instrucoes-teste-tecnico

Teste Técnico: Pipeline de Ingestão Periódica + API + Dashboard

Para visualizar o arquivo formatado, utilizar um visualizador de Markdown.

Contexto

Na Driva, trabalhamos com inteligência de mercado B2B através do nosso principal produto: o **HubDriva**. Uma das funcionalidades mais importantes da plataforma é o **Enriquecimento de Dados**, que permite aos nossos clientes (workspaces) agregarem informações valiosas a dados de empresas brasileiras.

Como funciona o Enriquecimento:

- Workspaces utilizam créditos para executar jobs de enriquecimento
- Cada job processa um conjunto de empresas/pessoas, agregando dados como e-mails, telefones, informações corporativas, etc.
- Os workspaces consomem esses dados enriquecidos para suas estratégias de vendas e marketing

O Problema:

O time de **Visibilidade** precisa monitorar a performance e qualidade dos enriquecimentos que estão sendo entregues na plataforma. Para isso, é necessário:

- Ter visibilidade sobre quantos enriquecimentos estão sendo realizados
- Analisar o desempenho (tempo de processamento, taxa de sucesso/falha)
- Identificar padrões e problemas
- Gerar insights para melhorias

Sua Missão:

Como membro do **time de Tech**, você foi designado para construir uma pipeline de dados que consulte periodicamente uma API interna que retorna informações sobre os enriquecimentos realizados na plataforma. Os dados devem ser processados e armazenados seguindo uma arquitetura de data warehouse em camadas **Bronze** (dados brutos) e **Gold** (dados processados e prontos para análise).

Além disso, você deverá disponibilizar esses dados por meio de uma **API** e construir um **dashboard simples** (frontend) para consumo do time de Visibilidade.

Objetivo da solução

1. Subir um ambiente local com Docker contendo **PostgreSQL**, **n8n**, **API** e (opcionalmente) o **frontend**
2. Construir uma **API** que:
   - Exponha um endpoint paginado simulando a API de enriquecimentos
   - Exponha endpoints de leitura/consulta sobre a camada Gold para o dashboard
3. Coletar dados de enriquecimentos (via n8n) **com um pooling de 5 minutos**
4. Armazenar os dados brutos na camada **Bronze**
5. Processar, validar e transformar os dados na camada **Gold**
6. Criar um **dashboard simples** que mostre métricas e listas relevantes (consumindo a API)

O Desafio

Seu desafio é construir e orquestrar um ambiente local com Docker, implementar uma API e criar workflows no **n8n** para ingestão e processamento periódico.

O desafio é dividido em **quatro etapas**:

1. **Configurar o Ambiente:** Usar **Docker** para subir as instâncias do PostgreSQL, n8n e a API, incluindo a inicialização automática das tabelas do banco de dados.
2. **Construir a API:** Implementar endpoints que simulam a fonte (enrichments) e expõem dados analíticos da camada Gold.
3. **Workflows no n8n (Bronze + Gold):** Criar workflows de ingestão e processamento (chamáveis) e um workflow orquestrador (agendado a cada 5 minutos).
4. **Dashboard (Frontend):** Criar um dashboard simples (tecnologia de frontend à sua escolha) consumindo a API.

> **Observação:** Pense em como os workflows devem ser estruturados e acionados para garantir que o pipeline completo seja executado periodicamente de forma eficiente e organizada.

Recursos Fornecidos

1) Arquitetura Local (Docker)

Você deve criar um arquivo `docker-compose.yml` para orquestrar os serviços.

⚠️ **OBRIGATÓRIO:** O `docker-compose.yml` deve incluir um volume que monte um arquivo `init.sql` para criar automaticamente todas as tabelas necessárias quando o PostgreSQL subir pela primeira vez.

Serviços mínimos obrigatórios:

- **Serviço 1:** `postgres`
  - Use uma imagem oficial do PostgreSQL (versão 14 ou superior)
  - Configure as variáveis de ambiente necessárias (usuário, senha, database)
  - Monte um script `init.sql` para criação das tabelas
  - Exponha a porta padrão (5432)
- **Serviço 2:** `n8n`
  - Use a imagem oficial do n8n
  - Deve rodar localmente (ex: `http://localhost:5678`)
  - Configure para persistir os workflows
  - Deve conseguir se conectar ao PostgreSQL
- **Serviço 3:** `api`
  - Linguagem/framework livre (ex.: Node/Nest/Express, Go, Python/FastAPI)
  - Deve rodar localmente (ex.: `http://localhost:3000`)
  - Deve conseguir se conectar ao PostgreSQL
  - Deve implementar os endpoints descritos na seção **API**
- **Serviço 4:** `frontend`
  - Subir o frontend via Docker (opcional), **ou**
  - Executar fora do Docker (ex.: Vite/Next) e documentar no README.
  - O importante é: o dashboard **funcionar** e consumir a API.

---

2) API (você deve construir)

A API terá **dois grupos de endpoints**:

1. **Fonte (simulação da API de enriquecimentos)** – consumida pelo n8n
2. **Analytics (leitura da camada Gold)** – consumida pelo dashboard

2.1) Autenticação

- A API requer autenticação via **API Key**
- Header: `Authorization: Bearer {API_KEY}`

- API Key (para o teste): `driva_test_key_abc123xyz789`

2.2) Endpoint de Fonte (para ingestão)

Endpoint: `GET /people/v1/enrichments`

Query Parameters (Paginação):

- `page` (int): Número da página atual (default: 1)
- `limit` (int): Itens por página (default: 50, max: 100)

⚠️ IMPORTANTE – PAGINAÇÃO E RATE LIMITING:

- A API deve ter **milhares de registros** (pode gerar em runtime ou seedar via SQL/arquivo JSON).
- A resposta deve incluir `meta.total_pages`, `meta.total_items`, etc.
- A API **pode retornar** `429 Too Many Requests` (simulação) — por exemplo, em bursts ou aleatoriamente.

Estrutura da Resposta (JSON):

```
{
  "meta": {
    "current_page": 1,
    "items_per_page": 50,
    "total_items": 5000,
    "total_pages": 100
  },
  "data": [
    {
      "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
      "id_workspace": "e6bb64bf-46e4-410d-8406-c61e267ea607",
      "workspace_name": "Tech Solutions Corp",
      "total_contacts": 1500,
      "contact_type": "COMPANY",
      "status": "COMPLETED",
      "created_at": "2025-11-15T14:30:00Z",
      "updated_at": "2025-11-15T14:35:22Z"
    }
  ]
}
```

> **Dica:** você pode armazenar esses enrichments em uma tabela no Postgres (seed) e apenas paginar via SQL. Isso facilita testar, manter determinismo e validar o pipeline.

2.3) Endpoints de Analytics (para dashboard)

Você deve criar endpoints para leitura da camada Gold. Sugestão mínima:

- `GET /analytics/overview`
  - Retorna KPIs para o dashboard (ex.: total de jobs, % sucesso, tempo médio, distribuição por categoria)
- `GET /analytics/enrichments`
  - Lista paginada/filtrável (ex.: por `id_workspace`, `status_processamento`, período)
- (Bônus) `GET /analytics/workspaces/top`
  - Ranking de workspaces por volume de enriquecimentos ou total_contatos

Requisitos gerais:

- Os endpoints devem consultar a camada Gold no Postgres.
- Devem ser rápidos e bem estruturados (pode usar índices).
- Devem devolver JSON simples (sem necessidade de autenticação extra além da API Key).

---

3) Regras de Negócio (Bronze e Gold)

Camada Bronze

- Deve armazenar **todos os dados de enriquecimento** retornados pela API
- Cada registro de enriquecimento deve ser armazenado individualmente
- Deve permitir rastreamento de quando cada job foi ingerido no DW
- Caso um registro já esteja no banco de dados, atualizar os dados
- **Campos obrigatórios de controle do DW:**
  - `dw_ingested_at` (TIMESTAMP): data/hora em que o registro foi ingerido pela primeira vez na Bronze
  - `dw_updated_at` (TIMESTAMP): data/hora da última atualização do registro na Bronze
- **Dica de modelagem:** Na Bronze, priorize captura fiel. Considere tipos flexíveis (ex.: `TEXT` / `JSONB`) para evitar falhas por incompatibilidade.

Camada Gold

- Deve processar apenas registros da Bronze que ainda não foram processados (ou garantir que a Gold reflita o estado mais atual)
- **Todos os nomes de colunas e valores devem estar em português**
- **Tradução de nomes de colunas (exemplos):**

    - `id` → `id_enriquecimento`
    - `id_workspace` → `id_workspace` (manter)
    - `workspace_name` → `nome_workspace`
    - `total_contacts` → `total_contatos`
    - `contact_type` → `tipo_contato`
    - `status` → `status_processamento`
    - `created_at` → `data_criacao`
    - `updated_at` → `data_atualizacao`
- **Criar novos campos calculados/transformados (aplicar):**
    - `duracao_processamento_minutos` (INTEGER/FLOAT): diferença em minutos entre `data_atualizacao` e `data_criacao`
    - `tempo_por_contato_minutos` (FLOAT): `duracao_processamento_minutos / total_contatos`
    - `processamento_sucesso` (BOOLEAN): `true` se `status_processamento = "CONCLUIDO"`
    - `categoria_tamanho_job` (VARCHAR): por `total_contatos`:
        - PEQUENO: < 100
        - MEDIO: 100-500
        - GRANDE: 501-1000
        - MUITO_GRANDE: > 1000
    - Traduzir valores de `tipo_contato`:
        - "PERSON" → "PESSOA"
        - "COMPANY" → "EMPRESA"
    - Traduzir valores de `status_processamento`:
        - "PROCESSING" → "EM_PROCESSAMENTO"
        - "COMPLETED" → "CONCLUIDO"
        - "FAILED" → "FALHOU"
        - "CANCELED" → "CANCELADO"
    - `necessita_reprocessamento` (BOOLEAN): `true` se status original = "FAILED" ou "CANCELED"
    - **Campo obrigatório:** `data_atualizacao_dw` (TIMESTAMP): data/hora da execução do processamento (snapshot)

---

4) Estrutura das Tabelas (init.sql)

Você deve criar um arquivo **`init.sql`** com as definições de todas as tabelas necessárias para o projeto:

- Tabelas do DW: **Bronze** e **Gold**
- (Recomendado) tabelas auxiliares para controle/estado (ex.: `dw_pipeline_state`, watermark, última página, etc.)
- (Se escolher) tabela(s) de seed da API (ex.: `api_enrichments_seed`) — opcional, mas recomendado

---

Requisitos dos Workflows (n8n)

Workflow de Ingestão (API → Bronze) — chamável

Este workflow deve:

1. Tratar paginação e buscar **todas as páginas** a cada execução (ou aplicar estratégia de watermark/atualização incremental — se fizer, documente).
2. Fazer requisição HTTP para a **sua API local**:
   - `GET http://api:3000/people/v1/enrichments?page={page}&limit=50` (dentro do Docker)
   - `Authorization: Bearer driva_test_key_abc123xyz789`
3. Tratar erros:
   - **429 Too Many Requests** (implementar retry com backoff)
4. Salvar na Bronze:
   - Insert/Upsert de todos os campos do enrichment
   - Preencher `dw_ingested_at` (na primeira ingestão) e `dw_updated_at` (em toda atualização)
   - Registrar página/execução (como preferir)

Workflow de Processamento (Bronze → Gold) — chamável

Este workflow deve:

1. Ler registros da Bronze e aplicar as transformações definidas
2. Popular a Gold garantindo:
   - colunas/valores em português
   - campos calculados preenchidos
   - `data_atualizacao_dw` preenchido
3. Garantir que a Gold reflita o estado mais atual da Bronze (upsert/snapshot/merge — escolha e documente)

Workflow Orquestrador (Scheduler) — agendado a cada 5 minutos

Você deve criar um workflow no n8n que:

1. Rode a cada **5 minutos**
2. Chame o workflow de ingestão
3. Ao finalizar, chame o workflow de processamento
4. Faça logging/observabilidade mínima (ex.: status, número de registros, erros)

---

5) Dashboard (Frontend)

Após a pipeline estar funcionando, você deve construir um **dashboard simples** que consuma os endpoints de **Analytics** da sua API.

Requisitos mínimos do dashboard:

- Uma página principal com KPIs (ex.: total de enriquecimentos, % sucesso, tempo médio de processamento)
- Um gráfico simples **ou** tabela resumida (ex.: distribuição por `categoria_tamanho_job` ou por `status_processamento`)
- Uma lista/tabela paginada (ou com filtro) de enriquecimentos

Tecnologia livre, por exemplo:

- React + Vite, Next.js, Vue, Svelte, Angular, etc.

> O objetivo é avaliar a capacidade de integrar ponta-a-ponta: DW + automação + API + visualização.

Critérios de Avaliação

1. **Funcionalidade ponta-a-ponta:** API + n8n + DW + dashboard funcionando conforme especificado
2. **Modelagem de Dados:** tabelas bem projetadas (Bronze/Gold, índices, consistência)
3. **Qualidade da API:** endpoints bem definidos, paginação, autenticação, boas práticas
4. **Qualidade dos Workflows:** organização, legibilidade, modularidade, retry/backoff, logs
5. **Qualidade do Dashboard:** simplicidade, clareza, consumo correto da API, UX básica
6. **Documentação:** README claro (como rodar tudo) + decisões registradas
7. **Boas Práticas:** tratamento de erro, padrão de commits (se aplicável), estrutura do projeto

---

Entregáveis

Para concluir o teste, você **DEVE** fornecer:

1. **Repositório no Github** com os seguintes arquivos:
    1. `docker-compose.yml`
    2. `init.sql`
    3. **Código da API** (com Dockerfile e instruções)
    4. **Código do Dashboard** (com instruções de execução)
    5. **JSONs dos Workflows** exportados do n8n
    6. `README.md` (principal) com:
        1. visão geral da solução
        2. como subir o ambiente
        3. como rodar o frontend (se fora do Docker)
        4. como importar/executar workflows
        5. exemplos de chamadas (curl) para os endpoints
2. **Vídeo de demonstração** apresentando o projeto

Boa sorte! 🚀