CENTRO TECNOLÓGICO DA UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
PROGRAMA DE CIÊNCIAS DA COMPUTAÇÃO

Luana Dalmarco Fronza

Proposta de uma Arquitetura de Processamento de Dados Para Armazéns Inteligentes Utilizando Apache Kafka

Florianópolis
2022

Luana Dalmarco Fronza

Proposta de uma Arquitetura de Processamento de Dados Para Armazéns Inteligentes Utilizando Apache Kafka

Trabalho de Conclusão de Curso submetido ao Programa de Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de bacharel em Ciências da Computação.
Orientadora: Profa. Dra. Patricia Della Méa Plentz

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Fronza, Luana Dalmarco
Proposta de uma Arquitetura de Processamento de Dados Para Armazéns Inteligentes Utilizando Apache Kafka / Luana Dalmarco Fronza ; orientador, Patricia Della Méa Plentz, 2022.
65 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, , Graduação em , Florianópolis, 2022.

Inclui referências.

1. . 2. Arquitetura de Software. 3. Tempo Real . 4. Armazéns Inteligentes. I. Plentz, Patricia Della Méa . II. Universidade Federal de Santa Catarina. Graduação em . III. Título.

Luana Dalmarco Fronza

Proposta de uma Arquitetura de Processamento de Dados Para Armazéns Inteligentes Utilizando Apache Kafka

O presente trabalho em nível de bacharelado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Profa. Dra. Patricia Della Méa Plentz,
Universidade Federal de Santa Catarina

Prof. Dr. Jônata Tyska Carvalho
Universidade Federal de Santa Catarina

Prof. Dr. Odorico Machado Mendizabal
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de bacharel em Ciências da Computação.

_____

Coordenação do Programa de Graduação

_____

Profa. Dra. Patricia Della Méa Plentz
Orientadora

Florianópolis, 2022.

Este trabalho é dedicado à minha família e aos meus amigos, que me apoiaram desde o início.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Adriana e Jocélio, que me apoiaram desde o início da graduação e foram essenciais na minha caminhada até a conclusão tanto deste trabalho quanto do curso. Agradeço também ao João Janini e ao Lucas Barzan, por caminharem ao meu lado durante toda a graduação até a apresentação deste trabalho, por toda a ajuda e parceria ao longo destes anos. Por fim agradeço também à professora Patricia, pela orientação e auxílio durante o processo de desenvolvimento desse projeto.

RESUMO

O comércio digital se tornou um dos principais meios de compra para a população nos últimos anos e consequentemente, com o grande volume de informações sendo geradas diariamente por essas plataformas, se torna necessário a busca por estratégias que tornem o processamento desses dados mais eficiente. Os armazéns inteligentes equipados com robôs que lidam com a extração e separação de produtos, se tornaram uma solução para a otimização do fluxo de entrega de pedidos antes realizados somente por pessoas. As literaturas apontam o planejamento de caminhos e alocação de tarefas como os principais problemas relacionados a essa área, entretanto, é essencial considerar a maneira que os dados estão sendo processados e gerenciados no caminho entre a plataforma digital e os robôs do armazém, pois de acordo com outros trabalhos, o processamento de pedidos não é feito em tempo real. Ao mesmo tempo, existem sistemas de mensageria de *streaming* de dados que podem realizar esta tarefa e trazer mais eficiência para o processo de entrega de pedidos em armazéns inteligentes. O objetivo desse projeto é propor uma arquitetura de software capaz de lidar com o grande fluxo de pedidos sendo feitos em tempo real em plataformas de vendas, utilizando o Apache Kafka (KAFKA, 2021), uma plataforma unificada, de alta capacidade e baixa latência como principal ferramenta de comunicação. Foram criados cenários de testes com diferentes frequências de chegada de pedidos e de organizações dos componentes do Kafka, considerando a quantidade de mensagens que são perdidas nestes cenários, com o intuito de validar a proposta. Os resultados obtidos mostraram que a arquitetura desenvolvida suporta o processamento de pedidos em cenários com e sem falha nos componentes do Apache Kafka e, além disso, lida com altas frequências de chegada de mensagens sem perda.

Palavras-chave: Armazéns Inteligentes. *Streaming* de Dados. Arquitetura. Tempo Real.

ABSTRACT

Digital commerce has become one of the main means of purchase for the population in recent years and, consequently, with the large volume of information being generated daily by these platforms, it becomes necessary to search for strategies that make the processing of this data more efficient. Smart warehouses with robots that handle the management and separation of products, have achieved a solution for optimizing the flow of delivery of orders previously carried out only by people. The literature points to path planning and task allocation as the main problems related to this area, however, it is essential to consider the way the data is being processed and managed in the path between the digital platform and the warehouse robots, because according to other jobs, order processing is not done in real time. At the same time, there are data streaming messaging systems that can accomplish this task and bring more efficiency to the order delivery process in smart warehouses. The objective of this project is to propose a software architecture capable of handling the large flow of orders being placed in real time on sales platforms, using Apache Kafka (KAFKA, 2021), a unified, high-capacity, low-latency platform as the main communication tool. Test scenarios were created with different frequencies of arrival of requests and organizations of Kafka components, considering the amount of messages that are lost in these scenarios, in order to validate the proposal. The results appreciated that the developed architecture supports the processing of requests in scenarios with and without failure in the Apache Kafka components and, in addition, it handles high frequencies of message arrivals without loss.

Keywords: Smart Warehouses. Data Streaming. Architecture. Real Time.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACKS	*Acknowledgment*
B2C	*Business To Consumer*
ISR	*In-Sync Replicas*
JSON	*JavaScript Object Notation*
MPP	*Multi-Robot Path-Planning*
MRFS	*Mobile Robot Fulfillment System*
MRS	*Multi Robot Systems*
MRTA	*Multi-Robot Task Allocation*

SUMÁRIO

1 INTRODUÇÃO

Em 2021, as vendas de comércio eletrônico totalizaram aproximadamente 5,2 trilhões de dólares em todo o mundo e, além disso, prevê-se que este número cresça 56% nos próximos anos (COPPOLA, 2022). Com esse crescimento, novos modelos de armazéns surgiram com o objetivo de lidar com o grande volume de vendas e as necessidades especiais de vendedores e clientes envolvidos no *e-commerce*. De acordo com (BOYSEN; KOSTER; WEIDINGER, 2019), os armazéns tradicionais traziam problemas que os novos modelos de armazenamento conseguem resolver, são eles: pedidos pequenos, grande variedade, cronogramas de entrega apertados e cargas de trabalho variadas. Os chamados armazéns inteligentes são compostos por tecnologias de automatização e estão organizados de forma a se adequar com as necessidades desse novo estilo de comércio. Algumas das principais mudanças são as estações de trabalho automatizadas, robôs móveis, armazenamento em prateleiras mistas e processamento dinâmico de pedidos. Apesar das mudanças citadas nos armazéns serem majoritariamente em relação ao *hardware* desse sistema, se faz necessário apontar que esses modelos automatizados de armazenamento devem se interconectar com componentes de *software* capazes de consumir, armazenar e processar o grande volume de dados que esse sistema produz de forma síncrona, segura, confiável e tolerante a falhas.

As plataformas de processamento de mensagens são ditas como solução para os problemas de manipulação de grandes massas de dados, elas têm por objetivo transferir dados de um sistema para outro, para que esses sistemas se preocupem com os dados e não com a sua transmissão e compartilhamento. Existem dois tipos principais de padrões de mensageria: o sistema de mensagens ponta-a-ponta e o sistema de mensagens publicação/assinatura (*publish/subscribe*).

Algumas das plataformas de mensageria mais tradicionais no mercado são: *ActiveMQ*, *MSMQ* e *RabbitMQ*, mas elas têm limitações em termos de desempenho e taxa de transferência (GOEL; M PRABHANATH; T VARGHESE, 2017). O Apache Kafka, plataforma de streaming de dados desenvolvida no Linkedin, é a tecnologia considerada pela maioria das principais empresas de Internet como uma das melhores alternativas (KAFKA, 2021) para lidar com os milhões de dados que suas aplicações geram.

Desenvolvido como um sistema de mensagens do tipo publicação/assinatura de forma centralizada e aceitando tipos genéricos de dados, o Kafka surgiu como uma solução moderna, escalável e inovadora para lidar com grandes volumes de dados que surgem em tempo real. É descrito como uma “plataforma de distribuição de *streaming*” que fornece um sistema de armazenamento de dados de forma durável, consistente e em ordem, que permite leitura e escrita de registros de forma determinística. Além disso, esse sistema permite inúmeras configurações diferentes que trazem proteções

adicionais contra falhas e melhoria de desempenho, por exemplo.

O diferencial do Apache Kafka, é que ele traz três recursos principais e essenciais que servem para a implementação de novos sistemas de *streaming* de mensagens, são eles:

- Publicação e assinatura de mensagens, incluindo importação/exportação contínua de seus dados para outros sistemas;
- Armazenamento de dados de forma durável e confiável pelo tempo necessário;
- Processamento de dados à medida que eles ocorrem.

Além disso, o Kafka permite o fácil monitoramento de seus componentes. Atualmente, existem inúmeros serviços prestados por terceiros que oferecem *dashboards* de visualização para acompanhamento, como o *Confluent Control Center*, o *Conduktor* e o *Lenses*.

Levando em consideração os novos modelos automatizados de armazéns utilizados pelas plataformas de *e-commerce*, a gigantesca quantidade de dados gerados por estes e as necessidades de componentes de *software* que devem acompanhar essa mudança, o produto deste trabalho de conclusão de curso é uma arquitetura de processamento de dados em tempo real, que garantiu a comunicação e troca de mensagens entre uma plataforma de compras online e um armazém inteligente. A arquitetura utiliza o Apache Kafka como principal ferramenta para lidar com o fluxo de pedidos vindos da plataforma digital e enviá-los ao armazém, que produz informações sobre os pedidos que vão sendo finalizados a todo momento. Isso resultou em um grande volume de streaming de dados que foi processado pelos componentes do Kafka.

Como um dos objetivos do trabalho é simular esse fluxo de forma funcional e eficiente, mantendo a semelhança com o que acontece no mundo real, os trabalhos realizados em (OLIVEIRA, G.; PLENTZ; CARVALHO, 2021) e (OLIVEIRA, G. S. *et al.*, 2022) foram utilizados como complemento da arquitetura. Deste modo, os testes da arquitetura proposta que utiliza o Kafka, se tornaram semelhantes a casos do mundo real. Além disso, a escolha das ferramentas para o desenvolvimento do fluxo desse projeto garantiu a qualidade da arquitetura, escalabilidade e também a tolerância a falhas.

O trabalho propõe a execução e análise de duas arquiteturas que utilizam o Kafka, a primeira, mais simples, serviu como prova de conceito para a segunda, a arquitetura proposta. Ambas as arquiteturas passaram por casos de testes que consideram a quantidade de mensagens trocadas em cenários com e sem falha nos seus componentes. Com o intuito de validar a arquitetura proposta desenvolvida, foram criados diferentes cenários de testes que levam em consideração diferentes frequências de chegada de pedidos e também possíveis falhas em componentes essenciais

do Kafka. Os testes executados nestes cenários utilizam a métrica de quantidade de mensagens perdidas, as quais foram quantificadas em pontos essenciais do fluxo de processamento da arquitetura. Os resultados obtidos da realização dos testes validam a utilização da arquitetura em casos reais, para fins de pesquisa ou mercado, existem cenários em que ocorreu a perda de mensagens, porém com as configurações corretas dos componentes do Kafka, a arquitetura resolve este problema e garante o processamento de dados até mesmo em casos mais extremos, com a maior frequência de chegada de pedidos testada.

No capítulo de Fundamentação Teórica e Trabalhos Relacionados serão apresentados alguns conceitos importantes relacionados ao domínio de armazéns inteligentes e sistemas de mensageria em tempo real, além de trazer uma discussão e apresentação do que já foi feito na literatura sobre os escopos de armazéns inteligentes, seu monitoramento e o Apache Kafka. No capítulo de Arquiteturas do Kafka para Integração com Sistemas de Armazéns Inteligentes é descrito como foram as implementações de ambas as arquiteturas, bem como o desenvolvimento de seus módulos principais e a inserção dos componentes do Apache Kafka para realizar a sua conexão. Os testes feitos para a validação das arquiteturas são descritos no capítulo de Testes e Resultados Obtidos, bem como são apresentados os cenários em que serão feitas as suas execuções. Por fim, no capítulo de Conclusão, são apresentadas as considerações finais quanto ao trabalho apresentado e a seção Trabalhos Futuros sugere possíveis expansões para a arquitetura proposta por este trabalho de conclusão de curso.

1.1 OBJETIVOS

A implementação de uma arquitetura eficiente e com tolerância a falhas para lidar com o fluxo de dados entre plataformas digitais e armazéns inteligentes exige a elaboração de algumas tarefas. Estas tarefas refletem o objetivo geral e se desdobram em objetivos específicos, os quais são descritos a seguir.

1.1.1 Objetivo Geral

Desenvolver uma arquitetura capaz lidar com o grande fluxo de dados que conectam uma plataforma digital a um armazém inteligente.

1.1.2 Objetivos Específicos

- Implementar a arquitetura de *software* em Java (JAVA, 2021) para lidar com o volume de pedidos de um comércio digital;
- Utilizar o Apache Kafka como ferramenta de processamento de dados;

- Utilizar os trabalhos relacionados (OLIVEIRA, G.; PLENTZ; CARVALHO, 2021) e (OLIVEIRA, G. S. *et al.*, 2022) que se referem a arquitetura de armazém inteligente, para auxiliar o componente de simulação de retirada e entrega de pedidos;
- Realizar testes em diferentes cenários para validar a arquitetura proposta;
- Analisar os resultados obtidos, avaliando a quantidade de mensagens perdidas em diferentes cenários.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Esse capítulo é reservado ao embasamento teórico dos conceitos fundamentais para a compreensão do domínio deste trabalho, além de apresentar alguns trabalhos que possuem correlação com esta monografia.

2.1 FUNDAMENTAÇÃO TEÓRICA

Os primeiros pontos exploram o funcionamento de um armazém inteligente bem como os componentes que estão inseridos nesse ambiente. Depois, são apresentadas definições importantes sobre sistemas de mensageria e fluxo de dados que se fazem essenciais para a compreensão do último ponto, sobre o Apache Kafka.

2.1.1 Comércio eletrônico (*E-commerce*)

O termo “Comércio eletrônico” faz referência ao novo meio de comercialização de bens de serviços que ocorre na Internet. Algumas empresas vendem produtos apenas *online*, mas para muitas, o comércio eletrônico é um canal de distribuição que permite atingir um número muito maior de clientes, devido à facilidade de acesso à loja por meio de vários lugares do país, ou até do planeta.

2.1.2 Atendimento de pedidos (*Order fulfillment*)

O atendimento de pedidos ou *order fulfillment* se refere a todo o fluxo de vida de um produto oferecido em um comércio digital, desde o processo de armazenamento, embalagem e envio até o tratamento de devoluções e troca. As empresas podem escolher gerenciar esse processo internamente ou utilizar combinações de atendimento interno e de terceiros.

2.1.3 Armazéns

O armazém é uma instalação que além de realizar a armazenagem, ou seja a retenção intermediária de mercadorias, realiza também o recebimento, a separação e o envio de pedidos entre os estágios sucessivos do *order fulfillment*.

Essa instalação contém elementos de *hardware* que podem ser subdivididos em dispositivos de armazenamento, como os sistemas de manuseio de materiais e as ferramentas de coleta, e processos que definem o fluxo de trabalho ao longo desses componentes. Para se adequar ao varejo *online*, várias combinações desses elementos podem ser críticas para o funcionamento de um armazém de maneira a influenciar na satisfação dos seus clientes.

2.1.3.1 Armazéns inteligentes

De acordo com (BOYSEN; KOSTER; WEIDINGER, 2019), o *e-commerce* precisa reunir pequenos pedidos de um grande sortimento, sob grande pressão de tempo e ajustar com flexibilidade os processos de atendimento de pedidos para cargas de trabalho variadas. Por conta disso, foram surgindo novos modelos de armazéns, chamados armazéns inteligentes, que visam aumentar a qualidade geral do serviço, a produtividade e a eficiência de todo o processo de entrega de um pedido, minimizando custos e falhas, com o objetivo de suprir as novas necessidades dos comércios digitais que atendem diretamente às demandas dos clientes finais no segmento *business-to-consumer* (B2C).

(GEEST; TEKINERDOGAN; CATAL, 2022) diz que algumas das principais vantagens de um sistema de armazenamento inteligente são:

- Fornecimento de informações em tempo real, o que não é possível em armazéns tradicionais;
- Tarefas manuais minimizadas e automação de tarefas maximizada;
- Escalabilidade operacional melhorada, pois atualizar a infraestrutura é mais fácil do que atualizar o capital humano dentro da organização;
- Decisões automatizadas são tomadas usando diferentes modelos de previsão;
- Quando demandas do cliente ou os processos mudam, os armazéns inteligentes são facilmente adaptados aos novos casos em comparação com os armazéns tradicionais;
- Uso de sensores inteligentes para monitorar equipamentos caros e, portanto, o tempo de inatividade é minimizado.

2.1.4 Sistema multi-robôs

Um Sistema Multi-Robôs ou do inglês *Multi-Robot Systems* (MRS), consiste em um grupo de robôs projetados com a finalidade de realizar algum objetivo coletivo que seria impossível ou inviável de ser alcançado por um único robô. Os robôs de um MRS conseguem executar tarefas de forma simultânea ou formando coalizões, organizando-se em grupos para atender ao objetivo final.

A resolução de tarefas complexas, o aumento do desempenho geral de um objetivo e a confiabilidade do sistema, são alguns dos vários benefícios que um sistema com vários robôs trazem quando comparados com sistemas de robôs únicos.

2.1.5 Sistema de atendimento por robô móvel (*Mobile robot fulfillment system*)

O *Mobile Robot Fulfillment System* (MRFS) é um sistema de armazenamento que emprega um sistema multi-robôs, no qual os robôs coletam pedidos das prateleiras e os levam até as estações de entrega de um armazém inteligente. Tendo um alto desempenho de coleta, o MRFS é um dos componentes que se tornaram essenciais para a evolução do modo de armazenamento, os tornando adequados para os prazos apertados do comércio eletrônico.

Para esse sistema, os diferentes tamanhos e variedade de produtos de pedidos não são problemáticos, desde que haja espaço suficiente no armazém para comportar os produtos de forma organizada, garantindo a melhor movimentação entre as prateleiras. Em contraste com outros sistemas de seleção de produtos, o MRFS tem um *layout* flexível e vem sem *hardware* instalado de forma fixa, se tornando escalável para cargas e tipos de trabalho variados.

2.1.6 Processamento de pedidos em armazéns de comércio eletrônico

Nos últimos anos, surgiram alguns recursos responsáveis por diferenciar o fluxo de trabalho de processamento de pedidos em armazéns de comércio eletrônico de armazéns convencionais. Como dito anteriormente, os armazéns inteligentes precisam reunir pequenos pedidos de um grande sortimento, sob grande pressão de tempo e ajustar com flexibilidade os processos de atendimento de pedidos para cargas de trabalho variadas. Por conta disso, à medida que armazéns automatizados foram surgindo, surgiram também diversas formas de implementação desse processo, visando buscar o máximo de recursos necessários possíveis. A seguir, serão descritas cada parte do novo fluxo de trabalho de processamento de pedidos que está sendo aplicado na grande maioria dos armazéns inteligentes atuais, de acordo com o estudo de (TANG *et al.*, 2022):

- **Pré-processamento:** ocorre depois que um pedido é solicitado, o sistema de armazenamento verifica o estoque de todos os produtos incluídos no pedido.
- **Agregação:** é a junção de pedidos que atendem às condições de separação em uma lista. Essa lista de separação contém vários pedidos individuais, que podem economizar distâncias de viagem e o tempo de separação de produtos, o que é crucial para a eficiência do armazém.
- **Seleção e classificação:** de acordo com as listas de separação (criadas na fase de Agregação), os responsáveis por coletar os pedidos (pessoas ou robôs móveis) vão às prateleiras e coletam as mercadorias, as quais irão precisar de uma classificação posterior se houver pedidos individuais isolados.

- **Embalagem e ponderação:** a qualidade das mercadorias são verificadas e comparadas com cada solicitação de pedido. Pedidos com produtos danificados ou que não pertencem às solicitações são bloqueados e aguardam pela Inspeção. Caso contrário, os produtos de um pedido são embalados em pacotes e pesados automaticamente em uma esteira, para conferir se atendem às exigências dos responsáveis pelo envio.
- **Saída:** é a parte final do fluxo, onde o pacote é passado para o responsável pelo transporte e, em seguida, é marcado como pedido concluído pelo armazém.
- **Inspeção:** é o processo de verificação de pedidos defeituosos e resolução do erro para atender à solicitação.
- **Redefinição:** é executado caso algum pedido que já tenha sido coletado e embalado, seja cancelado pelo cliente, o resto do processo é interceptado e os produtos devolvidos às prateleiras.

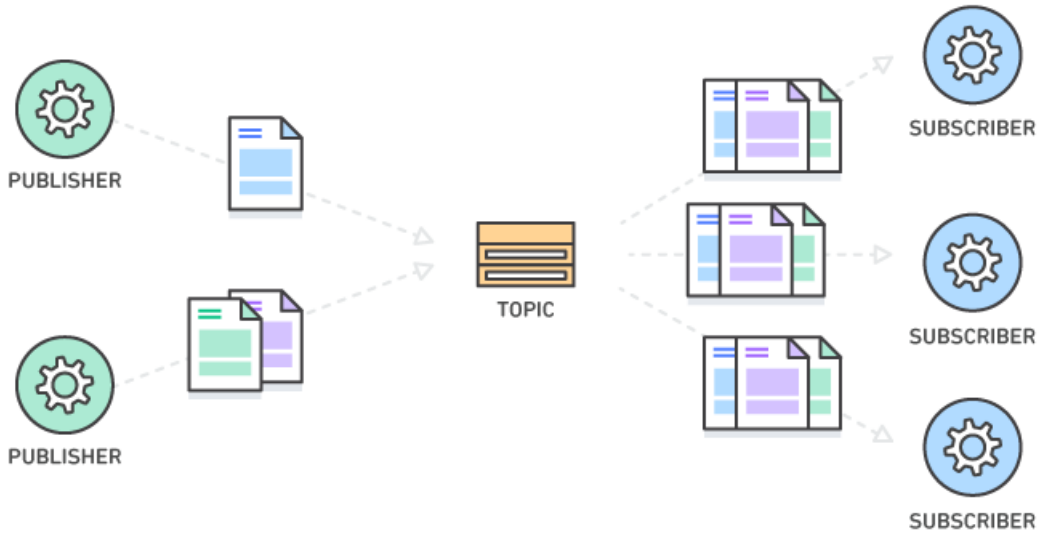
2.1.7 Sistema de mensageria

Um sistema de mensageria livra os aplicativos de se preocuparem com a transmissão e compartilhamento de seus dados com outros sistemas, deixando a responsabilidade de apenas gerar essas informações, ele é responsável por realizar essa função baseando-se no conceito de enfileiramento de mensagens. Nesse conceito, as mensagens são enfileiradas de forma assíncrona entre os aplicativos do cliente e o sistema de mensageria. Existem dois tipos principais de padrões de mensageria, o sistema de mensagens ponta-a-ponta e o sistema de mensagens publicação/assinatura (*publish/subscribe*). A próxima seção traz uma explicação sobre a segunda estrutura, a qual o Apache Kafka implementa.

2.1.7.1 Sistema de publicação/assinatura

Um sistema de mensageria no padrão de publicação/assinatura permite que mensagens sejam transferidas para diferentes partes de um sistema de forma assíncrona, na qual os remetentes são desassociados dos destinatários. Esse serviço, projetado para ser altamente confiável e escalável, tem alguns componentes essenciais para cumprir com esse objetivo, são eles: as mensagens, que são os dados transferidos; o tópico, que retém e transmite esses dados permitindo conexões dos componentes de envio e recebimento; o produtor ou *publisher* que faz o envio de mensagens e o assinante ou *subscriber* que os consome. O fluxo básico de uma mensagem que está sendo transferida por um sistema de publicação/assinatura pode ser visto na Figura 1.

Figura 1 – Arquitetura de um sistema de publicação/assinatura.



Fonte: (AMAZON, 2021)

Quando um ou mais produtores (*publisher*) querem transmitir uma mensagem, eles devem se conectar a um tópico e realizar o envio desse dado para o mesmo. O tópico irá processar essas mensagens e as enviar imediatamente para todos os assinantes (*subscriber*) que estão conectados nele ou somente para os assinantes que seguem uma política de filtragem que deve ser definida anteriormente.

2.1.8 *Streaming* de eventos

Um evento é uma mudança de estado de algum componente de um aplicativo, e para muitas empresas, é crucial para o seu negócio ter a capacidade de reagir a essas mudanças em tempo real. Sendo assim, surge o conceito de *streaming* de eventos, que é a prática de capturar esses dados em tempo real de fontes (como sensores, dispositivos móveis e aplicativos) e fazer o armazenamento e transmissão destes de forma escalável e durável para diferentes tecnologias de destino.

De acordo com (KAFKA, 2021), o *streaming* de eventos pode ser aplicado a uma ampla variedade de casos de uso de diferentes setores e organizações, como por exemplo: processamento de pagamentos e transações financeiras em tempo real (bolsas de valores, bancos e seguros), rastreamento e monitoramento de carros, caminhões, frotas e embarques, e coleta e reação imediata às interações e pedidos do cliente (como no varejo, no setor hoteleiro e de viagens e em aplicativos móveis).

2.1.9 Apache Kafka

Apache Kafka é um sistema que segue o padrão de mensageria de publicação/assinatura, é de código aberto e projetado com o objetivo de fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados que são gerados em tempo real. Para um melhor entendimento da plataforma como um todo, as próximas seções trazem um embasamento dos principais conceitos utilizados pelo Kafka.

2.1.9.1 Mensagem

Similarmente a um registro em um banco de dados tradicional, uma mensagem é como é chamada a unidade de dado dentro do Kafka. Esse dado nada mais é do que uma simples lista de *bytes* que pode ou não possuir uma outra lista de *bytes* de dados atrelada a si, que corresponde a uma chave. Esses dois componentes, por serem apenas uma estrutura de *bytes*, não significam nada para a plataforma em si, o que é uma vantagem pois assim, o sistema se torna adaptável para qualquer tipo de dado que queira ser transmitido.

As chaves são usadas quando mensagens precisam ser transmitidas com um maior controle, sendo utilizadas como uma espécie de filtro que as redireciona para o lugar correto de onde elas serão consumidas.

2.1.9.2 Tópico

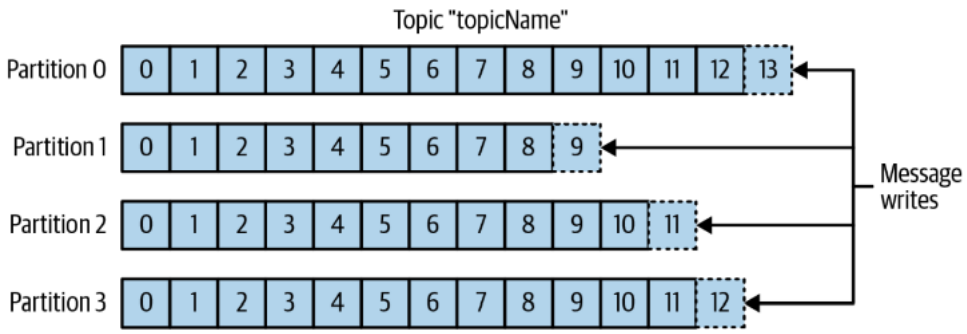
Os tópicos servem como se fossem pastas em um sistemas de arquivos. São neles que mensagens são armazenadas e consumidas pelos seus componentes conectados. No Kafka, um tópico pode possuir zero, um ou muitos produtores e consumidores ligados, que podem escrever e consumir mensagens de acordo com a frequência que for necessária. Os dados são escritos em um tópico apenas de forma anexada e são lidos em uma fila (do início ao fim), além disso, independente do tamanho, as mensagens podem ser armazenadas pelo tempo que for considerado adequado ao desenvolvedor da infraestrutura.

2.1.9.3 Partição

Manter um tópico restrito a operar em somente uma máquina colocaria um grande limite na capacidade de dimensionamento da plataforma. Como algumas das principais vantagens do Kafka são a sua escalabilidade e capacidade de tolerância a falhas, esse sistema permite a partição de tópicos. Diferentes servidores podem armazenar partições distintas ou cópias de uma mesma partição, garantindo redundância e escalabilidade.

Quando uma nova mensagem é escrita em um tópico, na verdade ela é anexada a uma de suas partições. Mensagens com as mesmas chaves são gravadas na mesma partição e o Kafka garante que qualquer consumidor de uma determinada partição sempre lerá os dados exatamente na mesma ordem em que foram gravados. A Figura 2 ilustra um tópico com 4 partições.

Figura 2 – Representação de um tópico (*topicName*) dividido em 4 partições.



Fonte: (SHAPIRA *et al.*, 2021)

2.1.9.4 Lote

Lotes são utilizados para trazer mais eficiência ao fluxo de transmissão, eles são um conjunto de mensagens que devem ser redirecionadas para uma mesma partição do mesmo tópico. Os lotes trazem uma redução de carga em uma rede, pois como as mensagens são enviadas a todo momento, as suas transmissões individuais podem causar uma sobrecarga excessiva. É necessário pontuar que esse conjunto de mensagens não pode ser muito grande, pois apesar de possibilitar que mais dados possam ser processados a cada segundo, mais tempo irá levar para uma mensagem se propagar.

2.1.9.5 Produtores e Consumidores

Produtores são os componentes que gravam mensagens em um ou mais tópicos do Kafka e consumidores são os que leem essas mensagens. Para alcançar a escalabilidade que a plataforma promete, os produtores e consumidores são totalmente dissociados uns dos outros, isto é, esses componentes se comunicam via um intermediário (tópico) e possuem desacoplamento espacial e temporal. No desacoplamento espacial, o produtor/consumidor não conhece ou não precisa conhecer a identidade do consumidor/produtor; e no desacoplamento temporal, o produtor e o consumidor podem ter independência do momento de existência. Por exemplo, um consumidor não precisa de outro para consumir dados de um mesmo tópico, o Kafka permite a capacidade de processamento de eventos no máximo uma vez.

2.1.9.6 *Broker*

Um *Broker* é um único servidor do Kafka, ele é responsável tanto pela recepção de mensagens de produtores quanto pelas solicitações de busca de partições e leitura de mensagens dos consumidores.

2.1.9.7 *Cluster*

Os *brokers* são organizados em *clusters*. Logo, um *cluster* é composto por um ou mais *brokers* que trocam informações de gerenciamento entre si. Neste contexto, um dos *brokers* é eleito como líder e administra os demais atribuindo partições e monitorando falhas.

Quando uma partição de um tópico é replicada, uma de suas cópias é eleita como líder e as demais são as seguidoras. Essa replicação permite redundância de mensagens na partição, de modo que um de seus seguidores pode assumir a liderança se houver uma falha do *broker* líder. Todos os produtores desta partição devem escrever mensagens em seu líder, porém os consumidores podem ler as mensagens de qualquer uma de suas réplicas.

A Figura 3 ilustra um exemplo de um *cluster* que possui 2 *brokers*, *Broker* 1 e *Broker* 2. Cada *broker* possui um tópico A com duas partições replicadas (partição 0 e 1), um produtor e um consumidor. O produtor envia mensagens aos dois *brokers*, garantindo redundância. No *Broker* 1 as mensagens estão sendo escritas na Partição 0 do Tópico A e no *Broker* 2, na Partição 1 do mesmo tópico, pois essas são as partições líderes de cada *broker*. Além disso, a cada dado escrito, ocorre o processo de cópia das mensagens entre um *broker* e outro (*Replicate A/0 e Replicate A/1*).

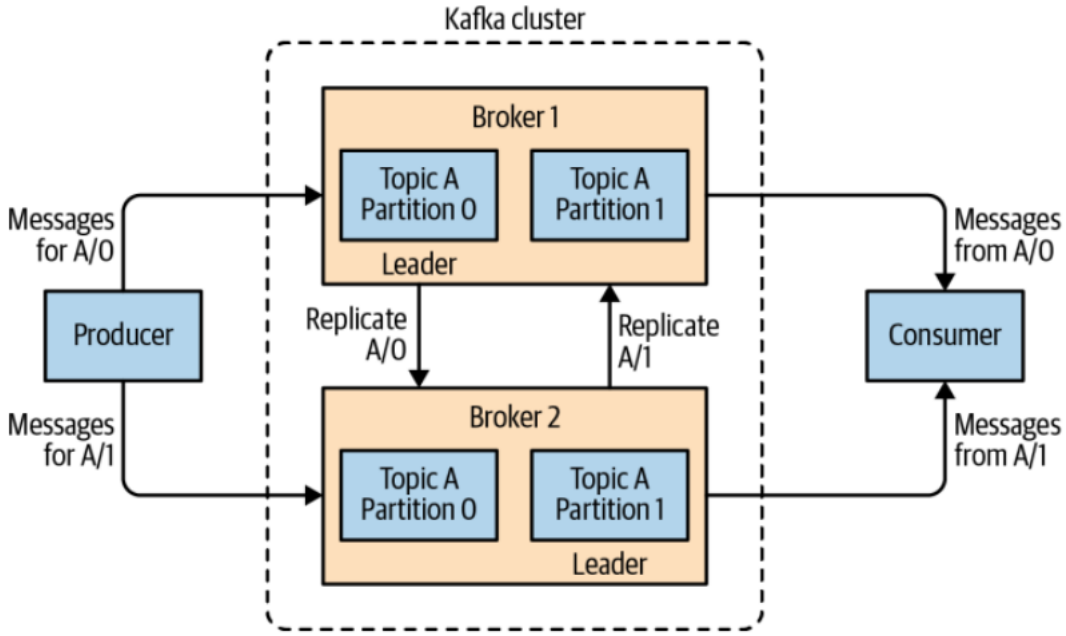
2.1.9.8 Réplicas Sincronizadas e ACKS

A ISR (*In-Sync Replicas*), ou réplicas sincronizadas, é uma lista que contém as partições que estão sincronizadas entre si, com a mesma quantidade de mensagens. Uma ISR sempre conterá a partição líder e zero ou mais seguidoras. Um seguidor é considerado “em sincronia” se está completamente atualizado com o líder em algum ponto de um período pré definido de tempo, chamado de “replica.lag.time.max.ms”, que por padrão é 500ms, isto é, estar em dia com o líder é ter uma cópia exata de suas mensagens. Um seguidor é removido da ISR se:

- Não faz uma solicitação de sincronização dentro do período “replica.lag.time.max.ms”;
- Se está desatualizado por mais tempo do que o período “replica.lag.time.max.ms”.

O ACKS (*Acknowledgements*) ou reconhecimento, é uma configuração que pode ser passada ao produtor de mensagens. Esse parâmetro denota o número de

Figura 3 – Representação de um *cluster* do Kafka com dois *brokers* e replicações de partição.



Fonte: (SHAPIRA *et al.*, 2021)

brokers que devem receber a mensagem antes dela ser considerada persistida com sucesso no tópico, suporta 3 valores: 0, 1 e *all*.

- **ACKS = 0:** com o valor em 0, o produtor não espera uma confirmação do *broker*, ele considera que a mensagem foi persistida no momento em que é enviada ao tópico, conhecido por "enviar e esquecer" ou "*fire-and-forget*";
- **ACKS = 1:** com a configuração 1, o produtor considera a gravação bem sucedida quando o *broker* que contém a partição líder do tópico persiste a mensagem. O *broker* responde imediatamente após a persistência na partição líder e os seguidores replicam a mensagem de forma assíncrona depois;
- **ACKS = all:** com o valor *all* o produtor só considera a publicação da mensagem bem sucedida quando todas as réplicas que estão na ISR persistirem o dado. O *broker* que contém a partição líder envia a confirmação ao produtor quando todos os seus seguidores confirmam a gravação da mensagem.

O último valor é o que possui mais garantia para que não aconteça a perda de dados, porém não é a melhor opção para casos em que a latência e taxa de transferência sejam mais importantes.

Com base nos conceitos acima descritos, este trabalho desenvolveu uma arquitetura de software que gerencie os pedidos vindos de uma plataforma de comércio

digital e os encaminhe para um armazém inteligente utilizando o Apache Kafka como a ferramenta de *streaming* desses dados. A próxima seção apresenta alguns trabalhos relacionados.

2.2 TRABALHOS RELACIONADOS

Essa seção apresenta alguns dos trabalhos mais recentes e relevantes que estão relacionados à área de armazéns inteligentes e sistemas de mensageria em tempo real, além de fazer um comparativo sobre como essas literaturas se aproximam ou se diferem da proposta deste trabalho.

2.2.1 Armazéns Inteligentes

Existem muitos trabalhos na área de armazéns inteligentes inseridos no cenário de *e-commerce*, a maioria deles propõem soluções para melhorar a performance de todo o fluxo, ou de apenas uma parte, do processamento de pedidos. Os autores apresentam melhorias para alguns pontos considerados essenciais para o funcionamento de armazéns modernos: o planejamento de caminhos (*Multi-Robot Path-Planning - MPP*), a alocação de tarefas (*Multi-Robot Task Allocation - MRTA*), ambos sobre os robôs que realizam a coleta e entrega de pedidos. Por exemplo, (LIU *et al.*, 2021) aborda o problema do MRTA e implementa, com diferentes algoritmos de alocação (e.g. *First-Come-First-Serve*), um sistema de *Tag* de Pedido que rotula itens que chegam dinamicamente no armazém e assim, garante a alocação prioritária de tarefas de pedidos primeiro, minimizando atrasos desnecessários. O trabalho de (SHI; HU; HUANG, 2021) aborda o MPP e o MRTA em um armazém com robôs independentes. Os autores propõem uma solução eficiente que pode aumentar o desempenho dessas máquinas trabalhando em um armazém com prateleiras e estantes de coleta pré-determinadas, sua solução consiste em uma abordagem descentralizada na qual os próprios robôs estimam seu tempo de conclusão de tarefas, além disso, eles projetam um esquema eficaz para evitar colisões entre as máquinas.

Em (OLIVEIRA, G.; PLENTZ; CARVALHO, 2021), os autores propõem uma arquitetura de *software* de um armazém inteligente que integra MRTA e MPP e lida com tempo e energia como restrições principais, e além disso, os autores propõem também o MCVB-TA (*Multi-Constrained Voronoi-Based Task Allocator*), um mecanismo que utiliza o diagrama de *Voronoi* para definir uma série de condições iniciais que são usadas como entrada dessa arquitetura. Essa estrutura é composta por 6 módulos: Pedidos, Organizador de Tarefas, Estimador de Custo, Alocador de Tarefas, Planejador de Caminhos e Monitor.

Os testes foram feitos utilizando um algoritmo que gera pedidos aleatórios dentro de um determinado intervalo e os inserindo em dois cenários na arquitetura: um que

realiza a alocação de tarefas de forma regular (executa o pedido assim que ele chega, sem realizar um processamento ou classificação sobre), e outro que utiliza o MCVB-TA proposto. Após as simulações, o mecanismo retorna o tempo que cada pedido levou para ser finalizado e o consumo de energia de cada robô. Os resultados mostram a eficiência do algoritmo proposto, que reduz o tempo e a energia da execução de tarefas quando comparado ao alocador normal.

Como outra versão do trabalho citado anteriormente, os autores propõem em (OLIVEIRA, G. S. *et al*., 2022) uma estratégia de alocação para MRFS que considera muitas das restrições de um armazém, como a quantidade de estações de entrega, a paralelização de tarefas entre robôs e a constância de recebimento de pedidos. O algoritmo, *Domain zoNe based Capacity and Priority Constrained Task Allocator* (DoNe-CPTA), leva em consideração cenários multi-restritos, tarefas dinâmicas e robôs heterogêneos, e utiliza como base os conceitos de (OLIVEIRA, G.; PLENTZ; CARVALHO, 2021) para atribuir tarefas aos robôs de forma eficiente. Os testes de validação do DoNe-CPTA foram realizados em um armazém que simula uma instalação real e um algoritmo que gera cerca de 6 pedidos aleatórios a cada 1 minuto, seguindo uma distribuição gama. A quantidade de produtos por pedido e seu respectivo peso foram definidos por uma distribuição lognormal. Os autores estimaram tais informações a partir de dados publicados em sites sobre grandes lojas de varejo e as consideram próximas de um ambiente de armazém inteligente real. Os resultados mostraram que o DoNe-CPTA pode reduzir até 45% do custo das rotas de robôs e 24% do tempo para atender aos pedidos.

2.2.2 Monitoramento

É importante reconhecer que o processamento de pedidos em armazéns inteligentes também possui grande influência na eficiência de entrega do *e-commerce*. Todo esse fluxo gera uma massa de dados de *streaming* em tempo real, na qual dados de pedidos podem ser afetados devido a situações imprevisíveis, por isso uma análise visual dessas mensagens é um conceito chave a ser abordado.

O autor em (XU; MEI; CHEN, 2016) propõe o ViDX, um sistema de análise com interface gráfica para diagnosticar eventos anormais em etapas de processamento de pedidos em armazéns inteligentes. Os autores aplicam uma técnica de agregação visual com reconhecimento de tempo e preservação de valores discrepantes para revelar possíveis anomalias históricas e as projetam em um gráfico radial, para que assim seja possível monitorar os processos. Entretanto, esse trabalho é insuficiente para monitorar e analisar eventos de *streaming* em tempo real, pois não contém detalhes sobre estações de trabalho paralelas e tem problemas de escalabilidade, além de assumir que a carga de trabalho e o cronograma de processamento são estáveis, o que se difere da situação de armazéns inteligentes com pedidos *online* imprevisíveis.

A abordagem de visualização proposta por (TANG *et al.*, 2022) considera esses padrões assíncronos de chegada e saída de dados. O sistema de análise visual *OrderMonitor* desenvolvido neste trabalho auxilia gerentes de armazéns a melhorar a eficiência do processamento de pedidos em tempo real, se baseando no *streaming* de dados de eventos que ocorrem nessa instalação. O fluxo é inserido em um novo *design* que facilita o monitoramento de pedidos em tempo real e indica potenciais pedidos anormais, considerando seus cronogramas e utilizando com base gráficos de Gantt e Marey. Apesar desse trabalho propor uma solução que processa e monitora dados de um armazém em tempo real, os autores não utilizam um sistema de mensageria que lida com esse tipo de *streaming*, mas sim um banco de dados que armazena os eventos em ordem.

2.2.3 Apache Kafka

Muitos trabalhos na literatura fazem estudos e avaliações de performance sobre o Apache Kafka e o elegem como principal plataforma para realizar o processamento e armazenamento de dados em tempo real. Os autores de (HIRAMAN; VIRESH M.; ABHIJEET C., 2018) fazem um estudo sobre a plataforma trazendo uma visão geral dos principais componentes e fazendo testes de troca de mensagens, além disso, eles trazem um levantamento das principais grandes empresas que utilizam o Kafka, bem como seus casos de uso:

- **Twitter:** infraestrutura de processamento de *streaming*;
- **LinkedIn:** *streaming* de dados em vários produtos como o *feed* de notícias e o sistema analítico *offline*;
- **Yahoo!:** análise de mídia em tempo real;
- **Netflix:** porta de entrada para coleta de dados, pois exige que bilhões de mensagens sejam processadas diariamente.

Em (SHARVARI; SOWMYANAG, 2019), os autores fazem uma análise sobre os principais sistemas de mensageria atuais (Apache Kafka, RabbitMQ e NATS Streaming) os comparando entre 7 de seus recursos principais, são estes: entrega e persistência de mensagens, ordenamento, taxa de transferência, latência, disponibilidade e escalabilidade. Os resultados desse levantamento trazem o Kafka como melhor opção para sistemas distribuídos que necessitam de uma análise de dados em tempo real de alto rendimento e baixa latência, e sistemas não críticos, que possuem grande volume de dados sendo produzidos em tempo real e não se preocupam com mensagens sendo perdidas, como sites de rastreamento de usuários, redes sociais e anúncios da Internet. Em (GOEL; M PRABHANATH; T VARGHESE, 2017), é implementada uma aplicação para auxiliar a visualização e pesquisa de mensagens do Apache Kafka, os autores fazem a escolha dessa plataforma pois, comparado a outros sistemas de

mensageria (MSMQ, RabbitMQ e ActiveMQ), o Kafka é um sistema que oferece escalabilidade e taxa de transferência mais rápida que as demais, cerca de 30%. A aplicação desenvolvida consegue buscar os dados que são enviados ao servidor do Kafka por um componente externo e mostrar todas as informações necessárias sobre esse fluxo em uma interface interativa.

Apesar desses, e da maioria dos trabalhos, recomendarem o Kafka e o inserirem em vários cenários de processamento de dados em tempo real, até o momento da finalização da etapa de pesquisas deste trabalho e do conhecimento da autora, somente uma literatura utiliza o Apache Kafka no contexto de armazéns inteligentes. Em (LOURENÇO *et al.*, 2021) é feito um estudo sobre a maneira mais simples de conexão entre o Kafka e ROS (*Robot Operating System*), uma plataforma que está sendo amplamente utilizada para aplicações robóticas, incluindo os armazéns inteligentes. Os autores desenvolvem um código ponte entre esses dois sistemas e realizam testes em três simulações com cenários reais, utilizando perfis de Qualidade de Serviço (*QoS*) fornecidos pelo Serviço de Distribuição de Dados (DDS) que permitem alcançar resultados de alta confiabilidade. Os resultados dessa pesquisa foram satisfatórios e apontam que o Kafka pode ser facilmente integrado com o ROS e um sistema de armazém inteligente.

2.2.4 Considerações Finais

Pensando no grande volume de dados em tempo real que o fluxo do *e-commerce* gera a todo momento, na necessidade de monitoramento desses dados para conseguir eficiência na entrega de produtos, na recomendação do Apache Kafka como plataforma escalável, distribuída e que traz resultados confiáveis na comunicação entre esse sistema e um sistema de robôs móveis inseridos em um armazém, este trabalho de conclusão de curso inseriu componentes dessa ferramenta no fluxo de processamento de pedidos de uma plataforma de vendas *online*. Foi desenvolvida uma arquitetura de software que utiliza elementos do Kafka e permite a conexão entre o site de *e-commerce*, onde são feitos os pedidos, o armazém inteligente e uma ponta com alta capacidade de conexão com um sistema de monitoramento, por exemplo. Inicialmente foi desenvolvida uma arquitetura base, que serviu como prova de conceito e que evoluiu até chegar na arquitetura do objetivo deste trabalho. Para realizar os testes, foi utilizada a arquitetura de armazém proposta em (OLIVEIRA, G.; PLENTZ; CARVALHO, 2021) com métricas de simulação de (OLIVEIRA, G. S. *et al.*, 2022) como componentes que representaram a coleta e entrega de produtos dos pedidos.

3 ARQUITETURAS DO KAFKA PARA INTEGRAÇÃO COM SISTEMAS DE ARMAZÉNS INTELIGENTES

Esse capítulo apresenta o desenvolvimento da arquitetura base que deu origem à arquitetura proposta por este trabalho, e está dividido em quatro partes principais. A primeira parte fornece detalhes sobre a metodologia utilizada para a composição do trabalho. Depois, é apresentado um detalhamento sobre os tipos de dados que serão utilizados, bem como de onde foram retirados, tanto para o desenvolvimento quanto para os testes das arquiteturas. A terceira parte descreve o desenvolvimento da arquitetura base, assim como dos seus módulos principais e suas conexões entre si utilizando os componentes do Apache Kafka. Por fim, é detalhado como a arquitetura base evoluiu para chegar até a arquitetura proposta e quais as suas características que contribuem para atingir o objetivo deste trabalho, que é realizar o processamento de pedidos entre um armazém e uma plataforma digital de forma síncrona, segura, confiável e tolerante a falhas.

3.1 METODOLOGIA

Inicialmente foram levantadas pesquisas relacionadas ao tema do trabalho, como funcionamento de armazéns inseridos no contexto do *e-commerce*, estrutura do Apache Kafka e conceitos e parâmetros importantes para a organização dos componentes da plataforma, para assim, definir uma base teórica para o desenvolvimento das arquiteturas.

Ambas arquiteturas foram implementadas em Java, linguagem nativa do Kafka, utilizando todas as bibliotecas necessárias para o envolvimento das partes. Para auxiliar os testes da proposta, alguns *scripts* de execução foram desenvolvidos em linguagem de comando. O desenvolvimento aconteceu em etapas, primeiro conectando as partes do fluxo de dados formando a arquitetura base e em seguida executando testes em cima desta. E, depois, a segunda arquitetura foi desenvolvida como uma extensão da primeira, adicionando os componentes necessários para suprir os requisitos da proposta. Por fim, foram executados testes novamente em diferentes cenários para a análise de perda de mensagens e tolerância a falhas.

3.2 DADOS

Para o desenvolvimento e testes das arquiteturas, foram utilizados conjuntos de dados de simulações de coleta e entrega de pedidos feitas em (OLIVEIRA, G. S. *et al.*, 2022). Cada arquivo disponibilizado possui uma série de informações e corresponde a uma simulação já executada e que se aproxima do mundo real. Para os objetivos deste projeto, apenas dados relevantes foram extraídos de cada arquivo, sendo estes a lista de pedidos e o tempo até cada pedido ser finalizado pela simulação.

Cada mensagem de pedido está no formato JSON (*JavaScript Object Notation*) e possui em sua estrutura um identificador único (id) e uma lista dos produtos que o compõem. Cada produto possui os seguintes dados: identificador único do produto dentro do pedido, coordenadas X e Y que indicam a sua posição no armazém, altura da prateleira e seu peso. A Figura 4 ilustra um exemplo de um pedido com 3 produtos. Os tempos também estão em um arquivo no formato JSON e são mapeados aos respectivos pedidos através do seu identificador único.

Figura 4 – Exemplo de um pedido com 3 produtos no formato JSON.


```
{
    "1": {
        "coord_x": 151,
        "coord_y": 40,
        "altura_prateleira": 0,
        "peso_total": 26.4
    },
    "2": {
        "coord_x": 131,
        "coord_y": 21,
        "altura_prateleira": 2,
        "peso_total": 1.4
    },
    "3": {
        "coord_x": 182,
        "coord_y": 51,
        "altura_prateleira": 1,
        "peso_total": 0.5
    }
},
```


Fonte: A autora (2022).

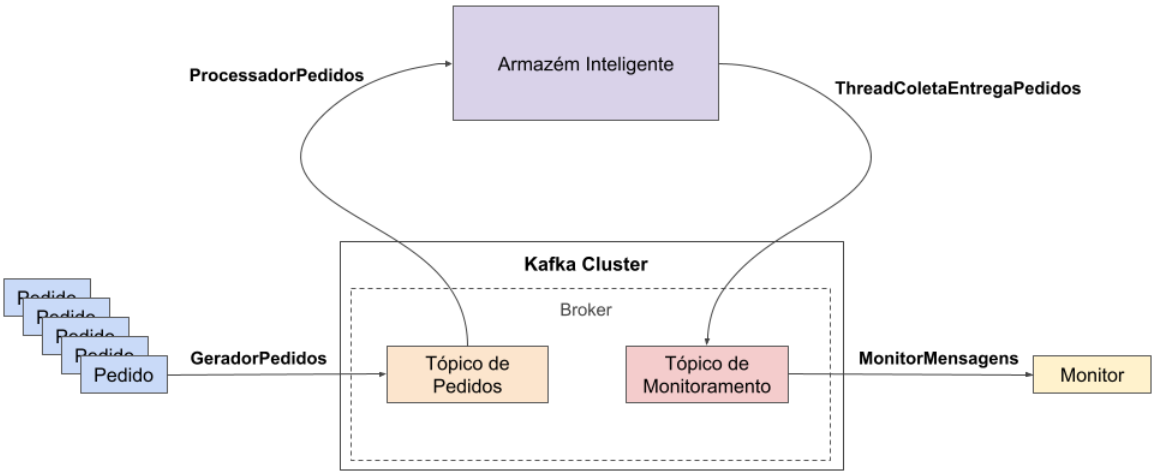
A utilização de dados que se aproximam de casos reais que já foram validados por outros trabalhos para os testes e desenvolvimento da arquitetura, se faz muito importante pois validaram a arquitetura proposta.

3.3 ARQUITETURA BASE

A arquitetura base ou *baseline* do projeto, foi pensada e desenvolvida para ser apenas uma prova do conceito da proposta deste trabalho. O seu desenvolvimento foi importante para realizar a implementação e conexão dos módulos principais, utilizando apenas componentes essenciais do Kafka, comprovando depois, quando testada, que o objetivo principal deste trabalho pode ser alcançado de forma segura contra falhas. A Figura 5 ilustra como os seus componentes estão conectados e organizados. Foram desenvolvidos quatro módulos de serviço que se conectam utilizando componentes do Kafka, sendo estes: um *cluster* com um *broker* e dois tópicos com uma partição

cada, inicialmente sem utilizar os conceitos de replicação. O primeiro tópico, chamado “Tópico de Pedidos” tem o módulo "GeradorPedidos"conectado como produtor e o módulo “ProcessadorPedidos” como consumidor de mensagens. O segundo, “Tópico de Monitoramento” contém as *threads* de coleta e entrega de pedidos como produtoras de dados e o módulo de monitoramento que os consome.

Figura 5 – Arquitetura base de processamento de pedidos utilizando Apache Kafka.



Fonte: A autora (2022).

3.3.1 Módulo GeradorPedidos

Funciona como um produtor, é responsável pela publicação de mensagens no tópico de pedidos. Esse módulo tem como entrada um arquivo com uma lista de pedidos que foram extraídos de uma simulação. Em uma determinada frequência de tempo, que será configurada nos testes, o próximo pedido da lista é enviado ao tópico de pedidos.

Como o pedido está em formato JSON e as mensagens no Kafka são apenas uma lista de *bytes*, antes de publicar no tópico, esse módulo faz a serialização do pedido o convertendo para este formato.

3.3.2 Módulo ProcessadorPedidos

É o consumidor do tópico de pedidos, responsável por encaminhar as mensagens às *threads* que simulam o fluxo do armazém. O Kafka permite consumir mensagens de um tópico no formato de lotes garantindo o não sobrecarregamento da rede.

Por conta disso, é possível configurar a frequência que os consumidores de um tópico fazem o carregamento de mensagens.

Considerando o padrão do Kafka, esse módulo faz a consumação do tópico a cada 1ms e para cada mensagem recebida nesse lote, faz a sua deserialização, permitindo ao programa ler os seus dados. Depois disso, para cada dado convertido, uma *thread* do tipo “ThreadColetaEntregaPedido” é criada recebendo o pedido como parâmetro.

3.3.3 ThreadColetaEntregaPedido

Como o foco deste projeto é propor uma arquitetura com Kafka que lide com pedidos que chegam e saem de um armazém, o objetivo de cada ThreadColetaEntregaPedido é simular todo o processo que acontece dentro dessa instalação, tomando como base a arquitetura proposta em (OLIVEIRA, G.; PLENTZ; CARVALHO, 2021), isto é, a divisão de cada pedido em tarefas, a distribuição dessas tarefas entre os robôs e a coleta e entrega de produtos até as estações finais.

Sendo assim, quando cada *thread* é criada, ela recebe como parâmetro o pedido que está responsável por simular. A partir do pedido, é possível acessar seu identificador único e mapear para o tempo que este levou para ser finalizado. Depois disso, a *thread* fica em estado de espera durante o tempo necessário e ao final cria uma mensagem de finalização, que contém dados do pedido e seu tempo de processamento.

As *threads* de coleta e entrega de pedidos funcionam também como produtores do tópico de monitoramento, as mensagens de finalização são convertidas em uma lista de *bytes* pelo próprio componente e depois publicadas no tópico, garantindo a frequência de publicação similar à que ocorre em um armazém no mundo real.

3.3.4 Módulo MonitorMensagens

É o módulo consumidor do tópico de monitoramento, sua função é receber as mensagens de finalização de pedidos criadas pelas *threads*. Seguindo o padrão do Kafka neste componente também, a cada 1 ms é executada a função para processar as mensagens que foram recebidas neste intervalo de tempo, no formato de lotes. Para cada dado recebido, é feita a sua conversão e as informações dos pedidos, seu identificador único e o tempo de execução, são apenas exibidas na tela. A Figura 6 mostra um exemplo.

A ideia do monitor é ser uma ponta livre na arquitetura, que permite várias conexões (uma das vantagens do Apache Kafka) com serviços que podem ser úteis à eficiência do armazém, como por exemplo um *dashboard* de análise e visualização dos dados, como o proposto em (TANG *et al.*, 2022).

Figura 6 – Exemplo de saída do módulo MonitorMensagens.

```
Pedido 1 finalizado: MensagemFinalizacao [idPedido=1, tempoTotalEntrega=67.1111]
Pedido 5 finalizado: MensagemFinalizacao [idPedido=5, tempoTotalEntrega=52.1944]
Pedido 10 finalizado: MensagemFinalizacao [idPedido=10, tempoTotalEntrega=35.4722]
Pedido 8 finalizado: MensagemFinalizacao [idPedido=8, tempoTotalEntrega=55.5556]
Pedido 6 finalizado: MensagemFinalizacao [idPedido=6, tempoTotalEntrega=87.0833]
Pedido 11 finalizado: MensagemFinalizacao [idPedido=11, tempoTotalEntrega=55.5556]
Pedido 15 finalizado: MensagemFinalizacao [idPedido=15, tempoTotalEntrega=33.0]
Pedido 2 finalizado: MensagemFinalizacao [idPedido=2, tempoTotalEntrega=180.889]
Pedido 4 finalizado: MensagemFinalizacao [idPedido=4, tempoTotalEntrega=147.306]
Pedido 16 finalizado: MensagemFinalizacao [idPedido=16, tempoTotalEntrega=48.6944]
Pedido 9 finalizado: MensagemFinalizacao [idPedido=9, tempoTotalEntrega=126.611]
...
```

Fonte: A autora (2022).

3.4 ARQUITETURA PROPOSTA

A partir da arquitetura base, a arquitetura proposta foi desenvolvida e pretende mostrar escalabilidade e capacidade de tolerância a falhas, conceitos que não foram abordados em um primeiro momento com a *baseline*. Essa seção detalha quais diferenças a levaram a ter essas características.

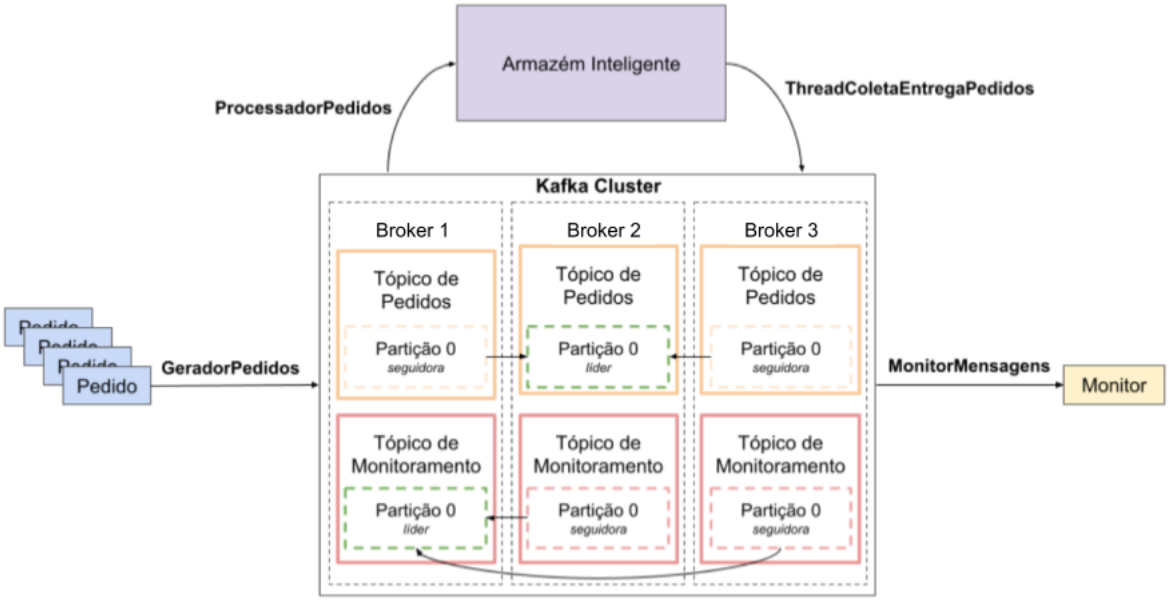
O principal motivo da arquitetura base não ser viável de utilização em um caso real é a sua tolerância a falhas. Como ela possui só um *broker* do Kafka, se este falha por algum motivo ou é reiniciado para manutenção, inúmeras mensagens de pedidos e de finalização serão perdidas pois não existe outro *broker* substituto para processá-las nesse tempo. Pensando nisso, a arquitetura proposta traz o conceito de replicação de *brokers*, já citado anteriormente na seção de fundamentos teóricos. A Figura 7 ilustra a organização dos módulos em conjunto com os componentes do Apache Kafka.

Para a arquitetura proposta, os módulos da arquitetura base continuaram realizando a mesma função, a diferença está na composição dos componentes do Kafka que agora conta com 1 *cluster* composto por 3 *brokers*, com os mesmos dois tópicos e com uma partição cada. Não foram necessárias adições de mais partições para um tópico neste caso de uso, visto que a ideia é que os pedidos não fiquem por muito tempo armazenados na partição, assim que eles são consumidos pelos consumidores, seu espaço já é liberado.

Como existem 3 *brokers* funcionais nessa arquitetura, é necessário que os tópicos façam a sua replicação em cada um, elegendo uma cópia de suas partições como líder e as restantes como seguidoras. A Figura 8 ilustra como é feita a escrita e leitura das mensagens com esta divisão, os produtores e consumidores de um tópico publicam e leem as mensagens na partição líder e, para ficarem sincronizadas e serem inseridas na lista de réplicas sincronizadas (ISR) garantindo redundância, as partições seguidoras mandam requisições para a líder para atualizar suas próprias mensagens.

Com isso, a tolerância a falhas do Kafka é colocada em prática. Se um dos

Figura 7 – Arquitetura proposta de processamento de pedidos utilizando Apache Kafka.



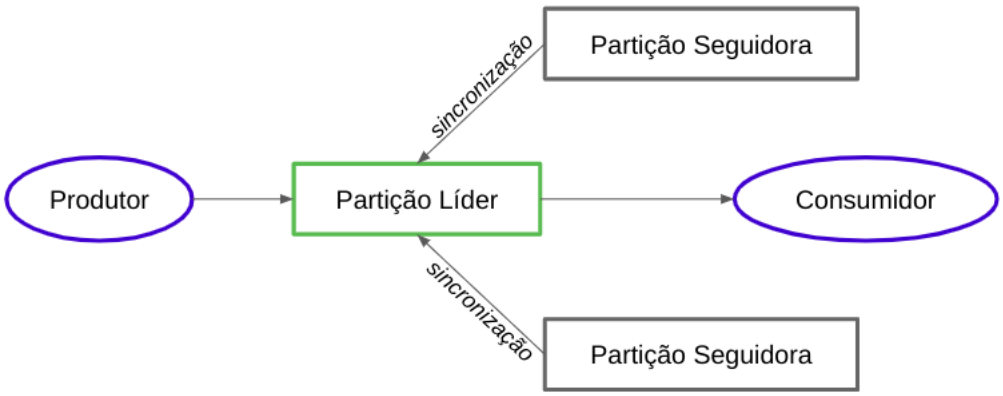
Fonte: A autora (2022).

brokers por algum motivo falha ou precisa ser reiniciado por algum tempo, as mensagens não são perdidas por conta desta redundância. Se o *broker* líder falha, um *broker* seguidor, que deve estar contido na ISR, é eleito novo líder e essa modificação é controlada e avisada aos outros pelo próprio *cluster* do Kafka, seus produtores e consumidores são redirecionados ao novo líder também. Se um *broker* seguidor falha, não é necessária uma troca, ele apenas é retirado pelo *cluster* da lista de réplicas que estão sincronizadas com a líder (ISR) e quando volta ao normal, se voltar, é inserido novamente depois que fizer a atualização das mensagens. Foram escolhidos 3 *brokers* para arquitetura, pois essa forma é fortemente recomendada pelo próprio Apache Kafka, dificilmente ocorre a perda de mensagens nesse caso.

A Figura 7, ilustra um exemplo de uma possível distribuição de lideranças de partição entre os *brokers*, o Tópico de Pedidos possui a sua partição líder no *Broker* 2 e o Tópico de Monitoramento possui sua partição líder no *Broker* 1. Essa distribuição é feita de maneira aleatória já na criação do *cluster* do Kafka. Sendo assim, o fluxo da arquitetura proposta funciona da seguinte forma:

- As mensagens dos pedidos são publicadas na partição do *Broker* 2 do Tópico de Pedidos pelo módulo GeradorPedidos;
- A cada 500ms, tempo padrão do *broker* Kafka, as partições do *Broker* 1 e 3

Figura 8 – Escrita e leitura de mensagens em uma partição líder e suas seguidoras.



Fonte: A autora (2022).

deste tópico fazem as suas atualizações com a líder;

- O módulo ProcessadorPedidos consome essas mensagens, também da partição líder do Tópico de Pedidos e faz seu processamento para cada uma delas, criando a ThreadColetaEntregaPedido;
- Cada ThreadColetaEntregaPedido, depois de realizar a sua função, faz seu papel de produtor do Tópico de Monitoramento, publicando as mensagens de finalização na sua partição líder, que se encontra no *Broker* 1;
- A cada 500ms as partições dos *Brokers* 2 e 3 do Tópico de monitoramento entram em sincronização com a líder;
- Por fim, o módulo de monitoramento MonitorMensagens faz a leitura das mensagens de seu respectivo tópico acessando a partição líder do *Broker* 1.

Além disso, a configuração de ACKS foi adicionada aos módulos produtores de mensagens. Inicialmente ambos possuem o valor 0, para que as mensagens sejam enviadas e logo confirmadas, mas a seguir, no capítulo de testes e resultados, outros valores serão testados em diferentes cenários e as diferenças de resultados serão indicadas.

3.5 CONSIDERAÇÕES FINAIS

Com a necessidade de desenvolver uma arquitetura capaz de realizar o processamento de pedidos entre um armazém inteligente e uma plataforma digital, utilizando dados que foram retirados de cenários reais, de forma síncrona, segura, confiável e tolerante a falhas, duas etapas de implementação foram realizadas neste trabalho.

A primeira etapa teve o objetivo de criar o fluxo básico cumprindo com a maioria dos objetivos propostos, desenvolvendo os módulos e suas conexões utilizando componentes do Apache Kafka, e criando uma arquitetura base funcional porém não tolerante a falhas. O desenvolvimento feito na segunda etapa, como uma evolução da primeira, supre essa necessidade bem como todos os objetivos propostos no início do projeto.

Com ambas as arquiteturas funcionais e com o fluxo de processamento de pedidos realizado do início ao fim, o próximo capítulo os testes que foram realizados, bem como um detalhamento sobre os diferentes cenários que possam ocorrer em uma utilização das mesmas no mundo real e os resultados obtidos, garantindo a utilidade da proposta para fins de pesquisa ou para o mercado.

4 TESTES E RESULTADOS OBTIDOS

Esse capítulo descreve a realização de diferentes casos de testes em diferentes cenários, exemplificando a utilização da arquitetura proposta, bem como os resultados obtidos que validaram o objetivo deste trabalho.

4.1 TESTES

Nessa seção são delineadas as métricas utilizadas nos casos de testes, as quais foram extraídas de casos reais de processamento de pedidos. Em seguida, serão apresentadas algumas características importantes dos dados que foram utilizados para a execução de todos os testes e por fim, serão detalhados os cenários de testes que as arquiteturas base e proposta foram testadas.

4.1.1 Métricas

Pensando nas possíveis falhas que podem ocorrer em um sistema com um grande volume de dados sendo processado em tempo real, e considerando o cenário de armazéns inteligentes e seu fluxo de mensagens, para este trabalho foi definido que a perda de mensagens é o fator que deve ser levado em consideração em um primeiro momento. Como as mensagens nesse ambiente são pedidos reais que pessoas fizeram em uma plataforma de vendas *online*, para um primeiro momento de testes deste trabalho, foi considerado que é mais problemático um pedido ser perdido, e o cliente não receber ou receber com atraso seus produtos, do que duplicado.

Assim, os testes que serão descritos a seguir levam em conta como métrica principal a quantidade de mensagens, para garantir que nenhuma mensagem é perdida no processo. A cada teste, todos os pedidos que entram no fluxo da arquitetura serão contabilizados e comparados à contagem de pedidos que chegam até o final do fluxo, no módulo de monitoramento.

4.1.2 Dados

Como já apresentado no capítulo anterior, os pedidos que serão utilizados para os testes das arquiteturas foram retirados de arquivos de simulações de (OLIVEIRA, G. S. *et al.*, 2022). Cada arquivo, representa uma simulação que já foi executada por este trabalho e por isso contém um número fixo de pedidos e outras informações sobre o ambiente do armazém inteligente. Para os testes das arquiteturas deste projeto, foram selecionados 4 arquivos de simulações, todas com duração de 180 minutos, em um armazém com uma frota heterogênea de 250 robôs (3 diferentes tipos) e com 52 estações de entrada, a única diferença entre elas é a quantidade de pedidos e a

quantidade de produtos de cada pedido, que é aleatória. A Tabela 1 mostra a relação entre cada arquivo de simulação e a quantidade de pedidos que cada uma possui.

Tabela 1 – Relação entre arquivos de simulação e sua quantidade de pedidos.

Arquivos	**Nome do Arquivo**	**Quantidade de Pedidos**
Arquivo 1	SIMU-i180-o6-r250-dHT03-d52-1.1	1080
Arquivo 2	SIMU-i180-o6-r250-dHT03-d52-2.1	1062
Arquivo 3	SIMU-i180-o6-r250-dHT03-d52-3.1	1140
Arquivo 4	SIMU-i180-o6-r250-dHT03-d52-4.1	1069

Fonte: A autora (2022).

Além disso, também é preciso definir para os testes qual a frequência de pedidos que chegam ao armazém, isto é, quantos pedidos por minuto o módulo GeradorPedidos deve publicar no seu respectivo tópico. As simulações feitas por (OLIVEIRA, G. S. *et al.*, 2022), que deram origem aos arquivos que foram utilizados, levam em consideração os armazéns inteligentes da Amazon, os quais recebem em média 6 pedidos por minuto. Por conta disso, a quantidade de pedidos de cada arquivo de simulação (que dura 180 minutos) é proporcional a essa frequência média.

Assim, partindo do objetivo de simular os testes nas arquiteturas de maneira que mais se aproxime do mundo real possível, a frequência de publicação de pedidos no tópico seguirá a mesma distribuição gama utilizada por (OLIVEIRA, G. S. *et al.*, 2022), com os parâmetros α = 6.34 e β = 1.0 (mais ou menos 6 pedidos por minuto).

Como já comentado nas seções anteriores, o Apache Kafka é uma plataforma que garante o processamento de um grande volume de dados recebidos frequentemente a cada segundo e por conta disso, a arquitetura implementada com esse sistema de mensageria, consegue lidar com uma frequência muito maior de pedidos por minuto do que ocorre no cenário da Amazon. Por conta disso, pensando na capacidade do Kafka e na possibilidade de existirem outros armazéns que recebem mais pedidos em menos tempo, além de testes com a frequência citada anteriormente, serão feitos e testados cenários de testes com diferentes cargas adicionais, para garantir a validação da arquitetura e também prever possíveis riscos que possam ocorrer na implementação da mesma em casos reais mais extremos.

Testes de carga são uma forma de validar a arquitetura de forma a simular cenários reais em que a frequência de entrada de mensagens é maior que a considerada "normal" (6 por minuto, nesse caso), permitindo a possibilidade de correção de falhas caso apareçam nesses cenários. Para aplicar esse conceito nos testes das arquiteturas deste trabalho, a frequência de publicação de pedidos no tópico (entrada de dados) pelo módulo GeradorPedidos será aumentada em 3 níveis, dividindo a frequência por 10 em cada um destes. Além disso, para garantir a mesma frequência de saída de dados, isto é, a simulação feita pelas *threads* ThreadColetaEntregaPedido e a frequência em que as mensagens de finalização são publicadas no tópico de monitoramento, a ve-

locidade de processamento será proporcional a que está sendo utilizada pela entrada de dados. A Tabela 2 ilustra a frequência de entrada de pedidos e sua relação com o tempo de processamento de um pedido aleatório.

Tabela 2 – Relação entre a frequência de entrada de pedidos e o tempo de processamento de um pedido aleatório pela ThreadColetaEntregaPedido.

Frequência de entrada de pedidos	**Tempo de processamento de um pedido aleatório**
1 pedido a cada 10 segundos	82,33 segundos
1 pedido a cada 1 segundo	8,23 segundos
1 pedido a cada 0,1 segundo	0,82 segundos
1 pedido a cada 0,01 segundo	0,08 segundos

Fonte: A autora (2022).

A partir dessas definições de dados para os testes, a seção a seguir detalha quais são os cenários de testes que serão testados em ambas as arquiteturas e posteriormente analisados com o objetivo de definir se a proposta deste trabalho é válida e utilizável para o mercado ou para futuras pesquisas.

4.1.3 Cenários de Testes

Para a execução dos testes que validam a proposta deste trabalho, foram estipulados alguns cenários para as duas arquiteturas desenvolvidas, levando em consideração as diferentes frequências de entrada de pedidos e falhas nos componentes do Kafka. Em cada caso de teste executado, as mensagens que passam pela arquitetura serão quantificadas e avaliadas no fim do fluxo de processamento, para que ao final deste trabalho os resultados possam ser mostrados e discutidos.

Um teste é composto de uma execução da arquitetura utilizando como entrada a lista de pedidos retirada de cada arquivo de simulação, o módulo GeradorPedidos pega os itens dessa lista em ordem e os envia ao tópico de pedidos seguindo uma das frequências estipuladas, cada mensagem percorre os outros módulos da arquitetura até chegar no MonitorMensagens e ser mostrado impresso na tela (o que significa que o pedido foi finalizado). Um teste termina quando todos os pedidos da lista são enviados ao tópico de pedidos e quando o último pedido que foi processado for finalizado, não necessariamente o último pedido da lista será o último pedido finalizado, pois o processamento destes nas *threads* ocorre de forma paralela.

Com o intuito de garantir a consistência dos resultados, foi executado um conjunto de 10 testes para cada lista de pedidos em cada cenário de testes. Por exemplo, a lista de pedidos do primeiro arquivo de simulação serviu de entrada para a arquitetura a cada um dos 10 testes executados separadamente, em um cenário. Sendo assim, as seções a seguir trazem um detalhamento dos cenários de testes para a arquitetura base e para a arquitetura proposta.

4.1.3.1 Cenários com a arquitetura base

A Tabela 3 mostra os cenários de testes em que a arquitetura base será testada, levando em consideração a frequência de entrada de pedidos e o *status* do *broker* durante o processamento.

Tabela 3 – Cenários de teste para a arquitetura base.

Cenários	**Frequência de entrada de pedidos**	***Status* do *Broker***
Cenário 1	1 pedido a cada 10 segundos	Sem falha
Cenário 2	1 pedido a cada 1 segundo	Sem falha
Cenário 3	1 pedido a cada 0,1 segundo	Sem falha
Cenário 4	1 pedido a cada 0,01 segundo	Sem falha
Cenário 5	1 pedido a cada 10 segundos	Com falha

Fonte: A autora (2022).

A mudança do *status* do *broker* indica uma falha que possa ocorrer na arquitetura, como já citado anteriormente, a qualquer momento um *broker* do Kafka pode se desconectar por determinado tempo, seja por uma questão de falha ou de manutenção. É importante rodar casos de teste com essa possibilidade pois é quase impossível manter uma arquitetura de processamento de *streaming* de dados em tempo real sem dar manutenção ao servidor com certa frequência.

Para simular a falha de um *broker* é necessário executar o comando de seu desligamento pelo terminal. Depois disso, o *cluster* do Kafka identifica a perda de conexão com este *broker* e tenta realizar eleição de um novo *broker* líder para continuar o processo, porém faz isso sem sucesso, já que a arquitetura base possui somente um desse componente.

Não é necessário criar outros casos de testes com falha no *broker* para a *baseline*, pois independente da velocidade em que os dados chegam, o processamento destes é interrompido a partir do momento em que um *broker* cai. Partindo deste cenário que a arquitetura proposta foi pensada, para ser tolerante a essas possíveis falhas de um *broker*.

4.1.3.2 Cenários com a arquitetura proposta

Os cenários nos quais serão executados os testes utilizando a arquitetura proposta estão apresentados pela Tabela 4. Foram levados em consideração cenários com e sem falha em um *broker* que contém a partição líder de um tópico, a frequência de pedidos que entram no fluxo e também o parâmetro ACKS passado aos produtores dos tópicos.

A diferença destes cenários para os cenários feitos para a arquitetura base, é que neste caso a arquitetura conta com 3 *brokers*, então em caso de falha as ações de reeleição da partição líder e redirecionamento dos produtores e consumidores

Tabela 4 – Cenários de teste para a arquitetura proposta.

Cenários	**Frequência de entrada de pedidos**	***Status* do *Broker***	**ACKS**
Cenário 1	1 pedido a cada 10 segundos	Sem falha	0
Cenário 2	1 pedido a cada 1 segundo	Sem falha	0
Cenário 3	1 pedido a cada 0,1 segundo	Sem falha	0
Cenário 4	1 pedido a cada 0,01 segundo	Sem falha	0
Cenário 5	1 pedido a cada 10 segundos	Com falha	0
Cenário 6	1 pedido a cada 1 segundo	Com falha	0
Cenário 7	1 pedido a cada 0,1 segundo	Com falha	0
Cenário 8	1 pedido a cada 0,01 segundo	Com falha	0
Cenário 9	1 pedido a cada 10 segundos	Com falha	1
Cenário 10	1 pedido a cada 1 segundo	Com falha	1
Cenário 11	1 pedido a cada 0,1 segundo	Com falha	1
Cenário 12	1 pedido a cada 0,01 segundo	Com falha	1
Cenário 13	1 pedido a cada 10 segundos	Com falha	*all*
Cenário 14	1 pedido a cada 1 segundo	Com falha	*all*
Cenário 15	1 pedido a cada 0,1 segundo	Com falha	*all*
Cenário 16	1 pedido a cada 0,01 segundo	Com falha	*all*

Fonte: A autora (2022).

para esta, podem resultar na perda de mensagens. Além disso foram levados em consideração os diferentes valores para o parâmetro ACKS adicionado aos produtores dos tópicos de pedidos e de monitoramento, cada opção selecionada gera um impacto direto quando um *broker* que contém a partição líder de um tópico falha.

Para os cenários em que não ocorre falha no *broker*, o ACKS dos produtores da arquitetura foram deixados no valor padrão zero, como não ocorre o desligamento do *broker* e consequentemente nenhuma mensagem corre o risco de ser perdida, não foi necessário realizar testes com outros valores.

Já nos outros cenários, foi estipulado que a falha sempre deve ocorrer em um *broker* que contém a partição líder, pois é nesse caso que ocorre a realocação dos demais componentes e que as mensagens podem ser perdidas. Nos casos em que um *broker* que possui uma partição seguidora falha, ele é apenas retirado da ISR e isso não afeta o processamento de dados do Kafka.

Sendo assim, para realizar os testes em situações que ocorrem a perda de um *broker* que contém a partição líder de um dos tópicos, é necessário acompanhar os componentes do Kafka por um sistema de monitoramento disponibilizado por terceiros, tornando possível a visualização de qual partição foi eleita líder (o Kafka faz a eleição aleatória quando o tópico é criado). O sistema de monitoramento utilizado neste trabalho é o Conduktor (CONDUKTOR, 2022), uma plataforma gratuita que contém várias funções de visualização de dados do Apache Kafka. Depois de identificado o *broker* que deve falhar, o mesmo comando utilizado nos testes da arquitetura base deve ser utilizado para derrubá-lo.

4.1.4 Execução

Os conjuntos de testes para cada lista de pedidos de cada arquivo de simulação foram executados subindo as arquiteturas e as configurações do Apache Kafka localmente, em um computador com as seguintes características: processador Intel ® Core (TM) i5-825oU 1.6GHz, 16GB de memória RAM e sistema operacional Ubuntu 10.04 LTS.

A seção a seguir apresenta quais foram os resultados obtidos da execução e também traz uma discussão e comparação entre os mesmos.

4.2 RESULTADOS OBTIDOS

Essa seção apresenta os resultados de testes para cada cenário descrito na seção anterior. Além disso, esses resultados foram comparados e analisados para realizar uma discussão sobre a validação e implementação da arquitetura proposta em casos reais de uso.

Como mencionado anteriormente, cada teste compreende a uma execução do fluxo da arquitetura recebendo como entrada uma lista de um dos arquivos de simulação. Com a intenção de garantir consistência de resultados, para cada lista foram executados 10 testes em cada cenário proposto para as arquiteturas. As mensagens foram monitoradas e quantificadas ao longo do fluxo da arquitetura em 4 pontos principais, os quais colocam os dados em maiores riscos caso haja a falha de um *broker*. A Figura 9 ilustra a arquitetura proposta e indica onde ficam esses pontos chaves, sendo eles: a persistência e a leitura de mensagens no tópico de pedidos e a persistência e leitura de mensagens no tópico de monitoramento.

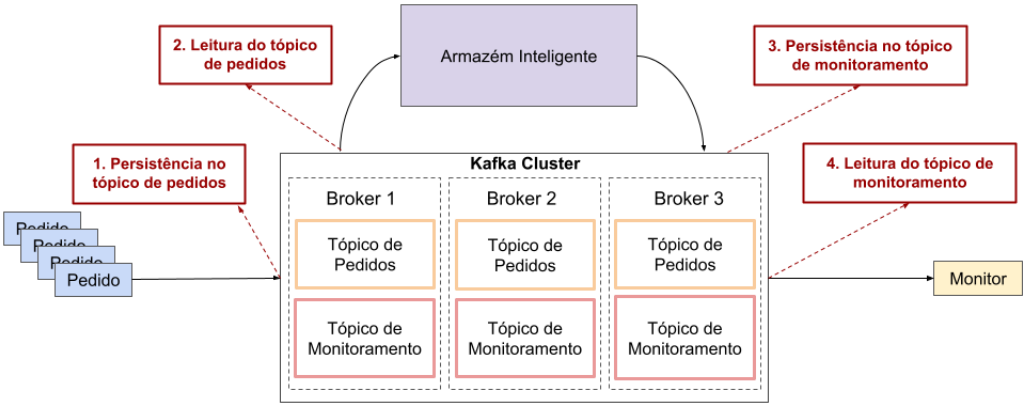
4.2.1 Resultados Referentes à Arquitetura Base

A Tabela 5 mostra os resultados dos testes de cada cenário proposto para a arquitetura base, listados na Tabela 3. Primeiramente são apresentados os cenários em que não ocorre falha no *broker* (de 1 a 4) e depois o cenário 5, o qual essa falha ocorre.

Como pode-se perceber pelos resultados dos casos de teste de 1 a 4 da arquitetura base, em todos os pontos de monitoramento de mensagens, a quantidade monitorada em todos os pontos chave se manteve a mesma, indicando que não ocorreu perda de mensagem em nenhum conjunto de testes feitos. Nesses cenários não ocorre falha do *broker* e por este motivo é esperado que o Kafka consiga lidar com o volume de dados testado em qualquer frequência de entrada de pedidos, até mesmo as utilizadas nos testes de carga (cenários 2, 3 e 4).

O cenário de teste 5 trouxe resultados ruins para a arquitetura base, como pode-se observar pela Tabela 5, quase 62% das mensagens foram perdidas. Isso aconteceu

Figura 9 – Arquitetura proposta com indicação dos pontos chave de troca de mensagens.



Fonte: A autora (2022).

Tabela 5 – Resultados dos cenários de testes da arquitetura base.

	Mensagens de entrada	**Mensagens persistidas no tópico de pedidos**	**Mensagens consumidas do tópico de pedidos**	**Mensagens persistidas no tópico de monitoramento**	**Mensagens consumidas do tópico de monitoramento**
			Cenários 1 a 4		
Arquivo 1	1080	1080	1080	1080	1080
Arquivo 2	1062	1062	1062	1062	1062
Arquivo 3	1140	1140	1140	1140	1140
Arquivo 4	1069	1069	1069	1069	1069
			Cenário 5		
Arquivo 1	1080	682	682	668	668

Fonte: A autora (2022).

pois o processamento de dados da arquitetura foi interrompido assim que ocorreu a falha no *broker*, como este é o único componente do *cluster*, não houve realocação das partições líderes dos tópicos e por conta disso as mensagens restantes não chegaram ao final do processo. Por conta disso, não foi necessário realizar mais testes com os outros arquivos de simulações.

Os testes com a arquitetura base mostraram que esta não tem nenhum tipo de tolerância a falhas e por isso se fez necessário a adaptação desta arquitetura para criar a arquitetura proposta, adicionando componentes que trazem essa característica essencial que o Apache Kafka proporciona.

4.2.2 Resultados Referentes à Arquitetura Proposta

As tabelas apresentadas nessa seção mostram os resultados dos testes de cada cenário criado para a arquitetura proposta, listados na Tabela 4. Primeiramente são apresentados os cenários em que não ocorre falha no *broker* (de 1 a 4), depois os cenários em que essa falha ocorre, divididos de acordo com o valor do ACKS.

Tabela 6 – Resultados dos cenários de testes 1 a 4 da arquitetura proposta.

	Mensagens de entrada	**Mensagens persistidas no tópico de pedidos**	**Mensagens consumidas do tópico de pedidos**	**Mensagens persistidas no tópico de monitoramento**	**Mensagens consumidas do tópico de monitoramento**
			Cenários 1 a 4		
Arquivo 1	1080	1080	1080	1080	1080
Arquivo 2	1062	1062	1062	1062	1062
Arquivo 3	1140	1140	1140	1140	1140
Arquivo 4	1069	1069	1069	1069	1069

Fonte: A autora (2022).

Os resultados mostrados na Tabela 6 comprovam que apesar de ser adicionada uma complexidade na arquitetura, se não ocorre falha em um *broker* que contém a partição líder, todas as mensagens enviadas à arquitetura chegam até o final do fluxo sem nenhuma perda, até mesmo para as frequências de testes de carga adicionais (cenários 2, 3 e 4).

Como podemos analisar pela Tabela 7, que mostra os resultados dos cenários em que o ACKS está configurado em 0, o Cenário 5 é o único em que nenhuma mensagem de pedido é perdida. Neste cenário, a frequência de envio de pedidos é a que foi considerada como "normal"(1 pedido a cada 10 segundos). Isso garante que a arquitetura proposta consegue lidar com essa carga, mesmo com possíveis falhas no *broker* que contém a partição líder, por conta disso, ela já poderia ser aplicada para realizar a conexão entre a plataforma de *e-commerce* da Amazon e seu armazém inteligente, por exemplo, validando o objetivo deste trabalho.

Já nos cenários 6 a 8, que indicam os testes com maiores cargas, pode-se perceber que ocorre a perda de mensagens. Com o valor do ACKS configurado em 0, não se tem garantia de persistência dos dados e, quando o *broker* falha, no tempo de troca de liderança da partição os pedidos são perdidos, cada vez em maior quantidade à medida que a frequência de chegada de mensagens aumenta.

A Tabela 8 traz as porcentagens de mensagens perdidas nos cenários de 6 a 8. No cenário 6, apenas 0,009% de mensagens foram perdidas; no cenário 7, em média, 0,021%; e, por fim, a porcentagem média de mensagens perdidas no cenário 8 é de 0,72%.

Tabela 7 – Resultados dos cenários de testes 5 a 8 da arquitetura proposta.

	Mensagens de entrada	**Mensagens persistidas no tópico de pedidos**	**Mensagens consumidas do tópico de pedidos**	**Mensagens persistidas no tópico de monitoramento**	**Mensagens consumidas do tópico de monitoramento**
			Cenários 5		
Arquivo 1	1080	1080	1080	1080	1080
Arquivo 2	1062	1062	1062	1062	1062
Arquivo 3	1140	1140	1140	1140	1140
Arquivo 4	1069	1069	1069	1069	1069
			Cenário 6		
Arquivo 1	1080	1080	1080	1080	1080
Arquivo 2	1062	1061,9	1061,9	1061,9	1061,9
Arquivo 3	1140	1140	1140	1140	1140
Arquivo 4	1069	1069	1069	1069	1069
			Cenário 7		
Arquivo 1	1080	1079,9	1079,9	1079,9	1079,9
Arquivo 2	1062	1061,8	1061,8	1061,7	1061,7
Arquivo 3	1140	1139,8	1139,7	1139,7	1139,7
Arquivo 4	1069	1068,8	1068,8	1068,8	1068,8
			Cenário 8		
Arquivo 1	1080	1074,5	1074,5	1073,9	1073,9
Arquivo 2	1062	1055,4	1055,4	1053,1	1053,1
Arquivo 3	1140	1131,5	1131,5	1130	1130
Arquivo 4	1069	1063,5	1063,5	1062,5	1062,5

Fonte: A autora (2022).

Como uma tentativa de impedir a perda de dados que aconteceram nos cenários em que o ACKS é 0, nos cenários de teste 9 a 12 o ACKS foi configurado em 1, para que pelo menos a partição líder confirme que persistiu a mensagem no seu tópico.

A partir dos resultados representados na Tabela 9, é interessante pontuar que nos cenários 10 e 11, os quais possuem uma carga maior de chegada de pedidos, não ocorreram perdas quando acontece a falha no *broker*, isso se deve ao valor do ACKS. Entretanto, no cenário 12 (que possui a maior frequência), apesar da partição líder persistir a mensagem, entre o tempo que seu *broker* falha e ocorre a troca de liderança e o tempo que leva até uma partição seguidora se atualizar com a líder, mensagens ainda são perdidas nesse processo, mas em menor quantidade. Isso é apresentado na Tabela 10, que a porcentagem média de mensagens perdidas, em média 0,03%, é bem menor nesse cenário quando comparado a do cenário 8 analisado anteriormente.

Por fim, os resultados dos cenários de 13 a 16, detalhados na Tabela 11, mostram que o valor do ACKS mais seguro pra lidar com a perda de dados é o *all*, pois nesses cenários, independente da carga, nenhuma mensagem é perdida. A garantia de que todas as mensagens são persistidas em todas as réplicas das partições, ga-

Tabela 8 – Porcentagem de mensagens perdidas nos cenários 6, 7 e 8 da arquitetura proposta.

Arquivo de Simulação	Porcentagem de mensagens perdidas
Cenário 6	
Arquivo 1	0,000%
Arquivo 2	0,009%
Arquivo 3	0,000%
Arquivo 4	0,000%
Cenário 7	
Arquivo 1	0,009%
Arquivo 2	0,028%
Arquivo 3	0,026%
Arquivo 4	0,019%
Cenário 8	
Arquivo 1	0,565%
Arquivo 2	0,838%
Arquivo 3	0,877%
Arquivo 4	0,608%

Fonte: A autora (2022).

Tabela 9 – Resultados dos cenários de testes 9 a 12 da arquitetura proposta.

	Mensagens de entrada	**Mensagens persistidas no tópico de pedidos**	**Mensagens consumidas do tópico de pedidos**	**Mensagens persistidas no tópico de monitoramento**	**Mensagens consumidas do tópico de monitoramento**
Cenários 9, 10 e 11					
Arquivo 1	1080	1080	1080	1080	1080
Arquivo 2	1062	1062	1062	1062	1062
Arquivo 3	1140	1140	1140	1140	1140
Arquivo 4	1069	1069	1069	1069	1069
Cenário 12					
Arquivo 1	1080	1079,8	1079,8	1079,8	1079,8
Arquivo 2	1062	1061,8	1061,8	1061,7	1061,7
Arquivo 3	1140	1139,6	1139,6	1139,6	1139,6
Arquivo 4	1069	1068,6	1068,6	1068,6	1068,6

Fonte: A autora (2022).

rante que os produtores só enviem a próxima mensagem quando a anterior é salva, mesmo que ele tenha que esperar a troca de líderes quando um *broker* falha.

4.2.3 Considerações Finais

Com todos os testes executados em diferentes cenários para cada arquitetura pode-se perceber a importância da configuração de componentes do Kafka que lidam com a tolerância a falhas. Os resultados dos cenários de teste da arquitetura base

Tabela 10 – Porcentagem de mensagens perdidas no cenário 12 da arquitetura proposta.

Arquivo de Simulação	Porcentagem de mensagens perdidas
Cenário 12	
Arquivo 1	0,019%
Arquivo 2	0,028%
Arquivo 3	0,035%
Arquivo 4	0,037%

Fonte: A autora (2022).

Tabela 11 – Resultados dos cenários de testes 13 a 16 da arquitetura proposta.

	Mensagens de entrada	Mensagens persistidas no tópico de pedidos	Mensagens consumidas do tópico de pedidos	Mensagens persistidas no tópico de monitoramento	Mensagens consumidas do tópico de monitoramento
			Cenários 13 a 16		
Arquivo 1	1080	1080	1080	1080	1080
Arquivo 2	1062	1062	1062	1062	1062
Arquivo 3	1140	1140	1140	1140	1140
Arquivo 4	1069	1069	1069	1069	1069

Fonte: A autora (2022).

mostram que, apesar dela conseguir lidar com todas as frequências de chegada de pedidos, em cenários que acontecem falha no seu único *broker*, a arquitetura se mostra inutilizável, pois o processamento de dados é interrompido a partir deste ponto e todos as próximas mensagens recebidas pelo fluxo são perdidas.

Assim, com a realização dos testes em cima da arquitetura proposta, que possui tolerância a falhas, dentre os 16 casos pensados, apenas em 4 deles houve a perda de mensagens, sendo que todos estes são casos de testes de carga. Destes casos, 3 possuíam o valor do ACKS configurado em 0 e, por conta disso, na troca de líderes de partição algumas mensagens foram perdidas devido à não confirmação de persistência. No último caso que ocorreu a perda de mensagens, o valor do ACKS estava configurado em 1, mas mesmo assim dados foram perdidos por conta da alta velocidade de chegada de pedidos na arquitetura. Para zerar a quantidade de mensagens perdidas pelo fluxo, nos últimos cenários de teste o parâmetro ACKS foi configurado em *all*, isso garantiu que todas as mensagens que entraram na arquitetura foram processadas e chegaram até o final da mesma, independente da frequência de chegada de pedidos.

A arquitetura proposta se mostrou uma ótima opção para lidar com casos de processamento de pedidos em casos reais, considerando as métricas de frequência de envio de mensagens e a sua organização tolerante a falhas. Caso a arquitetura seja utilizada para fins de mercado, é recomendada a utilização do parâmetro ACKS em 1

caso a frequência de chegada de mensagens seja de até 1 pedido por segundo, ou utilizar o ACKS em *all* para velocidades mais rápidas. O capítulo a seguir apresenta a conclusão deste trabalho e possíveis trabalhos futuros que possam ser feitos com base nesse.

5 CONCLUSÃO

O objetivo deste trabalho de conclusão de curso foi a proposta de uma arquitetura de *software* que utiliza o sistema de mensageria Apache Kafka e que seja capaz de lidar com o grande volume de dados de pedidos feitos em uma plataforma *online* de compra e venda de produtos e realizar a sua conexão com o armazém inteligente responsável pela sua coleta e entrega, o que foi demonstrado neste documento.

Foi desenvolvida uma arquitetura base (*baseline*), com uma estrutura simples que consegue realizar essa conexão, mas nos seus testes, pode-se perceber que esta não possui nenhum tipo de tolerância a falhas assim servindo somente como uma prova de conceito para a arquitetura principal. Com a evolução da *baseline* aplicando estruturas capazes de lidar com possíveis falhas que possam ocorrer, foi desenvolvida a arquitetura proposta para o objetivo mencionado. Foram utilizados os componentes principais do Apache Kafka para realizar toda a conexão entre os módulos desenvolvidos para compor o fluxo de processamento dos pedidos. Além disso, diversos testes sobre a proposta foram executados em diferentes cenários que simulam possíveis acontecimentos no ambiente em que a arquitetura poderá ser inserida, esses testes trouxeram resultados positivos e validam a sua utilização para fins de pesquisa ou para o mercado.

Por fim, todo o código deste projeto está disponível no repositório (GITHUB, 2022), o Apêndice A deste documento traz mais detalhes, caso seja necessária modificação ou adição de novos componentes e funcionalidades.

5.1 TRABALHOS FUTUROS

Esse projeto propôs uma arquitetura funcional e utilizável em casos reais para fazer a conexão entre uma plataforma de *e-commerce* e um armazém inteligente, processando os pedidos em tempo real. A arquitetura pode ser ampliada e novas funções e componentes do Apache Kafka podem ser inseridos a fim de expandir a sua capacidade de uso. Dentre as possíveis adições ao projeto, as seguintes foram consideradas, mas não implementadas devido a algumas limitações.

A arquitetura pode ser inserida dentro do próprio armazém inteligente, fazendo a conexão entre as mensagens que são trocadas neste ambiente, diversos componentes de *hardware* devem se comunicar dentro de um armazém, para garantir máxima eficiência de coleta e entrega de pedidos. Os robôs se comunicam entre si e com as tecnologias de monitoramento de espaço ou de gestão do armazém, essa comunicação deve gerar a todo momento uma inúmera quantidade de dados, a expansão da arquitetura proposta que utiliza um sistema de *streaming* de mensagens para lidar com esse caso, tornaria essa comunicação mais eficiente.

Outra proposta seria expandir a ponta de monitoramento da arquitetura para sis-

temas que criam um *dashboard* de visualização de dados em tempo real. A arquitetura pode ser conectada com esses sistemas e durante todo o processo de recebimento de pedidos, gestores do *e-commerce* e do próprio armazém podem visualizar a analisar todos os cenários de dados que passam pelo fluxo de processamento contribuindo com mais eficiência para detectar erros, caso aconteçam.

REFERÊNCIAS

AMAZON WEB SERVICES. **Pub/Sub Messaging**. [*S.l.*], 2021. Disponível em: https://aws.amazon.com/pt/pub-sub-messaging/. Acesso em: 27 set. 2021.

APACHE KAFKA. **Documentation**. [*S.l.*], 2021. Disponível em: https://kafka.apache.org/documentation/. Acesso em: 20 set. 2021.

BOYSEN, Nils; KOSTER, René de; WEIDINGER, Felix. Warehousing in the e-commerce era: A survey. *In*: 2. v. 277, p. 396–411.

CONDUKTOR. **Streamline Apache Kafka**. [*S.l.*], 2022. Disponível em: https://www.conduktor.io/. Acesso em: 10 nov. 2022.

GEEST, Maarten van; TEKINERDOGAN, Bedir; CATAL, Cagatay. Smart Warehouses: Rationale, Challenges and Solution Directions. **Applied Sciences**, v. 12, n. 1, 2022. ISSN 2076-3417.

GITHUB. **Kafka Architecture For Smart Warehouses Java**. [*S.l.*], 2022. Disponível em: https://github.com/luanadf/kafka-architecture-for-smart-warehouses-java. Acesso em: 20 dez. 2022.

GOEL, Shivkumar; M PRABHANATH, Nair; T VARGHESE, John. Building a GUI Application for Viewing and Searching Apache Kafka Messages. *In*: 6. v. 5, p. 1229–1231.

HIRAMAN, Bhole Rahul; VIRESH M., Chapte; ABHIJEET C., Karve. A Study of Apache Kafka in Big Data Stream Processing. *In*: 2018 International Conference on Information , Communication, Engineering and Technology (ICICET). [*S.l.*: *s.n.*], 2018. P. 1–3.

JAVA. **Java Oracle**. [*S.l.*], 2021. Disponível em: https://www.java.com/pt-BR/. Acesso em: 2 out. 2021.

LIU, Jialei; LIEW, Soung-Yue; OOI, Boon Yaik; QIN, Donghong. Dynamic Order-Based Scheduling Algorithms for Automated Retrieval System in Smart Warehouses. **IEEE Access**, v. 9, p. 158340–158352, 2021.

LOURENÇO, Luan Lucas; OLIVEIRA, George; MÉA PLENTZ, Patricia Della; RÖNING, Juha. Achieving reliable communication between Kafka and ROS through bridge codes. *In*: 2021 20th International Conference on Advanced Robotics (ICAR). [*S.l.*: *s.n.*], 2021. P. 324–329.

OLIVEIRA, George; PLENTZ, Patricia D. M.; CARVALHO, Jônata Tyska. Multi-Constrained Voronoi-Based Task Allocator for Smart-Warehouses. *In*: 2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI). [*S.l.*: *s.n.*], 2021. P. 515–520.

OLIVEIRA, George S.; RÖONING, Juha; CARVALHO, Jônata T.; PLENTZ, Patricia D. M. Efficient Task Allocation in Smart Warehouses with Multi-Delivery Stations and Heterogeneous Robots. *In*: 2022 25th International Conference on Information Fusion (FUSION). [*S.l.*: *s.n.*], 2022. P. 1–8.

SHAPIRA, Gwen; PALINO, Todd; SIVARAM, Rajini; PETTY, Krit. **Kafka: The Definitive Guide**: Real-Time Data and Stream Processing at Scale. 2. ed. [*S.l.*]: O'Reilly Media, Inc., 2021.

SHARVARI, T; SOWMYANAG, K. A study on Modern Messaging Systems- Kafka, RabbitMQ and NATS Streaming. **ArXiv**, abs/1912.03715, 2019.

SHI, Yinbin; HU, Biao; HUANG, Ran. Task Allocation and Path Planning of Many Robots with Motion Uncertainty in a Warehouse Environment. *In*: 2021 IEEE International Conference on Real-time Computing and Robotics (RCAR). [*S.l.*: *s.n.*], 2021. P. 776–781.

COPPOLA, Daniela. **Annual retail e-commerce sales growth worldwide from 2017 to 2026**. [*S.l.*], 2022. Disponível em: `https://kafka.apache.org/documentation/`. Acesso em: 22 ago. 2022.

TANG, J.; ZHOU, Y.; TANG, T.; WENG, D.; XIE, B.; YU, L.; ZHANG, H.; WU, Y. A Visualization Approach for Monitoring Order Processing in E-Commerce Warehouse. **IEEE Transactions on Visualization amp; Computer Graphics**, IEEE Computer Society, Los Alamitos, CA, USA, v. 28, n. 01, p. 857–867, jan. 2022. ISSN 1941-0506.

XU, Panpan; MEI, Honghui; CHEN, Wei. ViDX: Visual Diagnostics of Assembly Line Performance in Smart Factories. **IEEE Transactions on Visualization and Computer Graphics**, v. 23, p. 1–1, jan. 2016.

APÊNDICE A – CÓDIGO FONTE

O código fonte de todas as implementações feitas para o desenvolvimento deste trabalho constam no repositório (GITHUB, 2022). Para a execução do código é necessária a instalação das seguintes ferramentas: Java 8 (JAVA, 2021) e Apache Kafka (KAFKA, 2021).

A.1 EXECUÇÃO

Para realizar a execução das arquiteturas, antes de executar os arquivos Java, é necessário ter um *Kafka Cluster* com um ou mais *Kafka Brokers* executando, localmente ou na nuvem. O trabalho foi desenvolvido e testado subindo essas instâncias localmente, executando em dois terminais diferentes os seguintes comandos:

- Para executar a instância do *Kafka Cluster*:

```
~/kafka/bin/zookeeper-server-start.sh ~/kafka/config/zookeeper.properties
```

- Para executar uma instância do *Kafka Broker*:

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

APÊNDICE B – ARTIGO

Proposta de uma Arquitetura de Processamento de Dados Para Armazéns Inteligentes Utilizando Apache Kafka

Luana D. Fronza[1]
[1]Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística

Campus Universitário – Florianópolis, SC - Brasil

fronza.lu@gmail.com

Abstract. *Smart warehouses with robots that handle the management and separation of products, have achieved a solution for optimizing the flow of delivery of orders previously carried out only by people. The objective of this project is to propose a software architecture capable of handling the large flow of orders being placed in real time on sales platforms, using Apache Kafka [Kafka, 2021]. Test scenarios were created with different frequencies of arrival of requests and organizations of Kafka components. The results appreciated that the developed architecture supports the processing of requests in scenarios with and without failure in the Apache Kafka components and, in addition, it handles high frequencies of message arrivals without loss.*

Resumo. *Os armazéns inteligentes equipados com robôs que lidam com a extração e separação de produtos, se tornaram uma solução para a otimização do fluxo de entrega de pedidos antes realizados somente por pessoas. O objetivo desse projeto é propor uma arquitetura de software capaz de lidar com o grande fluxo de pedidos sendo feitos em tempo real em plataformas de vendas, utilizando o Apache Kafka [Kafka, 2021]. Foram criados cenários de testes com diferentes frequências de chegada de pedidos e de organizações dos componentes do Kafka. Os resultados obtidos mostraram que a arquitetura desenvolvida suporta o processamento de pedidos em cenários com e sem falha nos componentes do Apache Kafka e, além disso, lida com altas frequências de chegada de mensagens sem perda.*

1. Introdução

Em 2021, as vendas de comércio eletrônico totalizaram aproximadamente 5,2 trilhões de dólares em todo o mundo e, além disso, prevê-se que este número cresça 56% nos próximos anos [Coppola, 2022]. Com esse crescimento, novos modelos de armazéns surgiram com o objetivo de lidar com o grande volume de vendas e as necessidades especiais de vendedores e clientes envolvidos no e-commerce. De acordo com [Boysen, 2019], os armazéns tradicionais traziam problemas que os novos modelos de armazenamento conseguem resolver, são eles: pedidos pequenos, grande variedade, cronogramas de entrega apertados e cargas de trabalho variadas. Os chamados armazéns inteligentes são compostos por tecnologias de automatização e estão organizados de forma a se adequar com as necessidades desse novo estilo de comércio. Apesar das mudanças citadas nos armazéns serem majoritariamente em relação ao *hardware* desse sistema, se faz necessário apontar que esses modelos automatizados de armazenamento devem se interconectar com componentes de *software* capazes de consumir, armazenar e processar o grande volume de dados que esse sistema produz de forma síncrona, segura, confiável e tolerante a falhas.

As plataformas de processamento de mensagens são ditas como solução para os problemas de manipulação de grandes massas de dados, elas têm por objetivo transferir dados de um sistema para outro, para que esses sistemas se preocupem com os dados e não com a sua transmissão e compartilhamento. O Apache Kafka, plataforma de streaming de dados desenvolvida no Linkedin, é a tecnologia considerada pela maioria das principais empresas de Internet como uma das melhores

alternativas [Kafka, 2021] para lidar com os milhões de dados que suas aplicações geram. É descrito como uma "plataforma de distribuição de streaming" que fornece um sistema de armazenamento de dados de forma durável, consistente e em ordem, que permite leitura e escrita de registros de forma determinística. Além disso, esse sistema permite inúmeras configurações diferentes que trazem proteções adicionais contra falhas e melhoria de desempenho, por exemplo.

Levando em consideração os novos modelos automatizados de armazéns utilizados pelas plataformas de e-commerce, a gigantesca quantidade de dados gerados por estes e as necessidades de componentes de *software* que devem acompanhar essa mudança, o produto deste trabalho é uma arquitetura de processamento de dados em tempo real, que garantiu a comunicação e troca de mensagens entre uma plataforma de compras *online* e um armazém inteligente. A arquitetura utiliza o Apache Kafka como principal ferramenta para lidar com o fluxo de pedidos vindos da plataforma digital e enviá-los ao armazém, que produz informações sobre os pedidos que vão sendo finalizados a todo momento. Como um dos objetivos do trabalho é simular esse fluxo de forma funcional e eficiente, mantendo a semelhança com o que acontece no mundo real, os trabalhos realizados em [Oliveira, 2021] e [Oliveira, 2022] foram utilizados como complemento da arquitetura.

O trabalho propõe a execução e análise de duas arquiteturas que utilizam o Kafka, a primeira, mais simples, serviu como prova de conceito para a segunda, a arquitetura proposta. Ambas as arquiteturas passaram por casos de testes que consideram a quantidade de mensagens trocadas em cenários com e sem falha nos seus componentes. Com o intuito de validar a arquitetura proposta desenvolvida, foram criados diferentes cenários de testes que levam em consideração diferentes frequências de chegada de pedidos e também possíveis falhas em componentes essenciais do Kafka.

2. Fundamentação Teórica

Os primeiros pontos exploram o funcionamento de um armazém inteligente bem como os componentes que estão inseridos nesse ambiente. Depois, são apresentadas definições importantes sobre sistemas de mensageria e fluxo de dados que se fazem essenciais para a compreensão do último ponto, sobre o Apache Kafka.

2.1. Armazéns Inteligentes

Novos modelos de armazéns, chamados armazéns inteligentes, são instalações que visam aumentar a qualidade geral do serviço, a produtividade e a eficiência de todo o processo de entrega de um pedido, minimizando custos e falhas, com o objetivo de suprir as novas necessidades dos comércios digitais que atendem diretamente às demandas dos clientes finais no segmento *business-toconsumer* (B2C).

2.1. Apache Kafka

Apache Kafka é um sistema que segue o padrão de mensageria de publicação/assinatura, é de código aberto e projetado com o objetivo de fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados que são gerados em tempo real. Para um melhor entendimento da plataforma como um todo, as próximas seções trazem um embasamento dos principais conceitos utilizados pelo Kafka.

2.2.1 Tópico

Os tópicos servem como se fossem pastas em um sistemas de arquivos. São neles que mensagens são armazenadas e consumidas pelos seus componentes conectados. No Kafka, um tópico pode possuir zero, um ou muitos produtores e consumidores ligados, que podem escrever e consumir mensagens de acordo com a frequência que for necessária. Os dados são escritos em um tópico apenas de forma anexada e são lidos em uma fila (do início ao fim), além disso, independente do tamanho, as mensagens podem ser armazenadas pelo tempo que for considerado adequado ao desenvolvedor da infraestrutura.

2.2.2 Partição

Manter um tópico restrito a operar em somente uma máquina colocaria um grande limite na capacidade de dimensionamento da plataforma. Como algumas das principais vantagens do Kafka são a sua escalabilidade e capacidade de tolerância a falhas, esse sistema permite a partição de tópicos. Diferentes servidores podem armazenar partições distintas ou cópias de uma mesma partição, garantindo redundância e escalabilidade.

2.2.3 Produtores e Consumidores

Produtores são os componentes que gravam mensagens em um ou mais tópicos do Kafka e consumidores são os que leem essas mensagens. Para alcançar a escalabilidade que a plataforma promete, os produtores e consumidores são totalmente dissociados uns dos outros, isto é, esses componentes se comunicam via um intermediário (tópico) e possuem desacoplamento espacial e temporal. No desacoplamento espacial, o produtor/consumidor não conhece ou não precisa conhecer a identidade do consumidor/produtor; e no desacoplamento temporal, o produtor e o consumidor podem ter independência do momento de existência.

2.2.4 *Broker*

Um *Broker* é um único servidor do Kafka, ele é responsável tanto pela recepção de mensagens de produtores quanto pelas solicitações de busca de partições e leitura de mensagens dos consumidores.

2.2.5 *Cluster*

Os *brokers* são organizados em *clusters*. Logo, um *cluster* é composto por um ou mais *brokers* que trocam informações de gerenciamento entre si. Neste contexto, um dos *brokers* é eleito como líder e administra os demais atribuindo partições e monitorando falhas. Quando uma partição de um tópico é replicada, uma de suas cópias é eleita como líder e as demais são as seguidoras. Essa replicação permite redundância de mensagens na partição, de modo que um de seus seguidores pode assumir a liderança se houver uma falha do *broker* líder. Todos os produtores desta partição devem escrever mensagens em seu líder, porém os consumidores podem ler as mensagens de qualquer uma de suas réplicas.

2.2.6 Réplicas Sincronizadas e ACKS

A ISR (*In-Sync Replicas*), ou réplicas sincronizadas, é uma lista que contém as partições que estão sincronizadas entre si, com a mesma quantidade de mensagens. Uma ISR sempre conterá a partição líder e zero ou mais seguidoras. Um seguidor é considerado "em sincronia" se está completamente atualizado com o líder em algum ponto de um período pré definido de tempo, chamado de "replica.lag.time.max.ms", que por padrão é 500ms, isto é, estar em dia com o líder é ter uma cópia exata de suas mensagens. O ACKS (Acknowledgements) ou reconhecimento, é uma configuração que pode ser passada ao produtor de mensagens. Esse parâmetro denota o número de *brokers* que devem receber a mensagem antes dela ser considerada persistida com sucesso no tópico, suporta 3 valores: 0, 1 e *all*.

3. Trabalhos Relacionados

Essa seção apresenta alguns dos trabalhos mais recentes e relevantes que estão relacionados à área de armazéns inteligentes e sistemas de mensageria em tempo real, além de fazer um comparativo sobre como essas literaturas se aproximam ou se diferem da proposta deste trabalho.

Em [Oliveira, 2021], os autores propõem uma arquitetura de *software* de um armazém inteligente que integra MRTA e MPP e lida com tempo e energia como restrições principais, e além disso, os autores propõem também o MCVB-TA (*Multi-Constrained Voronoi-Based Task Allocator*), um mecanismo que utiliza o diagrama de Voronoi para definir uma série de condições iniciais que são usadas como entrada dessa arquitetura. Essa estrutura é composta por 6 módulos: Pedidos, Organizador de Tarefas, Estimador de Custo, Alocador de Tarefas, Planejador de Caminhos e Monitor. Os resultados mostram a eficiência do algoritmo proposto, que reduz o tempo e a energia da execução de tarefas quando comparado ao alocador normal. Como outra versão do trabalho citado anteriormente, os autores propõem em [Oliveira, 2022] uma estratégia de alocação para MRFS que considera muitas das restrições de um armazém, como a quantidade de estações de entrega, a paralelização de tarefas entre robôs e a constância de recebimento de pedidos. Os resultados mostraram que o DoNe-CPTA pode reduzir até 45% do custo das rotas de robôs e 24% do tempo para atender aos pedidos.

É importante reconhecer que o processamento de pedidos em armazéns inteligentes também possui grande influência na eficiência de entrega do e-commerce. Todo esse fluxo gera uma massa de dados de streaming em tempo real, na qual dados de pedidos podem ser afetados devido a situações imprevisíveis, por isso uma análise visual dessas mensagens é um conceito chave a ser abordado. A abordagem de visualização proposta por [Tang, 2022] considera esses padrões assíncronos de chegada e saída de dados. O sistema de análise visual OrderMonitor desenvolvido neste trabalho auxilia gerentes de armazéns a melhorar a eficiência do processamento de pedidos em tempo real, se baseando no streaming de dados de eventos que ocorrem nessa instalação. Apesar desse trabalho propor uma solução que processa e monitora dados de um armazém em tempo real, os autores não utilizam um sistema de mensageria que lida com esse tipo de streaming, mas sim um banco de dados que armazena os eventos em ordem.

Apesar da maioria dos trabalhos, recomendarem o Kafka e o inserirem em vários cenários de processamento de dados em tempo real, até o momento da finalização da etapa de pesquisas deste trabalho e do conhecimento da autora, somente uma literatura utiliza o Apache Kafka no contexto de armazéns inteligentes. Em [Lourenço, 2021] é feito um estudo sobre a maneira mais simples de conexão entre o Kafka e ROS (*Robot Operating System*), uma plataforma que está sendo amplamente utilizada para aplicações robóticas, incluindo os armazéns inteligentes. Os resultados

dessa pesquisa foram satisfatórios e apontam que o Kafka pode ser facilmente integrado com o ROS e um sistema de armazém inteligente.

4. Arquiteturas do Kafka para Integração com Sistemas de Armazéns Inteligentes

Essa seção apresenta o desenvolvimento da arquitetura base que deu origem à arquitetura proposta por este trabalho.

4.1. Dados

Para o desenvolvimento e testes das arquiteturas, foram utilizados conjuntos de dados de simulações de coleta e entrega de pedidos feitas em [Oliveira, 2022]. Cada arquivo disponibilizado possui uma série de informações e corresponde a uma simulação já executada e que se aproxima do mundo real. Para os objetivos deste projeto, apenas dados relevantes foram extraídos de cada arquivo, sendo estes a lista de pedidos e o tempo até cada pedido ser finalizado pela simulação.

4.2. Arquitetura Base

A arquitetura base ou baseline do projeto, foi pensada e desenvolvida para ser apenas uma prova do conceito da proposta deste trabalho. A Figura 1 ilustra como os seus componentes estão conectados e organizados. Foram desenvolvidos quatro módulos de serviço que se conectam utilizando componentes do Kafka, sendo estes: um *cluster* com um *broker* e dois tópicos com uma partição cada, inicialmente sem utilizar os conceitos de replicação. O primeiro tópico, chamado “Tópico de Pedidos” tem o módulo "GeradorPedidos"conectado como produtor e o módulo “ProcessadorPedidos” como consumidor de mensagens. O segundo, “Tópico de Monitoramento” contém as threads de coleta e entrega de pedidos como produtoras de dados e o módulo de monitoramento que os consome.

4.3. Arquitetura Proposta

A partir da arquitetura base, a arquitetura proposta foi desenvolvida e pretende mostrar escalabilidade e capacidade de tolerância a falhas, conceitos que não foram abordados em um primeiro momento com a baseline. Essa seção detalha quais diferenças a levaram a ter essas características.

O principal motivo da arquitetura base não ser viável de utilização em um caso real é a sua tolerância a falhas. Como ela possui só um *broker* do Kafka, se este falha por algum motivo ou é reiniciado para manutenção, inúmeras mensagens de pedidos e de finalização serão perdidas pois não existe outro *broker* substituto para processá-las nesse tempo. Pensando nisso, a arquitetura proposta traz o conceito de replicação de *brokers*, já citado anteriormente na seção de fundamentos teóricos. A Figura 2 ilustra a organização dos módulos em conjunto com os componentes do Apache Kafka.

Para a arquitetura proposta, os módulos da arquitetura base continuaram realizando a mesma função, a diferença está na composição dos componentes do Kafka que agora conta com 1 *cluster* composto por 3 *brokers*, com os mesmos dois tópicos e com uma partição cada. Não foram necessárias adições de mais partições para um tópico neste caso de uso, visto que a ideia é que os pedidos não fiquem por muito tempo armazenados na partição, assim que eles são consumidos pelos consumidores, seu espaço já é liberado.

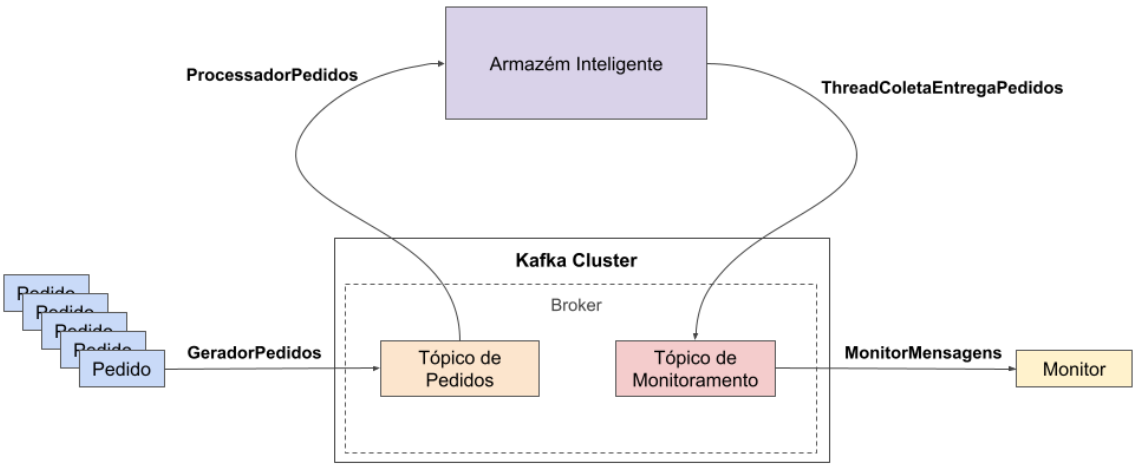


Figura 1. Arquitetura Base

Como existem 3 *brokers* funcionais nessa arquitetura, é necessário que os tópicos façam a sua replicação em cada um, elegendo uma cópia de suas partições como líder e as restantes como seguidoras. Com isso, a tolerância a falhas do Kafka é colocada em prática. Se um dos *brokers* por algum motivo falha ou precisa ser reiniciado por algum tempo, as mensagens não são perdidas por conta desta redundância. Se o *broker* líder falha, um *broker* seguidor, que deve estar contido na ISR, é eleito novo líder e essa modificação é controlada e avisada aos outros pelo próprio *cluster* do Kafka, seus produtores e consumidores são redirecionados ao novo líder também. Se um *broker* seguidor falha, não é necessária uma troca, ele apenas é retirado pelo *cluster* da lista de réplicas que estão sincronizadas com a líder (ISR) e quando volta ao normal, se voltar, é inserido novamente depois que fizer a atualização das mensagens. Foram escolhidos 3 *brokers* para arquitetura, pois essa forma é fortemente recomendada pelo próprio Apache Kafka, dificilmente ocorre a perda de mensagens nesse caso.

5. Testes e Resultados

Essa seção descreve a realização de diferentes casos de testes em diferentes cenários, exemplificando a utilização da arquitetura proposta, bem como os resultados obtidos que validaram o objetivo deste trabalho.

5.1. Testes

Como as mensagens nesse ambiente são pedidos reais que pessoas fizeram em uma plataforma de vendas *online*, para um primeiro momento de testes deste trabalho, foi considerado que é mais problemático um pedido ser perdido, e o cliente não receber ou receber com atraso seus produtos, do que duplicado. Assim, os testes que serão descritos a seguir levam em conta como métrica principal a quantidade de mensagens, para garantir que nenhuma mensagem é perdida no processo. A cada teste, todos os pedidos que entram no fluxo da arquitetura serão contabilizados e comparados à contagem de pedidos que chegam até o final do fluxo, no módulo de monitoramento.

5.2. Dados

Como já apresentado no capítulo anterior, os pedidos que serão utilizados para os testes das arquiteturas foram retirados de arquivos de simulações de [Oliveira, 2022]. Cada

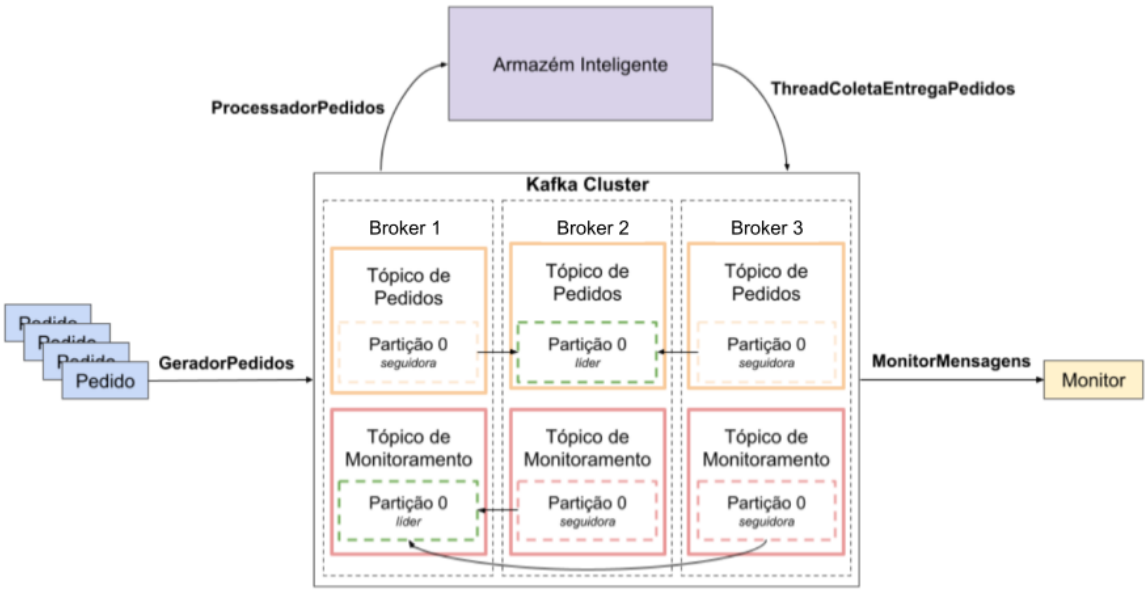


Figura 2. Arquitetura Proposta

arquivo, representa uma simulação que já foi executada por este trabalho e por isso contém um número fixo de pedidos e outras informações sobre o ambiente do armazém inteligente. Para os testes das arquiteturas deste projeto, foram selecionados 4 arquivos de simulações, todas com duração de 180 minutos, em um armazém com uma frota heterogênea de 250 robôs (3 diferentes tipos) e com 52 estações de entrada, a única diferença entre elas é a quantidade e tamanho de pedidos. A Tabela 1 mostra a relação entre cada arquivo de simulação e a quantidade de pedidos que cada uma possui.

Arquivos	Nome do Arquivo	Quantidade de Pedidos
Arquivo 1	SIMU-i180-o6-r250-dHT03-d52-1.1	1080
Arquivo 2	SIMU-i180-o6-r250-dHT03-d52-2.1	1062
Arquivo 3	SIMU-i180-o6-r250-dHT03-d52-3.1	1140
Arquivo 4	SIMU-i180-o6-r250-dHT03-d52-4.1	1069

Tabela 1. Relação entre arquivos de simulação e sua quantidade de pedidos

Além disso, também é preciso definir para os testes qual a frequência de pedidos que chegam ao armazém, isto é, quantos pedidos por minuto o módulo GeradorPedidos deve publicar no seu respectivo tópico. As simulações feitas por [Oliveira, 2022], que deram origem aos arquivos que foram utilizados, levam em consideração os armazéns inteligentes da Amazon, os quais recebem em média 6 pedidos por minuto. Por conta disso, a quantidade de pedidos de cada arquivo de simulação (que dura 180 minutos) é proporcional a essa frequência média. Assim, partindo do objetivo de simular os testes nas arquiteturas de maneira que mais se aproxime do mundo real possível, a frequência de publicação de pedidos no tópico seguirá a mesma distribuição gama utilizada por [Oliveira, 2022], com os parâmetros $\alpha = 6.34$ e $\beta = 1.0$ (mais ou menos 6 pedidos por minuto). Além de testes com a frequência citada anteriormente, serão feitos e testados cenários de testes de diferentes cargas adicionais, para garantir a validação da arquitetura e também prever possíveis riscos que possam ocorrer na implementação da mesma em casos reais mais extremos.

5.3. Cenários com a arquitetura proposta

Os cenários nos quais serão executados os testes utilizando a arquitetura proposta estão apresentados pela Tabela 2. Foram levados em consideração cenários com e sem falha em um *c,*que contém a partição líder de um tópico, a frequência de pedidos que entram no fluxo e também o parâmetro ACKS passado aos produtores dos tópicos.

Cenários	Frequência de entrada de pedidos	Status do Broker	ACKS
Cenário 1	1 pedido a cada 10 segundos	Sem falha	0
Cenário 2	1 pedido a cada 1 segundo	Sem falha	0
Cenário 3	1 pedido a cada 0,1 segundo	Sem falha	0
Cenário 4	1 pedido a cada 0,01 segundo	Sem falha	0
Cenário 5	1 pedido a cada 10 segundos	Com falha	0
Cenário 6	1 pedido a cada 1 segundo	Com falha	0
Cenário 7	1 pedido a cada 0,1 segundo	Com falha	0
Cenário 8	1 pedido a cada 0,01 segundo	Com falha	0
Cenário 9	1 pedido a cada 10 segundos	Com falha	1
Cenário 10	1 pedido a cada 1 segundo	Com falha	1
Cenário 11	1 pedido a cada 0,1 segundo	Com falha	1
Cenário 12	1 pedido a cada 0,01 segundo	Com falha	1
Cenário 13	1 pedido a cada 10 segundos	Com falha	*all*
Cenário 14	1 pedido a cada 1 segundo	Com falha	*all*
Cenário 15	1 pedido a cada 0,1 segundo	Com falha	*all*
Cenário 16	1 pedido a cada 0,01 segundo	Com falha	*all*

Tabela 2. Cenários para a arquitetura proposta

5.4. Resultados Obtidos

Com a realização dos testes em cima da arquitetura proposta, que possui tolerância a falhas, dentre os 16 casos pensados, apenas em 4 deles houve a perda de mensagens, sendo que todos estes são casos de testes de carga. Destes casos, 3 possuíam o valor do ACKS configurado em 0 e, por conta disso, na troca de líderes de partição algumas mensagens foram perdidas devido à não confirmação de persistência. No último caso que ocorreu a perda de mensagens, o valor do ACKS estava configurado em 1, mas mesmo assim dados foram perdidos por conta da alta velocidade de chegada de pedidos na arquitetura. Para zerar a quantidade de mensagens perdidas pela arquitetura, nos últimos cenários de teste o parâmetro ACKS foi configurado em all, isso garantiu que todas as mensagens que entraram na arquitetura foram processadas e chegaram até o final da mesma, independente da frequência de chegada de pedidos. A arquitetura proposta se mostrou uma ótima opção para lidar com casos de processamento de pedidos em casos reais, considerando as métricas de frequência de envio de mensagens e a sua organização tolerante a falhas. Caso a arquitetura seja utilizada para fins de mercado, é recomendada a utilização do parâmetro ACKS em 1 caso a frequência de chegada de mensagens seja de até 1 pedido por segundo, ou utilizar o ACKS em all para velocidades mais rápidas.

5. Conclusão

O objetivo deste trabalho de conclusão de curso foi a proposta de uma arquitetura de *software* que utiliza o sistema de mensageria Apache Kafka e que seja capaz de lidar com o grande volume de dados de pedidos feitos em uma plataforma *online* de compra e venda de produtos e

realizar a sua conexão com o armazém inteligente responsável pela sua coleta e entrega, o que foi demonstrado neste documento.

Foi desenvolvida uma arquitetura base (baseline), com uma estrutura simples que consegue realizar essa conexão, mas nos seus testes, pode-se perceber que esta não possui nenhum tipo de tolerância a falhas assim servindo somente como uma prova de conceito para a arquitetura principal. Com a evolução da baseline aplicando estruturas capazes de lidar com possíveis falhas que possam ocorrer, foi desenvolvida a arquitetura proposta para o objetivo mencionado. Foram utilizados os componentes principais do Apache Kafka para realizar toda a conexão entre os módulos desenvolvidos para compor o fluxo de processamento dos pedidos. Além disso, diversos testes sobre a proposta foram executados em diferentes cenários que simulam possíveis acontecimentos no ambiente em que a arquitetura poderá ser inserida, esses testes trouxeram resultados positivos e validam a sua utilização para fins de pesquisa ou para o mercado.

References

[Kafka, 2021] Apache Kafka. Documentation. [S.l.], 2021. Disponível em: https://kafka.apache.org/documentation/. Acesso em: 20 set. 2021.

[Coppola, 2022] Coppola, Daniela. Annual retail e-commerce sales growth worldwide from 2017 to 2026. [S.l.], 2022. Disponível em: https://kafka.apache.org/documentation/. Acesso em: 22 ago. 2022.

[Boysen, 2019] Boysen, Nils; Koster, René de; Weidinger, Felix. Warehousing in the e-commerce era: A survey. In: 2. v. 277, p. 396–411.

[Oliveira, 2021] Oliveira, George; Plentz, Patricia D. M.; Carvalho, Jônata Tyska. Multi-Constrained Voronoi-Based Task Allocator for Smart-Warehouses. In: 2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI). [S.l.: s.n.], 2021. P. 515–520.

[Oliveira, 2022] Oliveira, George S.; Roning, Juha; Carvalho, Jônata T.; Plentz, Patricia D. M. Efficient Task Allocation in Smart Warehouses with Multi-Delivery Stations and Heterogeneous Robots. In: 2022 25th International Conference on Information Fusion (FUSION). [S.l.: s.n.], 2022. P. 1–8.

[Tang, 2022] Tang, J.; Zhou, Y.; Tang, T.; Weng, D.; Xie, B.; Yu, L.; Zhang, H.; Wu, Y. A Visualization Approach for Monitoring Order Processing in E-Commerce Warehouse. IEEE Transactions on Visualization amp; Computer Graphics, IEEE Computer Society, Los Alamitos, CA, USA, v. 28, n. 01, p. 857–867, jan. 2022. ISSN 1941-0506

[Lourenço, 2021] Lourenço, Luan Lucas; Oliveira, George; Méa Plentz, Patricia Della; Roning, Juha. Achieving reliable communication between Kafka and ROS through bridge codes. In: 2021 20th International Conference on Advanced Robotics (ICAR). [S.l.: s.n.], 2021. P. 324–329.