PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Arquitetura de Software Distribuído

José Nunes Rodrigues de Assis

LOGÍSTICA BASEADA EM MICROSSERVIÇOS

Belo Horizonte
2022

José Nunes Rodrigues de Assis

LOGÍSTICA BASEADA EM MICROSSERVIÇOS

Trabalho de Conclusão de Curso de Especialização em Arquitetura de Software Distribuído como requisito parcial à obtenção do título de especialista.

Orientador(a): Pedro Alves de Oliveira

Belo Horizonte
2022

RESUMO

Este trabalho descreve uma proposta de arquitetura para o contexto de uma empresa de transporte de mercadorias que deseja evoluir sua área de TI através de novas soluções. Nesta arquitetura proposta é levado em consideração a integração de sistemas legados com as novas soluções. Sendo que um dos elementos principais desta arquitetura se baseia na adoção de um serviço de mensageria, Apache Kafka. Com a utilização deste serviço foi possível construir uma arquitetura desacoplada, com baixo custo e fácil integração. Não somente o Kafka, mas a maioria das tecnologias presentes na arquitetura proposta são de código aberto. Por fim, foi desenvolvido uma POC para poder realizar a validação de alguns requisitos funcionais e não funcionais em cima de 3 casos de uso. Todos testes feitos foram descritos para compravar a viabilidade de tal proposta.

Palavras-chave: arquitetura de software, sistema distribuido, serviço de mensageria, dados.

SUMÁRIO

1 Apresentação

Com o aumento da demanda de transporte de mercadorias no decorrer nos últimos anos, decorrente de serviços online, cresceu o número de empresas especializadas em entregas. Com o advento da pandemia acabou-se intensificando ainda mais estas demandas e com isto fazendo-se necessário otimizar processos de entrega, como definição de rotas e gestão de todas entregas.

A etapa de entrega para os clientes é crucial, podendo determinar a satisfação dos clientes. Muitos empreendedores recorrem a contratação de empresas especializadas em entrega afim de garantir mais resultados [1], trazendo mais agilidade e segurança.

1.1 Problema

Automatizar todos os processos de entrega realizados por ela, visando aprimorar os processos de apuração, conferência e faturamento e manter um nível de remuneração adequado;

Implementar integrações de seus sistemas com os de suas parceiras, de modo a propiciar que as entregas possam ser realizadas em parceria, em uma ou mais etapas do processo. Essas integrações requerem que os sistemas atuais sejam adaptados e novos componentes sejam incorporados visando a uma maior abertura, que será baseada na arquitetura orientada a serviços;

Utilizar geotecnologias em todos os procesos que envolvam localização, de forma a facilitar a identificação e atualização de informações relativas às entregas agendadas e realizadas;

Tornar viável o uso de todas as tecnologias da informação e softwares necessários para atender às demandas dos clientes, fornecedores e parceiros, conforme definido neste documento.

1.2 Objetivo do trabalho

Este trabalho tem como objetivo apresentar uma proposta de arquitetura para o problema apontado na seção anterior. Sendo que deve-se considerar a existência de soluções legadas e integrá-las as novas soluções. Para isto, este trabalho apresenta uma forma de realizar tais integrações com as soluções legadas e se aprofunda mais nos processos após a integração feita. Para validar a arquitetura e seus elementos foi realizado um POC que foca principalmente na recuperação de dados dos

sistemas tanto legados quanto os novos afim de disponibilizá-los para serem feitas análises e gerar relatórios para fins de auxílio de tomada de decisão. Ou seja, o foco deste trabalho, para realização da POC, é em dados.

1.3 Definições e Abreviaturas

CDC – *Change Data Capture*

2 Especificação da Solução

Esta seção descreve os requisitos comtemplados neste projeto arquitetural, divididos em dois grupos: funcionais e não funcionais.

2.1 Requisitos Funcionais

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	Atender, de forma seletiva (por perfil) a clientes, fornecedores e colaboradores;	B	A
RF02	Obter e manter informações de clientes, fornecedores, depósitos e mercadorias	A	A
RF03	Desenhar, analisar e acompanhar todos os processos de atendimento aos clientes existentes na empresa	A	A
RF04	Gerar relatórios para auxiliar na tomada de decisão a nível de negócio	B	A
RF05	Ingestão automatizada de eventos transacionais no data lake	B	A
RF06	Orquestrar execução de jobs permitindo uma fácil reexecução em caso de falhas	A	A
RF07	Monitorar execução de processo para identificação de falhas	B	B
RF08	Evoluir os dados no lake em um esquema de zonas até sua disponibilização para consulta	A	A
RF09	Alertar sobre falhas de identificadas em processos monitorados	B	A
RF10	Automatização de deploys através de CI/CD	M	B

*B=Baixa, M=Média, A=Alta.

2.2 Requisitos Não Funcionais

ID	Descrição	Prioridade B/M/A
RNF01	Possuir características de aplicação distribuída: abertura, portabilidade, uso de recursos de rede;	A
RNF02	Ser modular e componentizado, utilizando orientação a serviços;	A
RNF03	Ser de fácil implantação e utilização;	B
RNF04	Ser hospedado em nuvem híbrida, sendo a forma de hospedagem documentada;	A
RNF05	Suportar ambientes web e móveis;	B
RNF06	Possuir interface responsiva. Compatível com diversas resoluções de tela.	B
RNF07	Apresentar bom desempenho. Não tendo um tempo de resposta superior a 5 segundos.	B
RNF08	Apresentar boa manutenibilidade. Adicionar ou alterar recursos do serviço não pode levar mais que uma sprint para estar implantado em ambiente produtivo.	A
RNF09	Ser testável em todas as suas funcionalidades;	A
RNF10	Ser recuperável (resiliente) no caso da ocorrência de erro. O usuário não pode ficar sem ter conhecimento se as tarefas foram executadas ou não.	A
RNF11	Ser desenvolvido utilizando recursos de gestão de configuração, com integração contínua.	B
RNF12	Utilizar APIs ou outros recursos adequados para consumo de serviços;	A
RNF13	Estar disponível em horário integral (24 H), sete dias por semana;	A
RNF14	Utilizar GitHubActions para CI/CD	B
RNF15	Notificar a respeito das tarefas executadas e seus estados em caso de falha.	A

2.3 Restrições Arquiteturais

- R1: A solução deve possuir baixo custo
- R2: Facilidade de uso pelos colaboradores e associados, com possibilidades de acesso multicamadas, em tecnologias web e mobile.
- R3: O projeto deverá apresentar fácil manutenção dos componentes, não gerando altos custos futuros.
- R4: Os padrões tecnológicos definidos pelo modelo arquitetural proposto deverão atender plenamente aos objetivos de integração dos sistemas legados, de acordo com a descrição dos módulos, a fim de viabilizar que essa integração ocorra com baixo acoplamento e sem a necessidade de substituição dos ativos existentes.
- R5: robustez. Deve-se fazer uso de middlewares adequados, suportados por: Remote Proce ure Call (RPC), serviços de mensageria, web sockets, protocolos como HTTP, TCP/IP, SOAP, Rest, GraphQL. Com preferência para soluções de baixo custo e grande disseminação.

2.4 Mecanismos Arquiteturais

Análise	***Design***	**Implementação**
Persistência	ORM	SQLAlchemy
Front end	Single Page Application	React
Back end	APIs Restful	FastAPI
Integração	Publish / Subscribe	Apache Kafka
Log do sistema	Libs nativas de log da linguagem	Logger Python
Teste de Software	Testes unitários	Pytest
Deploy	CI/CD	GitHub Actions
Análise de Dados	Data Warehouse	Druid
Relatórios	Visualização de Dados	Apache Superset
Orquestração de cargas de trabalho	Orquestração	Apache Airflow
Observabilidade	Monitoramento	Grafana

3 Modelagem Arquitetural

Nesta seção será apresentado a modelagem arquitetural da solução proposta de forma a permitir seu completo entendimento visando à implementação da prova de conceito (seção 5).

3.1 Macroarquitetura

A Figura 1 mostra a especificação o diagrama geral da solução proposta, com todos os módulos, tanto os já existentes como os novos. Basicamente os sistemas já existentes irão se conectar, na maior parte, por meio de um serviço de *Publisher/Subscriber*, no caso sendo o proposto o *Apache Kafka*. O intuito de ser tal tecnologia para conectar novas soluções com as existentes permite desacoplamento entre as soluções. Sendo o Apache Kafka uma tecnologia de código aberto que não agregaria nenhum custo e reconhecida pelo sua capacidade de lidar com alto volume de dados em alta performance com poucos recursos.

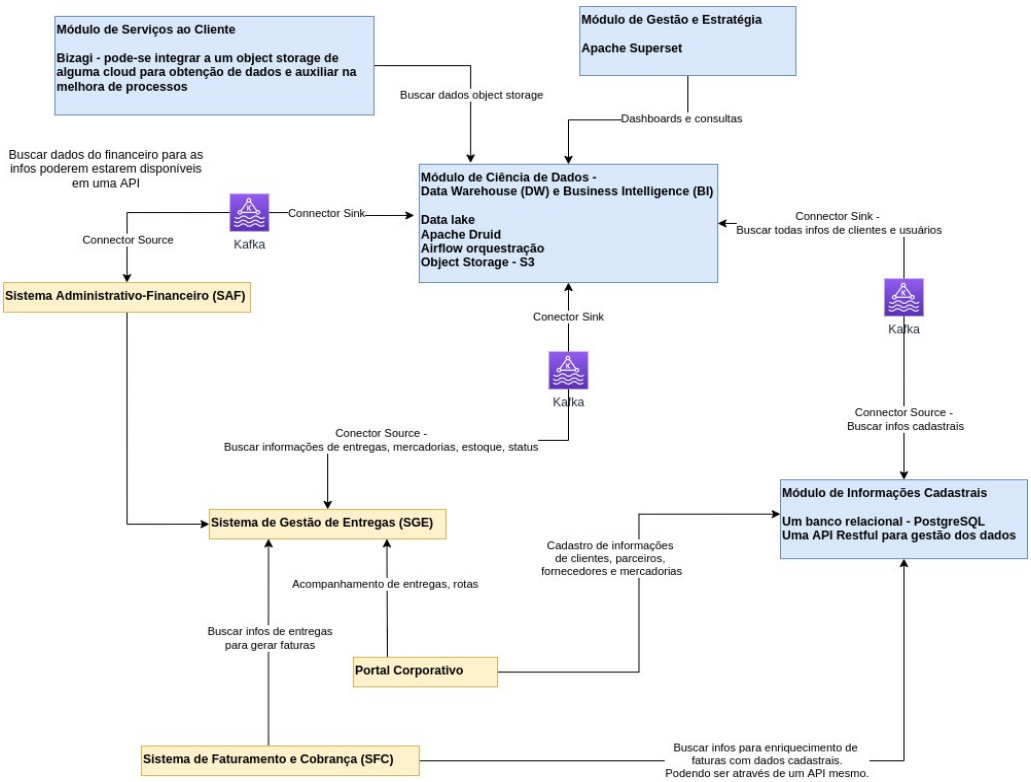

Figura 1: Visão Geral da Solução. Fonte: Autor.

Quando não utilizado o Kafka para realizar esta conexão entre sistemas poderá ser utilizado uma chamada direta via *HTTP* na API de outros serviços. Isto acontece o Portal Corporativo ao módulo de Informações Cadastrais, onde pode ser feito uma comunicação direta na API do módulo.

Para não ser necessário realizar modificações nas aplicações já existentes para obtenção de dados, pode-se utilizar de dispositivos como CDC (Change Data Capture). À medida que dados forem inseridos, modificados ou excluídos das bases dos sistemas legados estes dados seriam capturado e enviados para um tópico do Kafka. Sendo que para cada tabela haveria um tópico distinto. À partir destes tópicos haveria outros conectores no tópico responsáveis por ingerir estes eventos no Data Lake criado.

O objetivo do *data lake* é poder obter dados de diferentes fontes, de diferentes formatos e estruturas e com isto evoluir os mesmos para se obter informações revelantes para auxiliarem na tomada de decisão com relatórios gerados. O data lake seria constituído de zonas que representariam o quanto evoluído os dados se encontram. Sendo que na camada mais externa estariam os dados utilizados para geração de relatórios após serem feitos todos processos de tratamento e agregação necessários.

Com o uso do *data lake* ele ainda poderia auxiliar justamente o módulo de serviços ao cliente. Em que poderia se conectar diretamente e extrair métricas.

3.2 Descrição Resumida dos Casos de Uso

UC01 – NOME DO CASO DE USO 01	
Descrição	Change Data Capture de eventos transacionais
Atores	Kafka Connect Source
Prioridade	A
Requisitos associados	RF02, RNF01, RNF07, RNF08
Fluxo Principal	Conectores Kafka Source serão responsáveis por buscar eventos dos bancos de dados relacionais e enviar para os tópicos do kafka. https:// www.confluent.io/hub/debezium/debezium-connector-postgresql

UC01 – NOME DO CASO DE USO 02	
Descrição	Ingestão de dados no data lake de forma automatizada
Atores	Kafka Connect Sink
Prioridade	A
Requisitos associados	RF05, RNF01, RNF04, RNF07, RNF03
Fluxo Principal	Conectores Kafka serão responsáveis por buscar eventos nos tópicos e ingerir no data lake. Sendo eventos que foram enviados por microserviços ou por Kafka Connect Source.

UC01 – NOME DO CASO DE USO 03	
Descrição	Acompanhamento de execução de cargas de trabalho
Atores	Desenvolvedor
Prioridade	A
Requisitos associados	RF06, RF07, RNF05, RNF08, RNF10, RNF15
Fluxo Principal	O desenvolvedor, após logado na ferramenta de orquestração de workloads, poderá acompanhar processos que foram ou estão em execução para evolução dos dados no data lake. Podendo reprocessar ou encerrar processos.

3.3 Visão Lógica

Esta seção mostra a especificação dos diagramas da solução proposta, com todos os seus componentes, propriedades e interfaces. Na subseção a seguir é apresentado o Diagrama de Componentes.

3.3.1 Diagrama de Componentes

Na Figura 2 é apresentado um diagrama de componenetes da arquitetura proposta. O objetivo é realizar uma integração entre os sistemas legados com os novos através de um serviço de mensageria. No caso, para mensageria foi escolhido o Apache Kafka dado sua performance e escalabilidade já conhecidas no mercado, além de ser uma solução open source e não agregando custos. Utilizar o Apache Kafka para esta finalidade traz desacoplamento entre as soluções, além da escalabilidade e tolerância a falha que o serviço possui.

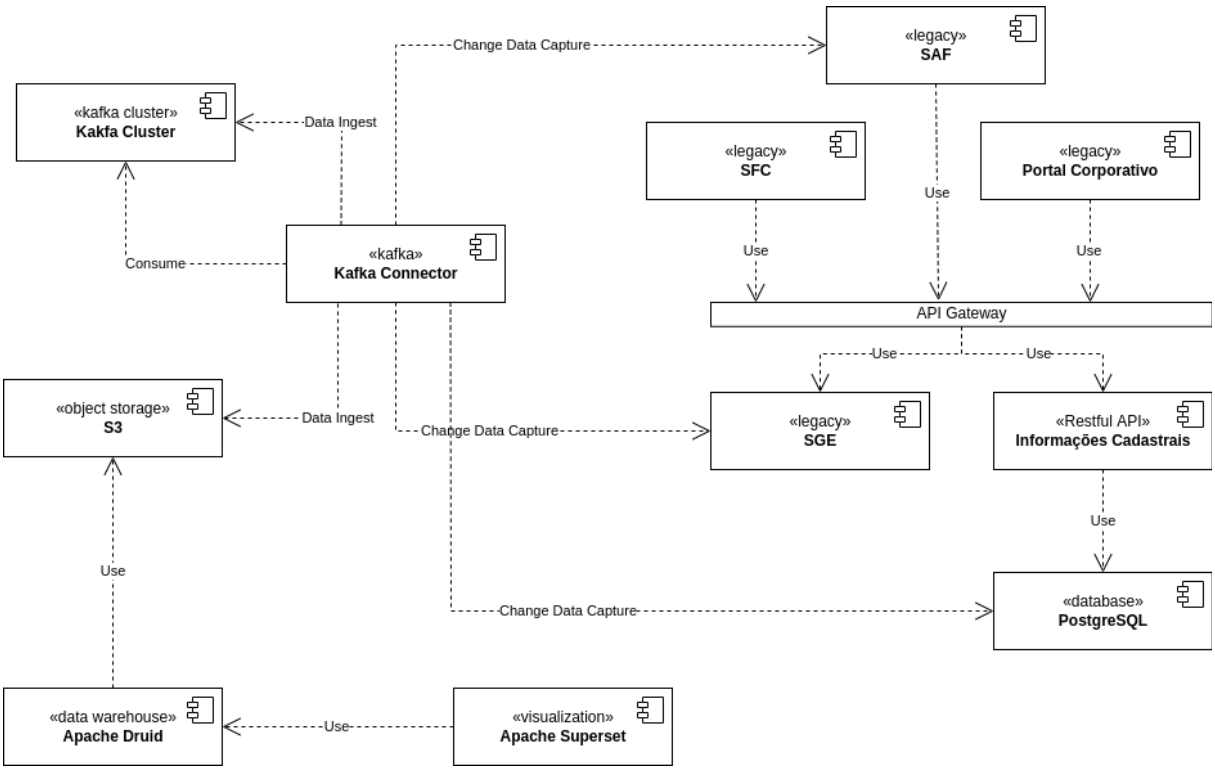

Figura 2: Diagrama de Componentes. Fonte: Autor.

Além do componente referente ao cluster Kafka, é necessário o componente Kafka Connector, que será responsável por buscar todas mudanças que ocorrerem nos sistemas legados e até mesmo nos novos e enviar os dados para um tópico no cluster do Kafka. A ideia é que para cada mudança nos bancos de dados relacionais possa ser capturado estas mudanças e assim serem enviadas para o serviço de integração.

Ainda utilizando o Kafka Connector, será feito a ingestão de todos os eventos nos tópicos em um serviço de armazenamento de objetos (S3). Com o S3 o objetivo é construir um data lake, que permite o armazenamento de dados de diferentes fontes e sem um formato fixo. A partir deste data lake os dados serão evoluídos e disponíbilizados em um Data Warehouse para análise dos mesmos e auxílio na tomada de decisão.

Sendo que para atuar como Data Warehouse neste cenário foi escolhido o Apache Druid, ferramenta também gratuita com escalibilidade para Big Data. Ele já possuí conectores nativos com o S3, já tendo fácil integração com diferentes tipos de armazenamento de objetos populares no mercado de vários provedores de nuvem.

Para ser feito as análises necessárias nos dados, criação de dashboards e apresentação das informações pode-se utilizar outra ferramente de código aberto,

Apache Superset. Com este componente já possui um controle de acesso de usuários não necessitando implementação de autenticação e autorização. Com conexão nativo ao Apache Druid, ficará fácil conectar as fontes de dados únicas e até mesmo diferentes fontes se necessário.

Sendo o SGE o princípal módulo (legado) pode-se utilizar um API Gateway na frente dele para padronizar o acesso ao módulo e facilitar o acesso por quaisquer novo serviço se necessário. Além de ser utilizado o mesmo API Gateway para acesso a informações cadastrais.

4 Prova de Conceito (POC)

Para realizar a POC foi utilizado *containers* docker para execução dos serviços especificados nos diagramas da arquitetura. O objetivo é trazer agilidade e flexibilidade para reprodução dos testes em diferentes ambientes sem a necessidade de realizar a instalação dos softwares de fato.

No seguinte link pode ser encontrado todos os artefatos gerados para realização da POC, https://github.com/josejnra/tcc-pos-puc. O projeto está hospedado no GitHub, repositório de projeto git. No projeto há arquivos *README.md* que trazem informações básicas de como levantar os serviços e o que pode ser feito.

4.1. Implementação

Os 3 casos de uso escolhidos são fortemente baseados no serviço de mensageria Apache Kafka. Sendo assim, na raíz do projeto pode ser encontrado um diretório chamado *kafka* que possui o código necessário para executar os serviços necessários para simular o caso de uso.

No diretório do *kafka* encontra-se um arquivo chamado *docker-compose.yaml* responsável por criar os containers responsáveis pela execução do Apache Kafka localmente. Além do Kafka, é criado um banco de dados utilizando o SGBD PostgreSQL para simular uma base de dados já existente da empresa visando simular a integração com novos sistemas. Este banco de dados ao ser criado é automaticamente populado com dados *mock* para ajudar na POC e validar o processo. Outro serviço que é criado é o *minio*, que é um serviço de armazenamento de objetos. O objetivo com o minio é servir como local de armazenamento para o *data lake* criado e suas zonas. Ele pode ser utilizado para fins de POC, mas há outros serviços em

nuvens públicas semelhantes que eventualmente poderiam ser utilizadas como: *Amazon Storage Service* da AWS e o *Azure Blob Storage* da Azure.

Além do banco de dados utilizado para simular uma base de dados legada, foi implementado um simples programa de linha de comando que auxilia na geração e inserção de dados mock para o banco de dados para poder acompanhar os processos do kafka connect em execução. O código implementado para isto pode ser encontrado no diretório *db_cli* na raíz do projeto. Dentro deste diretório há um arquivo *README.md* com todas instruções de uso do *script* implementado.

Outro serviço utilizado na POC é o *Apache Airflow*. Solução amplamente utilizada na indústria de dados para orquestração de cargas de trabalhos. Possuindo uma interface web e baixa curva de aprendizado na sua utilização. Para executar o Airflow localmente deve-se ir na pasta *airflow*, onde se encontra um arquivo docker-compose.yaml, que será responsável pela
execução do serviço. Tanto no diretório do *kafka* quanto no do *airflow* há instruções em um arquivo *markdown* para execução dos serviços.

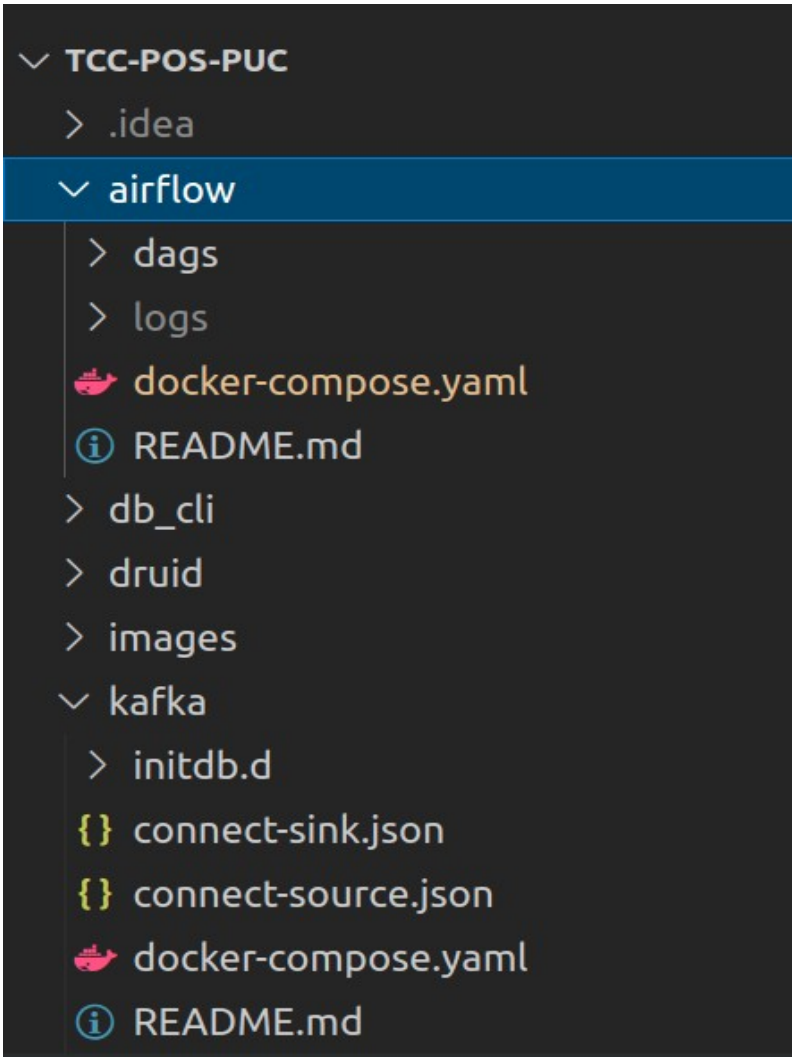

Figura 3: Estrutura de diretórios do projeto utilizado para a POC

Na Figura 3 pode ser conferido a estrutura de diretórios do projeto utilzado para a POC. Em que é mostrado os arquivos mencionados, docker-compose.yaml e README.md, para os serviços kafka e airflow com instruções de execução local.

4.1.1 Caso de Uso 1

Para o primeiro caso de uso, o objetivo é realizar a coleta de dados de bases de dados existentes e/ou de novos serviços para poder disponibilizar os dados para consumo de outras aplicações e assim poder haver integrações entre elas de forma escalável e desacoplada. Deste modo, não será necessário alterar serviços já existentes para se conectar ao cluster do kafka e enviar os eventos. Já que todas modificações nos registros do banco de dados serão enviadas para o tópico kafka. Este processo é conhecido como Change Data Capture [2].

Os requisitos associados a este caso de uso são:

- Possuir características de aplicação distribuída: abertura, portabilidade, uso de recursos de rede. Critério de aceitação: sistema permanecer em caso de falha em caso de queda de um dos brokers do kafka.
- Apresentar bom desempenho. não tendo um tempo de resposta superior a 5 segundos. Critério de aceitação: entre o dado ser gerado e estar disponível no tópico do kafka não pode ser superior a 5 segundos.
- Apresentar boa manutenibilidade. Adicionar ou alterar recursos do serviço não pode levar mais que uma sprint para estar implantado em ambiente produtivo. Critério de aceitação: adicionar um conector kafka não pode levar mais que uma hora para estar deployado e operacional.

Para este caso de uso deve-se ter o kafka em execução, o banco de dados e o kafka connect. Sendo que o fluxo se baseia em: para cada registro escrito, modificado ou excluído no banco de dado será buscado pelo conector em execução no kafka connect e o evento enviado para um tópico kafka. Na configuração deste conector é definido quais tabelas são de interesse coletar além dos parâmetros de conexão com o banco. Para cada tabela de interesse os eventos serão enviado para um tópico exclusivo dela, sendo uma proporção de 1:1 entre tabela e tópico. A configuração do tópico pode ser conferida no arquivo *connect-source.json*, que se encon-

tra no diretório do kafka. Na Figura 4 pode ser conferido as configurações do conector source utilizado para buscar os eventos em uma base de dados do PostgreSQL.

```
{
  "name": "connector-source",
  "config": {
      "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
      "plugin.name": "pgoutput",
      "tasks.max": "1",
      "database.hostname": "postgres",
      "database.port": "5432",
      "database.user": "postgres",
      "database.password": "root",
      "database.dbname" : "transportadora",
      "database.server.name": "dbserver1",
      "schema.whitelist": "public"
      }
}
```

Figura 4: Configuração do conector source do kafka connect

4.1.2 Caso de Uso 2

No segundo caso de uso o objetivo é realizar a ingestão dos dados no data lake. Os dados estando disponíveis no data lake será possível evoluí-los e gerar os relatórios para auxílio na tomada de decisão pela gerência. Para ser feito a ingestão dos dados será utilizando, também, o kafka connector só que um conector do tipo *sink*. Diferente do conector do tipo *source*, que busca o dado de algum lugar e salva em um tópico kafka, o conector do tipo *sink* pega eventos de um tópico kafka e envia-os para outro lugar. Neste caso, o objetivo é enviar os dados para o serviço de armazenamento de objetos, *minio*. O minio será onde o data lake, nesta POC, será implementado. Antes de executar o conector é necessário que o local de destino exista, por isto o bucket no minio deve ser criado previamente.

Este caso de uso possui como requisitos não funcionais:

- Possuir características de aplicação distribuída: abertura, portabilidade, uso de recursos de rede; Critério de aceitação: sistema permanecer em caso de falha em caso de queda de um dos brokers do kafka.
- Apresentar bom desempenho. Não tendo um tempo de resposta superior a 5 segundos. Critério de aceitação: entre o dado ser gerado e estar disponível no tópico do kafka não pode ser superior a 5 segundos.

- Ser de fácil implantação e utilização; Critério de aceitação: adicionar um conector kafka não pode levar mais que uma hora para estar deployado e operacional.

Para este caso de uso, deve-se ter o kafka em execução, e um tópico kafka populado, um bucket no minio e o kafka connect. Sendo que o fluxo é: um evento é escrito no tópico kafka, o conector kafka sink irá buscar tal evento no tópico e escrever em um *bucket* no data lake (minio). O conector possui uma estatégia de escrita em batch, que permite o acúmulo de eventos antes de realizar um escrita. Isto visa aumentar a performance para um grande volume de dados neste processo de escrita.

```
1  {
2    "name": "connector-sink",
3    "config": {
4      "name": "connector-sink",
5      "connector.class": "io.confluent.connect.s3.S3SinkConnector",
6      "tasks.max": "1",
7      "topics": "dbserver1.public.clientes,dbserver1.public.entregas,dbserver1.public.pedidos",
8      "key.converter": "org.apache.kafka.connect.storage.StringConverter",
9      "key.converter.schemas.enable": "false",
10     "key.converter.schema.registry.url": "http://schema-registry:8081",
11     "value.converter": "io.confluent.connect.avro.AvroConverter",
12     "value.converter.schema.registry.url": "http://schema-registry:8081",
13     "value.converter.schemas.enable": "false",
14     "errors.tolerance": "all",
15     "errors.deadletterqueue.topic.name": "dbserver1.public.tables-sink-errors",
16     "errors.deadletterqueue.topic.replication.factor": "1",
17     "errors.deadletterqueue.context.headers.enable": "true",
18     "schema.compatibility": "FORWARD",
19     "s3.bucket.name": "lake-raw-zone",
20     "flush.size": "2",
21     "rotate.interval.ms": "10000",
22     "format.class": "io.confluent.connect.s3.format.parquet.ParquetFormat",
23     "aws.access.key.id": "minioadmin",
24     "aws.secret.access.key": "minioadmin",
25     "s3.path.style.access.enabled": "true",
26     "storage.class": "io.confluent.connect.s3.storage.S3Storage",
27     "topics.dir": " ",
28     "store.url": "http://minio:9000",
29     "partitioner.class": "io.confluent.connect.storage.partitioner.TimeBasedPartitioner",
30     "partition.duration.ms": "3600000",
31     "path.format": "'year'=YYYY/'month'=MM/'day'=dd/'hour'=HH",
32     "locale": "en-US",
33     "timezone": "UTC",
34     "timestamp.extractor": "Record"
35   }
36 }
```

Figura 5: Configuração conector sink kafka

Na Figura 5 pode ser conferido um exemplo de configuração para o conector sink do kafka. Nele é necessário informar quais tópicos deve-se buscar os eventos (linha 7) e onde estes eventos serão salvos (linha 19). Sendo que para cada tópico

há um caminho distinto dentro do bucket criado, já com um particionamento definido por ano, mês, dia e hora (linha 31).

4.1.2 Caso de Uso 3

Para o terceiro caso de uso, foi escolhido o acompanhamento de execução de cargas de trabalho. O objetivo é facilitar a identificação e correção de execução de processos. Para isto foi escolhido a ferramenta Apache Airflow. Com o Airflow é possível termos uma visualização gráfico das cargas de trabalho que são executadas via agendamento. O Airflow já possui nativamente mecanismos que notificam desenvolvedores, e qualquer outro grupo de interesse, em caso de falha ou sucesso de uma operação. Este serviço é amplamente utilizado na indústria justamente pela facilidade de uso e a linguagem utilizada, python, o que torna sua traz uma adoção rápida pelas empresas. Caso uma carga de trabalho falhe, o desenvolvedor ou um responsável qualquer, pode simplesmente mandar reexecutar a carga através de uma interface web. O Airflow trabalha com um esquema de DAG, Directed acyclic graph (Grafo Acíclico Não Direcionado), em que é bom estabelecer dependências entre cargas de trabalho de forma bem simples.

Os requisitos não funcionais envolvidos neste caso de uso são:

- Suportar ambientes web e móveis; Critério de aceitação: ser possível visualizar e reexecutar tarefas pelos smarthphones via web browser.
- Apresentar boa manutenibilidade. Critério de aceitação: Adicionar ou alterar recursos do serviço não pode levar mais que uma sprint para estar implantado em ambiente produtivo.
- Monitoramento sobre os processos em execução. Critério de aceitação: o usuário não pode ficar sem ter conhecimento se as tarefas foram executadas ou não.

Para este caso de uso é necessário apenas executar o Airflow localmente, conforme já descrito na seção acima, através do *docker-compose.yaml*.

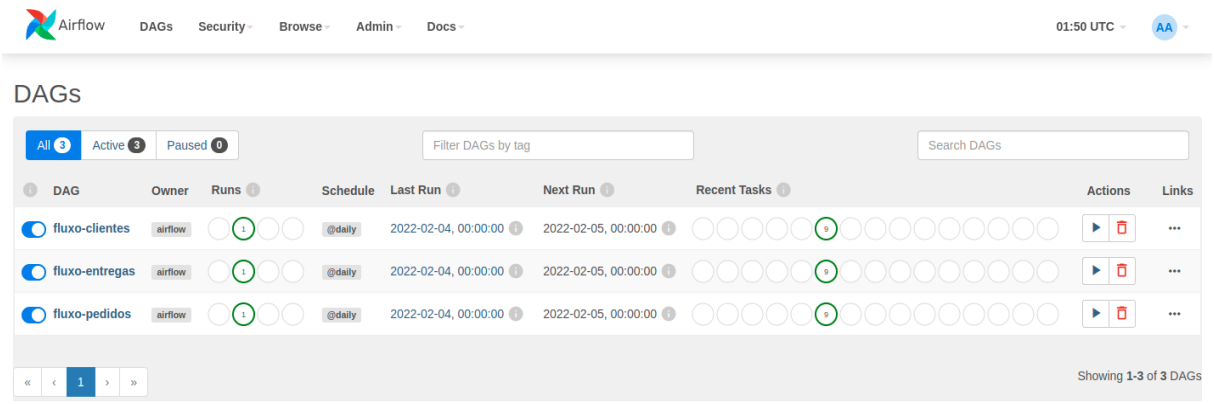

Figura 6: Página inicial do Airflow

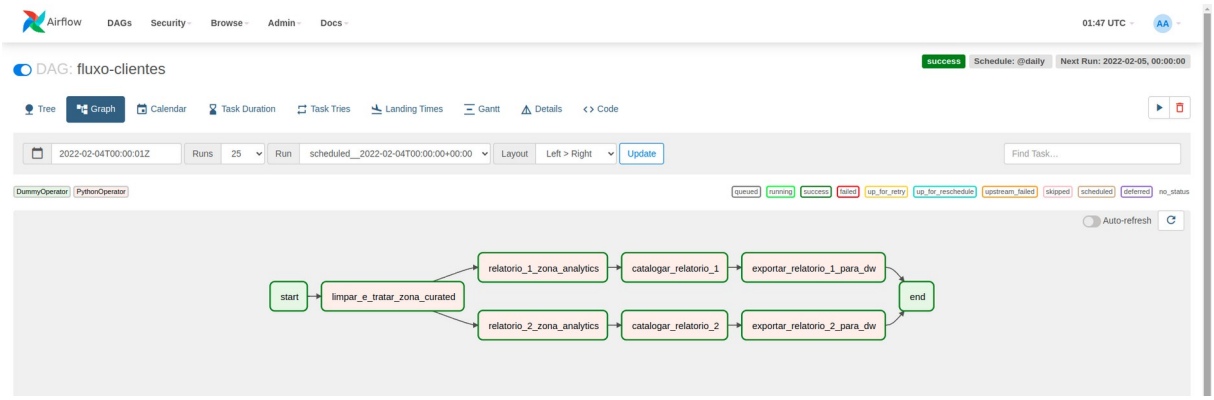

Figura 7: DAG representando um fluxo de execução para disponibilização de relatórios

Nas Figura 6 pode ser visualizado a tela inicial do Airflow, em que é listado todas as DAGs definidas e seus estados, além de informações como: última execução, próxima execução, quantas tarefa falharam e quantas deram sucesso além de outras informações.

Já na Figura 7 é mostrado a representação em grafo de uma DAG. Um fluxo em uma DAG no Airflow pode conter toda a evolução do dado, desde quando chegou na zona *raw* do data lake até a disponibilização de um relatório em algum serviço. Na figura é mostrado as dependências entras as tarefas e a ordem de execução, bem como se deram sucesso ao executar. Em uma DAG há várias configurações que podem serem feitas, como quem notificar, frequência de execução além de várias outras funcionalidades que evitam retrabalho para o desenvolvedor.

Na Figura 8 está presente o trecho de código utilizado para definição da DAG apresentada na Figura 7. Pode-se ver que com o Airflow pode-se construir longos fluxos com poucas linhas de código, o que acelera o processo de desenvolvimento e aumenta a manutenibilidade da solução. Da linha 44 até a 47 está definido o fluxo das tarefas que serão executadas e suas relaoes de dependência. Na linha 20 pode

ser visualizado um endereço de e-mail definido para ser notificado em caso de falha de execução da DAG.

```python
from datetime import datetime
import os
import sys
import time

from airflow.operators.python import PythonOperator

sys.path.insert(0, os.path.abspath(os.path.dirname(__file__)))

from airflow import DAG
from airflow.operators.dummy import DummyOperator

from airflow_utils import set_dag_id


args = {
    'owner': 'time-de-desenvolvimento',
    'description': 'Erro na execução da DAG de fluxo de clientes.',
    'start_date': datetime(2022, 1, 1),
    'email': ['fulano.beltrano@email.com'],
    'email_on_failure': True
}


with DAG(dag_id=set_dag_id(__file__),
         schedule_interval="@daily",
         max_active_tasks=1,
         default_args=args) as dag:

    start = DummyOperator(task_id='start')

    limpar_e_tratar = PythonOperator(task_id='limpar_e_tratar_zona_curated', python_callable=lambda: time.sleep(5))

    relatorio_1 = PythonOperator(task_id='relatorio_1_zona_analytics', python_callable=lambda: time.sleep(5))
    catalogar_relatorio_1 = PythonOperator(task_id='catalogar_relatorio_1', python_callable=lambda: time.sleep(5))
    exportar_relatorio_1_para_dw = PythonOperator(task_id='exportar_relatorio_1_para_dw', python_callable=lambda: time.sleep(5))

    relatorio_2 = PythonOperator(task_id='relatorio_2_zona_analytics', python_callable=lambda: time.sleep(5))
    catalogar_relatorio_2 = PythonOperator(task_id='catalogar_relatorio_2', python_callable=lambda: time.sleep(5))
    exportar_relatorio_2_para_dw = PythonOperator(task_id='exportar_relatorio_2_para_dw', python_callable=lambda: time.sleep(5))

    end = DummyOperator(task_id='end')

    start >> limpar_e_tratar

    limpar_e_tratar >> relatorio_1 >> catalogar_relatorio_1 >> exportar_relatorio_1_para_dw >> end
    limpar_e_tratar >> relatorio_2 >> catalogar_relatorio_2 >> exportar_relatorio_2_para_dw >> end

```

Figura 8: Implementação de uma DAG

5 Avaliação da Arquitetura

5.1. Análise das abordagens arquiteturais

Nesta seção será feito uma avaliação da arquitetura considerando o método Architecture Tradeoff Analysis Method (ATAM). Considerando os seguintes cenários:

Atributos de Qualidade	Cenários	Importância	Complexidade
Resiliência	Cenário 1: O sistema deve apresentar tolerância a falha	A	A
Manutenibilidade	Cenário 2: O sistema deve ter a manutenção facilitada.	M	M
Monitoramento	Cenário 3: O sistema deve notificar em caso de falhas	A	M

5.2. Cenários

Os cenários escolhidos para realizar a avalição utilizando o método ATAM foram: Cenário 1 - Resiliência: Ao cair um dos brokers do kafka, o sistema não pode perder dados ou deixar de operar.

Cenário 2 - Manutenibilidade: Havendo a necessidade de criar mais conectores kafka, seja source ou sink, basta ter as informações básicas de acesso das fontes e destino dos eventos e submeter esta configuração para o karfka connect. E imediatamente após submetido estar operando.

Cenário 3 - Monitoramento: Quando uma carga de trabalho falhar, os responsáveis pelo processo devem ser notificados imediatamente após a falha.

5.3. Evidências da Avaliação

Atributo de Qualidade:	Resiliência
Requisito de Qualidade:	O sistema deve apresentar tolerância a falha
Preocupação:	
Caso ocorra a queda de um dos brokers (nó) do kafka, o sistema deve continuar operando sem perdas de dados.	
Cenário(s):	
Cenário 1	
Ambiente:	
Sistema em operação normal	
Estímulo:	
Um dos brokers é desligado, simulando uma queda de um dos nós do cluster.	
Mecanismo:	
Criar um deploy de 2 brokers (nós) do kafka.	
Medida de resposta:	
Não constar perdas dos dados de um tópico.	
Considerações sobre a arquitetura:	
Riscos:	Se não configurado um fator de replicação de maior que 1 para os tópicos haverá perda dos dados caso um broker fique indisponível.
Pontos de Sensibilidade:	Brokers não podem estar implantados na mesma zona de disponibilidade em caso de

	deploy em nuvem pública.
Tradeoff:	Não há

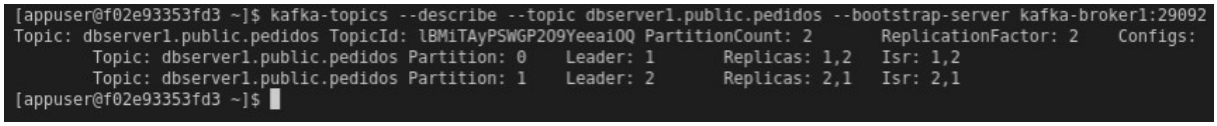

Figura 9: Detalhes do tópico antes de indisponibilidade de um broker

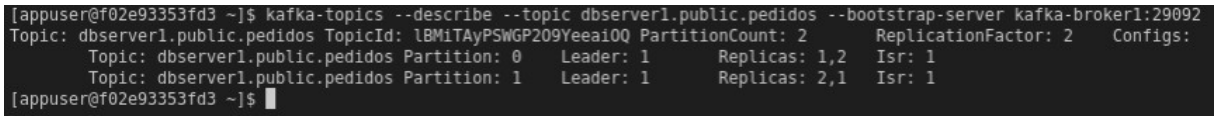

Figura 10: Detalhes do tópico após indisponibilidade de um broker

Na Figura 9 é mostrado uma descrição de um dos tópicos presente no cluster kafka levantado localmente com dois brokers. O tópico criado possui duas partições e fator de replicação igual a 2. Nos campos *Leader* pode ser visto que para cada partição um dos brokers seria o principal responsável: partição 0 seria principal no broker 1 e partição 1 seria principal no broker 2. E no campo *Isr* é mostrado quais réplicas estariam sincronizados. No campo *Replicas* é mostrado onde se encontram as replicas da partição. Neste caso, cada partição se encontra em ambos brokers. Por isto que em caso de queda de um dos brokers o sistema ainda irá continuar operando sem perda de dados.

Após simular a queda de um dos brokers, na Figura 10 pode ser visto como ficaria a disponibilidade dos dados do tópico. Para a partição 1 os dados já seriam encontrados no broker 1 (campo *Leader*) e não mais no broker 2, como na Figura 9. E o campo *Isr* apenas mostra que as réplicas presentes no broker 1 estão sincronizadas. Num cenário de perda de dados, no campo *Leader* a partição seria mostrada como None, caso os dados dela estivessem indisponíveis.

Atributo de Qualidade:	Manutenibilidade
Requisito de Qualidade:	O sistema deve ter a manutenção facilitada
Preocupação:	
Havendo a necessidade de criar mais conectores kafka, seja source ou sink, basta ter as informações básicas de acesso das fontes e destino dos eventos e submeter esta configuração para o karfka connect. E imediatamente após submetido estar operando.	
Cenário(s):	
Cenário 2	
Ambiente:	
Sistema em operação normal	
Estímulo:	
Submissão de arquivo de configuração do kafka connect source para busca dos dados de um banco de dados PostgreSQL.	
Mecanismo:	
Criar um arquivo JSON com a configuração do conector.	
Medida de resposta:	
Esforço em criar em definir configuração e criar o conector.	
Considerações sobre a arquitetura:	
Riscos:	Deve-se atendar no arquivo de configuração para não buscar mais tabelas do que se tem interesse e assim não gerar utilização de recursos desnecessariamente.
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

```
{
  "name": "connector-source",
  "config": {
      "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
      "plugin.name": "pgoutput",
      "tasks.max": "1",
      "database.hostname": "postgres",
      "database.port": "5432",
      "database.user": "postgres",
      "database.password": "root",
      "database.dbname" : "transportadora",
      "database.server.name": "dbserver1",
      "schema.whitelist": "public"
      }
}
```

Figura 11: Configuração do conector kafka

```
curl -i -X POST -H "Accept:application/json" -H  "Content-Type:application/json"
http://localhost:8083/connectors/ -d @connect-source.json
```

Figura 12: Comando utilizado para criar o conector source do kafka

Na Figura 10 é mostrado a configuração do conector a ser criado. Foi definido que o conector irá buscar dados de um banco de dados PostgreSQL, os parâmetros necessários para realizar a conexão com a fonte e qual workspace do kafka (*database.server.name*) estes dados estarão disponíveis no kafka. Sendo que os tópicos no kafka serão criados automaticamente e será buscado todas tabelas, já que não foi definido quais tabelas buscar.

Já na Figura 11 pode ser visto o comando utilizado para criar o conector kafka. Para isto somente é necessário o endereço do kafka connnect, no caso *localhost* na porta 8083, e qual o arquivo de configuração para o conector (*connect-source.json*).

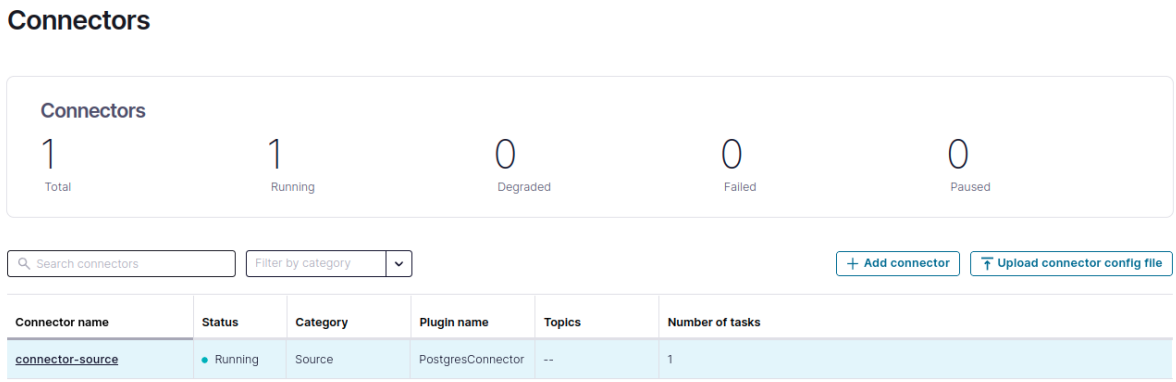

Figura 13: Lista de conectores existentes

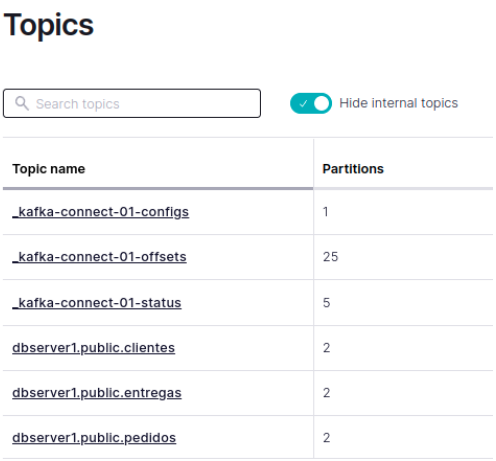

Figura 14: Lista de tópicos existente no Kafka

A Figura 13 mostra o *status* do conector criado (*running*), que no caso não houve nenhum erro na execução. E na Figura 14 é mostrado que foi criado 3 tópicos, cada um referente a uma tabela: dbserver1.public.clientes, dbserver1.public.entregas e dbserver1.public.pedidos.

Sendo assim, verifica-se que a obtenção de dados via CDC não requer muito esforço e a criação de conectores no kafka é bem simples, trazendo agilidade e baixa esforço para se criar novas extrações.

Atributo de Qualidade:	Monitoramento
Requisito de Qualidade:	O sistema deve notificar em caso de falhas
Preocupação:	
Quando uma carga de trabalho falhar, os responsáveis pelo processo devem ser notificados imediatamente após a falha.	
Cenário(s):	
Cenário 3	
Ambiente:	
Sistema em operação normal	
Estímulo:	
No Airflow, criar uma tarefa responsável por criar um relatório. Caso não encontre os dados relativo ao relatório deverá notificar os responsáveis pelo processo.	
Mecanismo:	
Criar uma DAG no Airflow responsável pela geração do relatório.	
Medida de resposta:	
Envio de notificação de falha por e-mail.	
Considerações sobre a arquitetura:	
Riscos:	Não há
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

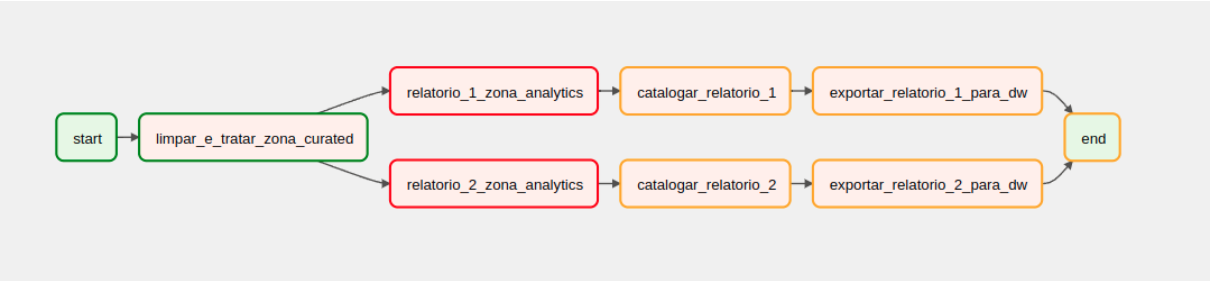

Figura 15: DAG do Airflow que possui falha no fluxo de trabalho

A Figura 15 mostra o fluxo de trabalho de uma DAG que possui falha. A falha no caso seria a obtenção de dados para gerar relatórios. Quando isto ocorrer deve-se enviar uma notificação para os e-mails definidos na construção da DAG. O que pode ser conferido na Figura 16, em que é mostrado um e-mail recebido pelo Airflow alertando sobre a falha na execução em uma das tarefas. Nesta notificação há uma mensagem básica alertando sobre a falha e links direcionando para o *log* da task que falhou.

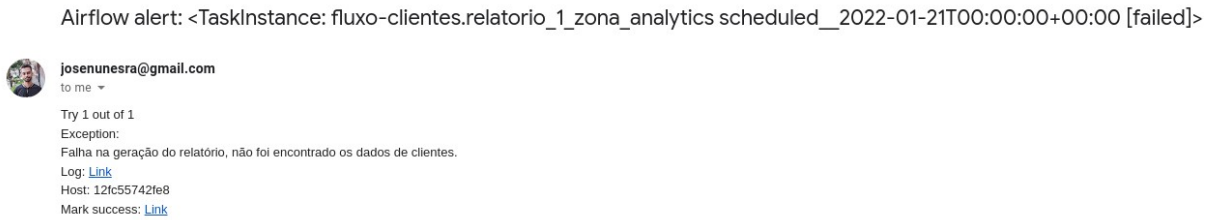

Figura 16: Notificação enviada pelo Airflow por email

5.4. Resultados

Requisitos Não Funcionais	Testado	Homologado
RNF01: Possuir características de aplicação distribuída: abertura, portabilidade, uso de recursos de rede;	SIM	SIM
RNF05: Suportar ambientes web e móveis;	SIM	SIM
RNF06: Possuir interface responsiva. Compatível com diversas resoluções de tela.	SIM	SIM
RNF08: Apresentar boa manutenibilidade. Adicionar ou alterar recursos do serviço não pode levar mais que uma sprint para estar implantado em ambiente produtivo.	SIM	SIM

RNF10: Ser recuperável (resiliente) no caso da ocorrência de erro. O usuário não pode ficar sem ter conhecimento se as tarefas foram executadas ou não.	SIM	SIM

O RNF01 foi testado e validado com a execução do kafka ao derrubar um dos brokers em excução. Após esta falha foi verificado que o sistema continuou operando normalmente sem perdas de dados. Para o RNF05 foi utilizado o Airflow principalmente por seu ambiente web, assim como o Kafka possui uma cliente web que permite visualizar vários atributos de configuração bem como as mensagens deisponíveis nos tópicos. Tanto o cliente web do airflow quanto do kafka possuem nativamente interface responsiva, atendendo ao RNF06. Ao criar um conector kafka, tanto para *source* quanto para *sink*, foi visto que adicionar ou alterar conectores existentes não representa uma dor para manutenção dos serviços.

6. Conclusão

Nete trabalho foi descrito uma proposta de arquitetura para o problema exposto e realizado uma POC para alguns dos pontos de interesse. Pode-se verificar que com a utilização de um serviço de mensageria seria possível realizar a integração com serviços legados e os novos. As tecnologias escolhidas são de código aberto, o que representaria um custo baixo para a empresa por não precisar comprar licenças. Além disto, são tecnologias já consolidadas no mercado com desempenho já reconhecido e fácil implantação e utilização. Com a POC desenvolvida, foi visto que atenderia perfeitamente os requisitos funcionais e não funcinais definidos. Sendo que os casos de uso escolhidos são relacionados a obtenção de dados e disponibilização para geração de relatórios que irão auxiliar na tomada de decisões pela gerência.

REFERÊNCIAS

1. Logística cresce na pandemia com aumento de compras pela internet. **G1 Globo**. Disponível em: https://g1.globo.com/economia/pme/pequenas-empresas-grandes-negocios/noticia/2020/12/06/logistica-cresce-na-pandemia-com-aumento-de-compras-pela-internet.ghtml. Acesso em: 12, dezembro, 2021.

2. Change Data Capture (CDC): What it is and How it Works. **Striim**. Disponível em: https://www.striim.com/change-data-capture-cdc-what-it-is-and-how-it-works/. Acessado em: 19, dezembro, 2021.

APÊNDICE - A

Código do projeto no GitHub: https://github.com/josejnra/tcc-pos-puc

URL do vídeo no YouTube: https://youtu.be/0rLDrVqVdXU