# Documento Técnico – Projeto MovieStream Analytics

## 1. Arquitetura da Solução

A arquitetura proposta utiliza Apache Airflow como orquestrador de pipelines de dados, com dois bancos de dados PostgreSQL: um como base de origem (dbinterview) e outro como destino para a área de landing (landing\_db). As DAGs do Airflow executam scripts Python que realizam a extração dos dados, executando cargas completas para a maioria das tabelas e Change Data Capture (CDC) para tabelas com maior volume de dados. A comunicação entre os containers e serviços ocorre via Docker Compose.

## 2. Estratégia de CDC Utilizada

Foi implementada uma abordagem de CDC baseada em coluna de controle temporal (`last\_update` e `payment\_date`).   
 As tabelas `rental` e `payment` são monitoradas para identificar alterações realizadas desde a última execução. O script Python armazena o timestamp processado em arquivos `.txt` e utiliza essas informações para aplicar inserções incrementais na base de destino (`landing\_db`).

## 3. Cronograma de Execução das DAGs

A DAG `pipeline\_ingestao\_cdc` foi configurada com agendamento diário (`@daily`) e executa os seguintes passos:  
1. Carga completa das tabelas de apoio e dimensões;  
2. Execução do CDC nas tabelas `rental` e `payment` com controle incremental por timestamp;  
3. Escrita incremental no banco `landing\_db`.  
  
 Além disso, foi criada a DAG `pipeline\_dbt`, também agendada diariamente, que depende da DAG de ingestão e executa os modelos DBT logo após a atualização dos dados.

## 4. Tecnologias Utilizadas

- Python: linguagem utilizada para scripts de ingestão de dados.  
- Apache Airflow: ferramenta de orquestração de workflows para agendar e monitorar as pipelines.  
- PostgreSQL: banco de dados relacional utilizado como origem (dbinterview) e destino (landing\_db).  
- Docker e Docker Compose: utilizados para criar e gerenciar os containers.  
- Pandas e SQLAlchemy: bibliotecas Python para movimentação dos dados.  
- DBT (Data Build Tool): ferramenta para modelagem e transformação de dados em camada de Data Warehouse.

## 5. Instruções de Execução

1. Clonar o repositório e acessar o diretório do projeto.  
2. Subir os containers com `docker compose up --build`.  
3. Acessar a interface do Airflow via `http://localhost:8080`.  
4. Ativar e executar manualmente (ou aguardar agendamento) das DAGs `pipeline\_ingestao\_cdc` e `pipeline\_dbt`.  
5. Verificar as transformações DBT diretamente no banco `landing\_db`, schema `public`.

## 6. Desafios Enfrentados

- Conexão entre containers e banco local (resolvido com `host.docker.internal` no Airflow).  
- Erros de dependência em tabelas com views no DBT, resolvidos ajustando o modo de escrita no script para `if\_exists='append'`.  
- Configuração da execução do DBT dentro do container do Airflow, com adição de `Dockerfile` e ajuste no `docker-compose.yml` para incluir o projeto e o perfil do DBT.  
- Erro na localização do arquivo `profiles.yml` resolvido com o parâmetro `--profiles-dir` e mapeamento correto do volume.

## 7. Modelagem e Transformações

A segunda fase do projeto teve como objetivo construir uma camada de Data Warehouse utilizando o DBT.   
 Foram aplicadas boas práticas de modularização e versionamento para garantir rastreabilidade e clareza na estrutura dos dados.

## 7.1 Camadas e Estrutura de Projeto

- `staging`: padronização, renomeação de colunas e tipagens dos dados extraídos do landing.  
- `marts`: camada analítica com indicadores de negócio prontos para análise.

## 7.2 Tabelas Derivadas Criadas

- `mart\_customer\_lifetime\_value`: valor total gasto por cliente, data da primeira locação e tempo de relacionamento.  
- `mart\_film\_popularity`: ranking de filmes mais alugados por mês/ano.  
- `mart\_store\_performance`: performance por loja (número de locações, receita e base de clientes distintos).

## 7.3 Boas Práticas Aplicadas

- Modularização clara com diretórios `staging` e `marts`.  
- Uso de `ref()` para controle de dependência entre modelos.  
- Documentação de modelos via `schema.yml`.  
- Materialização adequada (`view` em staging, `table` em marts).  
- Controle via Git com versionamento do projeto DBT.

## 8. SQL Analítico

As queries abaixo foram construídas em SQL padrão PostgreSQL para responder às perguntas de negócio propostas:

1. \*\*Top 5 clientes que mais geraram receita no último ano\*\*:  
```sql  
SELECT c.customer\_id, c.first\_name || ' ' || c.last\_name AS cliente, SUM(p.amount) AS receita\_total  
FROM payment p  
JOIN customer c ON p.customer\_id = c.customer\_id  
WHERE p.payment\_date >= CURRENT\_DATE - INTERVAL '1 year'  
GROUP BY c.customer\_id, cliente  
ORDER BY receita\_total DESC  
LIMIT 5;  
```

2. \*\*Média de dias entre aluguel e devolução por categoria de filme\*\*:  
```sql  
SELECT cat.name AS categoria, ROUND(AVG(r.return\_date - r.rental\_date), 2) AS media\_dias  
FROM rental r  
JOIN inventory i ON r.inventory\_id = i.inventory\_id  
JOIN film f ON i.film\_id = f.film\_id  
JOIN film\_category fc ON f.film\_id = fc.film\_id  
JOIN category cat ON fc.category\_id = cat.category\_id  
WHERE r.return\_date IS NOT NULL  
GROUP BY cat.name  
ORDER BY media\_dias DESC;  
```

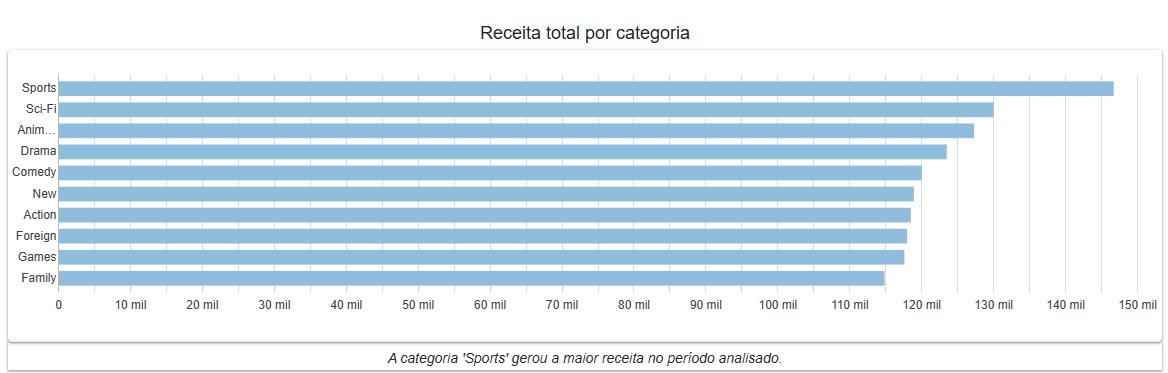
3. \*\*Top 3 cidades com maior volume de locações\*\*:  
```sql  
SELECT ci.city, COUNT(\*) AS total\_locacoes  
FROM rental r  
JOIN customer c ON r.customer\_id = c.customer\_id  
JOIN address a ON c.address\_id = a.address\_id  
JOIN city ci ON a.city\_id = ci.city\_id  
GROUP BY ci.city  
ORDER BY total\_locacoes DESC  
LIMIT 3;  
```

4. \*\*Ticket médio por loja\*\*:  
```sql  
SELECT s.store\_id, ROUND(SUM(p.amount)/COUNT(DISTINCT p.rental\_id), 2) AS ticket\_medio  
FROM payment p  
JOIN rental r ON p.rental\_id = r.rental\_id  
JOIN inventory i ON r.inventory\_id = i.inventory\_id  
JOIN store s ON i.store\_id = s.store\_id  
GROUP BY s.store\_id;  
```

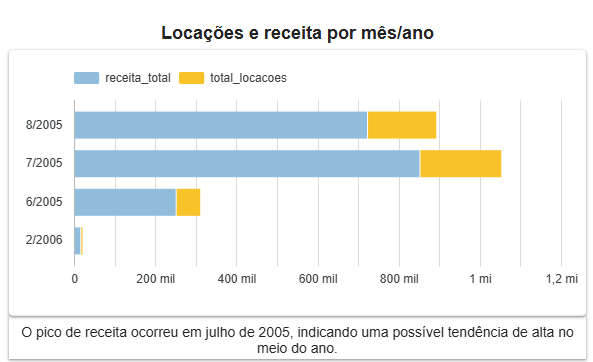
5. \*\*Receita mensal nos últimos 24 meses\*\*:  
```sql  
SELECT TO\_CHAR(p.payment\_date, 'YYYY-MM') AS mes\_ano, ROUND(SUM(p.amount), 2) AS receita\_total  
FROM payment p  
WHERE p.payment\_date >= DATE\_TRUNC('month', CURRENT\_DATE) - INTERVAL '24 months'  
GROUP BY mes\_ano  
ORDER BY mes\_ano;  
```

## 9. Visualização e Storytelling

As visualizações foram criadas utilizando o Google Looker Studio com base nos dados exportados do banco `landing\_db` e pode ser acessado através do link: <https://lookerstudio.google.com/reporting/3d7f94bc-7f79-44c0-b7c4-784983ec04cc>  
  
Indicadores apresentados:  
1. Receita por Categoria de Filme  
2. Locações por Mês (últimos 24 meses)  
3. Clientes Ativos por Loja  
  
📌 Insights:  
- As categorias que mais geram receita representam preferências claras do público. Utilizar essas informações para campanhas de marketing ou aquisição de novos filmes pode trazer alto retorno.



- Essa visão permite identificar padrões sazonais e avaliar crescimento/declínio da base de clientes.



- Ambas as lojas apresentaram a mesma quantidade de clientes ativos, indicando equilíbrio operacional entre unidades.

