Tutorial: Medallion Architecture

Databricks Free Edition

Pipeline de Dados End-to-End com
GitHub, CLI Local e Genie Spaces

Bronze >>> Silver >>> Gold >>> Genie

Autor: Thiago Charchar & Claude AI

Data: Fevereiro 2026

github.com/tlcharchar/databricks-medallion

Nivel: Intermediario

Sumário

1. Visão Geral do Projeto
2. Arquitetura Medallion
3. Stack Tecnológico
4. Fase 1 -- Setup e Integração
5. Fase 2 -- Pipeline Medallion (Bronze -> Silver -> Gold)
6. Fase 3 -- SQL Analytics e Genie Spaces
7. Fase 4 -- Orquestração com Workflows
8. Troubleshooting
9. Próximos Passos

1. Visão Geral do Projeto

Este tutorial guia você na construção de um pipeline de dados completo usando a Medallion Architecture (Bronze -> Silver -> Gold) no Databricks Free Edition.

O projeto integra três ambientes:

- Localhost -- Desenvolvimento local com Databricks CLI
- GitHub -- Versionamento e colaboração
- Databricks -- Processamento, SQL Analytics e Genie Spaces

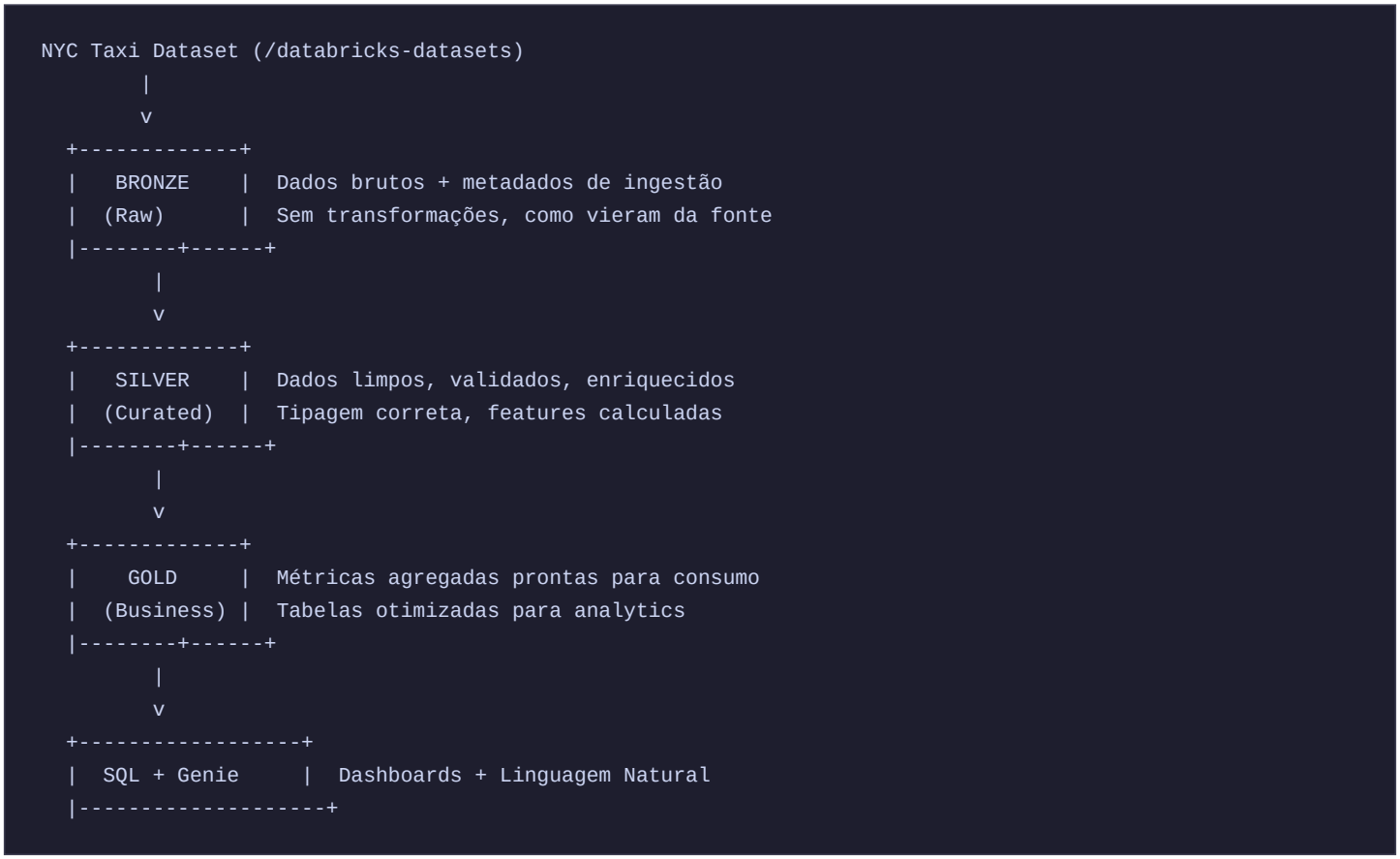
Ao final, você terá um pipeline funcional que ingere dados brutos, os transforma progressivamente, e permite fazer perguntas em linguagem natural usando o Genie Spaces ("Fale com seus dados").

Dataset

Usamos o NYC Yellow Taxi Trip Records, disponível nativamente no Databricks em /databricks-datasets/nyctaxi/tables/nyctaxi_yellow. O dataset contém milhões de registros de corridas de táxi em Nova York com informações de pickup/dropoff, tarifas, gorjetas e coordenadas geográficas.

2. Arquitetura Medallion

A Medallion Architecture organiza os dados em três camadas progressivas de qualidade:



Por que Medallion?

- Rastreabilidade: sempre é possível voltar ao dado bruto
- Qualidade progressiva: cada camada adiciona confiabilidade
- Reutilização: a Silver serve múltiplos casos de uso Gold
- Performance: tabelas Gold são pré-agregadas para queries rápidas

---

3. Stack Tecnológico

Componente	Tecnologia	Função
Plataforma	Databricks Free Editio..	Processamento e analytics
Processamento	PySpark / Spark SQL	Transformação de dados
Storage	Delta Lake (Unity Cata..	Tabelas versionadas e ACID
Versionamento	GitHub	Código e colaboração
CLI Local	Databricks CLI v0.287+	Automação local
Consumo	Databricks SQL	Queries e dashboards
NLP	Genie Spaces	Consultas em linguagem natural
Orquestração	Databricks Workflows	Automação de pipeline

---

4. Fase 1 -- Setup e Integração

4.1 Criar conta no Databricks Free Edition

1. Acesse databricks.com/try-databricks
2. Crie uma conta usando seu email
3. Escolha a cloud provider (AWS recomendado)
4. Aguarde o workspace ser provisionado
5. Anote a URL do workspace (ex: dbc-XXXXX.cloud.databricks.com)

> *Nota: O Databricks Free Edition dá acesso a Unity Catalog, SQL Warehouses, Genie Spaces e Serverless Compute -- tudo necessário para este tutorial.*

4.2 Instalar o Databricks CLI

No seu terminal (macOS/Linux):

```bash
# Instalação oficial
curl -fsSL https://raw.githubusercontent.com/databricks/setup-cli/main/install.sh | sudo sh

# Verificar instalação
databricks --version
# Esperado: Databricks CLI v0.287.0 (ou superior)
```

Alternativa sem sudo (download manual):

```bash
# Detectar arquitetura
ARCH=$(uname -m)  # arm64 ou x86_64
OS=$(uname -s | tr '[:upper:]' '[:lower:]')

# Mapear arquitetura
CLI_ARCH=$( [ "$ARCH" = "arm64" ] && echo "arm64" || echo "amd64" )

# Baixar última versão
LATEST=$(curl -fsSL -o /dev/null -w '%{url_effective}' \
  https://github.com/databricks/cli/releases/latest)
VERSION=$(basename "$LATEST")

# Instalar
mkdir -p ~/.local/bin
curl -fsSL -o /tmp/databricks_cli.zip \
  "https://github.com/databricks/cli/releases/download/${VERSION}/databricks_cli_${OS}_${CLI...
unzip -o /tmp/databricks_cli.zip -d ~/.local/bin/
chmod +x ~/.local/bin/databricks

# Adicionar ao PATH (se necessário)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc
```

4.3 Gerar Personal Access Token (PAT)

No Databricks UI:

1. Clique no seu avatar (canto superior direito)
2. Vá em Settings -> Developer
3. Em Access tokens, clique em Manage
4. Clique em Generate new token
5. Nome: cli-local, Lifetime: 90 days
6. Copie o token (ele só aparece uma vez!)

4.4 Autenticar o CLI

```bash
# Criar arquivo de configuração
cat > ~/.databrickscfg << EOF
[DEFAULT]
host  = https://dbc-XXXXX.cloud.databricks.com
token = dapi_SEU_TOKEN_AQUI
EOF

# Testar conexão
databricks current-user me
```

Resposta esperada:

```json
{
  "active": true,
  "emails": [{"value": "seu@email.com"}],
  "userName": "seu@email.com"
}
```

4.5 Criar repositório no GitHub

```bash
# Autenticar GitHub CLI (se ainda não fez)
gh auth login

# Criar repositório
gh repo create databricks-medallion \
  --public \
  --description "Medallion Architecture com Databricks Free Edition" \
  --clone

cd databricks-medallion
```

4.6 Estrutura do projeto

```bash
mkdir -p notebooks sql config
```

Estrutura final:

```
databricks-medallion/
|---- notebooks/
|   |---- 01_bronze_ingestion.py
|   |---- 02_silver_transformation.py
|   |---- 03_gold_aggregation.py
|---- sql/
|   |---- gold_queries.sql
|---- config/
|   |---- pipeline_config.yaml
|---- .gitignore
|---- README.md
```

4.7 Conectar GitHub ao Databricks

```bash
# Via CLI -- conecta o repo ao workspace
databricks repos create \
  https://github.com/SEU_USER/databricks-medallion gitHub \
  --path /Repos/seu@email.com/databricks-medallion
```

Resposta esperada:

```json
{
  "branch": "master",
  "head_commit_id": "abc123...",
  "path": "/Repos/seu@email.com/databricks-medallion",
  "provider": "gitHub"
}
```

> *Dica: Sempre que fizer push no GitHub, sincronize o Databricks com: `bash databricks repos update REPO_ID --branch master `*

---

5. Fase 2 -- Pipeline Medallion

5.1 Configuração do Pipeline

Arquivo: config/pipeline_config.yaml

```yaml
catalog: workspace
schema_bronze: medallion_bronze
schema_silver: medallion_silver
schema_gold: medallion_gold

source:
  name: nyctaxi
  path: /databricks-datasets/nyctaxi/tables/nyctaxi_yellow

tables:
  bronze: taxi_trips_raw
  silver: taxi_trips_cleaned
  gold_daily: taxi_daily_metrics
  gold_zone: taxi_zone_metrics
  gold_hourly: taxi_hourly_metrics
```

5.2 Bronze -- Ingestão Raw

Arquivo: notebooks/01_bronze_ingestion.py

O notebook Bronze faz a ingestão sem transformações, adicionando apenas metadados:

```python
# Configuração
CATALOG = "workspace"
SCHEMA_BRONZE = "medallion_bronze"

# Criar schema
spark.sql(f"CREATE SCHEMA IF NOT EXISTS {CATALOG}.{SCHEMA_BRONZE}")

# Ler dados brutos
from pyspark.sql.functions import current_timestamp, lit

SOURCE_PATH = "/databricks-datasets/nyctaxi/tables/nyctaxi_yellow"
df_raw = spark.read.format("delta").load(SOURCE_PATH)

# Adicionar metadados de ingestão
df_bronze = (
    df_raw
    .withColumn("_ingestion_timestamp", current_timestamp())
    .withColumn("_source", lit("nyctaxi_yellow"))
)

# Gravar como Delta Table
BRONZE_TABLE = f"{CATALOG}.{SCHEMA_BRONZE}.taxi_trips_raw"
(
    df_bronze.write
    .format("delta")
    .mode("overwrite")
    .option("overwriteSchema", "true")
    .saveAsTable(BRONZE_TABLE)
)
```

Schema da tabela Bronze:

Coluna	Tipo	Descrição
vendor_id	string	Empresa de táxi
pickup_datetime	timestamp	Início da corrida
dropoff_datetime	timestamp	Fim da corrida
passenger_count	int	Número de passageiros
trip_distance	double	Distância (milhas)
pickup_longitude	double	Longitude de origem
pickup_latitude	double	Latitude de origem
dropoff_longitude	double	Longitude de destino
dropoff_latitude	double	Latitude de destino
rate_code_id	int	Código da tarifa
payment_type	string	Forma de pagamento
fare_amount	double	Valor da tarifa
extra	double	Taxas extras
mta_tax	double	Imposto MTA
tip_amount	double	Gorjeta
tolls_amount	double	Pedágios
total_amount	double	Valor total
_ingestion_timestamp	timestamp	Quando foi ingerido
_source	string	Fonte dos dados

5.3 Silver -- Limpeza e Transformação

Arquivo: notebooks/02_silver_transformation.py

Transformações aplicadas:

1. Remoção de nulos em campos críticos (datetime, coordenadas)
2. Filtro de corridas inválidas (distância <= 0, tarifa <= 0, passageiros <= 0)
3. Filtro de coordenadas inválidas (lat/long = 0)
4. Duração da corrida calculada em minutos
5. Filtro de durações absurdas (< 1 min ou > 5h)
6. Velocidade média calculada (mph)
7. Filtro de velocidades absurdas (> 100 mph)
8. Features temporais: hora, dia da semana, data
9. Custo por milha e percentual de gorjeta
10. Remoção de duplicatas

python

```
from pyspark.sql.functions import (
    col, when, round as spark_round,
    unix_timestamp, hour, dayofweek,
    date_format, current_timestamp
)

df_silver = (
    df_bronze
    .filter(
        col("pickup_datetime").isNotNull() &
        col("dropoff_datetime").isNotNull() &
        col("pickup_latitude").isNotNull() &
        col("pickup_longitude").isNotNull()
    )
    .filter(
        (col("trip_distance") > 0) &
        (col("fare_amount") > 0) &
        (col("passenger_count") > 0)
    )
    .filter(
        (col("pickup_latitude") != 0) &
        (col("pickup_longitude") != 0) &
        (col("dropoff_latitude") != 0) &
        (col("dropoff_longitude") != 0)
    )
    .withColumn(
        "trip_duration_min",
        spark_round(
            (unix_timestamp("dropoff_datetime") -
             unix_timestamp("pickup_datetime")) / 60, 2
        )
    )
    .filter(
        (col("trip_duration_min") >= 1) &
        (col("trip_duration_min") <= 300)
    )
    .withColumn(
        "avg_speed_mph",
        spark_round(
            col("trip_distance") / (col("trip_duration_min") / 60), 2
        )
    )
    .filter(col("avg_speed_mph") <= 100)
    .withColumn("pickup_hour", hour("pickup_datetime"))
    .withColumn("pickup_day_of_week", dayofweek("pickup_datetime"))
    .withColumn("pickup_date",
                date_format("pickup_datetime", "yyyy-MM-dd"))
    .withColumn(
        "cost_per_mile",
        spark_round(col("fare_amount") / col("trip_distance"), 2)
    )
    .withColumn(
        "tip_percentage",
        spark_round(
            (col("tip_amount") / col("fare_amount")) * 100, 2
        )
    )
```

```
    )
    .drop("_ingestion_timestamp", "_source")
    .withColumn("_silver_timestamp", current_timestamp())
    .dropDuplicates()
)
```

> *Cuidado com isnan()! A função isnan() do PySpark só funciona com tipos DOUBLE e FLOAT. Se aplicada em colunas TIMESTAMP ou STRING, causa o erro: ` DATATYPE_MISMATCH.UNEXPECTED_INPUT_TYPE ` Solução: verificar o tipo da coluna antes de usar isnan(): `python from pyspark.sql.types import DoubleType, FloatType numeric_types = (DoubleType, FloatType) for field in df.schema.fields: if isinstance(field.dataType, numeric_types): # Usar isnull() | isnan() else: # Usar apenas isnull() `*

5.4 Gold -- Agregações de Negócio

Arquivo: notebooks/03_gold_aggregation.py

São criadas 3 tabelas Gold:

Gold 1 -- Métricas Diárias

```python
df_daily = (
    df_silver
    .groupBy("pickup_date")
    .agg(
        count("*").alias("total_trips"),
        spark_round(spark_sum("fare_amount"), 2)
            .alias("total_revenue"),
        spark_round(spark_sum("tip_amount"), 2)
            .alias("total_tips"),
        spark_round(avg("fare_amount"), 2).alias("avg_fare"),
        spark_round(avg("trip_distance"), 2)
            .alias("avg_distance_miles"),
        spark_round(avg("trip_duration_min"), 2)
            .alias("avg_duration_min"),
        spark_round(avg("tip_percentage"), 2)
            .alias("avg_tip_pct"),
        spark_round(avg("avg_speed_mph"), 2)
            .alias("avg_speed_mph"),
        spark_round(avg("passenger_count"), 1)
            .alias("avg_passengers"),
        spark_round(spark_sum("total_amount"), 2)
            .alias("total_amount_collected")
    )
)
```

Gold 2 -- Métricas por Zona Geográfica

Como o dataset usa lat/long (não Location IDs), criamos zonas arredondando coordenadas para 2 casas decimais (~1.1 km de precisão):

```python
df_zone = (
    df_silver
    .withColumn("zone_lat",
                spark_round(col("pickup_latitude"), 2))
    .withColumn("zone_lon",
                spark_round(col("pickup_longitude"), 2))
    .groupBy("zone_lat", "zone_lon")
    .agg(
        count("*").alias("total_trips"),
        spark_round(spark_sum("fare_amount"), 2)
            .alias("total_revenue"),
        spark_round(avg("fare_amount"), 2).alias("avg_fare"),
        spark_round(avg("trip_distance"), 2)
            .alias("avg_distance_miles"),
        spark_round(avg("tip_percentage"), 2)
            .alias("avg_tip_pct"),
        spark_round(avg("trip_duration_min"), 2)
            .alias("avg_duration_min")
    )
)
```

Gold 3 -- Métricas por Hora e Dia da Semana

```python
df_hourly = (
    df_silver
    .groupBy("pickup_hour", "pickup_day_of_week")
    .agg(
        count("*").alias("total_trips"),
        spark_round(avg("fare_amount"), 2).alias("avg_fare"),
        spark_round(avg("trip_distance"), 2)
            .alias("avg_distance_miles"),
        spark_round(avg("tip_percentage"), 2)
            .alias("avg_tip_pct"),
        spark_round(avg("trip_duration_min"), 2)
            .alias("avg_duration_min"),
        spark_round(avg("avg_speed_mph"), 2)
            .alias("avg_speed_mph")
    )
    .withColumn(
        "day_name",
        when(col("pickup_day_of_week") == 1, "Sunday")
        .when(col("pickup_day_of_week") == 2, "Monday")
        .when(col("pickup_day_of_week") == 3, "Tuesday")
        .when(col("pickup_day_of_week") == 4, "Wednesday")
        .when(col("pickup_day_of_week") == 5, "Thursday")
        .when(col("pickup_day_of_week") == 6, "Friday")
        .when(col("pickup_day_of_week") == 7, "Saturday")
    )
)
```

---

6. Fase 3 -- SQL Analytics e Genie Spaces

6.1 SQL Warehouse

O Databricks Free Edition já vem com um Serverless Starter Warehouse pré-configurado. Verifique via CLI:

```bash
databricks warehouses list
```

Resultado esperado:

```
ID                Name                          Size      State
4536ce4e53b91be3  Serverless Starter Warehouse  2X-Small  STOPPED
```

O warehouse inicia automaticamente quando você executa uma query no SQL Editor.

6.2 Queries Analíticas

Arquivo: sql/gold_queries.sql

Top 10 dias com maior receita

```sql
SELECT pickup_date, total_trips, total_revenue,
       total_tips, avg_fare, avg_distance_miles
FROM workspace.medallion_gold.taxi_daily_metrics
ORDER BY total_revenue DESC
LIMIT 10;
```

Top 20 zonas mais movimentadas

```sql
SELECT zone_lat, zone_lon, total_trips,
       total_revenue, avg_fare, avg_tip_pct
FROM workspace.medallion_gold.taxi_zone_metrics
ORDER BY total_trips DESC
LIMIT 20;
```

Padrão de corridas por hora do dia

```sql
SELECT pickup_hour,
       SUM(total_trips) AS total_trips,
       ROUND(AVG(avg_fare), 2) AS avg_fare,
       ROUND(AVG(avg_speed_mph), 2) AS avg_speed
FROM workspace.medallion_gold.taxi_hourly_metrics
GROUP BY pickup_hour
ORDER BY pickup_hour;
```

Dias úteis vs Fim de semana

```sql
SELECT
    CASE
        WHEN pickup_day_of_week IN (1, 7) THEN 'Weekend'
        ELSE 'Weekday'
    END AS period,
    SUM(total_trips) AS total_trips,
    ROUND(AVG(avg_fare), 2) AS avg_fare,
    ROUND(AVG(avg_tip_pct), 2) AS avg_tip_pct
FROM workspace.medallion_gold.taxi_hourly_metrics
GROUP BY
    CASE
        WHEN pickup_day_of_week IN (1, 7) THEN 'Weekend'
        ELSE 'Weekday'
    END;
```

6.3 Configurar Genie Space

O Genie Space permite fazer perguntas em linguagem natural sobre seus dados.

Passo a passo:

1. No menu lateral do Databricks, clique em Genie
2. Clique em New ou Create Genie Space
3. Configure:
   - Name: NYC Taxi Analytics
   - SQL Warehouse: Serverless Starter Warehouse
   - Tables: adicione as 3 tabelas Gold:
     - workspace.medallion_gold.taxi_daily_metrics
     - workspace.medallion_gold.taxi_zone_metrics
     - workspace.medallion_gold.taxi_hourly_metrics
4. Em General instructions, cole:

```
Você é um assistente de análise de dados de corridas de táxi
de Nova York (NYC Yellow Taxi).

Contexto dos dados:
- taxi_daily_metrics: métricas agregadas por dia
- taxi_zone_metrics: métricas por zona geográfica
- taxi_hourly_metrics: métricas por hora e dia da semana

Colunas importantes:
- total_trips = número de corridas
- total_revenue = receita total
- avg_fare = tarifa média por corrida
- avg_tip_pct = percentual médio de gorjeta
- avg_speed_mph = velocidade média (milhas/hora)
- pickup_day_of_week: 1=Domingo ... 7=Sábado
- day_name: nome do dia em inglês

Responda em português. Use formatos amigáveis para
valores monetários ($ com separador de milhar).
```

Exemplos de perguntas para testar:

- "Qual foi o dia com mais corridas?"
- "Qual a média de gorjeta no fim de semana vs dia útil?"
- "Quais são as 5 zonas com maior receita?"
- "Como varia a velocidade média por hora do dia?"
- "Qual o total de receita no mês de janeiro de 2013?"

---

7. Fase 4 -- Orquestração com Workflows

7.1 Criar o Workflow via CLI

O Databricks Free Edition usa Serverless Compute, então o job não define cluster -- o compute é gerenciado automaticamente:

```bash
databricks jobs create --json '{
  "name": "Medallion Pipeline - NYC Taxi",
  "tasks": [
    {
      "task_key": "bronze_ingestion",
      "notebook_task": {
        "notebook_path": "/Repos/seu@email.com/databricks-medallion/notebooks/01_bronze_inge...
        "source": "WORKSPACE"
      },
      "environment_key": "default"
    },
    {
      "task_key": "silver_transformation",
      "depends_on": [{"task_key": "bronze_ingestion"}],
      "notebook_task": {
        "notebook_path": "/Repos/seu@email.com/databricks-medallion/notebooks/02_silver_tran...
        "source": "WORKSPACE"
      },
      "environment_key": "default"
    },
    {
      "task_key": "gold_aggregation",
      "depends_on": [{"task_key": "silver_transformation"}],
      "notebook_task": {
        "notebook_path": "/Repos/seu@email.com/databricks-medallion/notebooks/03_gold_aggreg...
        "source": "WORKSPACE"
      },
      "environment_key": "default"
    }
  ],
  "environments": [
    {
      "environment_key": "default",
      "spec": {"client": "1"}
    }
  ],
  "format": "MULTI_TASK"
}'
```

7.2 Executar o Workflow

```bash
# Executar manualmente
databricks jobs run-now JOB_ID

# Verificar status
databricks jobs list
databricks runs list --job-id JOB_ID
```

7.3 Fluxo de execução

```
bronze_ingestion ---> silver_transformation ---> gold_aggregation
    (raw data)          (limpeza)               (agregações)
```

Cada task só inicia após a anterior completar com sucesso. Se uma task falhar, as dependentes não executam.

---

8. Troubleshooting

Erro: DATATYPE_MISMATCH.UNEXPECTED_INPUT_TYPE (isnan)

Causa: isnan() aplicado em coluna TIMESTAMP ou STRING. Solução: Verificar o tipo da coluna antes de usar isnan(). Usar apenas isnull() para tipos não-numéricos.

Erro: UNRESOLVED_COLUMN.WITH_SUGGESTION

Causa: Nome de coluna incorreto. O dataset NYC Taxi no Databricks usa nomes diferentes dependendo da versão. Solução: Sempre execute DESCRIBE tabela antes de escrever transformações:

```sql
DESCRIBE workspace.medallion_bronze.taxi_trips_raw
```

Nomes comuns que variam entre versões:

Versão antiga	Versão nova
pickup_datetime	tpep_pickup_datetime
pickup_latitude / longitude	PULocationID
dropoff_latitude / longit..	DOLocationID

Erro: Only serverless compute is supported

Causa: Databricks Free Edition não permite clusters customizados. Solução: Usar environment_key no job em vez de new_cluster.

Erro: source is set to GIT but no git_source

Causa: Ao criar um job com "source": "GIT", é necessário informar o git_source. Solução: Usar "source": "WORKSPACE" para apontar para o repo já sincronizado via Databricks Repos.

---

9. Próximos Passos

Após completar este tutorial, considere:

1. Dashboards -- Criar dashboards interativos no Databricks SQL
2. Agendamento -- Configurar schedule no Workflow (ex: diário)

3. Alertas -- Criar alertas SQL (ex: receita diária abaixo de threshold)
4. Dados reais -- Aplicar o Medallion pattern nos seus próprios dados
5. Delta Live Tables -- Migrar o pipeline para DLT (declarativo)
6. CI/CD -- Automatizar deploy via GitHub Actions
7. Data Quality -- Adicionar expectations e validações automáticas

---

Resumo do Fluxo Completo

```
Desenvolvimento Local (VS Code + CLI)
        |
        |  git push
        v
    GitHub Repo
        |
        |  databricks repos sync
        v
Databricks Workspace
        |
        |---- Repos (código versionado)
        |---- Notebooks (Bronze -> Silver -> Gold)
        |---- Unity Catalog (tabelas Delta)
        |---- SQL Warehouse (queries)
        |---- Genie Spaces (linguagem natural)
        |---- Workflows (orquestração)
```

Tutorial criado como parte do projeto databricks-medallion. Repositório: github.com/tlcharchar/databricks-medallion