Arquitetura

Tecnologias Utilizadas:

O projeto utiliza as seguintes tecnologias:

Componentes Docker:

- Airflow

- Kafka

- ElasticSearch

- Kibana

- PostegreSQL

- MongoDB

- Loghash

- Spark (jupyter notebook)

Frameworks:

- Streamlit

- OpenAI

- Ollama

- Langchain

- Pandas

- Pytorch

Explicando cada tecnologia e sua principal função no projeto:

**Airflow**

Apache Airflow é uma plataforma de código aberto usada para criar, agendar e monitorar workflows programaticamente. Ele permite que os usuários definam workflows como grafos acíclicos direcionados (DAGs), onde cada nó representa uma tarefa individual a ser executada. Airflow é amplamente utilizado para orquestração de dados, automação de processos e integração de sistemas.

Airflow oferece várias funcionalidades importantes:

* **Escalabilidade**: Pode ser facilmente escalado para lidar com grandes volumes de dados e tarefas complexas.
* **Monitoramento e Logging**: Fornece ferramentas para monitorar a execução de tarefas e visualizar logs detalhados.
* **Flexibilidade**: Suporta extensões através de operadores personalizados e integração com várias ferramentas de dados.
* **Reusabilidade**: Facilita a reutilização de componentes de workflow, economizando tempo e esforço.

**Utilização do Airflow no Projeto**

No projeto, o Airflow é o principal responsável pelo monitoramento do ambiente e verificação de integridade. Ele desempenha essas funções através de duas DAGs principais:

1. **Análise do Ambiente**: Esta DAG é responsável por monitorar continuamente o estado do ambiente. Ela realiza verificações periódicas para garantir que todos os componentes estejam operando dentro dos parâmetros normais.
2. **Envio de Alertas por Email**: Caso a DAG de análise do ambiente detecte algum problema ou anomalia, uma segunda DAG é acionada para enviar emails de alerta aos responsáveis, garantindo que problemas sejam tratados prontamente.

Além disso, o projeto utiliza processos de streaming e batch para o processamento de dados. O Airflow desempenha um papel crucial nesse contexto:

* **Processos Batch**: Airflow aciona a execução de processos batch, garantindo que eles sejam iniciados no momento correto e de acordo com a sequência definida pelo workflow.
* **Integração entre Processos**: Ele coordena a comunicação entre processos de streaming e batch, assegurando que os dados fluam corretamente entre diferentes etapas do pipeline de processamento.
* **Agendamento de Tarefas**: Airflow gerencia o agendamento das tarefas de batch, garantindo que elas sejam executadas de forma eficiente e sem conflitos com outras operações em andamento.

Com essas funcionalidades, o Airflow não só monitora e verifica o ambiente, mas também orquestra a execução de tarefas críticas, garantindo a integridade e eficiência do projeto como um todo. A flexibilidade e escalabilidade do Airflow tornam-no uma ferramenta indispensável para o sucesso do projeto.

**Kafka**

Apache Kafka é uma plataforma de streaming distribuída que pode ser usada para construir pipelines de dados em tempo real e aplicativos de streaming. Ele permite a publicação, assinatura, armazenamento e processamento de fluxos de registros em tempo real. Kafka é altamente escalável, tolerante a falhas e é usado por muitas grandes empresas para gerenciar seus dados em movimento.

Principais características do Kafka:

* **Alta Taxa de Transferência**: Capaz de processar milhões de mensagens por segundo.
* **Baixa Latência**: Permite a transmissão de dados em tempo real com baixa latência.
* **Durabilidade**: Utiliza logs distribuídos para garantir a durabilidade dos dados.
* **Escalabilidade**: Pode ser escalado horizontalmente para lidar com grandes volumes de dados.
* **Integração**: Facilmente integrado com outras ferramentas e sistemas de processamento de dados.

**Utilização do Kafka no Projeto**

No projeto, o Kafka é implementado como um serviço de mensageria para o banco de dados MongoDB. Ele desempenha um papel essencial na captura, transmissão e processamento de informações. Abaixo está uma descrição detalhada de como o Kafka é utilizado:

1. **Captura de Informações**: Quando uma nova fatura (invoice) é postada, o Kafka captura essa informação e a transmite para o Spark Streaming.
2. **Processamento em Spark Streaming**: Existem dois processos de Spark Streaming configurados:
   * **Primeiro Processo**: Este processo salva as mensagens exatamente como elas são recebidas pelo Kafka, garantindo que os dados brutos estejam armazenados para referência futura.
   * **Segundo Processo**: Este processo realiza a Extração, Transformação e Carregamento (ETL) das mensagens. Ele analisa os dados, transforma conforme necessário e, finalmente, salva os dados processados no banco de dados MongoDB.
3. **Consumers**: Para gerenciar o fluxo de dados, são utilizados dois consumers distintos:
   * **Consumer de Ingestão (Ingest Topic)**: Este consumer é responsável por receber as informações postadas e garantir que elas sejam encaminhadas corretamente para o primeiro processo de Spark Streaming.
   * **Consumer de Spark-Streaming**: Este consumer recebe as mensagens do Kafka e as encaminha para o segundo processo de Spark Streaming, onde ocorre o tratamento e análise dos dados.

Com essa arquitetura, o Kafka garante que as informações postadas sejam capturadas e processadas eficientemente, permitindo que os dados sejam analisados e armazenados no MongoDB de forma organizada e confiável. A flexibilidade e robustez do Kafka tornam-no uma escolha ideal para serviços de mensageria em projetos de grande escala e alta demanda.

**ElasticSearch**

ElasticSearch é um mecanismo de busca e análise distribuído, aberto e altamente escalável, projetado para lidar com grandes volumes de dados em tempo real. Ele é parte do Elastic Stack, que inclui também ferramentas como Kibana, Logstash e Beats, usadas para coleta, processamento e visualização de dados.

**Funcionalidades do ElasticSearch**

* **Busca em Tempo Real**: Permite realizar buscas rápidas e em tempo real em grandes volumes de dados.
* **Escalabilidade**: Capaz de escalar horizontalmente, distribuindo dados e cargas de trabalho em múltiplos nós.
* **Alta Disponibilidade**: Oferece replicação de dados para garantir alta disponibilidade e tolerância a falhas.
* **Análise de Dados**: Suporta agregações avançadas que permitem análises de dados complexas.
* **Fácil Integração**: Facilmente integrado com outras ferramentas e sistemas de processamento de dados, especialmente dentro do Elastic Stack.

**Utilização do ElasticSearch no Projeto**

No projeto, ElasticSearch é essencial para a recepção e análise de dados processados pelo Spark Streaming. Após o processo de ETL (Extração, Transformação e Carregamento) realizado pelo Spark, as informações são transmitidas para o ElasticSearch, onde são armazenadas em padrões de índice (index patterns).

* **Recepção de Dados**: ElasticSearch recebe os dados via streaming após o processamento ETL, garantindo que as informações estejam disponíveis imediatamente.
* **Armazenamento em Index Patterns**: As informações são armazenadas em seus respectivos padrões de índice, facilitando a organização e busca eficiente dos dados.
* **Dashboards e Análises**: Com os dados armazenados, são criados dashboards e análises usando ferramentas como Kibana, permitindo uma visão abrangente e detalhada do andamento do projeto.

**Importância do ElasticSearch no Projeto**

Utilizar ElasticSearch no projeto é crucial por várias razões:

* **Desempenho em Tempo Real**: A capacidade de ElasticSearch de realizar buscas e análises em tempo real permite que as informações mais recentes estejam sempre disponíveis para consulta e análise.
* **Escalabilidade e Flexibilidade**: A escalabilidade horizontal de ElasticSearch assegura que ele pode crescer conforme o volume de dados aumenta, mantendo o desempenho e a eficiência.
* **Análise e Visualização de Dados**: A integração com Kibana permite a criação de dashboards e visualizações detalhadas, proporcionando insights valiosos e facilitando a tomada de decisões baseada em dados.
* **Alta Disponibilidade e Resiliência**: Com sua arquitetura distribuída, ElasticSearch garante alta disponibilidade e resiliência, essencial para um projeto que depende de dados em tempo real.

Com essas funcionalidades, ElasticSearch não só armazena e organiza os dados eficientemente, mas também oferece uma plataforma robusta para análise e visualização, tornando-se uma peça fundamental para o sucesso do projeto.

**Kibana**

Kibana é uma plataforma de visualização de dados que faz parte do Elastic Stack, usada para buscas, visualizações e análise de dados armazenados no ElasticSearch. Ela permite a criação de gráficos, tabelas e dashboards interativos para explorar os dados em tempo real.

**Funcionalidades do Kibana**

* **Visualizações Interativas**: Suporta a criação de gráficos, tabelas, mapas e outras visualizações interativas.
* **Dashboards Customizáveis**: Permite a criação de dashboards personalizados que podem ser compartilhados e integrados em outras aplicações.
* **Análise em Tempo Real**: Oferece a capacidade de analisar dados em tempo real, fornecendo insights instantâneos.
* **Alertas e Notificações**: Configuração de alertas para eventos específicos nos dados.
* **Exploração de Dados**: Ferramentas avançadas para exploração e consulta de dados.

**Utilização do Kibana no Projeto**

No projeto, Kibana é utilizado para criar dashboards que consomem informações dos padrões de índice (index patterns) armazenados no ElasticSearch. Dois dashboards principais foram configurados:

1. **Dashboard Streaming**: Este dashboard recebe todas as informações via Spark Streaming e é atualizado automaticamente toda vez que um novo registro é computado no sistema. Ele fornece uma visão em tempo real do estado das informações e do andamento do projeto.
   * **Sincronização com o Banco de Dados**: O dashboard é sincronizado com o banco de dados para permitir uma análise dupla dos resultados, oferecendo uma visão compreensiva e detalhada dos dados.
2. **Dashboard Batch (Machine Learning)**: Este dashboard é criado por um comando batch e está agendado para ser executado uma vez por dia às 22h, mas também pode ser executado manualmente. Este processo realiza uma análise de machine learning e, após isso, salva os padrões de índice no ElasticSearch.
   * **Análise de Machine Learning**: O processo de batch faz uma análise detalhada utilizando algoritmos de machine learning para prever tendências e analisar dados históricos.
   * **Atualização do Dashboard**: Kibana consome os resultados do processo de machine learning e os apresenta no dashboard, mostrando o andamento do projeto e as previsões geradas.
   * **Previsões e Análises**: Além do andamento do projeto, o dashboard apresenta previsões e análises criadas pelos modelos de machine learning, oferecendo insights valiosos e suportando a tomada de decisões informadas.

Com Kibana, o projeto ganha uma ferramenta poderosa para visualização e análise de dados, essencial para monitorar o desempenho, identificar problemas e tomar decisões rápidas e informadas. A integração com ElasticSearch e Spark Streaming garante que os dados sejam atualizados em tempo real e as análises de machine learning sejam apresentadas de forma clara e acessível.

**PostgreSQL**

PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto, conhecido por sua robustez, escalabilidade e conformidade com padrões SQL. Ele é amplamente utilizado em diversas aplicações devido à sua confiabilidade e extensibilidade.

**Funcionalidades do PostgreSQL**

* **Conformidade com Padrões SQL**: Suporta a maioria das funcionalidades do SQL padrão.
* **Extensibilidade**: Permite a criação de tipos de dados personalizados, funções e operadores.
* **Escalabilidade e Performance**: Gerencia grandes volumes de dados e alto desempenho em consultas.
* **Segurança**: Oferece diversos mecanismos de segurança, incluindo autenticação e controle de acesso.
* **Suporte a Transações**: Implementa transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

**Utilização do PostgreSQL no Projeto**

No projeto, PostgreSQL é utilizado como o banco de dados relacional principal, armazenando informações críticas que são consumidas e consultadas em várias etapas do projeto. As principais tabelas e suas funções são:

* **Tabelas de Clientes**: Armazenam informações detalhadas dos clientes, servindo como base para relacionar dados entre clientes e outros elementos do sistema.
* **Tabelas de Reclamações e Criticidades**: Contêm registros de reclamações feitas pelos clientes e a criticidade de cada caso, essencial para análises de suporte e atendimento.
* **Tabelas de Machine Learning**: Armazenam dados preparados para serem utilizados em processos de machine learning, incluindo tokenização e embedding, facilitando o treinamento e aplicação de modelos de IA.
* **Tabelas de Agências e Localizações**: Guardam informações sobre as agências e suas localizações, permitindo consultas geográficas e relacionais.

**Importância do PostgreSQL no Projeto**

A escolha do PostgreSQL como banco de dados relacional é crucial por várias razões:

* **Amarração de Dados**: O banco de dados serve como o principal ponto de união entre diferentes entidades, como clientes, agências e invoices, garantindo a integridade e consistência dos dados.
* **Consultas Complexas**: Permite a execução de consultas SQL complexas e eficientes, necessárias para extrair insights e relacionar informações de diferentes tabelas.
* **Suporte a IA e Machine Learning**: Armazena dados pré-processados para machine learning, facilitando a integração com processos de IA.
* **Segurança e Confiabilidade**: Oferece mecanismos robustos de segurança e controle de acesso, garantindo a proteção dos dados sensíveis.

Com PostgreSQL, o projeto garante uma base de dados sólida e confiável, capaz de suportar operações complexas e de alto desempenho, essencial para o sucesso e eficiência das diversas etapas do projeto.

**MongoDB**

MongoDB é um banco de dados NoSQL orientado a documentos, conhecido por sua flexibilidade, escalabilidade e fácil integração com diversas aplicações. Ele armazena dados em formato BSON (Binary JSON), permitindo consultas rápidas e eficientes.

**Funcionalidades do MongoDB**

* **Documentos Flexíveis**: Armazena dados em documentos JSON-like, permitindo esquemas flexíveis que podem evoluir conforme necessário.
* **Escalabilidade Horizontal**: Suporta sharding para distribuir dados e cargas de trabalho em múltiplos servidores.
* **Consultas Poderosas**: Oferece uma linguagem de consulta rica e expressiva.
* **Alta Disponibilidade**: Implementa replicação para garantir alta disponibilidade e tolerância a falhas.
* **Índices e Agregações**: Suporta índices complexos e pipelines de agregação para análises avançadas.

**Utilização do MongoDB no Projeto**

No projeto, MongoDB é utilizado como banco de dados documental principal, com dois bancos de dados específicos: um para criticidade e outro para invoices.

1. **Banco de Dados de Invoices**: Armazena as mensagens em formato RAW, ou seja, exatamente como são criadas e enviadas via POST.
   * **Armazenamento de Dados Brutos**: Mantém os dados na forma original, permitindo análises e transformações subsequentes conforme necessário.
   * **Rastreabilidade**: Cada mensagem possui um InvoiceID único gerado no momento do POST, garantindo rastreabilidade e integridade dos dados.
2. **Banco de Dados de Criticidade**: Contém informações já tratadas, que passaram por processos de ETL (Extração, Transformação e Carregamento), prontas para serem consumidas por outras ferramentas.
   * **Dados Tratados**: Armazena dados que foram processados e transformados, prontos para análises e consumo imediato.
   * **Integração com Outras Ferramentas**: Facilita a integração e consumo dos dados por outras ferramentas analíticas dentro do projeto.

**Importância do MongoDB no Projeto**

A escolha do MongoDB como banco de dados documental é essencial por várias razões:

* **Flexibilidade e Evolução do Esquema**: Permite a evolução do esquema de dados sem grandes migrações, adaptando-se facilmente às mudanças nas necessidades do projeto.
* **Rastreabilidade e Análise**: O uso de InvoiceID como identificador único permite a rastreabilidade precisa e a correlação de dados entre os bancos de dados de invoices e criticidade.
* **Desempenho e Escalabilidade**: Suporta grandes volumes de dados com alto desempenho e pode escalar horizontalmente conforme necessário.
* **Alta Disponibilidade**: Garante que os dados estejam sempre disponíveis, mesmo em caso de falhas de hardware.

Com MongoDB, o projeto ganha um banco de dados documental robusto e flexível, capaz de atender às necessidades de armazenamento e análise de dados de forma eficiente e escalável.

**Spark (Jupyter Notebook)**

Apache Spark é uma plataforma de processamento de dados em grande escala que oferece suporte para processamento em tempo real e em lote. No projeto, Spark é utilizado para processos de ETL (Extração, Transformação e Carregamento) das informações.

**Utilização do Spark no Projeto**

No projeto, Spark é executado dentro de um container Docker que hospeda um Jupyter Notebook. Abaixo estão os detalhes da implementação:

* **Docker com Jupyter Notebook**: Utiliza um container Docker para rodar o Jupyter Notebook, proporcionando um ambiente isolado e consistente para a execução de códigos.
* **Códigos em Python**: Dentro do notebook, são escritos códigos em Python que utilizam Spark para realizar o ETL das informações.
  + **Extração**: Dados são extraídos de diversas fontes, como bancos de dados relacionais (PostgreSQL) e documentais (MongoDB).
  + **Transformação**: Dados são transformados, incluindo limpeza, agregação, e enriquecimento, utilizando APIs do Spark.
  + **Carregamento**: Dados transformados são carregados e enviados para os consumidores finais ou armazenados em bancos de dados apropriados.

**Benefícios do Uso de Spark no Jupyter Notebook**

* **Interatividade**: Jupyter Notebook oferece um ambiente interativo para escrever e testar código, facilitando o desenvolvimento e depuração dos processos de ETL.
* **Escalabilidade**: Spark permite o processamento eficiente de grandes volumes de dados, escalando horizontalmente conforme necessário.
* **Flexibilidade**: A utilização de Python e Spark no Jupyter Notebook permite flexibilidade na manipulação e análise dos dados, aproveitando bibliotecas e ferramentas do ecossistema Python.

**Resumo do Processo de ETL**

1. **Execução de Docker com Jupyter Notebook**: Criação e configuração de um container Docker que hospeda o Jupyter Notebook.
2. **Desenvolvimento de Códigos de ETL em Python**: Especificação de scripts Python dentro do notebook para realizar as etapas de extração, transformação e carregamento dos dados.
3. **Processamento de Dados com Spark**: Utilização das APIs do Spark para manipulação e transformação de dados em grande escala.
4. **Envio aos Consumidores**: Após o processamento, os dados são enviados aos consumidores ou armazenados em bancos de dados para análises futuras.

Com a combinação de Spark e Jupyter Notebook, o projeto garante um ambiente robusto e flexível para a realização de processos de ETL, facilitando a manipulação de grandes volumes de dados e proporcionando insights valiosos.

**Streamlit**

Streamlit é um framework open-source em Python que facilita a criação de aplicações web interativas e visualizações de dados de forma simples e rápida. Ele é amplamente utilizado por cientistas de dados, engenheiros de machine learning e desenvolvedores para construir dashboards e ferramentas de análise com poucas linhas de código.

**Funcionalidades do Streamlit**

* **Fácil de Usar**: Permite a criação de aplicações web com apenas algumas linhas de código Python.
* **Interatividade**: Suporta widgets interativos como sliders, botões, caixas de seleção e gráficos dinâmicos.
* **Atualização Automática**: As mudanças no código são refletidas instantaneamente na aplicação web sem necessidade de atualização manual.
* **Integração com Bibliotecas de Python**: Compatível com bibliotecas populares como NumPy, Pandas, Matplotlib, Plotly, e Scikit-learn.
* **Desenvolvimento Rápido**: Permite o desenvolvimento ágil e prototipagem rápida de aplicações web.

**Utilização do Streamlit no Projeto**

Streamlit pode ser utilizado no projeto para desenvolver dashboards interativos e ferramentas de visualização de dados, facilitando a análise e apresentação dos resultados.

**Benefícios do Uso do Streamlit**

* **Rapidez no Desenvolvimento**: Permite a criação rápida de protótipos e aplicações completas sem necessidade de conhecimento profundo de front-end.
* **Interatividade e Visualização**: Facilita a criação de interfaces interativas e visualizações dinâmicas, melhorando a experiência do usuário.
* **Adoção Amplamente Utilizada**: Utilizado por muitas empresas e profissionais de dados, com uma comunidade ativa que contribui para a melhoria contínua do framework.
* **Desempenho e Escalabilidade**: Adequado para aplicações de pequeno a médio porte, com a possibilidade de escalabilidade conforme necessário.

Streamlit é uma ferramenta poderosa para desenvolvedores que desejam criar aplicações web interativas de forma rápida e eficiente, aproveitando o ecossistema robusto de bibliotecas de Python.

**OpenAI**

OpenAI é uma organização de pesquisa em inteligência artificial (IA) que tem como missão garantir que a IA beneficie toda a humanidade. Fundada em dezembro de 2015, a OpenAI desenvolve modelos avançados de aprendizado de máquina e inteligência artificial, incluindo o GPT (Generative Pre-trained Transformer), uma série de modelos de linguagem natural que são amplamente utilizados para diversas aplicações como chatbots, assistentes virtuais, e geração de texto.

**Funcionalidades e Aplicações**

* **Modelos de Linguagem Natural**: Desenvolve modelos como GPT-3 e GPT-4 para tarefas de processamento de linguagem natural.
* **Pesquisa em IA**: Publica pesquisas avançadas em IA, contribuindo para o desenvolvimento científico da área.
* **API de IA**: Oferece APIs que permitem a desenvolvedores integrar capacidades de IA em suas aplicações.

**Ollama**

Ollama é uma plataforma que fornece ferramentas e serviços para a construção e gerenciamento de modelos de IA e aprendizado de máquina. Focada em facilitar o desenvolvimento e a implantação de soluções de IA, Ollama oferece um ambiente integrado que permite aos usuários desenvolver, treinar e implantar modelos de forma eficiente.

**Funcionalidades e Aplicações**

* **Desenvolvimento de Modelos**: Ferramentas para a criação e treinamento de modelos de IA.
* **Gerenciamento de Modelos**: Soluções para versionamento e monitoramento de modelos em produção.
* **Integração Fácil**: APIs e SDKs para integração com diversas plataformas e serviços.

**LangChain**

LangChain é uma biblioteca Python que facilita a criação de fluxos de trabalho complexos envolvendo modelos de linguagem natural (LLMs). Ela permite a construção de pipelines que combinam diferentes modelos e componentes de processamento de linguagem, facilitando a implementação de soluções avançadas de IA.

**Funcionalidades e Aplicações**

* **Construção de Pipelines**: Ferramentas para criar fluxos de trabalho complexos que combinam múltiplos modelos de linguagem.
* **Integração com LLMs**: Suporte para integração com diversos modelos de linguagem, facilitando a experimentação e o desenvolvimento.
* **Automatização de Tarefas**: Permite a automação de tarefas de processamento de linguagem natural, como tradução, resumo e geração de texto.

**Pandas**

Pandas é uma biblioteca open-source de Python amplamente utilizada para análise e manipulação de dados. Ela fornece estruturas de dados flexíveis e eficientes, como DataFrames, que facilitam a limpeza, transformação e análise de grandes conjuntos de dados.

**Funcionalidades e Aplicações**

* **Estruturas de Dados**: DataFrames e Series para manipulação eficiente de dados tabulares.
* **Operações de Dados**: Ferramentas para filtragem, agregação, e transformação de dados.
* **Leitura e Escrita**: Suporte para leitura e escrita de dados em diversos formatos, incluindo CSV, Excel, SQL, e HDF5.

**PyTorch**

PyTorch é uma biblioteca de aprendizado de máquina open-source desenvolvida pelo Facebook's AI Research lab (FAIR). Ela é conhecida por sua flexibilidade e facilidade de uso, sendo amplamente utilizada para pesquisa e desenvolvimento de modelos de deep learning.

**Funcionalidades e Aplicações**

* **Redes Neurais**: Ferramentas para construção e treinamento de redes neurais profundas.
* **Computação Tênsil**: Suporte para operações de tensor com aceleração por GPU.
* **Comunidade e Ecosistema**: Uma comunidade ativa que contribui com pacotes adicionais e extensões, facilitando a implementação de soluções avançadas de IA.

Essas ferramentas e plataformas são fundamentais no ecossistema de IA e aprendizado de máquina, cada uma contribuindo de maneira única para o desenvolvimento e aplicação de soluções inteligentes.

**Documentação Completa do Projeto**

**Visão Geral**

Este projeto tem como objetivo processar e analisar dados de invoices enviados via um serviço POST. O fluxo de dados envolve múltiplos componentes, incluindo Kafka, Spark Streaming, um script de ETL, e um sistema de Inteligência Artificial (IA) para análise de sentimentos e clusterização dos dados. A seguir, é detalhado o passo a passo do processo.

**Passo a Passo**

1. **Envio do Invoice via POST**
   1. **Descrição**: O invoice é enviado para o sistema através de uma requisição POST. Isso pode ser feito utilizando ferramentas como Postman, um frontend ou um sistema específico.
   2. **Entrada**: Dados do invoice em formato JSON ou outro formato suportado.
2. **Captura da Informação pelo Kafka**
   1. **Descrição**: Assim que o invoice é enviado, Kafka captura essa informação. Kafka atua como um sistema de mensageria que registra os dados e faz a transição para o próximo estágio.
   2. **Tópico Kafka**: ingest-topic
3. **Transição da Mensageria pelo Kafka**
   1. **Descrição**: Kafka realiza a transição da mensagem para o consumidor designado, que neste caso é um processo Spark Streaming.
   2. **Consumer Name**: ingest-topic
4. **Processamento com Spark Streaming**
   1. **Descrição**: O Spark Streaming, que está sendo executado dentro de um script Python no Jupyter Notebook, consome as mensagens do Kafka. Existem dois processos distintos de Spark Streaming:
      1. **Captura de Informação Raw (Original)**:
         1. **Descrição**: Este processo captura a informação original do POST e a salva no banco de dados de invoices.
         2. **Destino**: Banco de Dados invoice
      2. **Captura e Processamento ETL**:
         1. **Descrição**: Este processo captura a informação e a envia para um script de ETL (Extração, Transformação e Carregamento).
5. **Processamento ETL**
   1. **Descrição**: No script de ETL, os dados são transformados e preparados para análise. As principais etapas envolvem limpeza e formatação dos dados.
6. **Análise pela IA**
   1. **Descrição**: Após o processamento ETL, os dados são passados por um sistema de IA, cujo principal foco é analisar o campo description da informação.
   2. **Tarefas da IA**:
      1. **Análise de Sentimento**:
         1. **Descrição**: A IA realiza uma análise de sentimento para determinar se o sentimento expresso no description é positivo ou negativo.
         2. **Saída**: Campo Sentimental Analisys indicando positivo ou negativo.
      2. **Clusterização da Informação**:
         1. **Descrição**: A IA agrupa as informações em tópicos para facilitar a análise. Exemplos de tópicos incluem APP, Agência, Atendimento, etc.
         2. **Saída**: Campo indicando o tópico de clusterização.
7. **Salvamento e Disponibilização dos Dados**
   1. **Descrição**: As novas informações geradas, incluindo Sentimental Analisys e a clusterização, são salvas e disponibilizadas para os consumidores finais ou para análises futuras.
   2. **Destino**: Banco de Dados ou Sistema de Armazenamento apropriado.

**Resumo dos Componentes**

* **Ferramentas de Envio**: Postman, Frontend, Sistema Interno
* **Mensageria**: Kafka
* **Processamento**: Spark Streaming, Jupyter Notebook
* **ETL e IA**: Script Python, Ferramentas de IA para Análise de Sentimento e Clusterização
* **Banco de Dados**: Para armazenamento de dados brutos e processados

**Pipeline de Dados**

1. **Envio do Invoice → Kafka → Spark Streaming**
2. **Spark Streaming → Banco de Dados (Dados Raw)**
3. **Spark Streaming → ETL → IA (Análise de Sentimento e Clusterização) →Banco de Dados**
4. **IA → Banco de Dados (Dados Processados e Analisados)**

**Documentação Completa do Projeto - Segunda Etapa**

**Visão Geral**

A segunda etapa do projeto envolve a transmissão dos dados processados pelo Spark Streaming para o ElasticSearch. Isso é feito utilizando Index Patterns, permitindo que os dados sejam salvos no ElasticSearch em formato de append. Os dados também são alimentados simultaneamente no MongoDB. A seguir, é detalhado o passo a passo do processo.

**Passo a Passo**

1. **Transmissão de Dados via Spark Streaming para ElasticSearch**
   * **Descrição**: Após o processamento inicial, o Spark Streaming transmite os dados para o ElasticSearch.
   * **Método**: Utilização de conector Spark para ElasticSearch.
   * **Formato de Armazenamento**: Append (os novos registros são adicionados ao índice existente).
2. **Salvamento dos Dados no ElasticSearch**
   * **Descrição**: Os dados são salvos no banco de dados do ElasticSearch utilizando Index Patterns, que permitem a organização e consulta eficiente dos dados.
   * **Index Patterns**: Estruturas que facilitam o acesso e a consulta dos dados no ElasticSearch, permitindo a montagem de dashboards no Kibana.
3. **Alimentação Simultânea no MongoDB**
   * **Descrição**: Além de salvar os dados no ElasticSearch, o Spark Streaming também adiciona os registros processados no MongoDB.
   * **Formato de Armazenamento**: Append, semelhante ao ElasticSearch.
4. **Consumo para Kibana**
   * **Descrição**: Os Index Patterns fornecem os dados para o Kibana, uma ferramenta de visualização que permite a criação de dashboards e análises.
   * **Benefícios**: Facilita a visualização e análise dos dados processados, permitindo insights rápidos e eficientes.

**Pipeline de Dados - Segunda Etapa**

1. **Spark Streaming → ElasticSearch (Append via Index Patterns)**
2. **Spark Streaming → MongoDB (Append)**
3. **ElasticSearch → Kibana (Consumo via Index Patterns)**

**Componentes Envolvidos**

* **Spark Streaming**: Ferramenta de processamento de dados em tempo real.
* **ElasticSearch**: Motor de busca e análise distribuído.
* **Index Patterns**: Estruturas no ElasticSearch que facilitam a consulta e visualização dos dados.
* **MongoDB**: Banco de dados NoSQL para armazenamento adicional dos dados.
* **Kibana**: Ferramenta de visualização para criação de dashboards e análises.

**Resumo dos Benefícios**

* **Armazenamento Eficiente**: Uso de ElasticSearch e MongoDB para armazenamento eficiente e escalável.
* **Visualização e Análise**: Kibana fornece uma interface amigável para análise de dados e criação de dashboards.
* **Atualização em Tempo Real**: Com o Spark Streaming, os dados são processados e atualizados em tempo real, garantindo que as análises no Kibana estejam sempre atualizadas.

Esta documentação detalha a segunda etapa do processo, desde a transmissão dos dados pelo Spark Streaming até a visualização no Kibana, proporcionando uma visão clara e estruturada do fluxo de trabalho e dos componentes envolvidos.

**Documentação do Uso do Airflow**

**Visão Geral**

O Airflow está sendo utilizado no projeto de duas formas distintas: para monitoramento de processos e para execução de processos batch. A seguir, detalharemos cada um desses usos.

**1. Monitoramento de Processos**

* **Descrição**: O Airflow cria DAGs (Directed Acyclic Graphs) que monitoram os processos e análises de funcionalidade do projeto.
* **Funcionalidade**: As DAGs verificam se os processos estão funcionando conforme esperado e registram o andamento na interface de usuário (UI) do Airflow.
* **Benefícios**: Proporciona uma visão clara e contínua do status dos processos, facilitando a identificação de falhas e a manutenção do sistema.

**2. Execução de Processos Batch**

* **Descrição**: O Airflow é utilizado para executar processos batch, especialmente aqueles relacionados a análises de machine learning.
* **Funcionalidade**:
  + **Análises de Machine Learning**: Processos batch são configurados para rodar diariamente às 22:00, realizando análises de machine learning necessárias.
  + **Geração de Gráficos e Consumos via Kibana**: Os resultados das análises são utilizados para criar gráficos e alimentar dashboards no Kibana.
* **Configuração**: Este processo diário é executado e gerenciado pelo Airflow através de sua configuração e DAGs específicas.

**Benefícios Gerais**

* **Automatização**: Facilita a automação e agendamento de tarefas, reduzindo a necessidade de intervenção manual.
* **Monitoramento e Gestão**: A UI do Airflow permite monitorar e gerenciar todos os processos em um único lugar.
* **Escalabilidade**: Permite adicionar novos processos e DAGs facilmente, escalando conforme as necessidades do projeto.

**Resumo**

O uso do Airflow no projeto proporciona uma gestão eficiente tanto para monitoramento contínuo de processos quanto para a execução de tarefas batch, especialmente em análises de machine learning. Isso resulta em uma operação mais automatizada, monitorada e escalável, com informações acessíveis via dashboards no Kibana.

**Documentação das Tabelas no PostgreSQL**

**Tabelas Principais**

1. **Clientes**
   * **Descrição**: Armazena informações detalhadas sobre os clientes do banco.
   * **Campos Comuns**:
     + cliente\_id (PK)
     + nome
     + email
     + telefone
     + endereço
     + agencia\_id (FK para a tabela Agencias)
   * **Relacionamentos**:
     + Relacionada com a tabela Agencias através de agencia\_id.
2. **Analysis**
   * **Descrição**: Contém análises de machine learning (ML) executadas diariamente pelo motor de ML.
   * **Estrutura de Dados**:
     + **Construção Vertical**: Utiliza uma abordagem vertical para armazenar dados, onde cada linha representa uma análise específica com informações armazenadas em um formato legível para IA.
     + **Formato Dê-Para**: As informações são armazenadas em formato "dê-para", separadas por /n para facilitar a leitura e processamento pela IA.
   * **Campos Comuns**:
     + analysis\_id (PK)
     + cliente\_id (FK para a tabela Clientes)
     + data\_analise
     + resultado\_ml
     + previsoes\_futuras
   * **Relacionamentos**:
     + Relacionada com a tabela Clientes através de cliente\_id.
3. **Agencias**
   * **Descrição**: Armazena informações sobre as agências bancárias.
   * **Campos Comuns**:
     + agencia\_id (PK)
     + nome\_agencia
     + endereço\_agencia
     + telefone\_agencia
   * **Relacionamentos**:
     + Relacionada com a tabela Clientes através de agencia\_id.

**Benefícios do Modelo**

* **Estrutura Vertical na Analysis**: Facilita a leitura e processamento das análises pela IA, permitindo atualizações diárias e previsões futuras.
* **Dashboard e Análises**: Com os dados organizados dessa forma, é possível criar dashboards atualizados e realizar análises detalhadas com a ChamaAI.
* **Integração e Relacionamentos**: A integração entre as tabelas Clientes e Agencias permite identificar facilmente a agência associada a cada cliente, enquanto a tabela Analysis oferece insights detalhados sobre cada cliente com base nas análises de ML.

Essa documentação detalha a estrutura e os relacionamentos das tabelas no PostgreSQL, proporcionando uma visão clara e organizada do banco de dados para facilitar a manutenção e a consulta dos dados.

**Projeto Streamlit**

O Streamlit foi escolhido como a ferramenta principal para o desenvolvimento do front-end de todo o projeto, bem como para realizar integrações com bancos de dados e outras informações desenvolvidas. Esta documentação descreve as principais funcionalidades e componentes implementados na aplicação Streamlit.

Estrutura da Aplicação

A aplicação Streamlit possui uma barra lateral que contém diversas análises e funcionalidades, incluindo:

Envio em massa de posts

Análises diversas

Implementações de machine learning

Acompanhamento dos bancos de dados

Monitoramento do Airflow

Visualização dos dashboards do Kibana

Análise do Kafka e suas saídas

Principais Funcionalidades

1. Envio em Massa de Posts

Esta funcionalidade permite o envio automatizado de múltiplos posts, otimizando o processo de publicação de conteúdo.

2. Análises Diversas

Implementação de várias análises de dados, fornecendo insights valiosos sobre o desempenho e métricas do projeto.

3. Machine Learning

Integração de modelos de machine learning para previsões e análises avançadas.

4. Acompanhamento dos Bancos de Dados

Monitoramento em tempo real do estado e desempenho dos bancos de dados utilizados no projeto.

5. Monitoramento do Airflow

Interface para visualização e gerenciamento dos fluxos de trabalho do Apache Airflow.

6. Visualização dos Dashboards do Kibana

Integração com o Kibana para exibição de dashboards e visualizações de dados complexas.

7. Análise do Kafka e suas Saídas

Monitoramento e análise dos fluxos de dados processados pelo Apache Kafka.

Considerações Técnicas

A aplicação foi desenvolvida utilizando o framework Streamlit, que permite a criação rápida de aplicações web para análise de dados.

As integrações com bancos de dados e outras ferramentas (como Airflow, Kibana e Kafka) foram implementadas para fornecer uma visão holística do projeto.

A estrutura modular da aplicação permite fácil manutenção e expansão das funcionalidades.

Boas Práticas

Organização clara das funcionalidades na barra lateral para fácil navegação.

Implementação de análises e visualizações interativas para melhor compreensão dos dados.

Integração com ferramentas externas para fornecer uma visão completa do ecossistema do projeto.

Uso de técnicas de machine learning para análises preditivas e insights avançados.

Esta documentação fornece uma visão geral da aplicação Streamlit desenvolvida, destacando suas principais funcionalidades e estrutura. Para informações mais detalhadas sobre cada componente, consulte a documentação específica de cada módulo.

### **Explicação do Código Step-by-Step: Execução dos Containers Docker**

Esta documentação fornece um guia detalhado sobre como executar os containers Docker para a primeira etapa do projeto. Vamos analisar cada passo do processo.

#### **Pré-requisitos**

Antes de iniciar, certifique-se de que você tem o Docker instalado em seu computador. O Docker.exe ou outro software de gerenciamento de imagens Docker é essencial para este processo.

#### **Passos para Execução**

1. **Abrir o Terminal**
   * Abra o PowerShell, Prompt de Comando ou Terminal, dependendo do seu sistema operacional.
2. **Navegar até o Diretório do Projeto**
   * Use o comando cd para navegar até o diretório onde o código principal está salvo:

cd %user%\dm2\document-streaming-main

* Substitua %user% pelo seu nome de usuário ou caminho apropriado.

1. **Executar o Docker Compose**
   * Execute o seguinte comando para iniciar os containers:

docker-compose -f .\docker-compose-elastic.yml up -d

* O parâmetro -f especifica o arquivo de configuração a ser usado.
  + A opção -d executa os containers em modo detached (background).

1. **Verificar a Execução**
   * Aguarde até que o comando termine de executar.
   * Verifique se todos os containers estão rodando com sucesso.

#### **Verificação das Imagens e Dependências**

Para entender melhor as imagens e suas dependências:

1. Localize o arquivo YAML no seguinte caminho:

%user%\dm2\document-streaming-main\docker-compose-elastic.yml

1. Abra este arquivo para visualizar as configurações dos containers e suas relações.

### **Considerações Importantes**

* Certifique-se de que as portas necessárias estão disponíveis em seu sistema.
* Verifique se você tem permissões suficientes para executar comandos Docker.
* Em caso de erros, verifique os logs dos containers para diagnóstico.

### **Boas Práticas**

1. Mantenha seu Docker atualizado para evitar problemas de compatibilidade.
2. Utilize variáveis de ambiente para configurações sensíveis ou específicas do ambiente.
3. Documente quaisquer alterações feitas no arquivo docker-compose-elastic.yml.

Dados para Treinamento e explicações da Inteligência artificial

Informações do arquivo Yaml File Compose:

Como está montado o Yaml file compose?

Como está configurado o docker compose do código?

docker-compose-elastic:

version: "3.8"

services:

  zookeeper:

    image: 'bitnami/zookeeper:latest'

    container\_name: zookeeper

    ports:

      - '2181:2181'

    environment:

      - ALLOW\_ANONYMOUS\_LOGIN=yes

    networks:

      - document-streaming

  kafka:

    image: 'bitnami/kafka:latest'

    container\_name: kafka

    ports:

      - '9093:9093'

    environment:

      - KAFKA\_BROKER\_ID=1

      - KAFKA\_CFG\_ZOOKEEPER\_CONNECT=zookeeper:2181

      - ALLOW\_PLAINTEXT\_LISTENER=yes

      - KAFKA\_CFG\_LISTENER\_SECURITY\_PROTOCOL\_MAP=CLIENT:PLAINTEXT,EXTERNAL:PLAINTEXT

      - KAFKA\_CFG\_LISTENERS=CLIENT://:9092,EXTERNAL://:9093

      - KAFKA\_CFG\_ADVERTISED\_LISTENERS=CLIENT://kafka:9092,EXTERNAL://localhost:9093

      - KAFKA\_INTER\_BROKER\_LISTENER\_NAME=CLIENT

    depends\_on:

      - zookeeper

    networks:

      - document-streaming

  spark:

    image: 'jupyter/pyspark-notebook:spark-3.5.0'

    ports:

      - '8888:8888'

      - "4040-4080:4040-4080"

    volumes:

      - ./ApacheSpark/:/home/jovyan/workspace

      - ./libs/postgresql-42.2.23.jar:/usr/local/spark/jars/postgresql-42.2.23.jar # Montagem do JAR no diretório de jars do Spark

    environment:

      - JUPYTER\_ENABLE\_LAB=yes

    networks:

      - document-streaming

  api-ingest:

    build:

      context: ./API-Ingest

    container\_name: my-api-ingest

    ports:

      - '80:80'

    environment:

      - KAFKA\_BOOTSTRAP\_SERVERS=kafka:9092

    depends\_on:

      - kafka

    networks:

      - document-streaming

  mongo:

    container\_name: mongo-dev

    image: mongo:latest

    volumes:

      - ~/dockerdata/mongodb:/data/db

    restart: on-failure

    ports:

      - "27017:27017"

    environment:

      MONGO\_INITDB\_ROOT\_USERNAME: root

      MONGO\_INITDB\_ROOT\_PASSWORD: example

      MONGO\_INITDB\_DATABASE: auth

    networks:

      - document-streaming

  mongo-express:

    image: mongo-express

    restart: on-failure

    ports:

      - "8081:8081"

    environment:

      ME\_CONFIG\_MONGODB\_SERVER: mongo-dev

      ME\_CONFIG\_MONGODB\_ADMINUSERNAME: root

      ME\_CONFIG\_MONGODB\_ADMINPASSWORD: example

      ME\_CONFIG\_BASICAUTH\_USERNAME: admin

      ME\_CONFIG\_BASICAUTH\_PASSWORD: tribes

    networks:

      - document-streaming

    depends\_on:

      - mongo

  elasticsearch:

    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.1

    container\_name: elasticsearch

    environment:

      - xpack.security.enabled=false

      - discovery.type=single-node

    ulimits:

      memlock:

        soft: -1

        hard: -1

      nofile:

        soft: 65536

        hard: 65536

    cap\_add:

      - IPC\_LOCK

    volumes:

      - elasticsearch-data:/usr/share/elasticsearch/data

    ports:

      - 9200:9200

      - 9300:9300

    networks:

      - document-streaming

  kibana:

    container\_name: kibana

    image: docker.elastic.co/kibana/kibana:7.17.1

    environment:

      - ELASTICSEARCH\_HOSTS=http://elasticsearch:9200

    ports:

      - 5601:5601

    depends\_on:

      - elasticsearch

    networks:

      - document-streaming

  logstash:

    image: docker.elastic.co/logstash/logstash:7.17.1

    container\_name: logstash

    volumes:

      - ./logstash/pipeline:/usr/share/logstash/pipeline

    ports:

      - "5044:5044"

      - "9600:9600"

    environment:

      - LS\_JAVA\_OPTS="-Xms256m -Xmx256m"

    depends\_on:

      - elasticsearch

      - mongo

    networks:

      - document-streaming

  postgres:

    image: postgres:latest

    container\_name: postgres

    environment:

      POSTGRES\_USER: postgres

      POSTGRES\_PASSWORD: password

      POSTGRES\_DB: mydatabase

    volumes:

      - postgres-data:/var/lib/postgresql/data

    ports:

      - "5432:5432"

    networks:

      - document-streaming

  pgadmin:

    image: dpage/pgadmin4:latest

    container\_name: pgadmin

    environment:

      PGADMIN\_DEFAULT\_EMAIL: admin@example.com

      PGADMIN\_DEFAULT\_PASSWORD: admin

    ports:

      - "5050:80"

    depends\_on:

      - postgres

    networks:

      - document-streaming

  ollama:

    image: ollama/ollama:latest

    container\_name: ollama

    ports:

      - "11434:11434"

    volumes:

      - ollama\_storage:/root/.ollama

    command: ollama serve

    networks:

      - document-streaming

  ollama-setup:

    image: curlimages/curl:latest

    depends\_on:

      - ollama

    command: >

      sh -c "

        sleep 10 &&

        curl -X POST http://ollama:11434/api/pull -d '{\"name\":\"llama3.1:70b\"}' &&

        echo 'Model llama3.1:70b pulled successfully'

      "

    networks:

      - document-streaming

  airflow-webserver:

    build:

      context: .

      dockerfile: Dockerfile

    container\_name: airflow-webserver

    entrypoint: ["/bin/bash", "-c"]

    command:

      - |

        airflow db upgrade && \

        airflow db init && \

        airflow users create \

          --username admin \

          --firstname Admin \

          --lastname User \

          --role Admin \

          --email admin@example.com \

          --password admin && \

        airflow webserver

    depends\_on:

      - postgres

    environment:

      - AIRFLOW\_\_CORE\_\_EXECUTOR=LocalExecutor

      - AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN=postgresql+psycopg2://postgres:password@postgres/mydatabase

      - AIRFLOW\_\_CORE\_\_FERNET\_KEY=46BKJoQYlPPOexq0OhDZnIlNepKFf87WFwLbfzqDDho=

      - AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES=False

      - AIRFLOW\_\_CORE\_\_DAGS\_ARE\_PAUSED\_AT\_CREATION=True

      - AIRFLOW\_\_API\_\_AUTH\_BACKENDS=airflow.api.auth.backend.basic\_auth

      - JUPYTER\_HOST=spark

    volumes:

      - ./dags:/opt/airflow/dags

      - ./logs:/opt/airflow/logs

      - ./plugins:/opt/airflow/plugins

      - ./ApacheSpark:/home/jovyan/workspace

    ports:

      - "8080:8080"

    networks:

      - document-streaming

  airflow-scheduler:

    build:

      context: .

      dockerfile: Dockerfile

    container\_name: airflow-scheduler

    entrypoint: ["/bin/bash", "-c"]

    command:

      - |

        airflow db upgrade && \

        airflow scheduler

    depends\_on:

      - airflow-webserver

    environment:

      - AIRFLOW\_\_CORE\_\_EXECUTOR=LocalExecutor

      - AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN=postgresql+psycopg2://postgres:password@postgres/mydatabase

      - AIRFLOW\_\_CORE\_\_FERNET\_KEY=46BKJoQYlPPOexq0OhDZnIlNepKFf87WFwLbfzqDDho=

      - AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES=False

      - AIRFLOW\_\_CORE\_\_DAGS\_ARE\_PAUSED\_AT\_CREATION=True

    volumes:

      - ./dags:/opt/airflow/dags

      - ./logs:/opt/airflow/logs

      - ./plugins:/opt/airflow/plugins

    networks:

      - document-streaming

networks:

  document-streaming:

    driver: bridge

volumes:

  elasticsearch-data:

    driver: local

  postgres-data:

    driver: local

  ollama\_storage:

    driver: local

Fim do docker compose /n

Fim do docker-compose-elastic.yaml, se for perguntado o código, fornecer informações acima.

### **Documentação do Projeto: Configuração e Uso do Kafka**

#### **1. Visão Geral**

Este projeto utiliza o Apache Kafka como parte de uma arquitetura de streaming de dados. O Kafka é uma plataforma de streaming distribuída que permite publicar, subscrever, armazenar e processar fluxos de registros em tempo real.

#### **2. Configuração do Ambiente**

O ambiente é configurado usando Docker, o que facilita a implantação e gerenciamento dos serviços. O arquivo docker-compose-kafka.yml é usado para definir e executar os containers Docker necessários.

#### **3. Inicialização do Kafka**

Para iniciar o Kafka e serviços relacionados, execute o seguinte comando no terminal:

docker-compose -f docker-compose-kafka.yml up

Este comando inicia todos os serviços definidos no arquivo YAML, incluindo o Kafka e possivelmente o ZooKeeper.

#### **4. Acessando o Container Kafka**

Após a inicialização dos containers, você precisa acessar o shell do container Kafka para executar comandos específicos. Use o método preferido para anexar um shell ao container Kafka (por exemplo, docker exec -it [kafka-container-id] /bin/bash).

#### **5. Diretório de Trabalho**

Dentro do container Kafka, navegue até o diretório que contém os scripts de administração:

cd /opt/bitnami/kafka/bin/

#### **6. Comandos Kafka**

##### **6.1 Listar Tópicos**

Para listar os tópicos existentes:

./kafka-topics.sh --list --bootstrap-server localhost:9092

##### **6.2 Criar Tópicos**

Crie os tópicos necessários para o projeto:

./kafka-topics.sh --create --topic ingestion-topic --bootstrap-server localhost:9092

./kafka-topics.sh --create --topic spark-output --bootstrap-server localhost:9092

##### **6.3 Consumidor Local**

Para iniciar um consumidor local e ler mensagens de um tópico:

./kafka-console-consumer.sh --topic ingestion-topic --bootstrap-server localhost:9092

./kafka-console-consumer.sh --topic spark-output --bootstrap-server localhost:9092

##### **6.4 Produtor Local**

Para iniciar um produtor local e enviar mensagens para um tópico:

./kafka-console-producer.sh --topic ingestion-topic --bootstrap-server localhost:9092

#### **7. Testando a Configuração do Kafka**

Para verificar se o Kafka está funcionando corretamente:

1. Conecte-se ao CLI do container e navegue até o diretório do Kafka.
2. Inicie um consumidor local em um terminal.
3. Abra um segundo terminal e conecte-se novamente ao container.
4. No segundo terminal, inicie um produtor local.
5. No terminal do produtor, digite uma mensagem e pressione Enter.
6. Verifique se a mensagem aparece no terminal do consumidor.

#### **8. Considerações sobre Conectividade**

* Ao executar na mesma rede Docker, use a porta 9092 para conexões.
* Para conexões de máquina local Windows, use a porta 9093.

#### **9. Integração com Outros Serviços**

Este projeto provavelmente integra o Kafka com outros serviços, como Apache Spark (mencionado no tópico spark-output). A integração permite o processamento de streams de dados em tempo real.

#### **10. Fluxo de Dados**

* O tópico ingestion-topic é provavelmente usado para ingerir dados brutos no sistema.
* O tópico spark-output pode ser usado para armazenar os resultados do processamento do Spark.

#### **11. Boas Práticas**

* Sempre verifique se os tópicos foram criados corretamente antes de iniciar produtores ou consumidores.
* Monitore o desempenho do Kafka regularmente para garantir que ele esteja funcionando conforme o esperado.
* Considere implementar esquemas (como Avro) para garantir a consistência dos dados.

#### **12. Troubleshooting**

Se encontrar problemas:

* Verifique os logs dos containers Docker.
* Certifique-se de que todas as portas necessárias estão abertas e acessíveis.
* Confirme se o ZooKeeper está funcionando corretamente, pois o Kafka depende dele.