12 DE NOVEMBRO DE 2025

PROJETO DE ENGENHARIA DE DADOS
RAIO-X DA FROTA DA SPTRANS

Grupo 10
Guilherme Pereira Palmeira – RA: 25002998
Luiza Assis Pereira – RA: 25001987

Sumário

1. Projeto de Engenharia de Dados: Raio-X da Frota SPTrans

Este documento detalha a arquitetura de dados construída para capturar, processar e analisar em tempo quase real e em lote os dados de posicionamento dos ônibus da cidade de São Paulo, fornecidos pela API "Olho Vivo" da SPTrans.

2. Objetivo do Projeto

O objetivo principal deste projeto é construir uma plataforma de dados ponta a ponta para transformar dados brutos e efêmeros da API “Olho Vivo” da SPTrans, em insights persistentes e acionáveis.

A arquitetura foi projetada para superar as limitações da API de origem (que oferece apenas uma “foto” do momento) e responder algumas perguntas de negócio:

1. **Visão Estratégica (Histórica):** Como a performance da frota se compara entre diferentes horários? Quantos ônibus estão em congestionamentos entre diferentes horários?

2. **Visão Operacional (Tempo Quase Real):** Como está a performance da frota agora? Qual a velocidade média da frota agora? Quantos ônibus estão em congestionamentos agora?

3. Resultados Entregues

Como resultado, a arquitetura proporciona uma plataforma de análise de dados unificada, que entrega valor através de três componentes principais:

- **Dashboards de Business Intelligence (Metabase)**

    - **Dashboards de Tempo Quase Real:** Fornece uma visão operacional para monitoramento imediato, com KPIs críticos (Velocidade Média da Frota, Total da Frota Ativa, Total de Ônibus Congestionados) e um mapa de posições dos veículos atualizado a cada poucos minutos.

    - **Dashboards de Análise Histórica:** Apresenta uma visão precisa dos dados consolidados pelo pipeline de lote (batch). Permite a análise de tendências, como o volume de operação hora a hora (Gráfico de Linha) e a distribuição da quantidade de ônibus por linha (Mapa de Calor).

- **API de Dados (FastAPI)**

Expõe os principais KPIs e dados de posição através de endpoints RESTful. Isso permite que outras aplicações ou sistemas consumam os insights da plataforma de forma programática, desacoplando o consumo de dados do acesso direto ao banco de dados.

- **Data Lakehouse Centralizado (MinIO + Delta Lake)**

A Camada Gold no MinIO serve como uma fonte única da verdade, com dados limpos e agregados, pronta para alimentar futuros projetos de Data Science, Machine Learning ou outras iniciativas de análise avançada.

- **Acesso aos Dados de Forma Federada (Trino)**

Disponibiliza o acesso às tabelas Fato (Camada Gold do Lakehouse) e ao Postgres, possibilitando que Cientistas de Dados e camadas de visualização tenham acesso às principais tabelas da plataforma para realizar análises e consultas exploratórias.

4. Desenho da Arquitetura

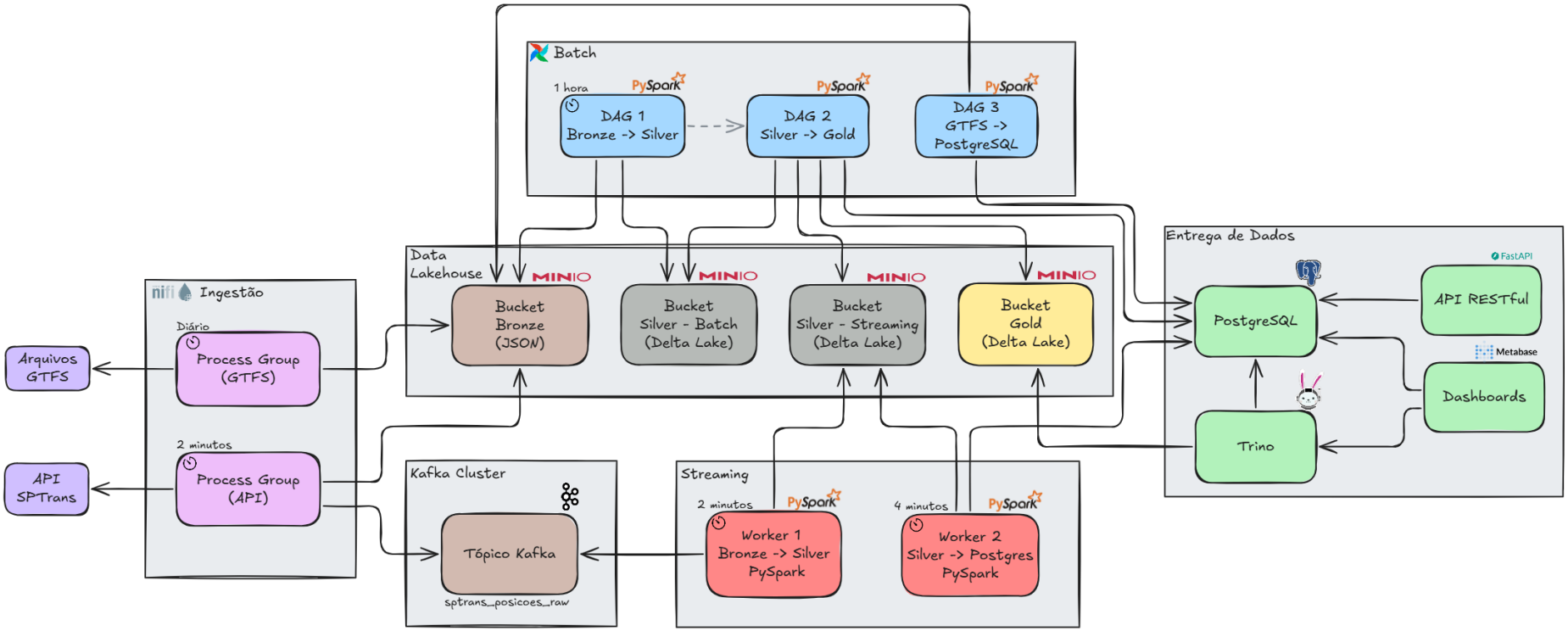
A solução implementa uma Arquitetura Lakehouse Medalhão que segue o padrão ELT (Extract, Load, Transform), projetada para fornecer tanto insights operacionais com baixa latência (streaming), quanto análises históricas (batch).

O objetivo principal é transformar o fluxo contínuo e efêmero de dados da API da SPTrans em um ativo de dados persistente e confiável, pronto para análise.

Ambos os caminhos (streaming e batch) processam os dados, com a Camada Gold do Lakehouse (MinIO + Delta Lake) atuando como a fonte única da verdade para os dados analíticos e históricos. Por fim, os dados processados (tanto históricos quanto de tempo real) são disponibilizados em uma Camada de Entrega de Dados, composta por um Data Warehouse em PostgreSQL.

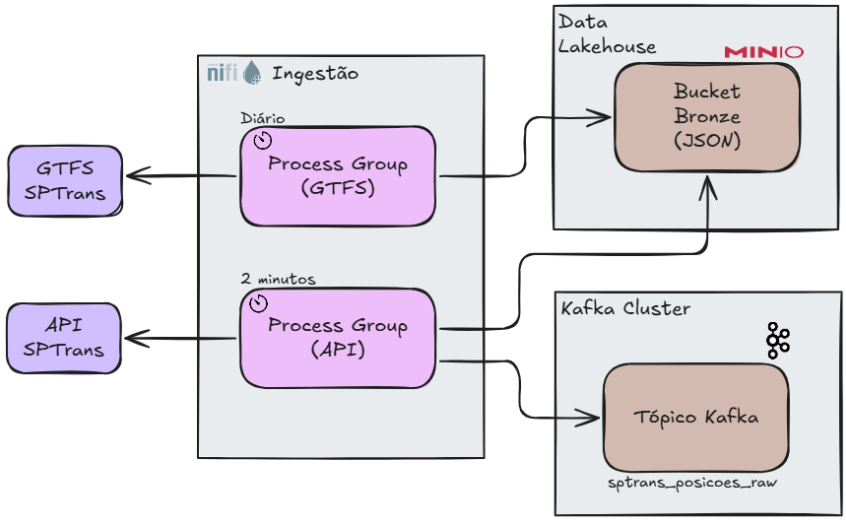
O resultado é uma plataforma unificada que entrega esses insights através de dashboards interativos no Metabase, uma API RESTful (FastAPI) e de forma federada através do Trino, fornecendo uma visão da operação da frota de ônibus.

A seguir, detalharemos cada parte da arquitetura:



4.1 Ingestão de Dados

A ingestão é o ponto de entrada dos dados no nosso ecossistema, responsável por capturar os dados brutos e direcioná-los para as camadas iniciais.



Tecnologia: Apache NiFi

Foi escolhido por ser uma ferramenta visual, low code e confiável para a automação e ingestão de fluxos de dados. Sua interface gráfica facilita a criação e o monitoramento de pipelines de ingestão, e sua capacidade de interagir com múltiplos sistemas (APIs, HDFS/S3, Kafka e etc) o torna ideal para a nossa arquitetura de ingestão dupla.

Fluxo de Dados: O NiFi gerência dois fluxos de ingestão distintos:

- **Dados Cadastrais (GTFS):** Um fluxo, executado diariamente, é responsável por copiar os arquivos estáticos do padrão GTFS (como o `routes.txt`) para a Camada Bronze. Estes arquivos fornecem os dados de enriquecimento (ex: nomes das linhas) que serão usados nos pipelines de transformação.

- **Dados de Posição:** Um segundo fluxo consulta a API `/Posicoes` da SPTrans a cada 2 minutos. A resposta do JSON completa é enviada simultaneamente para dois destinos:

    - **Apache Kafka:** Para alimentar o pipeline de streaming.

    - **MinIO (Camada Bronze):** Para persistência histórica e alimentação do pipeline de batch.

4.2 Data Lakehouse

Parte central do nosso armazenamento de dados, implementado com o padrão Medalhão (Medallion Architecture).

Tecnologias:

MinIO: Fornece um armazenamento de objetos escalável e compatível com o ecossistema Hadoop/S3, que é o padrão para Data Lakes modernos.

Delta Lake: Adiciona uma camada de confiabilidade sobre o MinIO, trazendo transações ACID, o que previne a corrupção de dados. Funções como MERGE e versionamento de dados facilitam o gerenciamento dos pipelines Batch e Streaming.

Estrutura (Medalhão):

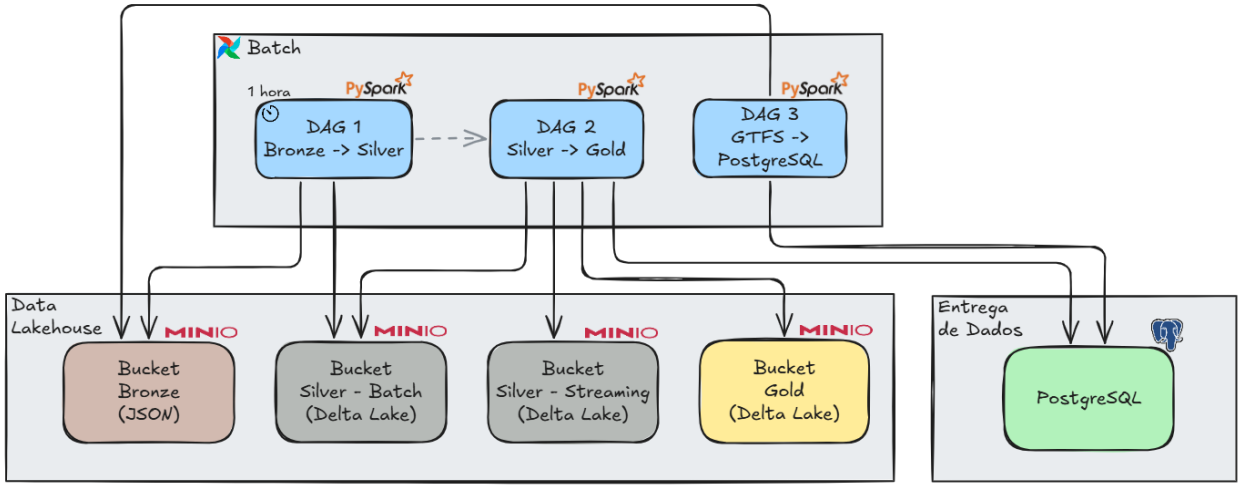
- **Bronze:** Armazena os dados brutos e imutáveis, da forma como são ingeridos, servindo como arquivo histórico.

- **Silver**: Armazena os dados limpos, com tipos corrigidos e em um formato tabular (Tabelas Delta), prontos para serem consumidos. Como nossa arquitetura suporta dois pipelines, esta camada contém três tabelas principais: `posicoes_onibus` e `kpis_historicos_para_processar` (para o batch) e `posicoes_onibus_streaming` (para o streaming).

- **Gold:** É a nossa camada de negócio. Para dados históricos e analíticos (`fato_operacao_linhas_hora`, `fato_velocidade_linha` e `fato_onibus_parados_linha`), esta camada no MinIO é a fonte única da verdade. Para dados de estado e KPIs de tempo real (como `nrt_posicao_onibus_atual`, `nrt_velocidade_linha` e `nrt_onibus_parados_linha`), a fonte da verdade é o PostgreSQL, para garantir a performance do streaming.

4.3 Processamento em Lote (Batch)

Este pipeline é responsável por criar a visão histórica dos dados, garantindo a precisão das informações.



Tecnologias:

Apache Airflow: É o orquestrador do pipeline de lote. Ele é o responsável por agendar a execução dos jobs, gerenciar as dependências (a DAG `silver_to_gold` só começa quando a DAG `bronze_to_silver` termina) e lidar com retentativas em casos de falhas.

Apache Spark: É a ferramenta de processamento distribuído que executa a lógica de negócio em si. Por ser um padrão bastante utilizado no mercado e sua capacidade de processar grandes volumes com eficiência e de forma paralela, fez com que fosse a escolha mais sensata.

Fluxo:

Duas DAGs principais rodam de hora em hora:

- **bronze_to_silver_dag.py:** Esta DAG aciona um job Spark (`bronze_to_silver_batch.py`) que lê os arquivos JSON brutos da Camada Bronze, os transforma e salva como uma Tabela Delta limpa na Camada Silver (`posicoes_onibus`).

- **silver_to_gold_dag.py:** Esta DAG aciona um segundo job Spark (`silver_to_gold_batch.py`) que:
    - **Consolida a Contagem:** Lê os dados da `posicoes_onibus` (Silver) da hora anterior, calcula a contagem definitiva de ônibus por linha e faz o MERGE na tabela `fato_operacao_linhas_hora` no Lakehouse (MinIO).

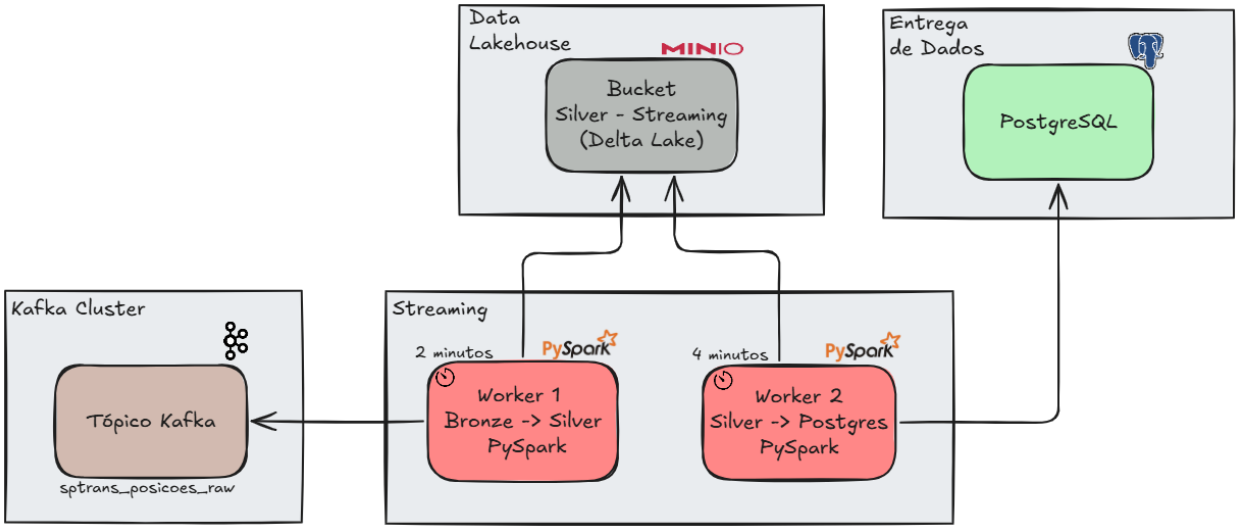
    - **Consolida os KPIs:** Lê os dados da tabela `kpis_historicos_para_processar` (Silver) da hora anterior (que foram gerados pelo streaming) e faz o MERGE nas tabelas `fato_velocidade_linha` e `fato_onibus_parados_linha` no Lakehouse (MinIO).

    - **Carrega a Camada de Servir:** Como passo final, o job grava as tabelas de fatos históricas (`fato_operacao_linhas_hora`, `fato_velocidade_linha` e `fato_onibus_parados_linha`) do PostgreSQL com os mesmos dados gravados no MinIO Gold, garantindo que a camada de servir tenha os dados condizentes com a camada histórica.

- **create_dimensions_dag.py:** Esta DAG aciona um job Spark (`create_dim_linha.py`) de forma diária, que lê o arquivo routes.txt disponibilizado via GTFS pela SPTrans e popula a tabela dim_linha.

4.4 Processamento em Tempo Quase Real (Streaming)

Esta é a Camada de Streaming da nossa arquitetura, responsável por fornecer os KPIs operacionais com menos latência.



Tecnologias:

Apache Kafka: Atua como buffer durável e escalável entre o produtor (NiFi) e o consumidor (Spark Streaming), permitindo que os sistemas operem de forma desacoplada. Se o pipeline de streaming falhar, os dados permanecem no Kafka, prontos para serem consumidos quando o pipeline voltar, garantindo que os dados não sejam perdidos.

Apache Spark Streaming: Solução de streaming nativa no ecossistema Spark. Sua integração com o Delta Lake permite a escrita de micro-lotes de forma transacional e confiável, além de gerenciar o estado através de checkpoints, o que cria resiliência contra falhas.

Fluxo: Dois workers (processos Spark) ficam em execução contínua:

- **bronze_to_silver_streaming.py:** A cada 2 minutos (conforme definido no trigger do job), consome as mensagens JSON do tópico do Kafka e escreve os dados limpos em uma Tabela Delta na camada Silver (`posicoes_onibus_streaming`).

- **silver_to_gold_streaming.py:** A cada 4 minutos (conforme o definido no trigger do job), lê os novos dados da Tabela Delta da camada Silver (`posicoes_onibus_streaming`) a e executa a seguinte lógica de negócio:

    - **Cálculo de KPIs:** Primeiro, o script compara os dados do lote atual com a "memória" de posições anterior (a tabela `nrt_posicao_onibus_atual`) diretamente do PostgreSQL, que é otimizado para leituras rápidas, para calcular a distância e o tempo decorrido para cada ônibus. Serão consideradas posições válidas os intervalos de posição maiores que 10 segundos. Com base nisso, as seguintes métricas são definidas:

- **Cálculo da Distância (Fórmula de Haversine):** Para determinar a distância percorrida por um ônibus entre duas atualizações de posição, o script utiliza a fórmula de Haversine. Esta é uma equação matemática precisa para calcular a distância entre pontos dadas as suas longitudes de latitudes. É a abordagem padrão para cálculos como este.

- **Velocidade Média da Linha:** Para evitar que ônibus parados ou com erros de GPS distorçam a métrica, a velocidade operacional é calculada usando o percentil 85 das velocidades de todos os ônibus de uma linha;

- **Ônibus em Congestionamento:** Um ônibus é considerado "parado" ou em congestionamento se atender a todas as seguintes regras:

    1) Sua velocidade média calculada (baseada na fórmula de Haversine) for inferior a 2.0km/h.
    2) A hora da captura não for o período da Madrugada (definido como 04:00 - 07:00 UTC, ou 01:00 - 04:00 BRT), para evitar ónibus estacionados na garagem e enviando sinal.

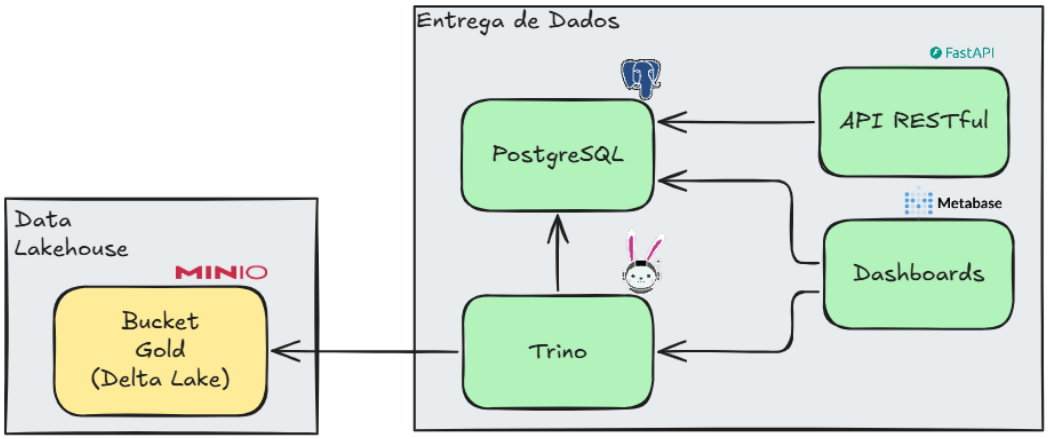
- **Atualização de Posições (PostgreSQL):** Atualiza a "memória" de posições, fazendo UPSERT (INSERT ... ON CONFLICT) da `nrt_posicao_onibus_atual` diretamente no PostgreSQL. Esta operação é rápida (segundos).

- **Atualização dos KPIs NRT (PostgreSQL):** Faz OVERWRITE (TRUNCATE + INSERT) dos KPIs sem `id_tempo` nas tabelas `nrt_velocidade_linha` e `nrt_onibus_parados_linha` no PostgreSQL, para alimentar os dashboards e API.

- **Persistência dos KPIs Históricos (Lakehouse Silver):** Os KPIs agregados com `id_tempo` são escritos (em modo append) em uma tabela Delta intermediária (`s3a://silver/kpis_historicos_para_processar/`). Esta operação é "write-only" e, portanto, bastante rápida, desacoplando o streaming da lentidão do MERGE na Camada Gold.

4.5 Entrega de Dados

É a camada final da arquitetura, projetada para entregar os insights processados aos usuários finais de forma rápida, intuitiva e acessível. Ela é composta por quatro componentes principais, destacados em verde.



4.5.1 Data Warehouse (PostgreSQL)

- **Papel:** Atua como nosso Data Warehouse. Ele armazena as tabelas de dimensão e as tabelas de fato já agregadas, otimizadas para consultas analíticas de baixa latência.

- **Função Tripla:**

  1) **Armazenamento de Estado:** É a fonte primária e o destino para a tabela de estado `nrt_posicao_onibus_atual`, permitindo UPSERTs de alta velocidade para o pipeline de streaming.

  2) **Camada de Consumo NRT:** É a fonte primária para as tabelas `nrt_velocidade_linha` e `nrt_onibus_parados_linha`, alimentando os dashboards de tempo quase real.

  3) **Camada de Consumo:** Armazena cópias otimizadas dos dados analíticos (como `fato_onibus_parados_linha`, `fato_velocidade_linha` e `fato_operacao_linhas_hora`) que são carregadas a partir da Camada Gold do Lakehouse (MinIO) pelo processamento Batch.

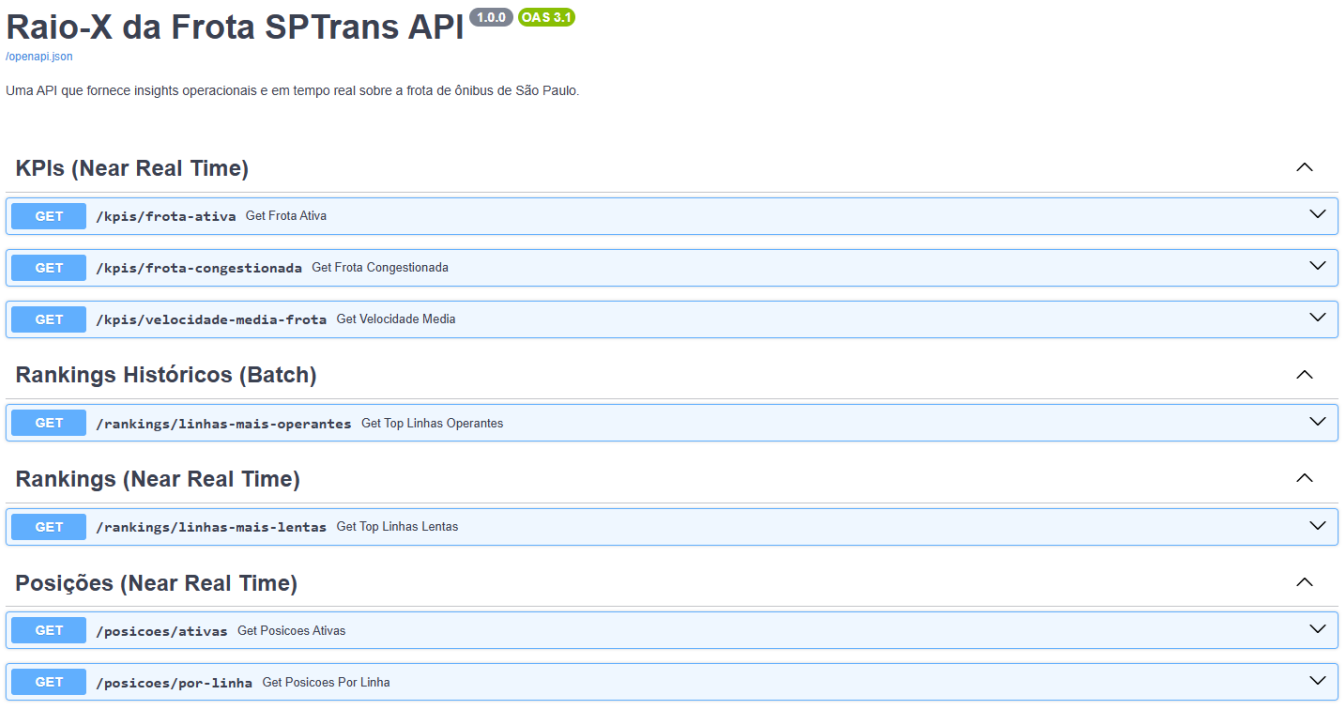
- **Fluxo:** O PostgreSQL é a fonte única de dados para a API, e uma das fontes de dados para os Dashboards no Metabase, junto com o Trino.

4.5.2 API RESTful (FastAPI)

- **Tecnologia:** FastAPI

- **Justificativa:** Escolhido por ser um framework Python moderno, de alta performance e fácil de usar para construir APIs RESTful.

- **Função:** Expõe os principais KPIs e dados de posição através de endpoints RESTful. Isso permite que outras aplicações ou sistemas consumam os insights da plataforma de forma programática, desacoplando o acesso direto ao banco de dados.



4.5.3 Visualização (Metabase)

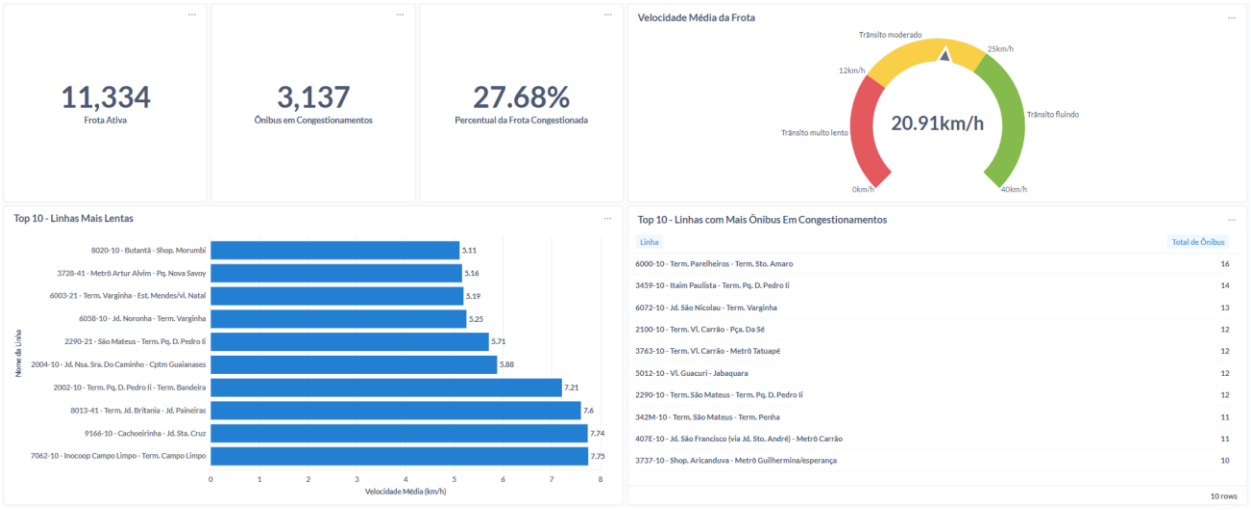
- **Tecnologia:** Metabase

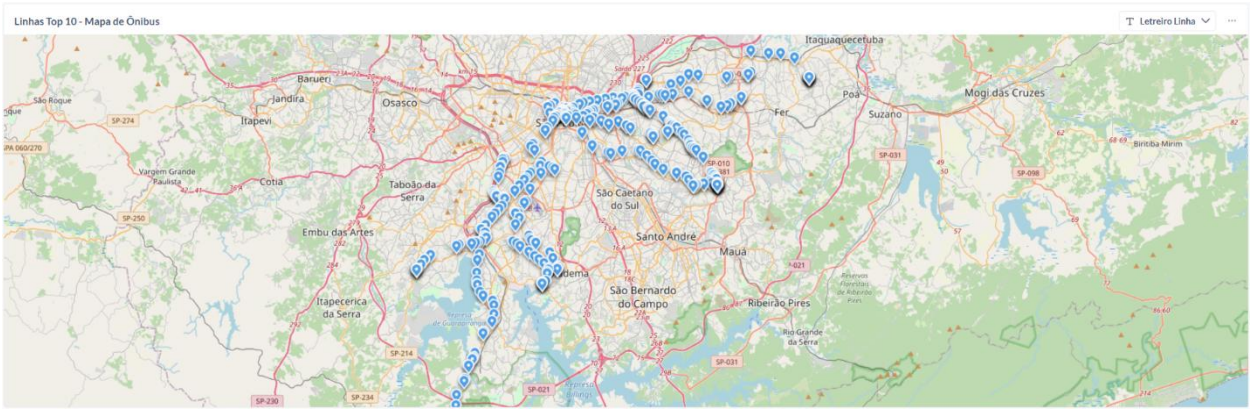
- **Justificativa:** Permite a criação de dashboards interativos e a exploração de dados de forma intuitiva, tanto por usuários técnicos (que podem escrever SQL) quanto por usuários de negócio (através do seu editor visual).

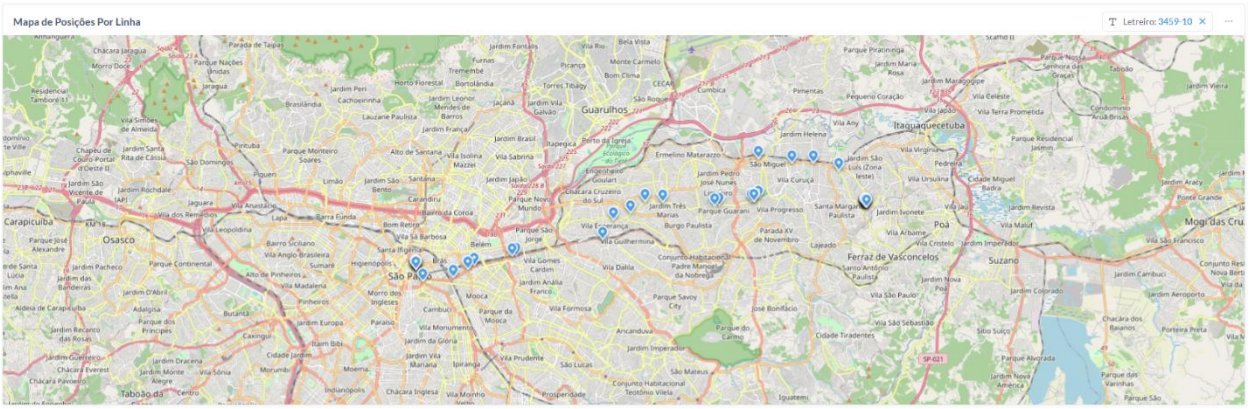
- **Função:** É a principal interface visual do projeto, onde os resultados são apresentados em dois tipos diferentes de Dashboard: Batch e Near Real Time. Conecta-se diretamente ao PostgreSQL para obter resposta de baixa latência.

Dashboard de Tempo Quase Real (Streaming)

Focado no monitoramento operacional, exibindo os KPIs como velocidade da frota, total da frota ativa, ônibus em congestionamento e o mapa de posições:

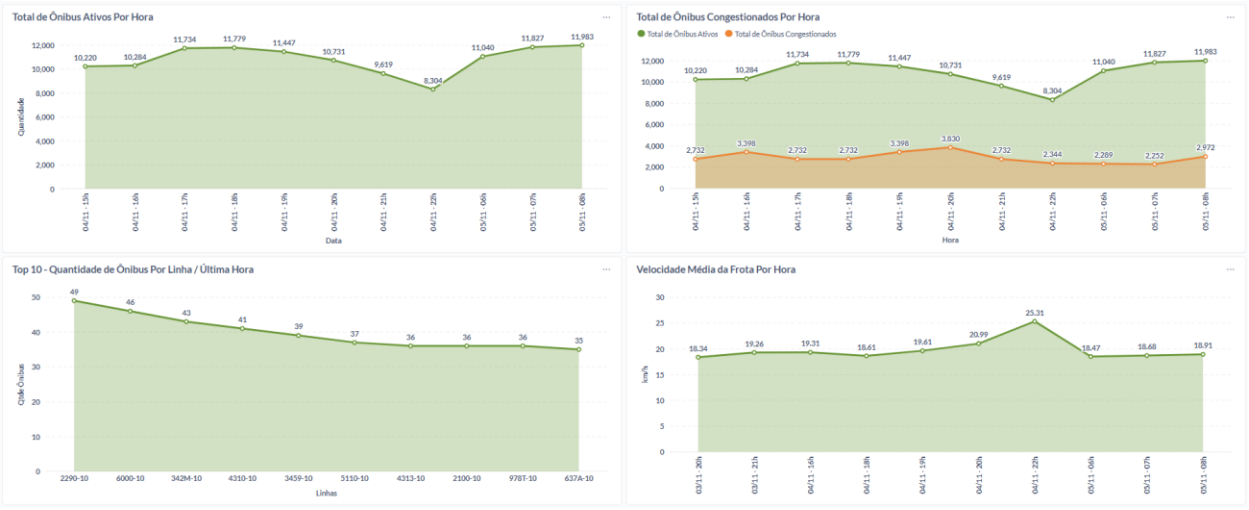






Dashboard de Análise Histórica (Batch)

Focado na análise estratégica, exibindo os dados consolidados pelo pipeline de lote (batch), permitindo a análise de tendências hora a hora:



Mapa de Calor - Quantidade de Ônibus Por Linha x Hora

Linha	03/11 - 16h	03/11 - 17h	03/11 - 18h	03/11 - 19h	03/11 - 20h	03/11 - 21h	04/11 - 15h	04/11 - 16h	04/11 - 17h	04/11 - 18h	04/11 - 19h	04/11 - 20h	04/11 - 21h	04/11 - 22h	05/11 - 06h	05/11 - 07h	05/11 - 08h
209P-1 - Cachoeirinha - Term. Pinheiros	7	7	7	7	7	6	7	7	7	8	8	7	6	4	6	7	7
2100-10 - Term. Vl. Carrão - Pça. Da Sé	35	36	28	28	19	15	30	35	35	35	30	19	16	14	24	29	36
2101-10 - Pça. Silvio Romero - Pça. Da Sé						2	2	2	2	2	2	2	2	2	2	2	2
2101-41 - Pça. Silvio Romero - Term. Vl. P...	1	1		1	1		1	1	1	1	1	1	1		1	1	1
2104-10 - Metrô Santana - Term. Pq. D. P...	8	8	5	6	5	5	8	8	8	8	6	5	5	5	7	7	8
2105-10 - Jd. Filhos Da Terra - Lgo. Da Co...	12	13	11	12	13	10	15	16	16	15	14	14	13	9	14	15	15
211L-10 - Mandaquí - Lapa	9	9	9	9	9	8	9	9	10	10	9	9	8	8	9	9	10
211V-10 - Vl. Paranaguá - Penha	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
2123-10 - Vl. Medeiros - Liberdade	9	9	9	8	8	8	9	9	10	11	10	9	9	8	9	9	9
2127-10 - Pq. Edu Chaves - Liberdade	12	12	12	12	11	11	10	13	14	13	13	12	11	11	12	12	14
213E-10 - Pq. Edu Chaves - Metrô Belém	5	5	5	5	5	5	5	5	5	5	5	5	5	4	5	5	5
2161-10 - Pq. Edu Chaves - Pça. Do Corr...	8	8	11	11	11	10	9	10	9	9	9	10	8	8	6	8	9
2175-10 - Pq. Edu Chaves - Term. Pq. D. P...	6	6	6	6	6	5	6	6	6	6	6	7	7	7	6	6	6
2182-10 - Jd. Brasil - Pça. Do Correio	16	16	15	15	15	15	16	18	18	17	16	17	16	15	15	17	18
2201-10 - Dív. De Ferraz - Cptm Gualana...	2	2	4	4	4	4	3	3	4	4	4	4	4	3	4	4	4
2202-10 - Jd. Das Oliveiras - Cptm Guala...	22	23	25	27	22	18	24	26	27	30	26	23	17	14	28	30	28
2203-10 - Burgo Paulista - Shop. Penha	8	6	7	7	7	7	8	8	8	8	8	8	8	7	8	8	8
2290-10 - Term. São Mateus - Term. Pq. D...	54	54	52	52	47	42	47	52	52	51	49	47	42	37	42	49	49
2290-21 - São Mateus - Term. Pq. D. Pedr...															6	7	4

4.5.4 Consulta Federada (Trino)

- **Tecnologia:** Trino (TrinoDB)

- **Papel:** Atua como motor de consulta SQL federado. Não armazena dados, mas fornece uma interface SQL única para consultar os dados de múltiplas fontes simultaneamente.

- **Justificativa:** Escolhido por sua capacidade de executar consultas federadas e JOINs unindo diferentes fontes.

- **Função:** Conecta-se ao Data Lakehouse (MinIO) para ler as tabelas fato das Camadas Gold e Silver, e ao PostgreSQL para ler as tabelas de Dimensão. Poderá ser usada como a principal ferramenta de Cientistas de Dados para a realização de consultas exploratórias.

5. Dicionário de Dados

5.1 Camada Bronze

Esta camada serve como o repositório inicial para todos os dados brutos, exatamente como foram ingeridos, antes de qualquer limpeza ou transformação.

5.1.1 Posições dos Ônibus (API SPTrans)

- **Fonte:** API `/Posicoes` da SPTrans.
- **Atualização:** A cada 2 minutos, via processo no NiFi.
- **Propósito:** Capturar o estado da frota de ônibus em tempo quase real. Este é o dado primário para todos os pipelines, tanto de lote (batch) quanto de tempo quase real (streaming).

5.1.2 Armazenamento em Data Lake (MinIO)

- **Localização:** `s3a://bronze/sptrans/posicao/`
- **Formato:** JSON
- **Particionamento:** Os arquivos são organizados em pastas por ano, mês, dia e hora (UTC) para otimizar as leituras do pipeline de lote.
- **Estrutura do JSON:** O arquivo JSON possui uma estrutura aninhada:

Campo	Nível	Tipo	Descrição
hr	Raiz	Texto	A hora da consulta à API (ex: "20:15").
l	Raiz	Array	Um array de objetos, onde cada objeto representa uma linha de ônibus que possui veículos em operação.
c	Em l	Texto	Letreiro completo da linha (ex: "8000-10").
cl	Em l	Número	Código identificador numérico da linha.
vs	Em l	Array	Um array de objetos, onde cada objeto representa um veículo (ônibus) daquela linha.
p	Em vs	Número	Prefixo do veículo, seu identificador único.
a	Em vs	Booleano	true se o veículo é acessível para pessoas com deficiência.
ta	Em vs	Texto	Timestamp (em UTC) da última atualização da posição do veículo.
py	Em vs	Número	Coordenada de Latitude da posição do veículo.
px	Em vs	Número	Coordenada de Longitude da posição do veículo.

5.1.3 Armazenamento em Barramento de Eventos (Kafka)

- **Tópico:** `sptrans_posicoes_raw`
- **Formato da Mensagem:** O corpo da mensagem é o mesmo arquivo JSON bruto descrito acima.
- **Propósito:** Servir como um buffer de alta performance para alimentar o pipeline de streaming em tempo quase real, permitindo que o worker, que atua como um consumer, processe os dados de forma independente.

5.1.4 Dados Cadastrais das Linhas (GTFS)

- **Fonte:** Arquivo `routes.txt` do repositório estático GTFS (General Transit Feed Specification).
- **Atualização:** Diária, os arquivos são colocados no MinIO.
- **Propósito:** Servir como uma fonte de dados de enriquecimento (dimensão) para associar os códigos das linhas aos seus nomes completos, tornando os relatórios mais legíveis.
- **Localização:** `s3a://bronze/gtfs/routes.txt`
- **Formato:** CSV (Comma-Separated Values) com cabeçalho.

Estrutura das Colunas

Nome da Coluna	Tipo de Dado	Descrição
route_id	Texto	Identificador único da linha, que corresponde ao letreiro_linha nos dados da API.
agency_id	Número	Identificador da agência de transporte.
route_short_name	Texto	Um nome curto para a linha (geralmente o mesmo que route_id).
route_long_name	Texto	O nome completo e descritivo da linha (ex: "Term. Lapa - Pça. Ramos de Azevedo").
route_type	Número	Tipo de transporte (ex: 3 para ônibus).
route_color	Texto	Cor associada à linha (código hexadecimal).
route_text_color	Texto	Cor do texto para ser usado sobre a route_color.

5.2 Camada Silver

A Camada Silver contém dados limpos, com tipos corrigidos, e estruturados, prontos para serem consumidos pelos pipelines de agregação da Camada Gold. Os dados nesta camada são armazenados no formato Delta Lake.

5.2.1 Tabela do Pipeline Histórico (Batch)

Tabela: posicoes_onibus

- **Localização:** `s3a://silver/posicoes_onibus`
- **Formato:** Delta Lake
- **Particionamento:** Os dados são particionados por ano, mês e dia para otimizar as leituras do pipeline de lote (batch), permitindo que o Spark leia apenas as pastas necessárias para uma determinada hora de processamento.
- **Atualização:** A cada hora, orquestrada pela DAG `bronze_to_silver` do Airflow.
- **Propósito:** Servir como a fonte de dados limpa e histórica para o cálculo da tabela `fato_operacao_linhas_hora`.

Estrutura das Colunas

Nome da Coluna	Tipo de Dado (Spark)	Descrição	Origem (Camada Bronze)
letreiro_linha	StringType	Identificador público da linha (ex: "8000-10").	l.c
codigo_linha	LongType	Código interno/oficial da linha.	l.cl
prefixo_onibus	LongType	Identificador único de cada ônibus.	vs.p
acessivel	BooleanType	true se o ônibus é adaptado para acessibilidade.	vs.a
timestamp_captura	TimestampType	Data e hora exata (em UTC) em que a posição foi registrada.	vs.ta
data_processamento	TimestampType	Data e hora em que o registro foi processado pelo Spark.	Gerado pelo Spark
ano	IntegerType	**(Coluna de Partição)** Ano extraído do timestamp_captura.	Gerado pelo Spark
mes	IntegerType	**(Coluna de Partição)** Mês extraído do timestamp_captura.	Gerado pelo Spark
dia	IntegerType	**(Coluna de Partição)** Dia extraído do timestamp_captura.	Gerado pelo Spark

5.2.2 Tabela do Pipeline de Tempo Quase Real (Streaming)

Tabela: posicoes_onibus_streaming

- **Localização:** `s3a://silver/posicoes_onibus_streaming`
- **Formato:** Delta Lake
- **Particionamento:** Os dados são particionados por ano, mes, dia e hora para permitir que o worker do pipeline de streaming leia apenas os dados mais recentes de forma eficiente, evitando a leitura completa da tabela.
- **Atualização:** Em micro-lotes (a cada 2 minutos), pelo processo `bronze_to_silver_streaming`.

- **Propósito:** Servir como a camada intermediária para os pipelines de streaming da camada Gold. Contém os dados de geolocalização necessários para alimentar os KPIs operacionais em tempo quase real (mapa, velocidade, etc.).

Estrutura das Colunas

Nome da Coluna	Tipo de Dado (Spark)	Descrição	Origem (Camada Bronze)
letreiro_linha	StringType	Identificador público da linha (ex: "8000-10").	l.c
codigo_linha	LongType	Código interno/oficial da linha.	l.cl
prefixo_onibus	LongType	Identificador único de cada ônibus.	vs.p
acessivel	BooleanType	true se o ônibus é adaptado para acessibilidade.	vs.a
timestamp_captura	TimestampType	Data e hora exata (em UTC) em que a posição foi registrada.	vs.ta
latitude	DoubleType	A coordenada de latitude do ônibus.	vs.py
longitude	DoubleType	A coordenada de longitude do ônibus.	vs.px
ano	IntegerType	**(Coluna de Partição)** Ano extraído do timestamp_captura.	Gerado pelo Spark
mes	IntegerType	**(Coluna de Partição)** Mês extraído do timestamp_captura.	Gerado pelo Spark
dia	IntegerType	**(Coluna de Partição)** Dia extraído do timestamp_captura.	Gerado pelo Spark
hora	IntegerType	**(Coluna de Partição)** Hora extraída do timestamp_captura.	Gerado pelo Spark

5.2.3 Tabela Intermediária de KPIs (Streaming -> Batch)

Tabela: kpis_historicos_para_processar

- **Localização:** `s3a://silver/kpis_historicos_para_processar`
- **Formato:** Delta Lake
- **Particionamento:** Os dados são particionados pelo id_tempo para que o pipeline de lote (batch) consolide os dados de forma eficiente.
- **Atualização:**
  1. **Streaming (Escrita APPEND):** A cada ~2 minutos, o job silver_to_gold_nrt_streaming anexa (faz APPEND) seus cálculos de micro-lote (os "rascunhos") na partição da hora correspondente.

  2. **Batch (Consolidação REPLACE):** A cada hora, o job silver_to_gold_batch (orquestrado pelo Airflow) lê todos os "rascunhos" da hora anterior, seleciona o mais recente (rank=1) e, em seguida, sobrescreve atomicamente (mode("overwrite") com replaceWhere) toda aquela partição, salvando apenas o registro rank=1.

- **Propósito:** O pipeline de streaming calcula os KPIs (`velocidade_media_kph`, `quantidade_onibus_parados`) e os armazena aqui. O pipeline de lote (batch) orquestrado pelo Airflow transforma em um log limpo, consolidando todos os cálculos daquela hora em um único registro final (o rank=1).

Estrutura das Colunas

Nome da Coluna	Tipo de Dado (Spark)	Descrição	Origem (Camada Bronze)
id_linha	LongType	Chave estrangeira para a dim_linha.	dim_linha (via Join)
id_tempo	IntegerType	**(Coluna de Partição)** Chave estrangeira para a dim_tempo, indicando a hora da medição.	dim_tempo (via Join)
velocidade_media_kph	DoubleType	KPI: Percentil 85 da velocidade dos ônibus da linha (que estavam > 5 km/h).	percentile_approx("velocidade_kph", 0.85)
quantidade_onibus_parados	LongType	KPI: Contagem de ônibus únicos parados (< 2 km/h) fora da madrugada.	countDistinct("prefixo_onibus")
updated_at	TimestampType	Data e hora em que o registro de KPI foi calculado e salvo.	current_timestamp()

5.3 Camada Gold

A Camada Gold é composta por duas partes: as tabelas de origem no Lakehouse (MinIO), que são a fonte da verdade para dados analíticos, e as tabelas da Camada de Entrega de Dados (PostgreSQL), que são otimizadas para consumo. A modelagem segue o padrão Modelo Estrela (Star Schema).

5.3.1 Tabelas de Dimensão (PostgreSQL)

As tabelas de dimensão descrevem as entidades de negócio e residem no PostgreSQL para serem facilmente acessadas por todos os processos.

Tabela: dim_tempo

- **Tipo:** Dimensão
- **Atualização:** Estática. Populada uma única vez via script SQL.
- **Propósito:** Enriquecer os dados com atributos de tempo detalhados.
- **Chave Primária:** id_tempo

Nome da Coluna	Tipo de Dado	Descrição
id_tempo	SERIAL	Chave primária única para cada hora de cada dia.
data_referencia	DATE	A data (ex: '2025-10-16').
hora_referencia	INTEGER	A hora do dia, em UTC (0-23).
ano	INTEGER	Ano da data de referência.
mes	INTEGER	Mês da data de referência.
dia	INTEGER	Dia da data de referência.
dia_da_semana	VARCHAR	O nome do dia da semana (ex: "Segunda-feira").
fim_de_semana	BOOLEAN	true se for Sábado ou Domingo.
periodo_do_dia	VARCHAR	Classificação da hora (ex: "Manhã", "Tarde", "Noite", "Madrugada").

Tabela: dim_linha

- **Tipo:** Dimensão
- **Atualização:** Diariamente, pela DAG `create_dimensions` do Airflow.
- **Propósito:** Fonte da verdade para os nomes e identificadores das linhas de ônibus.
- **Chave Primária:** id_linha

Nome da Coluna	Tipo de Dado	Descrição
id_linha	BIGINT	Chave primária única para cada linha.
letreiro_linha	VARCHAR	O identificador público da linha (ex: "8000-10").
nome_linha	VARCHAR	O nome completo descritivo da linha.

5.3.2 Tabelas de Fatos no Lakehouse (MinIO)

Estas são as tabelas principais e a fonte única da verdade para os dados agregados. Elas são armazenadas no formato Delta Lake para garantir transacionalidade (ACID) e versionamento.

Tabela: fato_operacao_linhas_hora

- **Localização:** `s3a://gold/fato_operacao_linhas_hora`
- **Propósito:** Armazenar a contagem de ônibus em operação por linha e por hora.

Nome da Coluna	Tipo de Dado (Spark)	Descrição
id_tempo	IntegerType	Chave estrangeira para dim_tempo.
id_linha	LongType	Chave estrangeira para dim_linha.
quantidade_onibus	LongType	Métrica com a contagem de ônibus.

Tabela: fato_velocidade_linha

- **Localização:** `s3a://gold/fato_velocidade_linha`
- **Propósito:** Armazenar o KPI de velocidade média operacional por linha e por hora.

Nome da Coluna	Tipo de Dado (Spark)	Descrição
id_tempo	IntegerType	Chave estrangeira para dim_tempo.
id_linha	LongType	Chave estrangeira para dim_linha.
velocidade_media_kph	DoubleType	Métrica de velocidade em km/h.
updated_at	TimestampType	Timestamp da última atualização do registro.

Tabela: fato_onibus_parados_linha

- **Localização:** `s3a://gold/fato_onibus_parados_linha`
- **Propósito:** Armazenar o KPI de contagem de ônibus parados/congestionados por linha e por hora.

Nome da Coluna	Tipo de Dado (Spark)	Descrição
id_tempo	IntegerType	Chave estrangeira para dim_tempo.
id_linha	LongType	Chave estrangeira para dim_linha.
quantidade_onibus_parados	LongType	Métrica da contagem de ônibus parados.
updated_at	TimestampType	Timestamp da última atualização do registro.

5.3.3 Tabelas de Fatos na Camada de Entrega de Dados (PostgreSQL)

Esta camada contém as tabelas prontas para consumo direto pela API e pelo Metabase. Ela serve a uma função tripla:

1) **Armazenamento de Estado (Primário):** É a fonte primária para a tabela de estado `nrt_posicao_onibus_atual`, que é otimizada para os UPSERTs de alta frequência do pipeline de streaming.

2) **Camada de Consumo NRT (Primário):** É a fonte primária para as tabelas `nrt_velocidade_linha` e `nrt_onibus_parados_linha`.

3) **Camada de Consumo (Cópia):** Armazena cópias das tabelas analíticas (`fato_operacao_linhas_hora`, `fato_velocidade_linha`, `fato_onibus_parados_linha`), carregadas a partir do Lakehouse.

Tabela: nrt_posicao_onibus_atual

- **Tipo:** Tabela de Estado (Streaming)
- **Atualização:** Em micro-lotes (a cada 4 minutos), pelo processo de Spark Streaming.
- **Propósito:** Esta é a tabela primária para o estado de posição dos ônibus.
- **Chave Primária:** prefixo_onibus

Tabela: nrt_velocidade_linha

- **Tipo:** Snapshot de KPI (Streaming)
- **Propósito:** Armazena o snapshot mais recente da velocidade média por linha, alimentando o dashboard NRT. É sobrescrita (Truncate + Insert) a cada micro-lote.
- **Chave Primária:** id_linha

Tabela: nrt_onibus_parados_linha

- **Tipo:** Snapshot de KPI (Streaming)
- **Propósito:** Armazena o snapshot mais recente da contagem de ônibus parados por linha, alimentando o dashboard NRT. É sobrescrita (Truncate + Insert) a cada micro-lote.
- **Chave Primária:** id_linha

As outras tabelas de fatos nesta camada (`fato_operacao_linhas_hora`, `fato_velocidade_linha` e `fato_onibus_parados_linha`) são cópias otimizadas das tabelas do Lakehouse (MinIO), carregadas ao final de cada pipeline para consultas de baixa latência.

6. Guia de Operação e Monitoramento

Esta seção descreve os passos práticos para configurar o ambiente, executar o projeto do zero e monitorar a saúde dos pipelines.

6.1 Executando o Projeto (Setup)

Siga os passos abaixo para iniciar a plataforma.

Pré-requisitos:

- Docker e Docker Compose instalados.
- Repositório do projeto clonado (https://github.com/guilhermepepa/FIA-Projeto-Final).

1. **Iniciar Todos os Serviços:** Na pasta raiz do projeto (onde está o `docker-compose.yml`), execute o seguinte comando:

   ```
   docker-compose up -d
   ```

   Este comando construirá as imagens e iniciará todos os serviços (NiFi, Kafka, Spark, Airflow, MinIO, PostgreSQL, Metabase e API) em segundo plano.

2. **Configurar o NiFi:**
   - Acesse a interface do NiFi em https://localhost:9443/nifi/login.
   - Arraste o arquivo de template `nifi-templates/sptrans-ingestion-v4.xml` para a tela para importar o fluxo de ingestão.
   - Clique com o botão direito no Process Group "`ingestao-sptrans`" e acesse a opção "`Configure`". Acesse a aba "`Controller Services`" e habilite o serviço "`AWSCredentialsProviderControllerService`".
   - Inicie o Process Group clicando no botão de play.

3. **Restaurar os Dashboards do Metabase:**
   - Acesse via o terminal o diretório `metabase/backup`.
   - Execute o comando abaixo para restaurar os dashboards, perguntas e configurações de conexão do Metabase a partir do backup.
   - No Windows (PowerShell):

     ```
     Get-Content .\metabase_backup.sql | docker-compose exec -T postgres psql -U admin -d metabase_meta
     ```

4. **Habilitar as DAGs no Airflow:**
   - Acesse a interface do Airflow em http://localhost:8081.
   - Na página principal, habilite as seguintes DAGs para execução automática, clicando no botão de toggle:
     - **create_dimensions:** Executa uma vez ao dia para criar as tabelas de dimensão.
     - **bronze_to_silver:** Roda de hora em hora para processar o lote de dados históricos.
     - **silver_to_gold:** Roda em seguida da bronze_to_silver para consolidar os dados na camada Gold.

- **delta_lake_maintenance:** Roda uma vez por dia (de madrugada) para otimizar as tabelas Delta. É recomendado mantê-la desabilitada e executar pontualmente, se necessário.

5. **Acessando as Interfaces:**
   - **Airflow UI:** http://localhost:8081
   - **Spark Master UI:** http://localhost:8080
   - **NiFi Canvas:** https://localhost:9443/nifi/login
   - **Metabase:** http://localhost:3000
   - **MinIO Console (Data Lake):** http://localhost:9001 (Login: admin, Senha: projetofinal)
   - **API Docs (Swagger):** http://localhost:8000/docs

6.2 Monitorando a Saúde do Pipeline

Para garantir que o pipeline está funcionando corretamente, os seguintes pontos podem ser observados:

1. **Airflow UI (http://localhost:8081):**
    - É o principal ponto de monitoramento para o pipeline batch.
    - Verificar se as DAGs estão sendo executadas nos horários agendadas e se as execuções terminam com o status de sucesso (verde).

2. **Spark Master UI (http://localhost:8080):**
    - Fornece uma visão geral de todas as aplicações Spark em execução.
    - Devem haver dois jobs de streaming (bronze_to_silver_streaming e silver_to_gold_streaming) no estado Running. Quando uma DAG do Airflow é executada, uma nova aplicação aparecerá brevemente enquanto o job de lote é processado.

3. **Logs dos Contêineres (Terminal):**
    - Principal meio de depuração em tempo real.
    - Para ver os logs de um serviço específico (Ex.: streaming Silver-Para-Gold):
      `docker-compose logs -f spark-streaming-silver-to-gold-nrt`

4. **NiFi Canvas (https://localhost:9443/nifi/login):**
    - Ponto principal de monitoração da ingestão.
    - Verificar se os processadores estão no estado Running, e se não há acúmulo excessivo de FlowFiles nas filas.

5. **Metabase (http://localhost:3000):**
    - Ponto final de validação dos dados.
    - Dashboards podem ser criados utilizando os selects contidos no path /metabase.

6. **MinIO (http://localhost:9001/login):**
    - Datalake onde todo os dados são armazenados.
    - Verificar se os dados estão sendo populados nas camadas bronze, silver e gold corretamente.