Uma solução para alerta de preço de combustível em tempo real baseada em processamento de eventos complexos

A real time solution for fuel price alerts based on complex event processing

José de Arimatea Rocha Neto [1] orcid.org/0000-0001-5299-1086
Jorge Cavalcanti Fonsêca [1,2] orcid.org/0000-0001-7954-2766

[1] Escola Politécnica de Pernambuco, Universidade de Pernambuco, Recife, Brasil,
[2] Faculdade de Ciência e Tecnologia, Universidade de Pernambuco, Caruaru, Brasil.

E-mail do autor principal: José Arimatea Rocha Neto jarn@ecomp.poli.br

Resumo

Com o aumento na quantidade de dados produzidos atualmente, métodos de consulta e análise, como os tradicionais SGBDs, passaram a não mais atender os requisitos da nova onda de aplicações. Para resolver esse problema surgiram as ferramentas de processamento em *batch* como Hadoop e Spark. Entretanto, as demandas de processamento de dados em tempo real continuaram a não ser. A partir daí surgiram outras ferramentas como Kafka e Storm. Alguns problemas comuns que necessitam desse tipo de processamento são: detecção de invasão em sistemas de segurança e monitoramento de ambientes. Este trabalho tem como objetivo apresentar uma solução que se utiliza das ferramentas de processamento em tempo real para facilitar o acesso da população da cidade do Recife às constantes atualizações de preços nos postos de combustível. Como resultado, espera-se obter alertas com informações úteis para os usuários da solução e abrir caminho para mais trabalhos na mesma área.

Palavras-Chave: Processamento de Streams; Processamento em Tempo Real; CEP; Processamento de Eventos Complexos; Kafka; KSQL.

Abstract

With the increasing amount of data available to be processed, traditional data processing methods are no longer enough. To solve this problem, batch processing tools as Hadoop and Spark came into play. However, a new category of problems arose. Those that require continuous processing and real time results. Then, a new set of tools was created, such as Kafka and Storm. Some common problems that require this type of data processing are: intrusion detection in security systems and environmental monitoring. This paper aims to present a solution, using real time processing tools, to facilitate the access of the population of the city Recife, Brazil to the constant fuel price updates at the gas stations. As a result, is expected from the solution to generate alerts with helpful information for the users and open the way for more work in the same area.

Key-words: *Stream Processing; Real Time Processing, CEP; Complex Event Processing; Kafka; KSQL.*

DOI: 1025286/repa.v5i1.1178

1 INTRODUÇÃO

A Internet tem sido usada cada vez mais nos dias de hoje. Segundo o *Our World in Data*, o número de usuários da Internet subiu de 44,4 milhões em 1995 para 3.4 bilhões em 2016 **[1]**. Além disso, em 2015 tínhamos 15,4 bilhões de dispositivos com acesso à Internet e estima-se que em 2025 serão 75,4 bilhões **[2]**.

Esse aumento no uso da Internet tem gerado um crescimento na quantidade de dados produzidos pelos dispositivos. De acordo com um relatório da Kleiner Perkins, em 2005 tínhamos cerca de 0.1ZB de dados trafegando na Internet. Em 2015 já eram cerca de 12ZB e estima-se que em 2025 esse volume possa chegar a 163ZB, como pode ser visto na Figura 1 **[3]**.

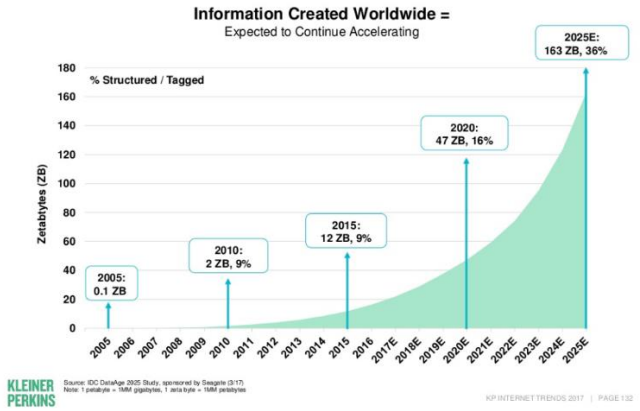

Figura 1: Crescimento do volume de dados na Internet.
Fonte: Kleiner Perkins **[2]**.

Esse crescimento na quantidade de dados tem impactado as formas de processamento das informações. Um exemplo são os SGBDs que, de acordo com Cugola, não conseguem atender aos requisitos não funcionais de performance desse tipo de processamento **[4]**.

Para trabalhar com uma grande quantidade de dados surgiram as ferramentas de processamento em *batch*, como Hadoop e Spark. Entretanto, existe um grande número de aplicações que requerem que os dados sejam processados em tempo real e de forma contínua, ou seja, sem interrupção. Para solucionar esse problema, emergiram os sistemas de processamento de fluxos de dados, ou chamados de *Event Streaming Processing Systems* **[4]**.

Esse trabalho tem como objetivo solucionar um problema que necessita de resposta em tempo real e de forma contínua através dos seguintes objetivos específicos:

- Entender as causas e consequências da variação nos preços de combustível no Brasil;
- Processar e gerar informações úteis a partir de dados de localização e alteração de preços;
- Facilitar o acesso da população às informações geradas pela solução.

Para que se possa entender melhor a solução proposta, a seção 2 faz um resumo do problema a ser solucionado. A seção 3 traz um referencial teórico das tecnologias utilizadas. A seção 4 faz uma análise de trabalhos relacionados. Já a seção 5 mostra toda a solução desenvolvida e seus resultados. A seção 6 descreve os resultados obtidos. Finalmente a seção 7 apresenta trabalhos futuros e levanta algumas conclusões sobre o trabalho.

2 O PROBLEMA

Apesar de como se imagina, o Brasil não está entre os países com o maior preço de gasolina no mundo. De acordo com um levantamento da *GlobalPetrolPrices*, o Brasil é apenas o 77° país nesse *ranking*, com um valor bem próximo da média mundial **[5].**

Apesar disso, o Brasil tem sofrido com uma constante alteração desse valor, como pode ser visto na Figura 4. Um dos motivos dessa variação é a abordagem adotada pela indústria brasileira onde o preço dos combustíveis tem acompanhado as oscilações internacionais, a variação do dólar e a cotação do petróleo **[6]**.

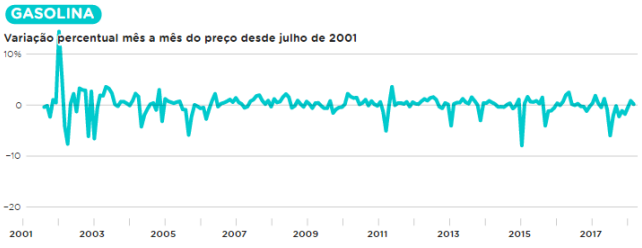

Figura 4: Variação percentual mês a mês do preço desde junho de 2001.
Fonte: Nexo **[7]**.

Como a volatilidade dos preços de combustível tem sido muito grande, os métodos de processamento que necessitam de dados armazenados para posterior processamento, como SGBDs e ferramentas de processamento em *batch,* não atendem os requisitos de velocidade de processamento gerados através da variação nos preços dos combustíveis.

Esse trabalho tem como objetivo desenvolver uma solução para que a população possa acompanhar essa variação de preço em tempo real, e que todos estejam informados sobre os valores atuais nos postos de combustível.

3 REFERENCIAL TEÓRICO

Esta seção apresenta os conceitos utilizados neste trabalho. A compreensão desses se faz importante para um melhor entendimento da pesquisa realizada. Para isso é preciso abordar alguns conceitos como o padrão *Publish/Subscribe*, Processamento de fluxos de informação em tempo real e Processamento de Eventos Complexos (CEP). Além disso, também é preciso entender as ferramentas Kafka, Kafka Streams e KSQL.

3.1 Padrão *Publish/Subscribe*

Publish/Subscribe (*pub/sub*) se tornou um popular paradigma de comunicação nos últimos anos. Ele provém uma forma de interação fracamente acoplada entre várias fontes de envio e de consumo de dados **[8].** Em termos gerais, o modelo pub/sub possibilita a disseminação de informação em sistemas distribuídos a partir de várias fontes (*publishers*) para um grupo de usuários interessados (*subscribers*) **[9]**.

Sistemas baseados no padrão pub/sub são normalmente classificados de acordo com suas restrições de distribuição de dados. Os dois modelos mais comuns são: baseado em tópicos e baseado em conteúdo **[9]**.

Em um sistema baseado em tópicos as mensagens são enviadas pelos *publishers* para canais de eventos abstratos chamados de tópicos. Usuários interessados em receber tais mensagens enviam requisições especificando seus tópicos de interesse. O sistema, então, se certifica de distribuir cada nova mensagem publicada em um tópico para todos os usuários que se registraram para aquele tópico **[10]**.

Já num sistema baseado em conteúdo, o grupo de usuários interessados em uma mensagem é definido em tempo de execução. Cada mensagem contém um cabeçalho que contém informações que ajudam a identificar os usuários que devem receber aquela mensagem **[9].**

O padrão *pub/sub* será utilizado na solução através da ferramenta Apache Kafka, que é detalhada mais a frente. Essa ferramenta será responsável por armazenar e facilitar todo o fluxo de dados da aplicação.

3.2 Processamento de fluxos de informação em tempo real

Ferramentas que possibilitam o processamento de fluxos de informação em tempo real tem a capacidade de processar de forma atemporal uma grande quantidade de dados que são enviados à ferramenta continuamente **[4]**. Os conceitos Tempo Real e Atemporal são cruciais para entender a necessidade dessa nova classe de sistemas **[4]**.

Sistemas de processamento de dados em tempo real precisam seguir oito regras que definem esse grupo de aplicações **[11]**. São elas:

1. Manter os dados em movimento;
2. Possuir um mecanismo de consultas SQL;
3. Saber lidar com imperfeições no fluxo;
4. Gerar resultados previsíveis;
5. Integrar o fluxo com dados armazenados;
6. Garantir segurança e disponibilidade dos dados;
7. Particionar e escalar aplicações de forma automática;
8. Processar e responder instantaneamente.

Em resumo, essas aplicações precisam estar disponíveis para receber os dados, processá-los e devolver uma resposta sem uma grande latência nesse processamento, como pode ser visto na Figura 2. Além disso, também é necessário que esse sistema garanta disponibilidade, segurança e seja fácil de escalar **[11]**.

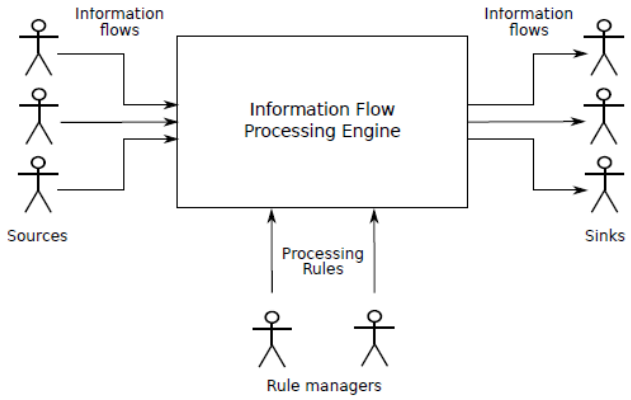

Figura 2: Visão em alto nível de um sistema de processamento de dados em tempo real.
Fonte: Cugola **[4]**.

DOI: 1025286/repa.v5i1.1178

3.3 Processamento de Eventos Complexos

Sistemas de processamento de eventos complexos oferecem um grupo de técnicas que facilitam a captura de eventos críticos observados em um fluxo de informação num dado período de tempo e que requer ação imediata **[12]**.

Essas aplicações normalmente processam eventos primários recebidos de fontes de dados diferentes e distribuídas, e - com base nesses eventos - conseguem derivar novos eventos que representam situações de interesse pré-definidas, como pode ser visto na Figura 3. Para isso, CEP normalmente utiliza técnicas de filtragem, correlação e operadores baseados em padrões **[13].**

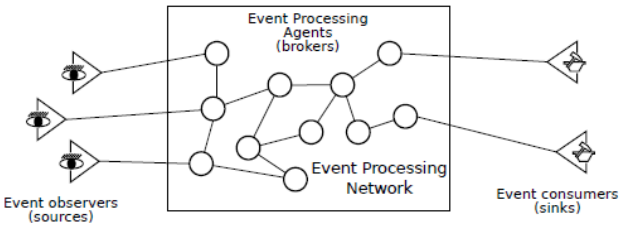

Figura 3: Visão em alto nível da arquitetura de uma aplicação CEP.
Fonte: Cugola (2012).

Sistemas CEP podem ser classificados com base no engenho CEP (centralizado, hierárquico, acíclico ou *peer-to-peer*) utilizado, o esquema de envio de mensagens e o modo como o processamento dos eventos é distribuído entre as instâncias do engenho **[13].**

A solução aqui apresentada faz uso do KSQL, que é detalhado mais a frente, para o processamento dos dados e aplicação de uma regra CEP e com isso conseguir derivar informações que vão acrescentar aos objetivos do trabalho.

3.4 Apache Kafka

O Apache Kafka, ferramenta utilizada nesse trabalho, é um sistema de mensagens distribuído *Open Source*. Ele provém uma solução para pub/sub em tempo real baseada em tópicos **[14]**. O Kafka, é uma das plataformas de mensagens mais utilizadas atualmente. Alguns exemplos de empresas que usam a mesma são: LinkedIn, Twitter e Foursquare **[14].**

3.5 Kafka Streams e KSQL

Kafka Streams é uma biblioteca cliente do Kafka que possibilita a construção de aplicações de processamento de informações em tempo real a partir da análise dos dados enviados para o cluster do Kafka **[15].** Como dito anteriormente essa biblioteca facilita a escalabilidade, tem tolerância a falhas e está totalmente integrada com o esquema de segurança do Kafka **[15]**.

KSQL é um engenho de consultas SQL *Open Source* para o Kafka. Essa solução foi construída em cima do Kafka Streams e fornece uma interface simples para fazer processamento de fluxo de dados a partir do Kafka. O engenho inclui operações como filtragem, agregação, transformações e junções, o que facilita do uso da ferramenta para análises de CEP **[16].**

4 TRABALHOS RELACIONADOS

Existem outros trabalhos que sustentam a metodologia escolhida e aplicada nesse artigo. Como exemplo temos a publicação de Cugola **[4]** que faz uma análise completa das tecnologias de processamento de fluxo de dados e de processamento de eventos complexos.

Outro exemplo é o trabalho de Terroso-Saenz [13] que, assim que este artigo, faz uso de técnicas de CEP para obter novas informações sobre um problema específico, que no caso daquele é a detecção de comportamentos anormais em ambientes marinhos.

Apesar de se basear nas mesmas técnicas de processamento de dados em tempo real dos trabalhos de Cugola **[4]** e Terroso-Saenz **[13]**, este trabalho se diferencia daqueles na escolha das ferramentas e na aplicação dessas com o objetivo de resolver um problema que impacta diretamente no dia a dia da população. E com isso provar que é possível aplicar técnicas já conhecidas na academia para solucionar problemas reais.

5 SOLUÇÃO

A solução proposta para resolver o problema citado na seção anterior faz uso das tecnologias e ferramentas de processamento de dados em tempo real citados no referencial teórico, seção 2.

Essa solução conta com duas fontes de dados, que serão detalhadas na seção 4.1. Um cluster Kafka, que possibilita o armazenamento e o processamento dos dados recebidos em tempo real, e uma aplicação KSQL, que faz a análise das informações recebidas e gera alertas caso as condições pré-definidas sejam cumpridas.

Para o contexto desse trabalho, é importante ressaltar que a solução foca na cidade do Recife e que apesar da prova de conceito ter sido desenvolvida numa única máquina em rede local, a arquitetura do Kafka facilita a escalabilidade da solução, possibilitando que a aplicação seja executada de forma distribuída com um esforço pequeno.

5.1 Arquitetura

Como falado anteriormente, esse trabalho faz uso de uma aplicação KSQL para analisar os eventos recebidos e gerar alertas a partir de consultas pré-definidas. Um resumo da arquitetura pode ser visto na Figura 5.

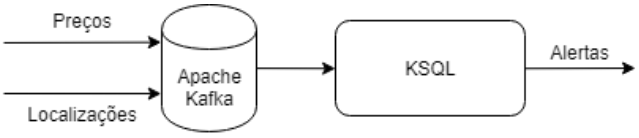

Figura 5: Visão superficial da arquitetura da solução proposta nesse trabalho.

Como existem dois fluxos de dados, detalhados nas seções 4.2.1 e 4.2.2, que são usados como entrada, foi necessário gerar dois Streams (estrutura de dados do KSQL que cria uma visão com todos os dados recebidos em um tópico do Kafka), detalhados nas seções 4.3.1 e 4.3.2, um para cada entrada.

Além disso, também foi preciso criar um novo Stream, detalhado na seção 4.3.3, como resultado da consulta pré-definida. Esse último irá receber os alertas identificados pela aplicação.

5.2 As Bases de Dados

Como dito anteriormente, esse trabalho utilizou duas fontes de dados em CSV para a solução do problema citado na seção 3. A primeira delas é uma base com preços dos combustíveis nos postos da cidade do Recife e a segunda conta com localizações em latitude e longitude de usuários na mesma cidade. As duas serão tratadas em detalhes nas seções 4.2.1 e 4.2.2 respectivamente.

5.2.1 Base de Atualização dos Preços

A base de atualização de preços utilizada nesse trabalho é uma base derivada a partir do resumo semanal de preços por município disponibilizado pela Agência Nacional de Petróleo (ANP) [17].

Os dados contidos nessa base seguem o formato que pode ser visto na Tabela 1. Os dados foram gerados de forma aleatória e para isso foi necessário um processamento que segue os seguintes passos:

1. Criação de CSV contendo todos os postos da cidade do Recife a partir da base da ANP;
2. Inclusão das coordenadas geográficas obtidas a partir da API do Google Maps no CSV para cada posto;
3. Seleção de um posto aleatório a partir da lista de postos da cidade;
4. Geração de um preço aleatório entre o maior e o menor preço encontrado na base da ANP;
5. Junção dos dados gerados para posterior processamento e envio para o Kafka.

Tabela 1: Formato dos dados de atualização de preços.

Campo	Descrição
Id	Identificador único do posto de combustível
Razão Social	Razão Social do posto de combustível
Endereço	Endereço do posto de combustível
Bairro	Bairro onde o posto está localizado
Bandeira	A marca do posto de combustível
Latitude	A latitude da localização do posto
Longitude	A longitude da localização do posto
Tipo	O tipo de combustível utilizado
Preço	O novo valor a ser atualizado
Recordtime	O horário de registro do dado

Ao executar o gerador de atualização de preços, os dados serão gerados initerruptamente a cada 60 segundos e serão enviados para o cluster Kafka para posterior processamento. A Figura 6 mostra um exemplo do dado gerado e enviado para o cluster.


```
{
  "id":1,
  "razao social":"Afogados Comercio de Combustivel Ltda - Epp",
  "endereco":"Rua Cosme Viana, 721 Loja 1",
  "bairro":"Afogados",
  "bandeira":"SETTA DISTRIBUIDORA",
  "latitude":"-8.0710581",
  "longitude":"-34.9096468",
  "tipo":"gasolina",
  "preco":4.19,
  "recordtime":1539079200000
}
```


Figura 6: Exemplo de atualização de preço enviada ao Kafka.

DOI: 1025286/repa.v5i1.1178

5.2.2 Base de Localizações

A base de localizações utilizada nesse trabalho é uma base criada totalmente em tempo real a partir das coordenadas geográficas da cidade do Recife obtidas através de ferramenta Google Maps. Seguem as coordenadas base utilizadas:

- Latitude mínima: -8.008605
- Latitude máxima: -8.154150
- Longitude mínima: -34.865000
- Longitude máxima: -34.968500

Os dados contidos nessa base seguem o formato que pode ser visto na Tabela 2. Para a geração dos dados foi necessário um processamento que segue os seguintes passos:

1. Foi gerada uma base de usuários identificados de 1 a 100;
2. A cada rodada, um desses usuários é escolhido aleatoriamente e uma localização é gerada a partir das coordenadas base;
3. Junção dos dados gerados e envio para o cluster Kafka.

Tabela 2: Formato dos dados de localizações.

Campo	Descrição
userid	Identificador único do usuário
Latitude	A latitude da localização do usuário
Longitude	A longitude da localização do usuário
Recordtime	O horário de registro do dado

Ao executar o gerador de localizações, os dados são gerados initerruptamente a cada 1 segundo e enviados para o cluster Kafka para posterior processamento. A Figura 7 mostra um exemplo do dado gerado e enviado para o cluster.


```
{
  "userid":"10",
  "lat":8.070081,
  "long":34.909722,
  "recordtime":1539079200000
}
```


Figura 7: Exemplo de localização enviada ao Kafka.

É importante ressaltar que o gerador de localizações pode ser executado independentemente do restante da solução.

5.3 Os Streams

Streams são estruturas de dados do KSQL que facilitam o processamento em cima de dados armazenados nos tópicos do Kafka. Nos Streams os dados são disponibilizados de forma contínua assim que eles chegam no cluster do Kafka.

Para a solução proposta nesse trabalho foi necessária a criação de 3 Streams. O primeiro para representar os dados de preços, o segundo para os dados de localização e o terceiro para os alertas gerados. Esses Streams são detalhados nas seções 4.3.1, 4.3.2 e 4.3.3 respectivamente.

5.3.1 Stream de Preços

O *Stream* de preços é uma estrutura de dados do KSQL que foi criada para representar os dados recebidos no tópico gas_*prices* do Kafka. Esses dados são os mesmos que foram criados pelo gerador de atualização de preços mencionado na seção 4.2.1.

Para gerar esse *Stream*, como pode ser visto na Figura 8, foram utilizados apenas os campos que foram usados para o processamento dos dados. São eles: *stationid*, *lat*, *long*, *price* e *record_time*. Além do campo *joinner*, que foi adicionado para facilitar o *join* entre os *streams*.

```
CREATE STREAM gas_prices ( \
  stationid VARCHAR,
  lat DOUBLE,
  long DOUBLE,
  price DOUBLE,
  recordtime BIGINT,
  joinner INT
) \
WITH (KAFKA_TOPIC='gas_prices', VALUE_FORMAT='JSON');
```

Figura 8: Script SQL para a criação do *stream* de preços.

5.3.2 Stream de Localização

O *Stream* de localização é uma estrutura de dados do KSQL que foi criada para representar os dados recebidos no tópico *locations* do Kafka. Esses dados são os mesmos que foram criados pelo gerador de localizações mencionado na seção 4.2.2.

Para gerar esse *stream*, como pode ser visto na Figura 9, foram usados apenas os campos que foram usados para o processamento dos dados. São eles: *userid*, *lat*, *long* e *record_time*. Além do campo *joinner*, que foi adicionado para facilitar o *join* entre os *streams*.

```
CREATE STREAM locations ( \
  userid VARCHAR,
  lat DOUBLE,
  long DOUBLE,
  recordtime BIGINT,
  joinner INT
) \
WITH (KAFKA_TOPIC='locations', VALUE_FORMAT='JSON');
```

Figura 9: Script SQL para a criação do stream de localizações.

5.3.3 Processamento dos Dados

O processamento dos dados se dá através da análise de dois eventos primários para a produção de um evento complexo. São eles:

- **Evento 1**: Alteração de preço no posto X;
- **Evento 2**: Usuário Y está na localização Z;
- **Evento Complexo**: Caso o usuário Y esteja a até 500 metros de distância do posto X e a última alteração desse posto tenha sido há menos de uma hora, gera-se um alerta.

Para que seja possível gerar o alerta é preciso que haja um processamento e um *stream* para receber os alertas. O processamento se dá da seguinte forma:

1. Os dados de alteração de preços e localização são recebidos no cluster Kafka;
2. A aplicação KSQL compara todo novo dado recebido com todos os dados que já haviam sido recebidos no outro tópico na última hora;
3. Para todo caso que houver alteração de preço na última hora e haja um usuário há até 500 metros de distância, envia-se um alerta para o *stream* de alertas.

O necessário para fazer o processamento acima pode ser visto na Figura 10.

```
CREATE STREAM alerts AS \
SELECT \
  L.userid, L.lat, L.long, L.recordtime,
  P.stationid, P.price, P.lat, P.long, P.recordtime \
FROM locations L
INNER JOIN gas_prices P WITHIN 1 HOURS ON L.joinner = P.joinner
WHERE GEO_DISTANCE(P.lat, P.long, L.lat, L.long, 'KM') < 0.5 \
AND L.recordtime - P.recordtime <= 3600000;
```

Figura 10: Processamento e criação do Stream de alertas.

6 RESULTADOS

A partir da solução desenvolvida foi possível gerar alertas para solucionar o problema citado na seção 3 adequadamente. Foram feitos cerca de 15 testes e a aplicação desempenhou bem, com respostas recebidas por volta de dois segundos após o recebimento dos dados. Esses testes foram feitos em uma máquina com processador i7 com quatro núcleos e 16gb de memória RAM. Além disso, o tempo de processamento não foi alterado mesmo com o aumento no volume de dados enviados para 100 envios por segundo.

Um exemplo de resultado obtido pela aplicação pode ser visto a seguir. A Figura 11 mostra um exemplo de dado recebido no *Stream gas_prices.* A Figura 12 mostra os dados recebidos no *Stream locations.* Finalmente, a Figura 13 mostra os resultados no Stream *alerts*.


```
[ ⊟
  { ⊟
    "stationid":"1",
    "lat":8.071058,
    "long":34.909646,
    "price":4.50,
    "recordtime":1539079200000,
    "joinner":1
  }
]
```


Figura 11: Exemplo de dado recebido no *Stream gas_prices.*

Como pode ser visto na Figura 13, o resultado obtido é o que se espera. Nos dados de entrada da Figura 11 e Figura 12, pode-se ver que que a alteração de preços do posto com *stationid* 1 e as localizações dos usuários com *userid* 10 e 11 foram recebidas na mesma hora. Entretanto, apenas o usuário com *userid* 10 estava a menos de 500 metros ao posto naquele momento.

DOI: 1025286/repa.v5i1.1178

```
[ ⊟
  { ⊟
    "userid":"10",
    "lat":8.070081,
    "long":34.909722,
    "recordtime":1539079200000,
    "joinner":1
  },
  { ⊟
    "userid":"11",
    "lat":8.065163,
    "long":34.909089,
    "recordtime":1539079200000,
    "joinner":1
  },
  { ⊟
    "userid":"12",
    "lat":8.070081,
    "long":34.909722,
    "recordtime":1539086400000,
    "joinner":1
  }
]
```


Figura 12: Exemplo de dado recebido no Stream *locations*.


```
[ ⊟
  { ⊟
    "P.stationid":"1",
    "P.price":4.50,
    "P.lat":8.071058,
    "P.long":34.909646,
    "P.recordtime":1539079200000,
    "L.userid":"10",
    "L.lat":8.070081,
    "L.long":34.909722,
    "L.recordtime":1539079200000
  }
]
```


Figura 13: Alerta gerado no Stream *alerts*.

Já o usuário com *userid* 12 enviou uma localização próxima ao posto em questão, mas essa localização foi registrada duas após a alteração de preço enviada pelo posto. Esse usuário não foi notificado pois essa diferença de tempo quebra a regra pré-definida de uma hora de diferença entre os dados.

Além de ter tido sucesso quanto aos resultados dos alertas, essa solução valida o uso das ferramentas Apache Kafka, Kafka Streams e KSQL para se trabalhar com problemas de processamento de eventos complexos em tempo real.

Todas as ferramentas são simples de usar e tem um excelente desempenho mesmo quando toda a arquitetura está rodando em uma única máquina. Outra vantagem dessa arquitetura e da utilização dessas ferramentas é a possibilidade de escalar para uma solução distribuída sem maiores problemas.

7 CONCLUSÕES E TRABALHOS FUTUROS

A solução desenvolvida nesse trabalho inclui funcionalidades importantes e que facilitam o acompanhamento da população dos preços nos postos de combustível.

Entretanto, para que a população consiga usar esta solução e tirar total proveito de seus benefícios, outros trabalhos complementares podem ser desenvolvidos. Segue sugestões de trabalhos futuros:

- Ampliação da solução para funcionar em todo o território nacional;
- Desenvolvimento de um *endpoint* para que seja possível requisitar os preços atuais em determinado posto;
- Desenvolvimento de um aplicativo mobile para que a população tenha acesso a solução.

Devido ao uso de ferramentas de processamento de dados em tempo real, este trabalho mostrou que é possível criar uma solução para deixar a população melhor informada sobre as alterações de preços nos postos de combustível.

O objetivo foi desenvolver uma solução inicial para que, no futuro, outros trabalhos venham a contribuir para a solução final do problema.

Também, como dito anteriormente, foi possível validar o uso das ferramentas utilizadas na solução em um problema real. Dessa forma, foi possível validar o uso dessas para que outros trabalhos possam utilizá-las sabendo dos seus potenciais.

REFERÊNCIAS

[1] ROSER, M.; RITCHIE, H.; ORTIZ-OSPINA, E. **Internet**. 2016. Disponível em: https://ourworldindata.org/internet. Último acesso: 18/11/18.

[2] COLUMBUS, L. **2017 Roundup of Internet of Things Forecasts**. 2017. Disponível em: https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#6cc12eee1480. Último acesso 18/11/18.

[3] MEEKER, M. **Internet Trends 2017. Code Conference**. 2017. Disponível em: https://www.kleinerperkins.com/perspectives/internet-trends-report-2017. Último acesso em: 18/11/18.

[4] CUGOLA, G.; MARGARA, A. **Processing flows of information: From data stream to complex event processing**. ACM Computing Surveys, vol. 44(3), 2012.

[5] TAKAR, T. **Gasolina do Brasil é uma das mais caras? Há 76 países com valor maior**. 2018. Disponível em: https://economia.uol.com.br/noticias/redacao/2018/05/25/preco-gasolina-brasil-alto-mundo.htm. Último acesso em: 18/11/18.

[6] MOTA, C. V. **6 perguntas para entender a alta nos preços da gasolina e do diesel**. 2018. Disponível em: https://www.bbc.com/portuguese/brasil-44217446. Último acesso em: 18/11/18.

[7] ALMEIDA, R.; ZANLORENSSI, G. **A trajetória do preço do combustível no Brasil nos últimos 17 anos**. 2017. Disponível em: https://www.nexojornal.com.br/grafico/2017/10/16/A-trajet%C3%B3ria-do-pre%C3%A7o-do-combust%C3%ADvel-no-Brasil-nos-%C3%BAltimos-17-anos. Último acesso em: 18/11/18.

[8] VINAY, S.; *et al*. **Poldercast: Fast, robust, and scalable architecture for P2P topic-based pub/sub**. Proceedings of the 13th International Middleware Conference, 2012.

[9] ONICA, E.; *et al*. **Confidentiality-preserving publish/subscribe: a survey**. ACM Computing Surveys, vol. 49(2), 2016.

[10] CHOCKLER, G., *et al*. **Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication**. Proceedings of the 2007 inaugural international conference on Distributed event-based systems, 2007.

[11] STONEBRAKER, M.; ÇETINTEMEL, U.; ZDONIK, S. **The 8 requirements of real-time stream processing**. ACM Sigmod Record, vol 34(4), pp.42-47, 2005.

[12] DÁVID, I.; RÁTH, I.; VARRÓ, D. **Foundations for streaming model transformations by complex event processing**. Software & Systems Modeling. Vol. 17(1). pp. 135-162, 2018.

[13] **Terroso-Saenz**, F.; VALDES-VELA, M.; SKARMETA-GOMEZ, A. F. **A complex event processing approach to detect abnormal behaviours in the marine environment**. Information Systems Frontiers, vol 18 (4), pp. 765-780, 2016.

[14] GARG, N. **Apache Kafka**. Packt Publishing Ltd, 2013.

[15] SOFTWARE FOUNDATION, **Apache. Kafka Streams: Introduction**. 2018. Disponível em: https://kafka.apache.org/documentation/streams. Último acesso em: 18/11/18.

[16] INC, Confluent. **KSQL: Introduction**. 2018. Disponível em: https://docs.confluent.io/current/ksql/docs/index.html. Último acesso em: 18/11/18.

[17] AGÊNCIA NACIONAL DE PETRÓLEO. **Resumo por Município**. 2018. Disponível em: https://preco.anp.gov.br/include/Resumo_Por_Municipio_Index.asp. Último acesso em: 18/11/18.

DOI: 1025286/repa.v5i1.1178