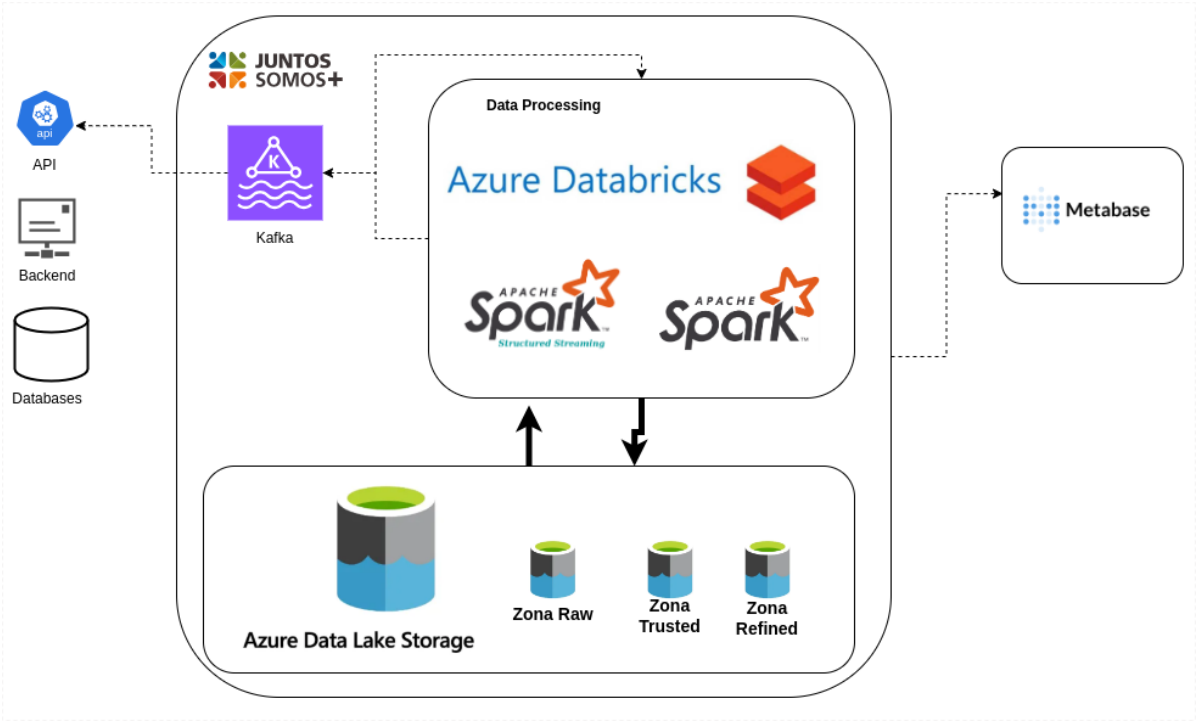
Proposta de Preliminar de Arquitetura e Desenvolvimento para Ingestão de Dados

Arariglenо Fernandes

Esta proposta preliminar apresenta uma arquitetura de dados em tempo real destinada à squad de pedidos, utilizando o Azure Databricks como plataforma central de processamento e armazenamento de dados. O objetivo é fornecer uma solução escalável e eficiente, capaz de processar grandes volumes de dados transacionais em tempo real, integrando ferramentas como Kafka para ingestão de dados e Metabase para visualização. A proposta abrange desde a ingestão de dados até o armazenamento e análise, oferecendo uma base sólida para o desenvolvimento subsequente pela equipe, com a possibilidade de ajustes conforme necessário para atender às demandas específicas do projeto.



Descrição

1. Fontes de Dados

- API e Backend: A arquitetura captura dados de uma API e de sistemas de backend que alimentam o fluxo de dados transacionais.
- Databases: Dados também são obtidos de bancos de dados existentes.

2. Ingestão de Dados

- Kafka: O Kafka atua como o middleware para ingestão de dados em tempo real, capturando eventos e dados de diversas fontes, incluindo a API e os sistemas de backend.

3. Processamento de Dados
- Azure Databricks: O Azure Databricks é o núcleo do processamento de dados na arquitetura, utilizando o Apache Spark e o Spark Structured Streaming para processar dados em tempo real.
  - Data Processing: No Databricks, os dados são processados em tempo real, utilizando pipelines que limpam, transformam e agregam os dados conforme necessário.

4. Armazenamento de Dados
- Azure Data Lake Storage: Após o processamento no Databricks, os dados são armazenados no Azure Data Lake Storage, que é dividido em três zonas:
    - Zona Raw: Armazena os dados brutos, diretamente ingeridos sem qualquer transformação.
  - Zona Trusted: Armazena os dados que foram processados e estão prontos para uso por outras aplicações.
  - Zona Refined: Contém os dados refinados, que passaram por mais transformações e estão prontos para análises e relatórios detalhados.

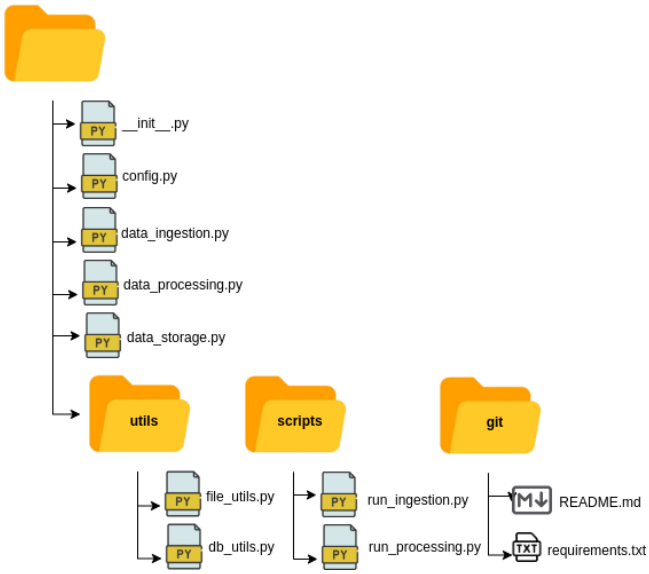
5. Visualização e Análise
- Metabase: Os dados refinados podem ser visualizados e analisados utilizando o Metabase, que se conecta ao Azure Data Lake Storage para fornecer dashboards e relatórios baseados nos dados processados.

6. Integração e Comunicação
- A arquitetura está projetada para integrar diversos componentes de forma eficiente, permitindo a comunicação fluida entre APIs, Kafka, Databricks, Azure Data Lake, e Metabase.

Considerações
Esta arquitetura permite o processamento de dados em tempo real, garantindo que a squad possa trabalhar com informações atualizadas e precisas para tomadas de decisão rápidas. Ela também é flexível o suficiente para escalar conforme o volume de dados e a complexidade das análises aumentam.

Projeto Desenvolvimento

Esta estrutura de projeto representa uma proposta preliminar para a organização de um projeto Python voltado ao desenvolvimento de processos de ingestão, processamento e armazenamento de dados. A estrutura aqui apresentada visa fornecer um ponto de partida para a implementação do projeto, mas poderá ser ajustada conforme as necessidades específicas da equipe e os requisitos técnicos sejam melhor compreendidos ao longo do desenvolvimento.



Etapa do processo:

1. Diretório Raiz (my_data_project/)
   - Este é o diretório principal que contém todos os arquivos e subdiretórios do projeto. É a estrutura base onde o código, as bibliotecas e a documentação são organizados.

2. Arquivos e Diretórios Dentro do Diretório Principal
- `__init__.py`: Marca o diretório como um pacote Python, permitindo que os arquivos dentro dele sejam importados como módulos.

- `config.py`: Contém as configurações e variáveis globais do projeto, como credenciais, caminhos de diretórios, e outras configurações necessárias para o funcionamento do sistema.

- `data_ingestion.py`: Script responsável por ingerir dados de diferentes fontes (bancos de dados, arquivos CSV, APIs, etc.) e armazená-los em uma área de preparação ou processamento.

- `data_processing.py`: Script que contém as rotinas de processamento dos dados ingeridos, como transformações, limpeza, e preparação dos dados para análise ou armazenamento em um formato refinado.

- `data_storage.py`: Script que lida com o armazenamento final dos dados processados, seja em bancos de dados, sistemas de arquivos, ou data lakes.

- `utils/`: Diretório que contém funções auxiliares e utilitárias que são utilizadas em diferentes partes do projeto. Exemplos incluem funções para manipulação de arquivos e conexões com bancos de dados.

- `file_utils.py`: Contém funções utilitárias para operações com arquivos, como leitura, escrita, e manipulação de diretórios.

- `db_utils.py`: Contém funções para interações com bancos de dados, como conexões, consultas, e transações.

- `scripts/`: Diretório que contém scripts de automação ou execução do projeto.

- `run_ingestion.py`: Script para executar o processo de ingestão de dados, chamando as funções definidas em `data_ingestion.py`.

- `run_processing.py`: Script para executar o processo de processamento de dados, utilizando as funções de `data_processing.py`.

- `git/`: Diretório usado para armazenar informações relacionadas ao controle de versão do projeto com Git.

- `README.md`: Arquivo de documentação do projeto, explicando como configurar, executar e contribuir para o projeto.

- `requirements.txt`: Lista de dependências do projeto que precisam ser instaladas para o ambiente de desenvolvimento, normalmente usadas com o gerenciador de pacotes `pip`.

Cada um desses componentes desempenha um papel crucial na organização e funcionalidade do projeto, garantindo que o código seja modular, reutilizável e fácil de manter. A estrutura segue boas práticas de desenvolvimento, permitindo a escalabilidade e a colaboração em equipe.