com tratamento sofisticado de dados atrasados e fora de ordem [3]. Ele foi desenvolvido para priorizar o streaming. Justamente por conter todas essas características, ele foi pensado para a utilização de captações de picos de temperatura em ambientes, conectado em sensores.

II. FUNDAMENTOS TEÓRICOS

O Apache Flink é usado para criar os mais diferentes tipos de aplicações de transmissão e em lotes, devido ao amplo conjunto de atributos. As aplicações mais comuns desenvolvidas com essa ferramenta são:

Aplicações orientadas por eventos: ingerindo eventos de um ou mais fluxos de eventos e executando cálculos, atualizações de estado ou ações externas.

Aplicações de análise de dados: extraindo informações e insights de dados. É executada consultando conjuntos de dados finitos e reexecutando as consultas ou alterando os resultados para que consiga incorporar novos dados. Com ele, é possível que ocorra a atualização de dados em tempo real, de forma constante.

Aplicações de pipelines de dados: transformando os dados a serem removidos de um armazenamento para outro. Tradicionalmente, extract-transform-load (ETL) é executado periodicamente, em lotes. Vide (Figura 1).

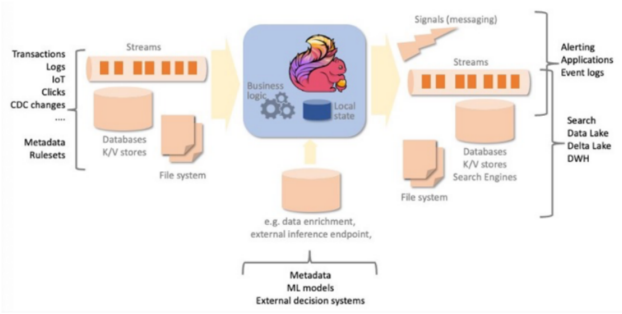

Fig. 1. Execução de aplicações.

O Flink é um mecanismo de processamento de fluxo de alto throughput e baixa latência. Uma aplicação do Flink consiste em um gráfico de fluxo de dados acíclico complexo e arbitrário, composto por fluxos e transformações. Os dados são ingeridos de uma ou mais fontes de dados e enviados para um ou mais destinos. Os sistemas de fonte e destino podem ser fluxos, filas de mensagens ou datastores e incluem arquivos, bancos de dados populares e mecanismos de busca. As transformações podem ser com estado, como agregações ao longo de janelas de tempo ou detecção de padrões complexos [3].

A tolerância a falhas é obtida por dois mecanismos separados: pontos de verificação automática e periódica do estado da aplicação, copiados para um armazenamento persistente, para permitir a recuperação automática em caso de falha; pontos de salvamento sob demanda, salvando uma imagem consistente do estado de execução, para permitir a pausa e retomada, atualizar ou bifurcar sua tarefa do Flink, mantendo o estado da aplicação em todas as pausas e reinicializações. Os mecanismos de ponto de verificação e ponto de salvamento são assíncronos, capturando um snapshot consistente do estado sem "parar o mundo", enquanto a aplicação continua processando eventos. Vide (Figura 2).

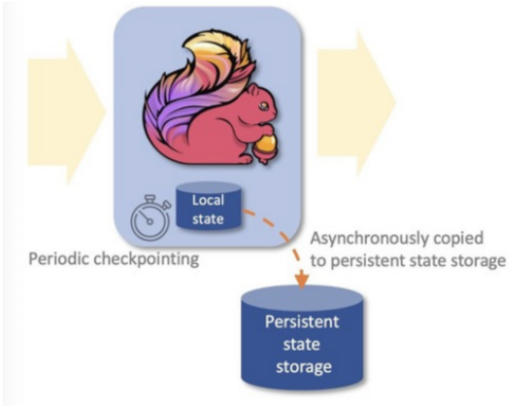

Fig. 2. Mecanismo de funcionamento.

Os benefícios de utilização do Apache Flink incluem:

Processamento de conjunto de dados ilimitados e limitados: o Apache Flink fornece algoritmos e estruturas de dados para suportar processamento limitado e ilimitado por meio da mesma interface de programação. As aplicações que processam dados ilimitados são executadas continuamente. As aplicações que processam dados limitados encerrarão sua execução ao chegarem ao final dos conjuntos de dados de entrada;

Execução de aplicações em grande escala: o Apache Flink foi projetado para executar aplicações com estado em praticamente qualquer escala. O processamento é paralelo a milhares de tarefas, distribuído em várias máquinas, simultaneamente;

Desempenho na memória: os dados que fluem pela aplicação e pelo estado são particionados em várias máquinas. Portanto, o cálculo pode ser concluído acessando dados locais, geralmente na memória;

Consistência de estado exatamente uma vez: o Apache Flink garante a consistência do estado interno, mesmo em caso de falha e durante a parada e reinicialização da aplicação. O efeito de cada mensagem no estado interno é sempre aplicado exatamente uma vez, independentemente de a aplicação receber duplicatas da fonte de dados na recuperação ou na reinicialização;

Ampla variedade de conectores: vários conectores comprovados para sistemas populares de mensagens e streaming, armazenamentos de dados, mecanismos de pesquisa e sistemas de arquivos. Alguns exemplos são Apache Kafka, Amazon Kinesis Data Streams, Amazon SQS, Active MQ, Rabbit MQ, NiFi, OpenSearch e Elasticsearch, DynamoDB, HBase e qualquer banco de dados que forneça o cliente JDBC;

Vários níveis de abstrações: o Apache Flink oferece vários níveis de abstração para a interface de programação. Do SQL de streaming de alto nível e da API de tabela, usando abstrações familiares, como tabela, junções e agrupamento. A API DataStream oferece um nível mais baixo de abstração, mas também mais controle, com a semântica de fluxos, janelas e mapeamento. E, finalmente, a API ProcessFunction oferece controle rigoroso sobre o processamento de cada mensagem e controle direto do estado. Vide (Figura 3).

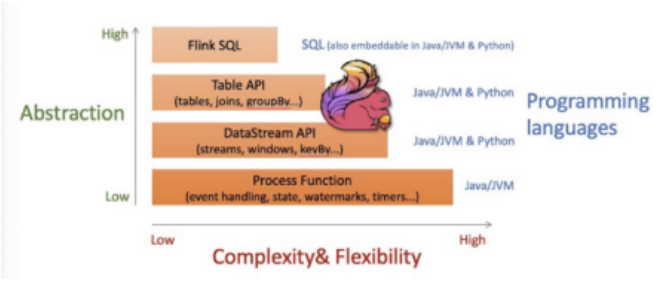

Fig. 3. Abstrações.

Várias linguagens de programação: o Apache Flink pode ser programado com várias linguagens, desde o SQL de streaming de alto nível até Python, Scala, Java, mas também com outras linguagens JVM, como Kotlin.

A IoT é um conceito que está fora do âmbito das tecnologias, pois não deriva delas, e sim as utiliza para cumprir uma série de funcionalidades. Em termos de protocolos diversos

e capazes de expandir a nova “rede de objetos”, há ainda protocolos adicionais do ambiente industrial e predial, pois os objetos existentes e já operandos em tais sistemas não só podem como serão envolvidos na grande rede de interligação de objetos.

As funcionalidades do objeto na IoT já estão postas pelo mercado e pelas organizações normativas, e as tecnologias associadas continuam em desenvolvimento. A Intel, que fabrica os componentes para funcionalidade de “processamento” dos objetos, definindo assim suas características, acredita em “inteligente devices to deliver intelligence where needed and to acquire and filter data from the field” [5]. Esses sistemas irão compor sistemas inteligentes, integrando bilhões de dispositivos e provendo soluções e análises para valorizar soluções fim a fim dos clientes.

Para o lado dos desenvolvedores de aplicações, há a multiplicação de novas soluções, e novas expressões podem ser citadas: smart buildings, smart cities, smart transport, smart grid, smart energy, smart health, entre outras [5].

No conjunto das características do objeto na IoT, existem as seguintes atribuições [5]:

Processamento: capacidade de processamento computacional inserida no objeto, ou inteligência capaz de fazê-lo agir e responder às requisições de IoT;

Endereçamento: capacidade do objeto de ser encontrado na IoT, ou seja, ser localizado na rede por roteamento;

Identificação: identidade de cada objeto;

Localização: atributo relacionado ao local físico em que o objeto se encontra.

Ao conectar objetos com diferentes recursos a uma rede, potencializa-se o surgimento de novas aplicações. Neste sentido, conectar esses objetos à Internet significa criar a Internet das Coisas. Na IoT, os objetos podem prover comunicação entre usuários, dispositivos. Com isto emerge uma nova gama de aplicações, tais como coleta de dados de pacientes e monitoramento de idosos, sensoriamento de ambientes de difícil acesso e inóspitos, entre outras.

A arquitetura básica dos objetos inteligentes é composta por quatro unidades: processamento/memória, comunicação, energia e sensores/atuadores. São descritos a seguir a interligação do componentes de dispositivos:

- Unidade(s) de processamento/memória: composta de uma memória interna para armazenamento de dados e programas, um microcontrolador e um conversor analógico-digital para receber sinais dos sensores. As CPUs utilizadas nesses dispositivos são, em geral, as mesmas utilizadas em sistemas embarcados e comumente não apresentam alto poder computacional. Frequente- mente existe uma memória externa do tipo flash, que serve como memória secundária, por exemplo, para manter um “log” de dados. As características desejáveis para estas unidades são consumo reduzido de energia e ocupar o menor espaço possível.
- Unidade(s) de comunicação: consiste de pelo menos um canal de comunicação com ou sem fio, sendo mais comum o meio sem fio. Neste último caso, a maioria das plata- formas usam rádio de baixo custo e baixa potência. Como consequência, a comunicação é de curto alcance e apresentam perdas frequentes. Esta unidade básica será objeto de estudo mais detalhado na próxima seção.
- Fonte de energia: responsável por fornecer energia aos componentes do objeto inteligente. Normalmente, a fonte de energia consiste de uma bateria (recarregável ou não) e um conversor AC-DC e tem a função de alimentar os componentes. Entretanto, existem outras fontes de alimentação como energia elétrica, solar e mesmo a captura de energia do ambiente através de técnicas de conversão (e.g., energia mecânica em energia elétrica), conhecidas como energy harvesting.
- Unidade(s) de sensor(es)/atuador(es): realizam o monitoramento do ambiente no qual o objeto se encontra. Os sensores capturam valores de grandezas físicas como temperatura, umidade, pressão e presença. Atualmente, existem literalmente centenas de sensores diferentes que são capazes de capturar essas grandezas. Atuadores, como o nome indica, são dispositivos que produzem alguma ação, atendendo a comandos que podem ser manuais, elétricos ou mecânicos.

A seção subsequente discute uma das aplicações práticas dessas ferramentas, destacando um caso de uso comum e implementação exemplar.

III. METODOLOGIA

O objetivo central da pesquisa construir em tempo real um meio de comunicação para que analisar e gerar métricas pautadas nos dados de temperatura residenciais e periódicos coletados por sensores, com seu sistemas individuais de processamento, com a finalidade de detectar picos inesperados. Portanto, fundamentalmente a solução definida foi a construção de um pipeline na biblioteca de Python, PyFlink, que seja capaz de receber os dados coletados, analisá-los sistematicamente, gerar métricas em cima da coletas e ser capaz de apresentar dashboards de simples entendimento para informar o usuário da interface.

Inicialmente, foram necessárias a instalação de tecnologias localmente com finalidades diversas como apresentado na referência bibliográfica [5], como será descrito a seguir em ordem de prioridade:

- **Python:** A instalação do Python, primordialmente é indispensável por ser a linguagem trabalhada no core do desenvolvimento.
- **Docker:** O docker atua como uma camada de virtualização do sistema físico de um computador, capaz de carregar consigo inúmeras pré-configurações de bibliotecas que serão acrescentadas no projeto e sem a preocupação de configuração milimetricamente do ambiente de desenvolvimento.
- **Apache Flink:** O Apache é projetado para processar grandes volumes de dados em tempo real e também oferece capacidades de processamento de dados em lote.

- **Apache Kafka:** Será responsável por armazenar os dados coletados e transferir através de APIs para as tecnologias que solicitarem.
- **Kibana:** Uma interface de usuário (UI) que funciona em cima do Elasticsearch, proporcionando recursos avançados de visualização e análise de dados.
- **Elastic Search:** É um mecanismo de busca e análise de dados em tempo real.
- **Zookeeper:** Uma tecnologia responsável por centralizar a manutenção das configurações, sinconização e serviços.
- **Amazon EC2:** A máquina virtual da Aws servirá como ferramenta poderosa de nuvem neste projeto, pois não será necessário realizar as instalações e configurações localmente, todo o procedimento poderá ter um ambiente dedicado para sua realização, economizando espaço local, ganhando velocidade de processamento e garantindo a confiabilidade de um serviço em nuvem.

Há utilização do Kafka para armazenar dados de entrada sobre temperatura. Um script atuando como gerador de dados simples para simular o recebimento de dados de uma IoT, chamado `generate_source_data.py` é fornecido para gravar em tempo real novos registros no `temperature_msg` tópico Kafka. Cada registro foi estruturado da seguinte forma: `{"createTime": "2020-08-12 06:29:02", "orderId": 1597213797, "tempAmount": 29.159}`

- **createTime:** A hora da criação.
- **orderId:** O id da temperatura atual.
- **tempAmount:** O valor da temperatura.

Os dados de temperatura serão processados com PyFlink usando o script Python `temperature_msg.py`. Este script primeiro mapeará os orderId registros e calculará a soma dos valores das transações para cada id. O ambiente é baseado no Docker Compose. Ele usa uma imagem personalizada para ativar o Apache Flink (JobManager + TaskManager), Kafka+Zookeeper, o gerador de dados e contêineres Elastic-Search+Kibana.

Ademais, foi construída uma máquina virtual através do Amazon EC2, onde configurou-se uma imagem Docker e sendo baixadas todas as imagens das tecnologias mencionadas anteriormente. Após a ambientação estar completa os serviços disponibilizaram no url `http://localhost:8081` as métricas necessárias para análise. Durante todo o processo para o fortalecimento da metodologia foram realizados testes por grandes quantidades de tempo com temperaturas fictícias variadas.

IV. RESULTADOS E CONCLUSÕES

Por fim, obteve-se uma profunda quantidade amostral de dados capaz da geração de diversas métricas importantes para a análise. Sendo capaz, portanto, de aprofundar os saberes no tópico Apache Flink para processamento em tempo real de dados IoT na nuvem, sendo capaz futuramente de impulsionar trabalhos científicos interessados em utilizar dados oriundo de IoT e processados em tempo real na nuvem para automatizar ambientes residenciais ou empresariais.

Os resultados obtidos foram muito satisfatórios para o esperado do projeto, as métricas apresentadas estavam de acordo com o esperado do espaço amostral com uma eficácia de `100%`. Portanto, concluí-se a oportunidade tecnológica para um cenário que apresenta como core este processamento, sem a necessidade da utilização de recursos locais.

A figura a seguir não estará apresentando um resultado oriundo dos resultados obtidos neste artigo, porém servirão como exemplo prático do produto final de entrega semelhante:

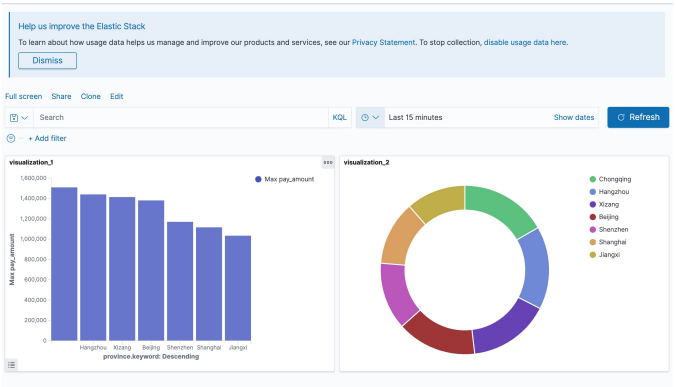

Fig. 4. Dashboard das métricas no Kibana.

REFERENCES

[1] Faccioni Filho, Mauro. "Internet das coisas." Unisul Virtual(2016)
[2] https://aws.amazon.com/pt/what-is/apache-flink/
[3] Santos, Bruno P., et al. "Internet das coisas: da teoria à prática." Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos 31 (2016): 16.
[4] Santaella, Lucia, et al. "Desvelando a Internet das coisas." Revista GEMInIS 4.2 (2013): 19-32.
[5] https://github.com/apache/flink-playgrounds/tree/master/pyflink-walkthrough
[6] https://thingspeak.com/channels/public
[7] https://medium.com/@daeynasvistas/a-iot-internet-das-coisas-surgiu-como-a-nova-gera
[8] https://www.confluent.io/blog/apache-flink-stream-processing-use-cases-with-examples/

V. Apêndice

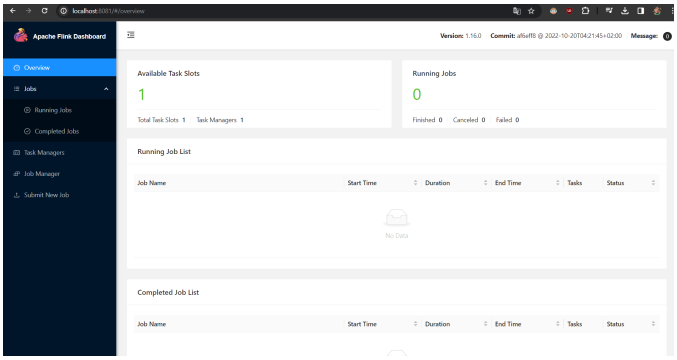

Fig. 5. Interface Web Apache Flink.

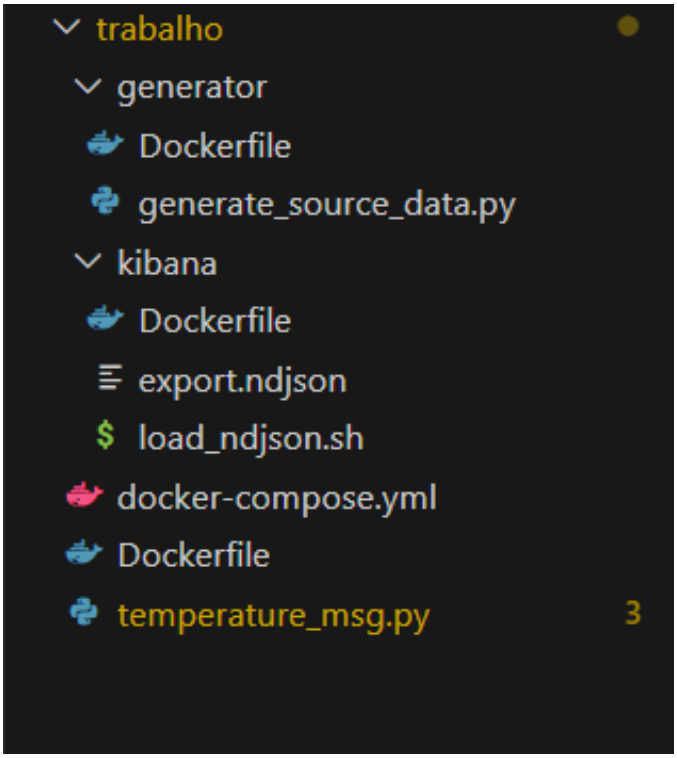

Fig. 6. Hierarquia da Árvore do Projeto.

```yaml
version: '2.1'
services:
  jobmanager:
    build: .
    image: pyflink/pyflink:1.16.0-scala_2.12
    volumes:
      - .:/opt/trabalho
    hostname: "jobmanager"
    expose:
      - "6123"
    ports:
      - "8081:8081"
    command: jobmanager
    environment:
      - JOB_MANAGER_RPC_ADDRESS=jobmanager
  taskmanager:
    image: pyflink/pyflink:1.16.0-scala_2.12
    volumes:
    - .:/opt/trabalho
    expose:
      - "6121"
      - "6122"
    depends_on:
      - jobmanager
    command: taskmanager
    links:
      - jobmanager:jobmanager
    environment:
      - JOB_MANAGER_RPC_ADDRESS=jobmanager
  zookeeper:
    image: wurstmeister/zookeeper:3.4.6
    ulimits:
      nofile:
        soft: 65536
        hard: 65536
    ports:
      - "2181:2181"
  kafka:
    image: wurstmeister/kafka:2.13-2.8.1
    ports:
      - "9092:9092"
    depends_on:
      - zookeeper
    environment:
      HOSTNAME_COMMAND: "route -n | awk '/UG[ \t]/{print $$2}'"
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_CREATE_TOPICS: "temperature_msg:1:1"
  generator:
    build: generator
    image: generator:1.0
    depends_on:
      - kafka
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.8.0
    environment:
      - cluster.name=docker-cluster
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
      - discovery.type=single-node
    ports:
      - "9200:9200"
      - "9300:9300"
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
  kibana:
    image: docker.elastic.co/kibana/kibana:7.8.0
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch
  load-kibana-dashboard:
    build: ./kibana
    command: ['/bin/bash', '-c', 'cat /tmp/load/load_ndjson.sh | tr -d "\r" | bash']
    depends_on:
      - kibana

```

Fig. 7. Código para Configurar o Docker-Compose.

```python
import random
import time, calendar
from random import randint
from kafka import KafkaProducer
from kafka import errors
from json import dumps
from time import sleep


def write_data(producer):
    data_cnt = 20000
    order_id = calendar.timegm(time.gmtime())
    max_temp = 100
    topic = "temperature_msg"

    for i in range(data_cnt):
        ts = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        rd = random.random()
        order_id += 1
        temp_amount = max_temp * rd
        cur_data = {"createTime": ts, "orderId": order_id, "tempAmount": temp_amount}
        producer.send(topic, value=cur_data)
        sleep(0.5)

def create_producer():
    print("Connecting to Kafka brokers")
    for i in range(0, 6):
        try:
            producer = KafkaProducer(bootstrap_servers=['kafka:9092'],
                                     value_serializer=lambda x: dumps(x).encode('utf-8'))
            print("Connected to Kafka")
            return producer
        except errors.NoBrokersAvailable:
            print("Waiting for brokers to become available")
            sleep(10)

    raise RuntimeError("Failed to connect to brokers within 60 seconds")

if __name__ == '__main__':
    producer = create_producer()
    write_data(producer)
```

trabalho > temperature_msg.py > process_data

```python
from pyflink.datastream import StreamExecutionEnvironment, TimeCharacteristic
from pyflink.table import StreamTableEnvironment, DataTypes
from pyflink.table.udf import udf

def process_data(t_env):
    try:
        print('2-')

        create_kafka_source_ddl = """
            CREATE TABLE temperature_msg(
                createTime VARCHAR,
                orderId BIGINT,
                tempAmount DOUBLE
            ) WITH (
              'connector' = 'kafka',
              'topic' = 'temperature_msg',
              'properties.bootstrap.servers' = 'kafka:9092',
              'properties.group.id' = 'test_3',
              'scan.startup.mode' = 'latest-offset',
              'format' = 'json'
            )
            """

        print("4-")
        create_es_sink_ddl = """
                CREATE TABLE es_sink(
                    orderId BIGINT PRIMARY KEY,
                    temp_amount DOUBLE
                ) with (
                    'connector' = 'elasticsearch-7',
                    'hosts' = 'http://elasticsearch:9200',
                    'index' = 'platform_temp_amount_1',
                    'document-id.key-delimiter' = '$',
                    'sink.bulk-flush.max-size' = '42mb',
                    'sink.bulk-flush.max-actions' = '32',
                    'sink.bulk-flush.interval' = '1000',
                    'sink.bulk-flush.backoff.delay' = '1000',
                    'format' = 'json'
                )
        """
        print('5-')

        t_env.execute_sql(create_kafka_source_ddl)
        t_env.execute_sql(create_es_sink_ddl)
        print('6-')

        t_env.from_path("temperature_msg") \
            .select("tempAmount") \
            .group_by() \
            .select("avg(tempAmount) as avg_temperature") \
            .execute_insert("es_sink")

        print('7-')
    except Exception as e:
        print(f'Error: {e}')

if __name__ == '__main__':
    print('1-')

    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env)
    t_env.get_config().get_configuration().set_boolean("python.fn-execution.memory.managed", True)

    # Chamando a função de processamento
    process_data(t_env)

    print("Execution completed.")
```