UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CAROLINY REGINA VALENÇA LEANDRO

APRIMORAMENTO DE MECANISMOS DE TRATAMENTO DE LOGS NÃO ENVIADOS EM UMA PLATAFORMA DE COMÉRCIO ELETRÔNICO

CAMPINA GRANDE - PB

2023

CAROLINY REGINA VALENÇA LEANDRO

APRIMORAMENTO DE MECANISMOS DE TRATAMENTO DE LOGS NÃO ENVIADOS EM UMA PLATAFORMA DE COMÉRCIO ELETRÔNICO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Tiago Lima Massoni

CAMPINA GRANDE - PB

2023

CAROLINY REGINA VALENÇA LEANDRO

APRIMORAMENTO DE MECANISMOS DE TRATAMENTO DE LOGS NÃO ENVIADOS EM UMA PLATAFORMA DE COMÉRCIO ELETRÔNICO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Tiago Lima Massoni

Orientador – UASC/CEEI/UFCG

Franklin de Souza Ramalho

Examinador – UASC/CEEI/UFCG

Melina Mongiovi Cunha Lima Sabino

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 17 de NOVEMBRO de 2023.

CAMPINA GRANDE - PB

RESUMO

A observabilidade desempenha um papel importante no desenvolvimento e na manutenção de software. Podemos dizer que um sistema é observável quando pode-se entender e explicar qualquer estado em que o mesmo possa entrar, podendo ele ser corriqueiro ou algo totalmente novo. Juntamente com métricas e traces, os logs representam um dos pilares da observabilidade, desempenhando um papel vital na depuração dos estados de um sistema. Isso ressalta sua importância como fonte de dados e a necessidade de seu tratamento e armazenamento. Nesse contexto, o OpenTelemetry emerge como um framework e conjunto de ferramentas que se propõe a facilitar a coleta e a gestão de dados de observabilidade em sistemas. Sendo independente de fornecedores e ferramentas, e adotando um modelo de código aberto, o OpenTelemetry se revela um software altamente versátil, adaptável às necessidades individuais de seus usuários, tornando-se uma escolha ideal na implementação de observabilidade em sistemas. O foco deste trabalho está no aprimoramento de um módulo utilizado em um coletor OpenTelemetry, cuja função principal é receber logs em uma plataforma de comércio eletrônico. Esse módulo compreende dois componentes: o WAL, responsável por detectar falhas no envio de logs ao OpenSearch e armazenar logs não enviados em um serviço de armazenamento de objetos; e um Replayer de logs, que tenta reenviar os logs armazenados posteriormente ao OpenSearch. Entretanto, o Replayer de logs enfrenta desafios relacionados a disponibilidade de recursos de hardware, instabilidade em ambientes variáveis e limitações na configuração, o que impacta negativamente em sua eficácia no envio de logs ao OpenSearch. Além disso, a ausência de dados sobre a saúde e o desempenho do WAL pode dificultar a manutenção e depuração deste componente, devido à falta de informações relevantes. Diante desse cenário, este trabalho tem como objetivo aprimorar o Replayer de logs, visando melhorar a disponibilidade e a utilização dos recursos de hardware, aumentar sua confiabilidade no envio de logs ao OpenSearch e torná-lo mais flexível em termos de configuração. Também, pretende-se adicionar capacidades de observabilidade ao mecanismo WAL com o objetivo de garantir maior visibilidade do funcionamento do mecanismo e facilitar a depuração do mesmo.

IMPROVEMENT OF MECHANISMS FOR TREATMENT OF UNSUBMITTED LOGS IN AN E-COMMERCE PLATFORM

ABSTRACT

Observability plays an important role in software development and maintenance. We can say that a system is observable when any state it can enter can be understood and explained, whether it is routine or something completely new. Along with metrics and traces, logs represent one of the pillars of observability, playing a vital role in debugging system states. This highlights its importance as a source of data and the need for its treatment and storage. In this context, OpenTelemetry emerges as a framework and set of tools that aims to facilitate the collection and management of observability data in systems. Being independent of vendors and tools, and adopting an open-source model, OpenTelemetry proves to be highly versatile software, adaptable to the individual needs of its users, making it an ideal choice in implementing observability in systems. The focus of this work is on improving a module used in an OpenTelemetry collector, whose main function is to receive logs on an e-commerce platform. This module comprises two components: the WAL, responsible for detecting failures in sending logs to OpenSearch and storing unsent logs in an object storage service; and a log Replayer, which attempts to resend the stored logs to OpenSearch later. However, the log Replayer faces challenges related to the availability of hardware resources, instability in variable environments, and limitations in configuration, which negatively impact its effectiveness in sending logs to OpenSearch. In addition, the absence of data on the health and performance of the WAL can make maintenance and debugging of this component difficult due to the lack of relevant information. Given this scenario, this work aims to improve the log Replayer, aiming to improve the availability and utilization of hardware resources, increase its reliability in sending logs to OpenSearch, and make it more flexible in terms of configuration. Additionally, it is intended to add observability capabilities to the WAL mechanism to ensure greater visibility of the mechanism’s operation and facilitate debugging of the same.

Aprimoramento de mecanismos de tratamento de logs não enviados em uma Plataforma de Comércio Eletrônico

Caroliny Regina Valença Leandro
Universidade Federal de Campina Grande
Campina Grande, Brasil
caroliny.leandro@ccc.ufcg.edu.br

RESUMO

A observabilidade desempenha um papel importante no desenvolvimento e na manutenção de software. Podemos dizer que um sistema é observável quando pode-se entender e explicar qualquer estado em que o mesmo possa entrar, podendo ele ser corriqueiro ou algo totalmente novo [7]. Juntamente com métricas e traces, os logs representam um dos pilares da observabilidade, desempenhando um papel vital na depuração dos estados de um sistema. Isso ressalta sua importância como fonte de dados e a necessidade de seu tratamento e armazenamento. Nesse contexto, o OpenTelemetry emerge como um framework e conjunto de ferramentas que se propõe a facilitar a coleta e a gestão de dados de observabilidade em sistemas [2]. Sendo independente de fornecedores e ferramentas, e adotando um modelo de código aberto, o OpenTelemetry se revela um software altamente versátil, adaptável às necessidades individuais de seus usuários, tornando-se uma escolha ideal na implementação de observabilidade em sistemas. O foco deste trabalho está no aprimoramento de um módulo utilizado em um coletor OpenTelemetry, cuja função principal é receber logs em uma plataforma de comércio eletrônico. Esse módulo compreende dois componentes: o WAL, responsável por detectar falhas no envio de logs ao OpenSearch e armazenar logs não enviados em um serviço de armazenamento de objetos; e um Replayer de logs, que tenta reenviar os logs armazenados posteriormente ao OpenSearch. Entretanto, o Replayer de logs enfrenta desafios relacionados a disponibilidade de recursos de hardware, instabilidade em ambientes variáveis e limitações na configuração, o que impacta negativamente em sua eficácia no envio de logs ao OpenSearch. Além disso, a ausência de dados sobre a saúde e o desempenho do WAL pode dificultar a manutenção e depuração deste componente, devido à falta de informações relevantes. Diante desse cenário, este trabalho tem como objetivo aprimorar o Replayer de logs, visando melhorar a disponibilidade e a utilização dos recursos de hardware, aumentar sua confiabilidade no envio de logs ao OpenSearch e torná-lo mais flexível em termos de configuração. Também, pretende-se adicionar capacidades de observabilidade ao mecanismo WAL com o objetivo de garantir maior visibilidade do funcionamento do mecanismo e facilitar a depuração do mesmo.

1 INTRODUÇÃO

O conceito de observabilidade, criado pelo engenheiro Rudolf E. Kálmán em 1960, descreve uma abordagem para medir o quão eficazmente os estados internos de um sistema podem ser inferidos a partir do conhecimento de suas saídas externas [7]. Inicialmente desenvolvida para atender às necessidades de sistemas físicos, essa definição foi posteriormente adaptada para sistemas de software, evoluindo para a concepção atual. Portanto, quando aplicado a sistemas de software, a observabilidade se torna uma medida que avalia a capacidade de entender e explicar qualquer estado em que o sistema possa se encontrar, independentemente de ser um estado comum ou completamente inédito. Se é possível depurar diversos estados com diferentes níveis de complexidade e combinações de dados, considera-se que o sistema tem uma boa observabilidade [7].

Um Log é um registro textual de um evento ocorrido dentro do sistema, uma das classes primárias de dados coletadas por um sistema observável e possui uma enorme importância na depuração de erros e entendimento de estados de um sistema [10]. Para que os logs sejam utilizados ativamente, é necessário que sejam exportados para uma ferramenta que armazene e trate esses dados.

No contexto deste trabalho, a responsabilidade de coletar, processar e exportar logs recai sobre um coletor OpenTelemetry customizado pela equipe de profissionais de uma plataforma de comércio eletrônico (neste caso, a VTEX[1]). O OpenTelemetry, um framework de observabilidade e conjunto de ferramentas desenvolvidos para auxiliar na criação e gerenciamento de dados de telemetria [2], é a base desse processo, enquanto o OpenSearch, que é um mecanismo distribuído de pesquisa e análise fundamentado no Apache Lucene [8], se configura como o receptor dos logs coletados neste trabalho. Contudo, o processo de envio de logs do coletor para o OpenSearch ocasionalmente apresenta falhas, resultando em perdas de dados. Dependendo do nível da falha ou da importância dos dados não enviados, isso pode prejudicar a análise de problemas que possam surgir nos sistemas alvo da coleta de dados de telemetria.

Diante desse cenário, a equipe de profissionais da plataforma de comércio eletrônico desenvolveu um mecanismo denominado WAL (nome derivado do termo Write-Ahead Logging), no coletor utilizado, para capturar logs dos sistemas e encaminhá-los para o OpenSearch. Esse mecanismo entra em ação quando ocorre uma falha no processo de envio de logs para o OpenSearch, e sua função é captar os logs que não foram enviados e mandá-los para um sistema de gerenciamento de arquivos, o Amazon S3[1]. Posteriormente, um outro componente, que opera de forma independente do coletor, se encarregará de tentar reenviar esses logs para o destino final. Contudo, *o WAL carece de observabilidade*. Isso resulta no surgimento de um ponto cego para os profissionais encarregados de lidar com esse software, pois a visibilidade adequada do funcionamento do mecanismo está ausente, tornando a depuração do sistema mais complexa em casos de falhas.

Para completar o ciclo de reenvio de logs para o OpenSearch, dispomos do Replayer, um outro mecanismo desenvolvido pelos profissionais da plataforma de comércio eletrônico para encaminhar os logs armazenados no Amazon S3 para o OpenSearch. Nesse contexto, o Replayer atua como uma função anônima ativada por meio

[1] site da VTEX

de um gatilho externo. Quando acionada, essa função acessa a instância do S3 que armazena os logs não enviados e tenta reencaminhá-los para o OpenSearch. Na versão original do Replayer, o mecanismo segue um esquema no qual cada arquivo de log permanece na lista de reenvio por um período definido. No entanto, se o reenvio não ocorrer dentro desse intervalo de tempo, o arquivo é descartado. Essa abordagem evidencia que o Replayer não desempenha eficazmente sua função de evitar a perda de logs. Além disso, uma vez que o Replayer é uma função hospedada no serviço Lambda AWS, ele está sujeito a limitações de recursos, incluindo RAM, CPU e o timeout da função, todos definidos pela AWS. Esses recursos muitas vezes não são suficientes para garantir o desempenho adequado do Replayer, impactando negativamente seu funcionamento, especialmente quando há um grande volume de dados para processar ou quando os dados são muito extensos. Além disso, o mecanismo é carente de flexibilidade no que diz respeito às configurações e é totalmente dependente da AWS, havendo a possibilidade de comprometimento do seu funcionamento caso ocorra alguma indisponibilidade nos serviços da empresa. Na Figura 1 é possível visualizar o fluxo que os logs percorrem durante a sua coleta e exportação pro OpenSearch.

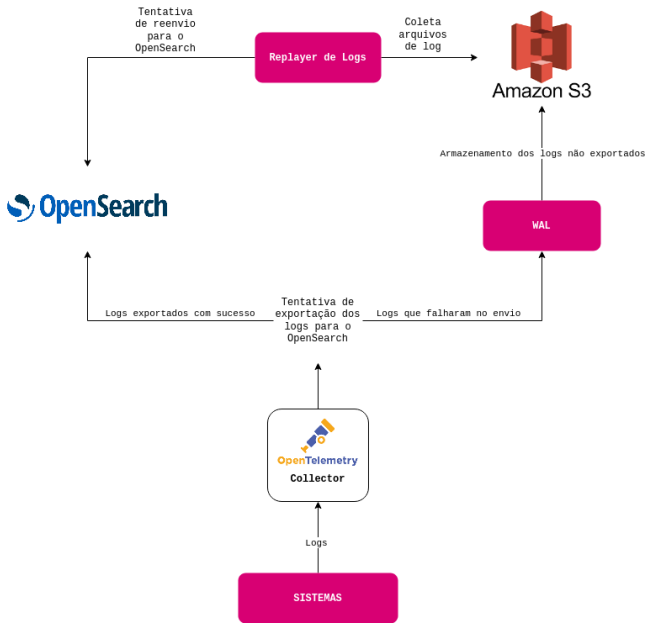

Figura 1: Diagramação do fluxo feito pelos logs recebidos e exportados pelo coletor OpenTelemetry utilizado no trabalho.

Considerando os fatores mencionados anteriormente, delineamos como objetivo central deste trabalho a redução da perda de logs por meio do aprimoramento dos mecanismos WAL e Replayer de Logs. No contexto do WAL, uma possível melhoria consiste na incorporação de métricas com o propósito de aprimorar a observabilidade desse sistema, contribuindo assim na simplificação da depuração do mecanismo. Quanto ao Replayer, a proposta envolve a reconstrução do sistema em Go, visando aprimorar o desempenho e manter a consistência com o padrão de linguagem adotado pelo coletor OpenTelemetry. Além disso, busca-se transformar o mecanismo em um sistema independente da AWS, capaz de operar sem esforço em um cluster Kubernetes[6], garantindo assim uma maior disponibilidade de recursos. O objetivo é também torná-lo altamente configurável, observável e, acima de tudo, mais confiável no que diz respeito ao envio de logs para o OpenSearch em comparação com a versão atualmente disponível. Portanto, essas são as contribuições deste trabalho:

- A adição de observabilidade no mecanismo WAL.
- A reescrita do Replayer de Logs de forma a sanar o problema de limitação de recursos e falta de flexibilidade relacionado a configuração do mecanismo.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Telemetria

Telemetria engloba os dados relacionados aos comportamentos de um sistema que são gerados por ele próprio. Esses dados podem ser classificados em logs, métricas ou traces [2]. É evidente que os logs, como previamente discutido, ocupam uma posição central neste trabalho. No entanto, as métricas também desempenham um papel significativo nas soluções, registrando eventos específicos dos softwares e seus valores.

2.1.1 Métricas. Uma métrica representa uma representação numérica de dados de um evento específico capturado em tempo de execução [10], sendo tipicamente empregada para registrar informações quantitativas, podendo ou não ter tags associadas. Existem quatro tipos distintos de métricas: *counter*, *gauge*, *histogram* e *summary* [3], porém, nas soluções em questão, apenas os três primeiros tipos são utilizados.

Um counter corresponde a uma métrica cumulativa, cujo valor só pode ser incrementado ou zerado em caso de reinicialização, utilizado para medir valores que apenas variam para cima, como o número de requisições recebidas por um sistema. Já o gauge representa uma métrica que reflete um único valor, sujeito a flutuações tanto para cima quanto para baixo, como temperaturas. O histogram, por sua vez, é uma métrica que coleta valores e os distribui em diversos intervalos criados a partir de uma faixa de valores, como um histogram que registra durações de requisições, além de fornecer uma soma dos valores coletados e a quantidade de eventos observados [3].

As métricas, particularmente as personalizadas, têm a capacidade de proporcionar *insights* relacionados à disponibilidade e desempenho de um sistema, ao mesmo tempo em que podem servir como alertas para possíveis falhas e gargalos que o sistema possa vir a enfrentar.

2.2 OpenTelemetry

O OpenTelemetry, um projeto do Cloud Native Computing Foundation (CNCF), é um framework e um conjunto de ferramentas desenvolvidos com o propósito de criar e gerenciar dados de telemetria, abrangendo traces, métricas e logs [2]. O OpenTelemetry se destaca por não estar vinculado a nenhuma empresa específica, o que o torna uma ferramenta altamente versátil e adaptável, permitindo a conexão com diversos backends de observabilidade. Além disso, oferece recursos para a instrumentação, possibilitando tornar

sistemas observáveis tanto de forma manual como automática, e é compatível com diversas linguagens de programação, bem como componentes capazes de lidar com os dados gerados por esses sistemas.

A maior contribuição do OpenTelemetry para a observabilidade, no entanto, é a criação e popularização de protocolos e convenções que padronizam a codificação, transporte, entrega e formato de exibição de dados de telemetria. Essa padronização no gerenciamento de dados de telemetria traz uma flexibilidade considerável para os sistemas que, de alguma forma, lidam com esses dados, pois permite a troca de informações com diversos outros sistemas sem complexidades excessivas. Devido ao seu código aberto, o OpenTelemetry também oferece a liberdade de adicionar novos recursos à ferramenta, como, por exemplo, suporte para instrumentação em outras linguagens além do escopo inicialmente estabelecido pelo OpenTelemetry.

2.3 Coletor OpenTelemetry

Um coletor OpenTelemetry é uma aplicação destinada a coletar, processar e exportar dados de telemetria [2]. Trata-se de um software independente e não vinculado a fornecedores específicos, o que elimina a necessidade de configurar, executar e manter diversas implementações de exportação de dados para backends que lidam com telemetria. Isso simplifica consideravelmente o processo de tornar sistemas observáveis. Essa simplificação é, em parte, viabilizada pelo Protocolo OpenTelemetry (OTLP), que estabelece diretrizes para a codificação, transporte e entrega de dados de telemetria entre as fontes desses dados e o destino final.

Dessa forma, o processo é padronizado, reduzindo a complexidade da integração entre as fontes, coletores e backends. Isso, por sua vez, torna o fluxo mais flexível e adaptável às diferentes necessidades. Vale ressaltar que um coletor pode ser personalizado de acordo com as exigências específicas do usuário.

2.4 OpenSearch

O OpenSearch é um motor distribuído de busca e análise. Uma vez que os dados são adicionados à ferramenta, é possível realizar buscas baseadas em texto de diversas formas. O OpenSearch é amplamente utilizado como backend para aplicações de busca, como a Wikipedia ou lojas de e-commerce, devido ao seu desempenho e escalabilidade[8].

Além disso, essa ferramenta é adequada para a análise de logs devido à sua capacidade avançada de busca e visualização de dados juntamente com a utilização do OpenSearch Dashboards, sendo essa ferramenta capaz de construir diversos tipos de visualizações de dados[8].

No contexto deste trabalho, o OpenSearch é escolhido como o destino final dos logs de sistemas recebidos pelo coletor devido à sua ampla gama de funcionalidades relacionadas à análise de dados. Dado o grande volume de dados recebidos diariamente por uma plataforma de comércio eletrônico, é de extrema importância o uso de ferramentas que facilitem a extração de padrões desses dados e ofereçam diversas maneiras de compreender as informações contidas nesses registros.

2.5 WAL (Write-Ahead Logging)

Write-Ahead Logging, mais reconhecido entre profissionais de bancos de dados, é o mecanismo que assegura a integridade de transações. Esse método envolve a prática de registrar todas as alterações em bases de dados em um sistema de armazenamento seguro antes de aplicá-las de forma definitiva. Isso permite que qualquer perda de dados ou inconsistência no banco possa ser rastreada e corrigida por meio da consulta a esses registros [9].

É possível estabelecer um paralelo entre esse conceito e a terminologia empregada no mecanismo que armazena os logs não enviados para o OpenSearch, uma vez que esse mecanismo opera como um sistema de armazenamento seguro. Seu propósito é armazenar dados de transações realizadas por outros sistemas que enfrentaram alguma falha no envio para o destino final.

2.6 Armazenamento de Objetos

O armazenamento de objetos é uma tecnologia que lida com dados em formato não estruturado, tal como fotos, vídeos, e-mails, arquivos de áudio e dados de sensores. Os sistemas de armazenamento de objetos em nuvem distribuem esses dados em vários dispositivos físicos e proporcionam aos usuários acesso remoto por meio de um repositório de armazenamento virtual. Essa tecnologia é particularmente vantajosa para o desenvolvimento de aplicações nativas de nuvem que demandam flexibilidade e escalabilidade, além de facilitar questões relacionadas a backup, importação de dados para análise e arquivamento [1].

O sistema de armazenamento de objetos combina partes de dados para criar um arquivo, adicionando a ele todos os metadados criados pelo usuário e associando um identificador personalizado. Isso resulta na formação de uma estrutura plana, isenta de hierarquias ou camadas, denominada *bucket*. Essa abordagem permite a recuperação e análise de qualquer objeto, independentemente do tipo de arquivo, com base apenas em sua função e características [1].

Neste trabalho, a tecnologia em questão é empregada para armazenar conjuntos de logs que enfrentaram falhas no processo de envio para o OpenSearch. Dessa maneira, o WAL recebe esses logs e encaminha-os para um serviço de armazenamento de objetos, que, no contexto, é o Amazon S3[1].

3 IMPLEMENTAÇÃO

Neste trabalho, temos dois componentes que passaram por aprimoramentos: o WAL e o Replayer. O WAL está integrado ao coletor OpenTelemetry customizado pela VTEX, e não foi necessário realizar modificações significativas, já que as melhorias foram incorporadas ao código existente. Por outro lado, o Replayer passou por mudanças substanciais, incluindo a transição da versão anterior escrita em Python para a nova versão em Go, o que exigiu a reconstrução do código. Portanto, a execução deste trabalho ocorreu em quatro etapas distintas: i) O estudo e a prática da linguagem Go, com foco na criação de um protótipo capaz de enviar logs armazenados em um bucket do Amazon S3 para uma instância de teste do OpenSearch, ii) a implementação de indicadores de saúde e desempenho para o WAL no coletor, iii) a reescrita do Replayer de Logs em Go, juntamente com a implementação das melhorias previamente mencionadas e iv) a realização de testes de carga e a

análise do comportamento do Replayer, bem como a coleta de dados relacionados às métricas implementadas no WAL e no Replayer.

3.1 Implementação dos indicadores de saúde e funcionamento do WAL

Conforme mencionado anteriormente, a função do WAL é armazenar os logs que não foram enviados ao OpenSearch e retê-los em um bucket S3 para posterior reenvio. Quando ocorre uma falha na requisição de envio de logs ao OpenSearch por parte do coletor, o WAL é acionado com objetivo de armazenar os itens não enviados para o OpenSearch em um buffer local. O WAL incorpora um método denominado flush, que envia os logs armazenados no buffer local para o Amazon S3. Esse método é acionado periodicamente após um intervalo de tempo fixo ou quando o buffer atinge uma capacidade específica. Dessa forma, o mecanismo envia regularmente arquivos de logs para o S3, minimizando a perda de logs. No entanto, esse funcionamento não possui uma visibilidade adequada, uma vez que não ocorre a coleta de dados de telemetria do mecanismo durante o processo.

Para contornar esse problema, foi necessária a implementação e coleta de métricas que visam mapear o comportamento do WAL. Durante a definição dos requisitos iniciais para este trabalho, ficou evidente que as informações mais relevantes a serem coletadas são:

- O estado de ativação do WAL.
- A quantidade de eventos de log recebidos e adicionados ao buffer.
- As respostas de erro fornecidas pelo OpenSearch em casos de falhas no envio de logs.
- O total de chamadas do método flush (com indicação se foi acionado periodicamente ou devido ao atingimento da capacidade máxima do buffer).
- O espaço do buffer utilizado no momento do flush.
- A quantidade de requisições de envio de logs para o S3 que falharam.

As métricas do WAL e seu processo de coleta foram implementadas de acordo com a documentação do OpenTelemetry[2]. Para esse fim, foi criado um arquivo independente que descreve todas as métricas a serem coletadas, seus nomes e descrições conforme o padrão indicado na documentação do Prometheus. Nesse mesmo arquivo, todas as métricas foram inicializadas. As métricas que indicam o estado de ativação do WAL e o percentual de espaço ocupado no buffer durante o flush são do tipo Gauge, enquanto as demais são do tipo Counter.

3.2 Reescrita do Replayer de Logs

A versão anterior do Replayer consistia em uma função anônima escrita na linguagem Python e hospedada no serviço AWS Lambda. Esse mecanismo recuperava os arquivos de logs armazenados no Amazon S3 e tentava reenviá-los para o OpenSearch por até 12 horas a partir do momento em que foram originalmente guardados no bucket de logs não enviados. Se um arquivo permanecesse no bucket por mais de 12 horas, era temporariamente descartado em um bucket DLQ (Dead Letter Queue), resultando em possível perda de logs.

Devido aos problemas previamente mencionados relacionados ao Replayer, chegou-se a conclusão de que a melhoria principal deveria envolver a reescrita desse código em uma linguagem de programação diferente para que o Replayer não fosse mais uma função Python anônima e possuísse um gerenciamento melhor de recursos por si próprio. Além disso, deveria ser desenvolvido de tal forma que fosse independente, agnóstico em relação a qualquer serviço e capaz de operar eficientemente em um ambiente de contêineres.

A linguagem escolhida para criar a nova versão foi o Go[4]. Isso se deve ao fato de que o Go já é utilizado no coletor OpenTelemetry, é de fácil configuração, possui baixo consumo de recursos e oferece alto desempenho. Aproveitando que o Go é uma linguagem que facilita a criação e o gerenciamento de threads, o mecanismo foi reescrito de modo a permitir o envio de logs para o OpenSearch em paralelo. Além disso, o comportamento de descartar logs não reenviados em até 12 horas para um bucket DLQ foi eliminado, visando melhorar a confiabilidade do sistema no que diz respeito ao reenvio. No mais, o funcionamento geral do sistema permaneceu inalterado.

Além dos pontos citados acima, os demais objetivos a serem alcançados compreendiam:

- Parametrizar o mecanismo de forma que fosse viável controlar mais precisamente o seu funcionamento.
- Instrumentar a solução para permitir a coleta de dados de telemetria.

A instrumentação foi realizada de acordo com as diretrizes presentes na documentação do OpenTelemetry para a linguagem Go. Para uma visão mais ampla da do fluxo de atividades da solução, é possível consultá-lo na Figura 2.

4 AVALIAÇÃO

Para avaliar a implementação do Replayer, foi utilizado o Grafana k6, uma ferramenta de teste de carga que simula usuários realizando requisições para um determinado servidor. Com essa ferramenta é possível construir diversos cenários onde usuários enviam ou recebem dados [5]. Neste contexto, foram construídos cenários em que usuários virtuais geram e enviam logs para o coletor responsável por direcionar esses logs para o OpenSearch, ocasionando todo o fluxo que envolve o WAL e Replayer de logs. O objetivo principal deste experimento é validar se o Replayer funciona da forma esperada independentemente da carga de logs. É esperado que o Replayer pegue os arquivos de logs armazenados no bucket, tente enviá-los para o OpenSearch e quando o envio for realizado, o arquivo deve ser excluído do bucket.

Neste experimento, o Replayer funciona numa instância do serviço de nuvem Amazon Elastic Compute Cloud, que possui 8 CPU virtuais e 64Gb de memória disponíveis. Além disso, o coletor foi configurado de forma que os logs recebido são enviados direto para o WAL, sem precisar necessariamente passar pela tentativa de envio para o OpenSearch no início e o Replayer está configurado para trabalhar com paralelismo, onde o número máximo de threads que podem trabalhar ao mesmo tempo é 20.

4.1 Avaliação do Replayer

4.1.1 Primeiro cenário de testes. Buscando maior proximidade com a realidade, o primeiro cenário de testes foi configurado com valores semelhantes aos do ambiente de produção da plataforma de e-commerce mencionada neste trabalho. Assim, a ferramenta k6 foi

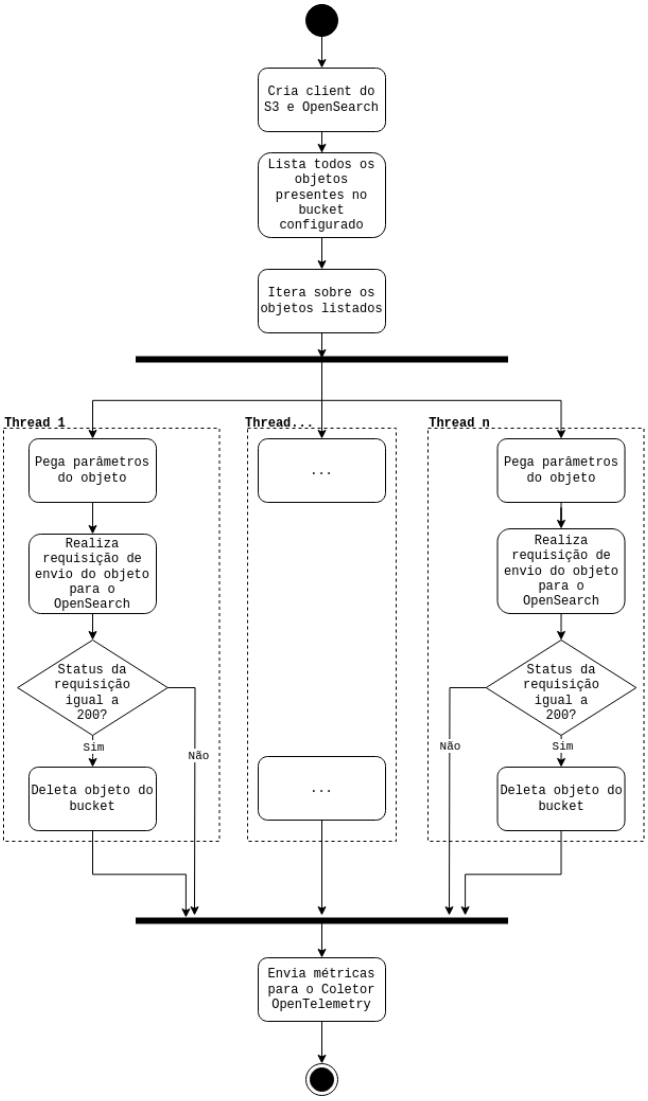

Figura 2: Diagramação do fluxo realizado pelo Replayer na nova versão.

configurada para enviar aproximadamente 100 mil logs por minuto em um período de 5 minutos para o coletor OpenTelemetry, embora esse valor possa variar para mais. O experimento teve início às 17:44:06.

Na Figura 3, podem ser observados gráficos que exibem a quantidade de requisições de envio de logs recebidas pelo coletor e posteriormente exportadas por ele. O pico de recebimento e exportação atingiu 100 mil requisições por segundo. É importante observar o horário em que as requisições de envio de logs começaram a chegar ao coletor, marcado às 17 horas e 45 minutos, momento em que a curva do gráfico passa a crescer de forma mais evidente.

Na Figura 4, apresentam-se os gráficos relacionados ao WAL. O gráfico intitulado *Total received events* indica a quantidade de eventos de logs enviados pelo coletor e recebidos pelo mecanismo. É perceptível que os horários mantêm uma certa constância, com o gráfico mostrando um aumento entre as 17 horas e 44 minutos e as 17 horas e 45 minutos. Com base nesses dados, podemos concluir que o coletor exporta os logs que recebe para o WAL de maneira ágil.

Na Figura 5, apresenta-se uma representação visual da quantidade de logs recebidos pelo OpenSearch após o trabalho do replayer. Mais uma vez, nota-se que o horário mantém-se constante. Nesse contexto, a ferramenta k6 gera logs em tempo de execução, e cada log possui um timestamp indicando o momento de sua geração, armazenado junto com outros atributos. Levando em consideração o horário de início e a duração do experimento, é possível observar a consistência dos logs que chegaram ao OpenSearch. A figura retrata o recebimento de logs gerados entre as 17 horas e 44 minutos até as 17 horas e 49 minutos, abrangendo um intervalo de 5 minutos, que corresponde ao tempo configurado no k6 para a geração e envio de logs para o coletor.

Assim, podemos concluir que o fluxo se mantém constante e que o Replayer opera de maneira adequada em um cenário semelhante ao de produção, uma vez que o OpenSearch demonstra um padrão de logs recebidos que se alinha com o cenário de teste do k6.

4.1.2 Segundo cenário de testes. Este cenário de teste foi construído com o intuito de estressar o fluxo de reenvio de logs com uma quantidade e logs muito grande para analisar se o comportamento, especialmente do Replayer, permanece coerente. A ferramenta k6 foi configurada para enviar aproximadamente 1 milhão de logs por minuto, durante um intervalo de 5 minutos, para o coletor OpenTelemetry, embora esse valor possa variar. O experimento teve início às 16:20:19.

Na Figura 6, é possível examinar os gráficos que ilustram a quantidade de requisições de envio de logs recebidas e exportadas pelo coletor, alcançando um pico de até 200 mil requisições por segundo. As requisições de envio de logs começaram a chegar ao coletor às 16 horas e 20 minutos.

Na Figura 7, no gráfico denominado *Total received events*, é possível observar que os logs começam a ser recebidos pelo WAL a partir das 16 horas e 20 minutos, e a entrada de logs cessa aproximadamente às 16 horas e 29 minutos, em consonância com os horários de recebimento de logs registrados pelo coletor.

Por fim, na Figura 8, é possível verificar que os timestamps dos logs recebidos estão registrados entre as 16 horas e 20 minutos e as 16 horas e 25 minutos, indicando que esses logs foram gerados durante o período de tempo correspondente ao cenário de teste do k6.

Portanto, temos evidências de que o fluxo funciona corretamente mesmo em situações de sobrecarga e que nenhum dos componentes gera gargalos para o sistema.

4.2 Avaliação das métricas do WAL

Em relação à validação das métricas, sua eficácia como uma melhoria do WAL pode ser comprovada ao observar os resultados dos testes analisados acima, já que desempenharam um papel importante como evidência das experimentações. Elas forneceram informações cruciais sobre o funcionamento do mecanismo e os dados recebidos por ele. Dessa forma, podemos afirmar que as métricas do WAL são importantes para a análise do mecanismo por parte

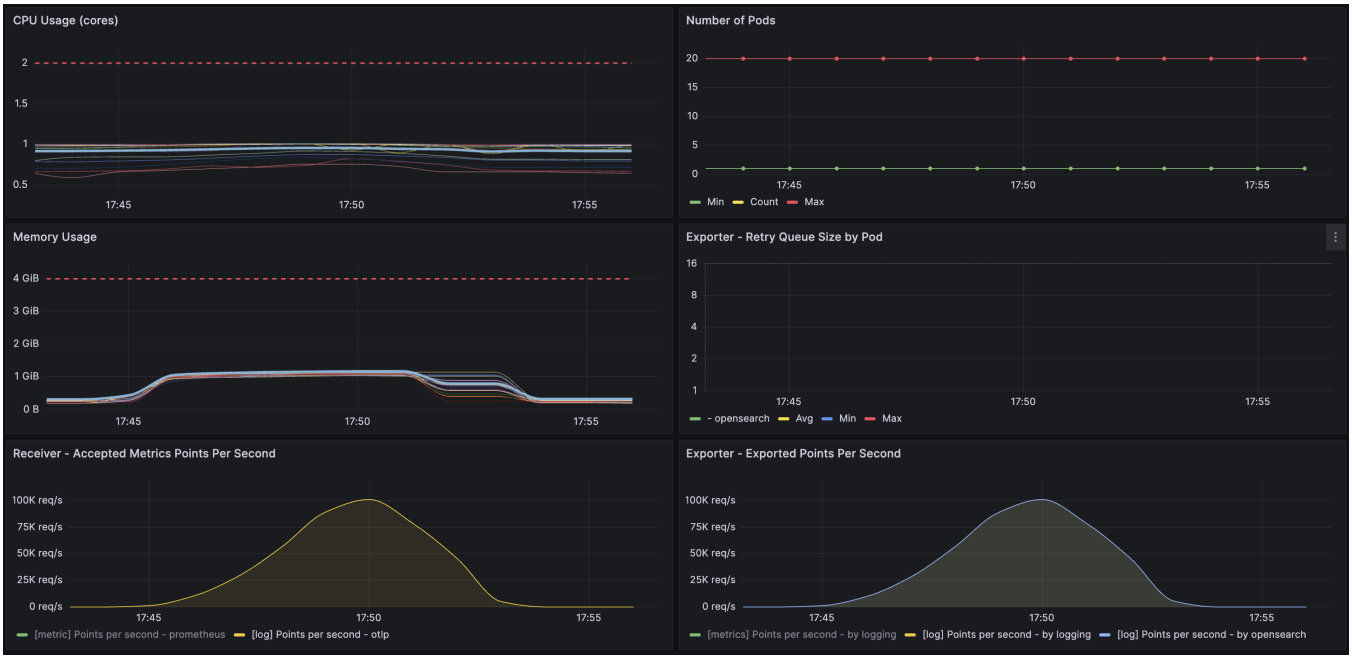

Figura 3: Gráficos de métricas retiradas do coletor OpenTelemetry utilizado para os testes durante o cenário 1.

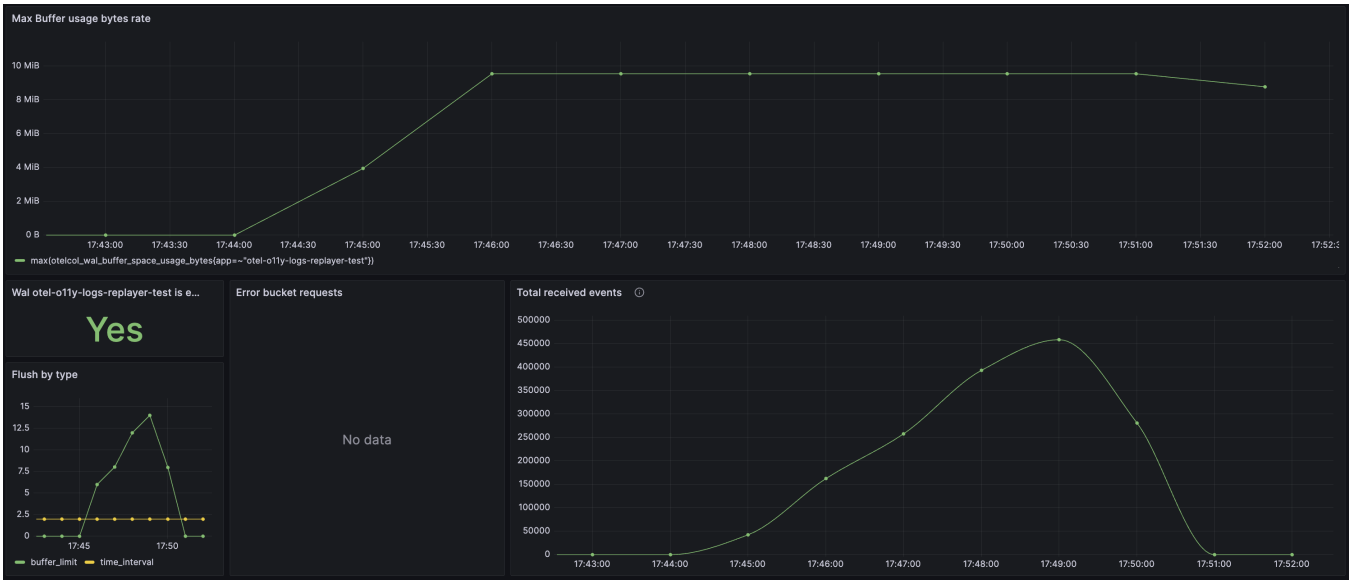

Figura 4: Gráficos de métricas retiradas do WAL durante o cenário de teste 1.

dos profissionais que precisam lidar com ele. Com esse tipo de informação, torna-se mais fácil compreender o que está acontecendo e abordar falhas de comportamento da melhor forma possível.

5 CONCLUSÃO

Considerando tudo o que foi desenvolvido, é possível concluir que houve contribuições significativas para os objetos de foco deste trabalho. Foram entregues o WAL com recursos de observabilidade e métricas que contribuem para uma melhor compreensão do funcionamento do mecanismo, bem como um novo Replayer de Logs mais flexível, resiliente e com maior disponibilidade de recursos. Como resultado, pode-se afirmar que a perda de logs durante o fluxo de coleta até a chegada ao OpenSearch será reduzida, e em caso de falhas, será possível realizar análises mais precisas sobre o que aconteceu, uma vez que agora existem dados disponíveis sobre o desempenho do WAL e do Replayer.

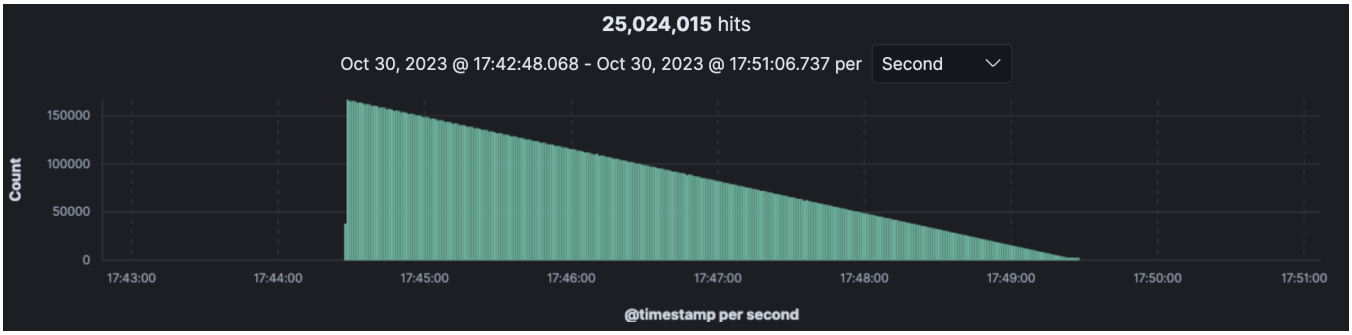

Figura 5: Gráfico de logs recebidos pelo OpenSearch durante o cenário de teste 1.

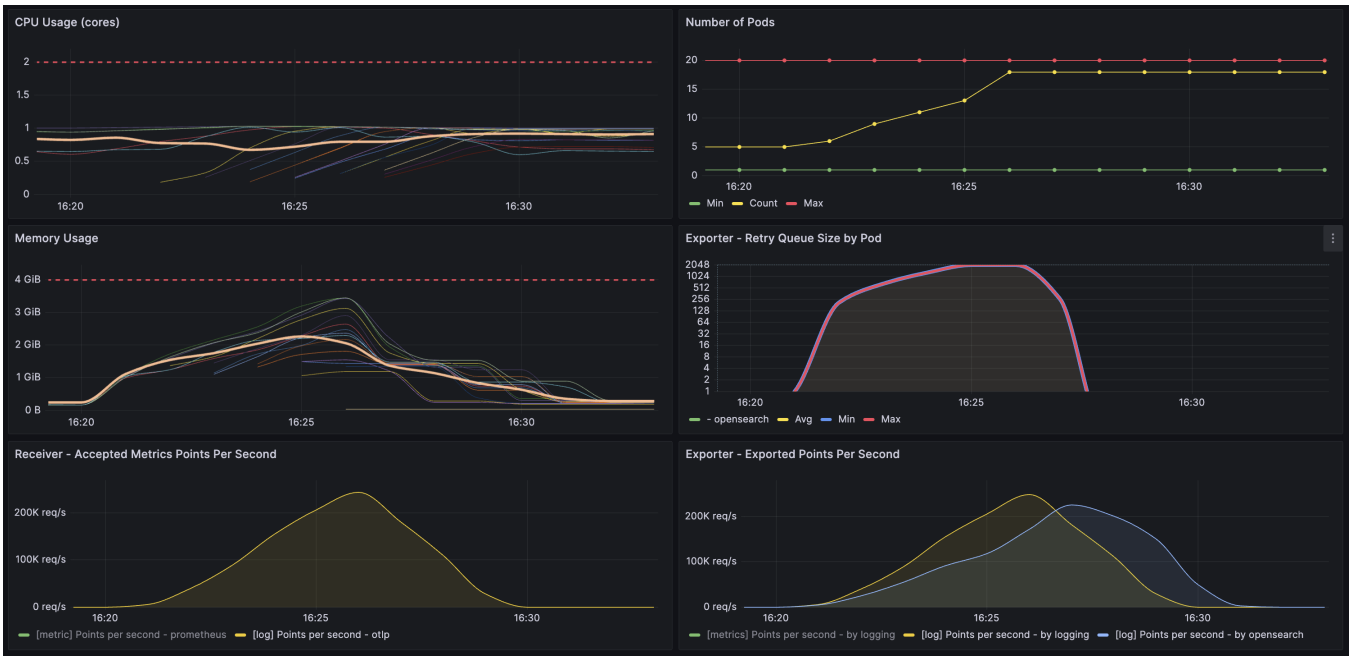

Figura 6: Gráficos de métricas retiradas do coletor OpenTelemetry utilizado para os testes durante o cenário 2.

5.1 Limitações e Trabalhos Futuros

5.1.1 Infraestrutura. Um dos principais desafios deste trabalho está relacionado à infraestrutura necessária para a validação dos sistemas em um ambiente local. Visto que o trabalho se concentra em dois mecanismos que, embora relativamente pequenos, dependem de outras ferramentas e serviços, além de fazerem parte de um fluxo de funcionamento mais complexo, a montagem de uma infraestrutura que reflita o ambiente de produção localmente foi inviável.

5.1.2 Adição de métricas mais descritivas. Uma evolução interessante deste trabalho incluiria a adição de novas métricas e aprimoramentos nas métricas existentes no WAL e no Replayer para proporcionar uma descrição mais abrangente do funcionamento de ambos. As métricas atualmente disponíveis atendem a várias necessidades, mas ainda há informações adicionais a serem coletadas e expostas, o que contribuiria significativamente para a aprimoração da observabilidade desses sistemas.

6 AGRADECIMENTOS

REFERÊNCIAS

[1] Amazon Web Services, Inc. 2023. *What is Object Storage?* Amazon Web Services, Inc. Disponível em https://aws.amazon.com/what-is/object-storage/.

[2] Cloud Native Computing Foundation (CNCF) 2023. *OpenTelemetry Documentation.* Cloud Native Computing Foundation (CNCF). Disponível em https://opentelemetry.io/docs/.

[3] Cloud Native Computing Foundation (CNCF) 2023. *Prometheus Documentation.* Cloud Native Computing Foundation (CNCF). Disponível em https://prometheus.io/docs/.

[4] Google 2023. *Go Documentation.* Google. Disponível em https://go.dev/doc/.

[5] Grafana Labs 2023. *Grafana k6 Documentation.* Grafana Labs. Disponível em https://k6.io/docs/.

[6] Kubernetes 2023. *Kubernetes Documentation.* Kubernetes. Disponível em https://kubernetes.io/docs/home/.

[7] Charity Majors, Liz Fong-Jones, and George Miranda. 2022. *Observability Engineering.* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA

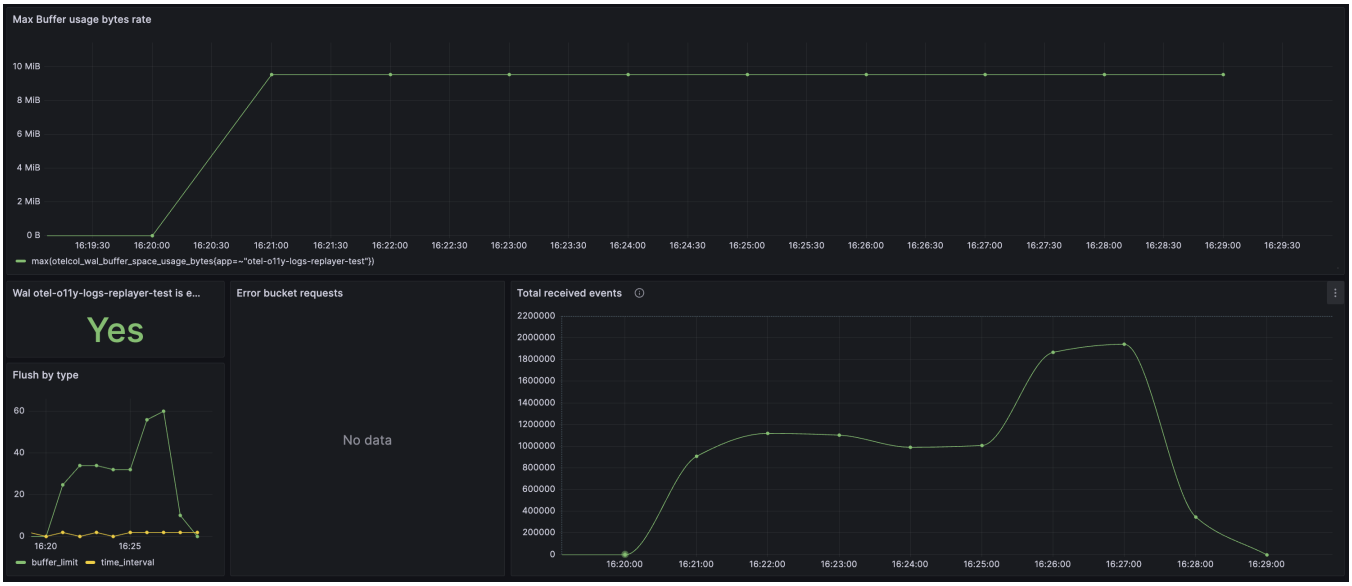

Figura 7: Gráficos de métricas retiradas do WAL durante o cenário de teste 2.

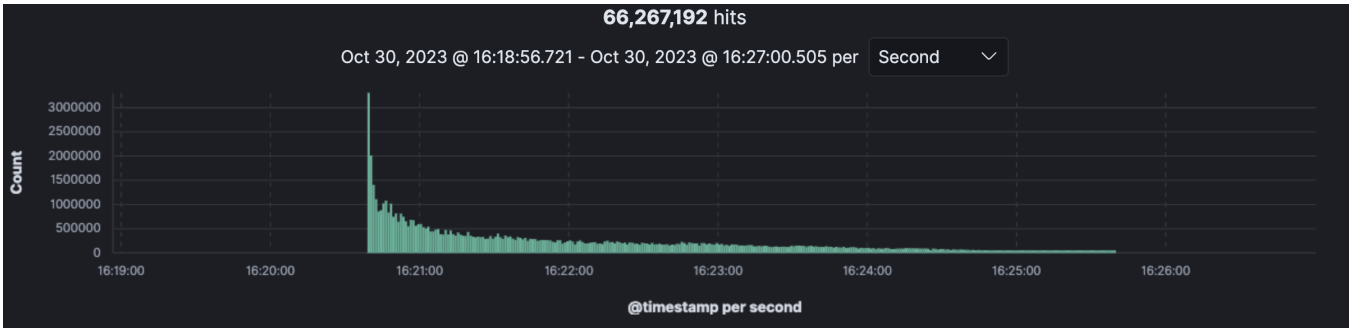

Figura 8: Gráfico de logs recebidos pelo OpenSearch durante o cenário de teste 2.

95472.
[8] OpenSearch 2023. *OpenSearch Documentation*. OpenSearch. Disponível em https://opensearch.org/docs/.
[9] PostgreSQL 2023. *PostgreSQL Documentation*. PostgreSQL. Disponível em https://www.postgresql.org/docs/.
[10] Cindy Sridharan. 2018. *Distributed Systems Observability*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.