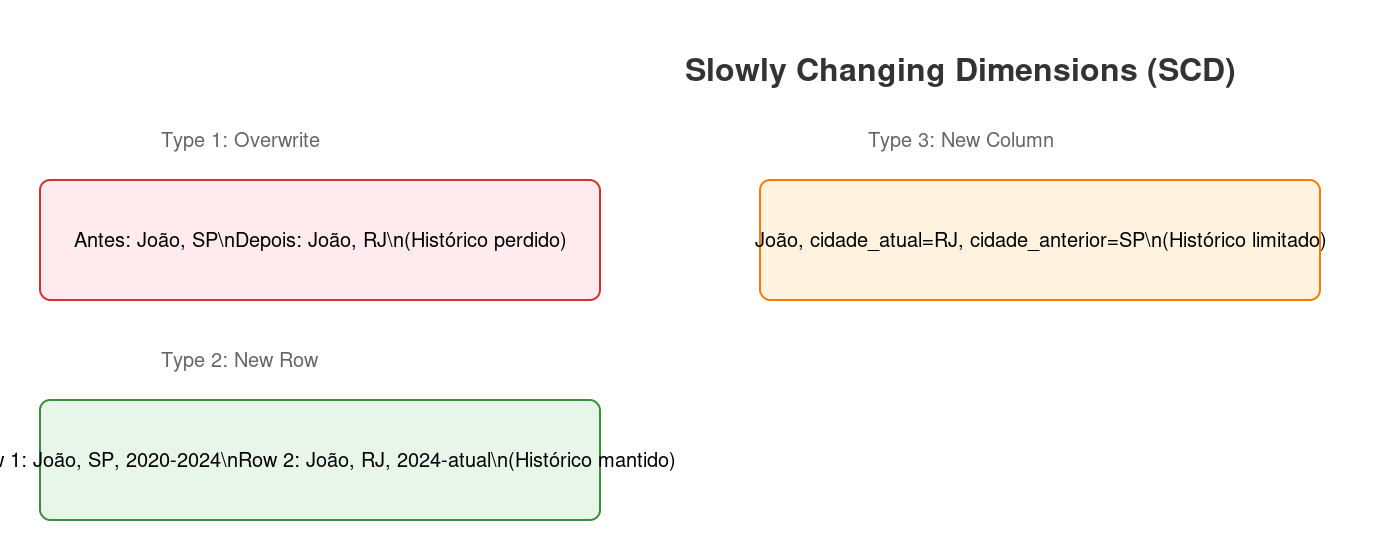
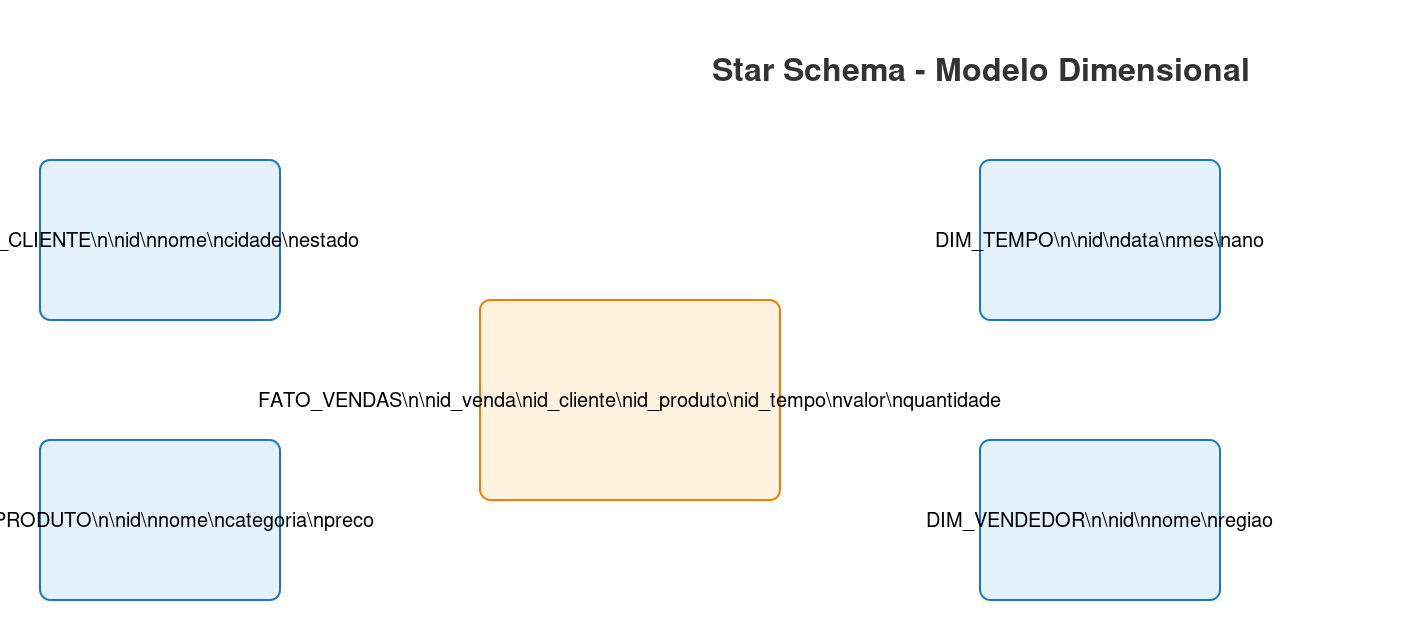
# Capítulo 8: Streaming e Orquestração com Delta Lake



*Cap08 SCD Types*



*Cap08 Star Schema*

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender Structured Streaming no Apache Spark
* Processar dados em tempo real com Delta Lake
* Criar pipelines de streaming (fonte → sink)
* Implementar Change Data Capture (CDC)
* Usar checkpointing para tolerância a falhas
* Monitorar streaming queries
* Orquestrar pipelines Delta com Azure Data Factory
* Implementar batch + streaming na mesma tabela
* Otimizar streaming performance

**Tempo estimado:** 4-5 horas

**Custo Azure:** Incluído no Databricks (mesmo custo de cluster)

## 8.1 O que é Structured Streaming?

**Structured Streaming** é a API de streaming do Apache Spark que processa dados em tempo real usando DataFrames.

### Streaming vs Batch

|  |  |  |
| --- | --- | --- |
| Característica | Batch | Streaming |
| **Processamento** | Dados finitos (estáticos) | Dados infinitos (contínuos) |
| **Latência** | Minutos/Horas | Segundos/Milissegundos |
| **Trigger** | Manual/Agendado | Contínuo/Microbatch |
| **Use cases** | ETL diário, reports | IoT, eventos, logs em tempo real |

### Como Funciona

**Conceito:** Trata streaming como tabela infinita que cresce continuamente.

Batch DataFrame:

+---+-------+

| id| value |

+---+-------+

| 1 | Alice |

| 2 | Bob |

+---+-------+

Streaming DataFrame (conceitual):

+---+-------+

| id| value |

+---+-------+

| 1 | Alice | ← Dados antigos

| 2 | Bob |

| 3 | Carol | ← Novos dados chegando

| 4 | David | ← Continuamente...

...

**Spark processa em microbatches:**

* A cada intervalo (ex: 10s), processa novos dados
* Mantém estado (checkpointing)
* Garante exactly-once processing

## 8.2 Fontes de Streaming Suportadas

**Principais fontes:**

1. **File Source (Auto Loader):**
   * Monitora pasta no Data Lake
   * Processa novos arquivos automaticamente
   * Formatos: CSV, JSON, Parquet, Delta
2. **Kafka:**
   * Apache Kafka (event streaming platform)
   * Azure Event Hubs (Kafka compatível)
3. **Delta Lake:**
   * Tabela Delta como fonte de streaming (CDC)
4. **Azure Event Hubs:**
   * Streaming de eventos
5. **Socket Source:**
   * Teste/desenvolvimento (não use em produção)

## 8.3 Hands-On: Streaming Básico (File Source)

Vamos criar streaming que monitora pasta no Data Lake e processa novos arquivos.

### Passo 1: Configurar Source (Auto Loader)

**PYTHON**

# Caminho de entrada (pasta monitorada)

input\_path = “/mnt/datalake/streaming\_input/”

# Schema dos dados

from pyspark.sql.types import StructType, StructField, IntegerType, StringType

schema = StructType([

StructField(“id”, IntegerType(), True),

StructField(“nome”, StringType(), True),

StructField(“evento”, StringType(), True),

StructField(“timestamp”, StringType(), True)

])

# Criar streaming DataFrame

df\_stream = spark.readStream \

.format(“cloudFiles”) \

.option(“cloudFiles.format”, “json”) \

.schema(schema) \

.load(input\_path)

print(“Streaming source configurado!”)

**cloudFiles (Auto Loader):**

* Databricks-specific (otimizado)
* Detecta novos arquivos automaticamente
* Suporta schema inference e evolution

### Passo 2: Transformações (Mesmo que Batch)

**PYTHON**

from pyspark.sql.functions import col, current\_timestamp

# Transformações (igual batch!)

df\_transformed = df\_stream \

.filter(col(“evento”) == “compra”) \

.withColumn(“processed\_at”, current\_timestamp())

# Streaming DataFrame se comporta como batch DataFrame!

### Passo 3: Configurar Sink (Delta Lake)

**PYTHON**

# Caminho de saída

output\_path = “/mnt/datalake/silver/eventos\_compra/”

checkpoint\_path = “/mnt/datalake/checkpoints/eventos\_compra/”

# Escrever streaming para Delta Lake

query = df\_transformed.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.start(output\_path)

print(f”Streaming query iniciado: {query.id}”)

**Parâmetros importantes:**

* **outputMode:**
  + append - Adiciona novos registros (padrão, mais comum)
  + complete - Reescreve tabela inteira (só para agregações)
  + update - Atualiza registros existentes (agregações)
* **checkpointLocation:**
  + Pasta onde Spark salva estado
  + Garante exactly-once processing (tolerância a falhas)
  + **OBRIGATÓRIO** em produção!

### Passo 4: Simular Dados Chegando

**Em outro notebook ou célula, adicionar arquivos:**

**PYTHON**

# Simular chegada de dados

import time

for i in range(5):

data = [

(i\*10 + 1, “Cliente A”, “compra”, “2024-02-09 08:00:00”),

(i\*10 + 2, “Cliente B”, “visualizacao”, “2024-02-09 08:01:00”),

(i\*10 + 3, “Cliente C”, “compra”, “2024-02-09 08:02:00”)

]

df\_batch = spark.createDataFrame(data, schema)

# Gravar como JSON (simula novo arquivo)

df\_batch.write \

.mode(“append”) \

.json(f”{input\_path}/batch\_{i}.json”)

print(f”Arquivo {i} adicionado!”)

time.sleep(10) # Aguardar 10s

**Streaming automaticamente processa novos arquivos!**

### Passo 5: Monitorar Query

**PYTHON**

# Status da query

query.status

# Aguardar terminar (não usar em produção - roda infinitamente!)

# query.awaitTermination()

# Parar query

query.stop()

**Ver dados processados:**

**PYTHON**

# Ler tabela Delta (batch)

df\_result = spark.read.format(“delta”).load(output\_path)

df\_result.show()

**Output:**

+---+----------+-------+-------------------+-------------------+

| id| nome| evento| timestamp| processed\_at|

+---+----------+-------+-------------------+-------------------+

| 1|Cliente A| compra|2024-02-09 08:00:00|2024-02-09 08:05:10|

| 3|Cliente C| compra|2024-02-09 08:02:00|2024-02-09 08:05:10|

| 11|Cliente A| compra|2024-02-09 08:00:00|2024-02-09 08:05:20|

...

## 8.4 Delta Lake como Fonte de Streaming

Delta Lake pode ser **fonte** de streaming (Change Data Capture - CDC).

### Use Case: Processar Mudanças em Tempo Real

**PYTHON**

# Tabela Delta como fonte

source\_path = “/mnt/datalake/silver/clientes\_delta/”

# Streaming de mudanças (CDC)

df\_changes = spark.readStream \

.format(“delta”) \

.load(source\_path)

# Processar mudanças

df\_changes.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, “/mnt/datalake/checkpoints/clientes\_cdc/”) \

.start(“/mnt/datalake/gold/clientes\_processados/”)

**Vantagem:** Toda INSERT/UPDATE/DELETE na tabela origem dispara processamento downstream!

### Filtrar por Versão (Incremental)

**PYTHON**

# Processar apenas mudanças após versão 10

df\_incremental = spark.readStream \

.format(“delta”) \

.option(“startingVersion”, 10) \

.load(source\_path)

## 8.5 Triggers (Modos de Execução)

**Triggers** controlam quando/como processar microbatches.

### Default (Microbatch Contínuo)

**PYTHON**

# Processa assim que novos dados chegam (latência mínima)

query = df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.start(output\_path)

### Fixed Interval (Intervalo Fixo)

**PYTHON**

# Processa a cada 30 segundos

query = df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.trigger(processingTime=“30 seconds”) \

.start(output\_path)

**Use quando:**

* Não precisa latência baixíssima
* Reduzir overhead de microbatches frequentes

### Once (Uma Vez)

**PYTHON**

# Processa uma vez e para

query = df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.trigger(once=True) \

.start(output\_path)

query.awaitTermination()

**Use quando:**

* Processar dados acumulados (quase batch)
* Agendamento externo (ADF chama notebook)

### Available Now (Processar Tudo Disponível)

**PYTHON**

# Processa tudo disponível no momento e para

query = df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.trigger(availableNow=True) \

.start(output\_path)

query.awaitTermination()

## 8.6 Agregações em Streaming

### Agregações Simples (Append Mode)

**PYTHON**

from pyspark.sql.functions import count

# Contar eventos por tipo

df\_agg = df\_stream.groupBy(“evento”).count()

# Sink (Complete Mode - reescreve resultado)

df\_agg.writeStream \

.format(“memory”) \

.outputMode(“complete”) \

.queryName(“contagem\_eventos”) \

.start()

# Consultar (em outra célula)

spark.sql(“SELECT \* FROM contagem\_eventos”).show()

**Output:**

+-------------+-----+

| evento|count|

+-------------+-----+

| compra| 15|

|visualizacao | 30|

+-------------+-----+

### Window Aggregations (Janelas de Tempo)

**PYTHON**

from pyspark.sql.functions import window

# Converter timestamp string para timestamp type

from pyspark.sql.functions import to\_timestamp

df\_stream\_ts = df\_stream.withColumn(

“timestamp”,

to\_timestamp(col(“timestamp”), “yyyy-MM-dd HH:mm:ss”)

)

# Agregação por janela de 10 minutos

df\_windowed = df\_stream\_ts \

.groupBy(

window(col(“timestamp”), “10 minutes”),

“evento”

) \

.count()

df\_windowed.writeStream \

.format(“delta”) \

.outputMode(“complete”) \

.option(“checkpointLocation”, “/mnt/datalake/checkpoints/windowed/”) \

.start(“/mnt/datalake/gold/eventos\_windowed/”)

**Output:**

+------------------------------------------+-------+-----+

|window | evento|count|

+------------------------------------------+-------+-----+

|{2024-02-09 08:00:00, 2024-02-09 08:10:00}| compra| 5|

|{2024-02-09 08:10:00, 2024-02-09 08:20:00}| compra| 8|

...

## 8.7 Checkpointing e Tolerância a Falhas

**Checkpointing** é essencial para produção.

### Como Funciona

Checkpoint Directory:

commits/ (transações commitadas)

metadata/ (configuração da query)

offsets/ (posição de leitura de cada fonte)

state/ (estado de agregações)

**Garantia:**

* Se cluster falhar, Spark retoma do último checkpoint
* Exactly-once processing (nenhum dado processado 2x ou perdido)

### Recovery Após Falha

**PYTHON**

# Query com checkpoint

query = df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, “/mnt/datalake/checkpoints/my\_query/”) \

.start(“/mnt/datalake/output/”)

# Se falhar e reiniciar, Spark:

# 1. Lê checkpoint

# 2. Retoma do offset exato

# 3. Continua processamento

## 8.8 Monitoramento de Streaming

### Query Progress

**PYTHON**

# Últimas métricas

query.lastProgress

**Output (JSON):**

**JSON**

{

“id”: “abc123”,

“runId”: “xyz789”,

“name”: null,

“timestamp”: “2024-02-09T08:10:00.000Z”,

“batchId”: 5,

“numInputRows”: 1000,

“inputRowsPerSecond”: 100.0,

“processedRowsPerSecond”: 200.0,

“durationMs”: {

“addBatch”: 450,

“getBatch”: 50,

“latestOffset”: 10,

“queryPlanning”: 40,

“triggerExecution”: 550,

“walCommit”: 50

},

“stateOperators”: [],

“sources”: [

{

“description”: “CloudFilesSource[/mnt/datalake/input]”,

“startOffset”: 100,

“endOffset”: 200,

“numInputRows”: 1000,

“inputRowsPerSecond”: 100.0,

“processedRowsPerSecond”: 200.0

}

],

“sink”: {

“description”: “DeltaSink[/mnt/datalake/output]”

}

}

**Métricas importantes:**

* numInputRows - Linhas processadas neste batch
* inputRowsPerSecond - Taxa de entrada
* processedRowsPerSecond - Taxa de processamento
* durationMs - Tempo gasto em cada fase

### Monitorar Todas Queries Ativas

**PYTHON**

# Listar queries ativas

spark.streams.active

# Informações de cada query

for query in spark.streams.active:

print(f”Query ID: {query.id}”)

print(f”Status: {query.status}”)

print(f”Last progress: {query.lastProgress}”)

## 8.9 Integração com Azure Data Factory

ADF pode orquestrar streaming jobs Databricks.

### Opção 1: Notebook com Trigger Once

**No Databricks notebook:**

**PYTHON**

# Streaming com trigger once (processa e para)

df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.trigger(once=True) \

.start(output\_path) \

.awaitTermination()

print(“Processamento concluído!”)

**No ADF:**

* Notebook Activity chamando este notebook
* Schedule Trigger (ex: a cada 15 min)

**Resultado:** ADF executa notebook periodicamente, processando dados acumulados.

### Opção 2: Streaming Contínuo Gerenciado

**No Databricks:**

* Job permanente (não notebook)
* Cluster dedicado com auto-restart

**No ADF:**

* Não necessário (streaming roda independente)
* ADF só usado para orquestrar ETL batch

## 8.10 Batch + Streaming na Mesma Tabela Delta

**Vantagem Delta Lake:** Mesma tabela serve batch e streaming!

**PYTHON**

# Streaming WRITE

df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.start(“/mnt/datalake/unified\_table/”)

# Batch READ (em outro notebook)

df\_batch = spark.read.format(“delta”).load(“/mnt/datalake/unified\_table/”)

df\_batch.show()

# Streaming READ (CDC)

df\_stream\_read = spark.readStream.format(“delta”).load(“/mnt/datalake/unified\_table/”)

**Sem conflitos!** Delta Lake gerencia concorrência.

## 8.11 Otimizações de Streaming

### Auto Loader vs readStream (File Source)

**Auto Loader (cloudFiles):**

* Detecta novos arquivos eficientemente (file notification + queue)
* Schema inference automático
* Escalável (milhões de arquivos)

**readStream (Spark nativo):**

* Lista arquivos toda vez (lento em pastas grandes)
* Sem schema inference

**Recomendação:** Use Auto Loader!

### Particionamento de Sink

**PYTHON**

# Particionar saída por data (melhora leitura batch posterior)

df\_stream.writeStream \

.format(“delta”) \

.outputMode(“append”) \

.option(“checkpointLocation”, checkpoint\_path) \

.partitionBy(“date”) \

.start(output\_path)

### Tuning de Microbatch Size

**PYTHON**

# Aumentar tamanho de microbatch (menos overhead)

spark.conf.set(“spark.sql.streaming.schemaInference”, “true”)

spark.conf.set(“spark.sql.streaming.fileSource.log.cleanupDelay”, “3600000”) # 1 hora

# Reduzir arquivos por microbatch (se muito grande)

df\_stream.writeStream \

.format(“delta”) \

.option(“checkpointLocation”, checkpoint\_path) \

.option(“maxFilesPerTrigger”, 100) \

.start(output\_path)

## 8.12 Change Data Capture (CDC) Completo

### Cenário: Replicar SQL Database → Delta Lake

**Fonte:** Tabela SQL com CDC habilitado (captura INSERT/UPDATE/DELETE)

**Pipeline:**

1. **CDC Source (SQL):**
   * Ferramenta: Debezium, Azure Data Factory CDC
   * Grava mudanças em Event Hub ou Data Lake (JSON)
2. **Streaming Databricks:**
   * Lê mudanças (stream)
   * Aplica em Delta Lake (MERGE)

**Exemplo simplificado:**

**PYTHON**

# Ler CDC stream (JSON com tipo de operação)

df\_cdc = spark.readStream \

.format(“cloudFiles”) \

.option(“cloudFiles.format”, “json”) \

.schema(cdc\_schema) \

.load(“/mnt/datalake/cdc\_events/”)

# Schema CDC

cdc\_schema = StructType([

StructField(“operation”, StringType()), # INSERT, UPDATE, DELETE

StructField(“id”, IntegerType()),

StructField(“nome”, StringType()),

StructField(“idade”, IntegerType())

])

# Processar cada microbatch

def process\_cdc\_batch(df, batch\_id):

from delta.tables import DeltaTable

delta\_table = DeltaTable.forPath(spark, “/mnt/datalake/silver/clientes/”)

# Separar operações

inserts = df.filter(df.operation == “INSERT”).drop(“operation”)

updates = df.filter(df.operation == “UPDATE”).drop(“operation”)

deletes = df.filter(df.operation == “DELETE”)

# Aplicar INSERT

inserts.write.format(“delta”).mode(“append”).save(“/mnt/datalake/silver/clientes/”)

# Aplicar UPDATE (MERGE)

delta\_table.alias(“target”).merge(

updates.alias(“source”),

“target.id = source.id”

).whenMatchedUpdateAll().execute()

# Aplicar DELETE

for row in deletes.collect():

delta\_table.delete(f”id = {row.id}”)

# Executar

df\_cdc.writeStream \

.foreachBatch(process\_cdc\_batch) \

.option(“checkpointLocation”, “/mnt/datalake/checkpoints/cdc/”) \

.start()

## 8.13 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar Structured Streaming
* [ ] Diferenciar batch vs streaming
* [ ] Entender microbatches e checkpointing

**Práticos:**

* [ ] Criar streaming source (Auto Loader)
* [ ] Aplicar transformações em streaming DataFrame
* [ ] Gravar em Delta Lake (sink)
* [ ] Configurar checkpointing
* [ ] Usar diferentes triggers (once, processingTime)
* [ ] Fazer agregações em streaming (window)
* [ ] Monitorar queries (lastProgress)
* [ ] Integrar com ADF (trigger once)

**Avançado:**

* [ ] Implementar CDC com Delta Lake
* [ ] Usar Delta como fonte de streaming
* [ ] Otimizar performance (Auto Loader, partitioning)
* [ ] foreachBatch para lógica customizada

## 8.14 Próximos Passos

No **Capítulo 9** (último da Parte 2), você vai:

Padrões de ETL avançados (Incremental Load, Deduplication)

Data Quality Checks (validações automatizadas)

Error Handling e Dead Letter Queues

Monitoramento e Alertas em produção

CI/CD para pipelines Databricks

## 8.15 Recursos Complementares

**Documentação Oficial:**

* [Structured Streaming Guide](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html)
* [Delta Lake Streaming](https://docs.delta.io/latest/delta-streaming.html)
* [Auto Loader](https://learn.microsoft.com/azure/databricks/ingestion/auto-loader/)

**Tutoriais:**

* [Streaming with Delta Lake](https://learn.microsoft.com/azure/databricks/delta/delta-streaming)
* [CDC Tutorial](https://docs.delta.io/latest/delta-change-data-feed.html)

## 8.16 Exercícios Práticos

**Exercício 1: Streaming Básico**

Crie streaming pipeline:

1. Source: Auto Loader monitora pasta /streaming\_input/ (JSON)
2. Transformação: Filtrar apenas eventos tipo “compra”
3. Sink: Delta Lake /silver/compras/
4. Simular chegada de 10 arquivos JSON
5. Verificar dados processados

**Exercício 2: Window Aggregation**

Crie streaming com agregação:

1. Source: Mesma pasta anterior
2. Agregação: Contar eventos por tipo, janela de 5 minutos
3. Sink: Delta Lake /gold/eventos\_agregados/
4. Usar trigger processingTime=“30 seconds”

**Exercício 3: Delta CDC**

Implementar CDC:

1. Criar tabela Delta produtos
2. Configurar streaming que lê mudanças (readStream Delta)
3. Processar mudanças e gravar em tabela auditoria
4. Simular INSERT, UPDATE, DELETE
5. Verificar auditoria capturou tudo

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Structured Streaming:** Processar dados em tempo real com API DataFrame

**Auto Loader:** Monitorar pastas e processar novos arquivos automaticamente

**Delta Lake Streaming:**

* Sink (gravar stream em Delta)
* Source (ler mudanças de Delta - CDC)

**Triggers:** Default, processingTime, once, availableNow

**Agregações:** Simples e window aggregations

**Checkpointing:** Tolerância a falhas, exactly-once processing

**Monitoramento:** lastProgress, active queries

**Integração ADF:** Trigger once para orquestração

**CDC:** Change Data Capture com foreachBatch

**Batch + Streaming:** Mesma tabela Delta para ambos

**Próximo capítulo:** Padrões de ETL e Data Quality!

# Capítulo 9: Padrões de ETL e Data Quality

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Implementar Incremental Load (carga incremental)
* Realizar Deduplication (remover duplicatas)
* Criar Data Quality Checks automatizados
* Implementar Error Handling (tratamento de erros)
* Usar Dead Letter Queues (DLQ)
* Criar pipelines idempotentes
* Implementar logging e auditoria
* Monitorar e alertar falhas
* Aplicar best practices de ETL

**Tempo estimado:** 4-5 horas

**Custo Azure:** Incluído nos serviços já provisionados

## 9.1 Incremental Load (Carga Incremental)

**Problema:** Reprocessar toda tabela a cada ETL é ineficiente.

**Solução:** Carregar apenas dados novos/modificados.

### Estratégias de Incremental Load

**1. Watermark (Coluna de Data/Timestamp)**

**Requisito:** Tabela tem coluna last\_modified ou created\_at

**PYTHON**

# Ler última data processada (salva em arquivo de controle)

last\_run\_date = spark.read.text(“/mnt/datalake/control/last\_run.txt”) \

.collect()[0][0]

print(f”Última execução: {last\_run\_date}”)

# Carregar apenas dados novos

df\_incremental = spark.read \

.format(“delta”) \

.load(“/mnt/datalake/bronze/vendas/”) \

.filter(f”last\_modified > '{last\_run\_date}'“)

# Processar dados incrementais

df\_processed = df\_incremental.transform(minha\_transformacao)

# Gravar em silver

df\_processed.write \

.format(“delta”) \

.mode(“append”) \

.save(“/mnt/datalake/silver/vendas/”)

# Atualizar watermark

from pyspark.sql.functions import max as spark\_max, current\_timestamp

new\_watermark = df\_incremental.select(spark\_max(“last\_modified”)).collect()[0][0]

spark.createDataFrame([(str(new\_watermark),)], [“watermark”]) \

.write.mode(“overwrite”).text(“/mnt/datalake/control/last\_run.txt”)

print(f”Novo watermark: {new\_watermark}”)

**2. Change Data Capture (CDC)**

**Visto no Cap 8:** Delta Lake como fonte de streaming.

**PYTHON**

# Ler apenas mudanças desde última versão

df\_changes = spark.read \

.format(“delta”) \

.option(“readChangeFeed”, “true”) \

.option(“startingVersion”, 10) \

.load(“/mnt/datalake/bronze/clientes/”)

df\_changes.show()

**3. Merge (Upsert) Incremental**

**Cenário:** Recebe dados diários, alguns updates, alguns novos.

**PYTHON**

from delta.tables import DeltaTable

# Dados novos (incremental)

df\_new = spark.read.parquet(“/mnt/datalake/landing/vendas\_2024-02-09.parquet”)

# Tabela destino

delta\_table = DeltaTable.forPath(spark, “/mnt/datalake/silver/vendas/”)

# MERGE

delta\_table.alias(“target”).merge(

df\_new.alias(“source”),

“target.venda\_id = source.venda\_id”

).whenMatchedUpdateAll() \

.whenNotMatchedInsertAll() \

.execute()

print(“Merge incremental concluído!”)

## 9.2 Deduplication (Remover Duplicatas)

### Estratégia 1: dropDuplicates (Simples)

**PYTHON**

# Remover duplicatas baseado em coluna(s)

df\_unique = df.dropDuplicates([“customer\_id”, “order\_id”])

df\_unique.write \

.format(“delta”) \

.mode(“overwrite”) \

.save(“/mnt/datalake/silver/vendas\_unique/”)

### Estratégia 2: Window Function (Manter Mais Recente)

**Cenário:** Múltiplos registros para mesmo ID, manter apenas mais recente.

**PYTHON**

from pyspark.sql.window import Window

from pyspark.sql.functions import row\_number, desc

# Window particionada por ID, ordenada por timestamp

window\_spec = Window.partitionBy(“customer\_id”).orderBy(desc(“last\_modified”))

# Ranquear

df\_ranked = df.withColumn(“rank”, row\_number().over(window\_spec))

# Filtrar apenas rank 1 (mais recente)

df\_deduplicated = df\_ranked.filter(“rank = 1”).drop(“rank”)

df\_deduplicated.show()

### Estratégia 3: Deduplicate ao Gravar (Delta Lake)

**Delta Lake pode deduplicar automaticamente no MERGE:**

**PYTHON**

delta\_table.alias(“target”).merge(

df\_new.alias(“source”),

“target.id = source.id”

).whenMatchedUpdate(

condition=“source.last\_modified > target.last\_modified”, # Só atualiza se mais novo

set={“\*”: “source.\*”}

).whenNotMatchedInsertAll() \

.execute()

## 9.3 Data Quality Checks

**Data Quality Checks** validam dados automaticamente antes de gravar.

### Framework de Validações

**PYTHON**

from pyspark.sql.functions import col, count, isnan, when

def data\_quality\_checks(df, table\_name):

“““

Executa checks de qualidade e retorna DataFrame com falhas.

“““

checks = []

# 1. Null checks (colunas obrigatórias)

required\_cols = [“id”, “nome”, “data”]

for col\_name in required\_cols:

null\_count = df.filter(col(col\_name).isNull()).count()

checks.append({

“table”: table\_name,

“check”: f”{col\_name}\_not\_null”,

“passed”: null\_count == 0,

“failed\_records”: null\_count

})

# 2. Duplicates check

total\_records = df.count()

unique\_records = df.dropDuplicates([“id”]).count()

duplicates = total\_records - unique\_records

checks.append({

“table”: table\_name,

“check”: “no\_duplicates”,

“passed”: duplicates == 0,

“failed\_records”: duplicates

})

# 3. Range check (ex: idade entre 0-120)

if “idade” in df.columns:

invalid\_age = df.filter((col(“idade”) < 0) | (col(“idade”) > 120)).count()

checks.append({

“table”: table\_name,

“check”: “idade\_range\_0\_120”,

“passed”: invalid\_age == 0,

“failed\_records”: invalid\_age

})

# 4. Format check (ex: email válido)

if “email” in df.columns:

invalid\_email = df.filter(~col(“email”).rlike(r”^[\w\.-]+@[\w\.-]+\.\w+$”)).count()

checks.append({

“table”: table\_name,

“check”: “email\_format”,

“passed”: invalid\_email == 0,

“failed\_records”: invalid\_email

})

# Retornar DataFrame de checks

df\_checks = spark.createDataFrame(checks)

return df\_checks

# Executar checks

df = spark.read.format(“delta”).load(“/mnt/datalake/bronze/clientes/”)

df\_quality = data\_quality\_checks(df, “clientes”)

df\_quality.show(truncate=False)

**Output:**

+----------+------------------+------+---------------+

|table |check |passed|failed\_records|

+----------+------------------+------+---------------+

|clientes |id\_not\_null |true |0 |

|clientes |nome\_not\_null |false |3 |

|clientes |no\_duplicates |true |0 |

|clientes |idade\_range\_0\_120 |false |2 |

|clientes |email\_format |true |0 |

+----------+------------------+------+---------------+

### Salvar Resultados de Quality

**PYTHON**

# Gravar histórico de checks

from pyspark.sql.functions import current\_timestamp

df\_quality\_with\_ts = df\_quality.withColumn(“check\_timestamp”, current\_timestamp())

df\_quality\_with\_ts.write \

.format(“delta”) \

.mode(“append”) \

.save(“/mnt/datalake/monitoring/data\_quality\_checks/”)

### Fail Pipeline se Checks Falham

**PYTHON**

# Verificar se algum check falhou

failed\_checks = df\_quality.filter(“passed = false”).count()

if failed\_checks > 0:

print(“ Data Quality Checks falharam!”)

df\_quality.filter(“passed = false”).show()

raise Exception(“Pipeline abortado devido a falhas de qualidade.”)

else:

print(“ Todos checks passaram!”)

# Continuar pipeline...

## 9.4 Error Handling (Tratamento de Erros)

### Try-Except em PySpark

**PYTHON**

try:

# Transformações

df\_transformed = df.transform(complex\_transformation)

# Gravar

df\_transformed.write.format(“delta”).mode(“append”).save(output\_path)

print(“ Pipeline concluído com sucesso!”)

except Exception as e:

print(f” Erro no pipeline: {str(e)}”)

# Logar erro

error\_log = spark.createDataFrame([{

“timestamp”: str(current\_timestamp()),

“pipeline”: “vendas\_etl”,

“error”: str(e),

“status”: “FAILED”

}])

error\_log.write.format(“delta”).mode(“append”).save(“/mnt/datalake/logs/errors/”)

# Re-raise para falhar pipeline

raise

### Dead Letter Queue (DLQ)

**DLQ:** Gravar registros com erro para análise posterior.

**PYTHON**

from pyspark.sql.functions import col, when, lit

def safe\_transformation(df):

“““

Aplicar transformação, separando registros válidos e inválidos.

“““

# Adicionar coluna de erro

df\_with\_error = df.withColumn(

“error\_message”,

when(col(“idade”).isNull(), “idade\_nula”)

.when((col(“idade”) < 0) | (col(“idade”) > 120), “idade\_invalida”)

.otherwise(None)

)

# Separar válidos e inválidos

df\_valid = df\_with\_error.filter(col(“error\_message”).isNull()).drop(“error\_message”)

df\_invalid = df\_with\_error.filter(col(“error\_message”).isNotNull())

return df\_valid, df\_invalid

# Executar

df\_valid, df\_invalid = safe\_transformation(df)

# Gravar válidos em silver

df\_valid.write.format(“delta”).mode(“append”).save(“/mnt/datalake/silver/clientes/”)

# Gravar inválidos em DLQ

if df\_invalid.count() > 0:

df\_invalid.withColumn(“dlq\_timestamp”, current\_timestamp()) \

.write.format(“delta”).mode(“append”).save(“/mnt/datalake/dlq/clientes/”)

print(f” {df\_invalid.count()} registros enviados para DLQ”)

## 9.5 Idempotência

**Idempotência:** Executar pipeline múltiplas vezes produz mesmo resultado.

### Problema: Append Não É Idempotente

**PYTHON**

# Se executar 2x, duplica dados!

df.write.format(“delta”).mode(“append”).save(output\_path)

### Solução 1: Overwrite

**PYTHON**

# Sempre sobrescreve (idempotente)

df.write.format(“delta”).mode(“overwrite”).save(output\_path)

**Problema:** Perde dados históricos.

### Solução 2: Merge com Deduplicação

**PYTHON**

from delta.tables import DeltaTable

# Criar tabela se não existir

if not DeltaTable.isDeltaTable(spark, output\_path):

df.limit(0).write.format(“delta”).save(output\_path)

delta\_table = DeltaTable.forPath(spark, output\_path)

# MERGE (idempotente)

delta\_table.alias(“target”).merge(

df.alias(“source”),

“target.id = source.id AND target.data = source.data” # Chave composta

).whenMatchedUpdateAll() \

.whenNotMatchedInsertAll() \

.execute()

**Executar 2x → Mesmo resultado!**

### Solução 3: Run ID / Execution Tracking

**PYTHON**

import uuid

# Gerar run ID único

run\_id = str(uuid.uuid4())

# Adicionar ao DataFrame

df\_with\_run\_id = df.withColumn(“run\_id”, lit(run\_id))

# Gravar

df\_with\_run\_id.write.format(“delta”).mode(“append”).save(output\_path)

# Salvar run ID em tabela de controle

spark.createDataFrame([{“run\_id”: run\_id, “timestamp”: str(current\_timestamp())}]) \

.write.format(“delta”).mode(“append”).save(“/mnt/datalake/control/runs/”)

# Verificar se run\_id já existe antes de processar (evitar duplicação)

existing\_runs = spark.read.format(“delta”).load(“/mnt/datalake/control/runs/”) \

.filter(f”run\_id = '{run\_id}'“).count()

if existing\_runs > 0:

print(“Run ID já processado, pulando...”)

dbutils.notebook.exit(“SKIPPED”)

## 9.6 Logging e Auditoria

### Framework de Logging

**PYTHON**

from datetime import datetime

def log\_pipeline\_start(pipeline\_name):

“““Logar início de pipeline.”““

log\_entry = spark.createDataFrame([{

“pipeline\_name”: pipeline\_name,

“status”: “STARTED”,

“timestamp”: datetime.now().isoformat(),

“records\_processed”: 0,

“error\_message”: None

}])

log\_entry.write.format(“delta”).mode(“append”).save(“/mnt/datalake/logs/pipeline\_runs/”)

print(f” Pipeline '{pipeline\_name}' iniciado”)

def log\_pipeline\_end(pipeline\_name, records\_processed, status=“SUCCESS”, error=None):

“““Logar fim de pipeline.”““

log\_entry = spark.createDataFrame([{

“pipeline\_name”: pipeline\_name,

“status”: status,

“timestamp”: datetime.now().isoformat(),

“records\_processed”: records\_processed,

“error\_message”: str(error) if error else None

}])

log\_entry.write.format(“delta”).mode(“append”).save(“/mnt/datalake/logs/pipeline\_runs/”)

print(f” Pipeline '{pipeline\_name}' concluído: {status}”)

# Usar em pipeline

pipeline\_name = “vendas\_bronze\_to\_silver”

try:

log\_pipeline\_start(pipeline\_name)

# Processar

df = spark.read.format(“delta”).load(bronze\_path)

df\_transformed = df.transform(my\_transformation)

df\_transformed.write.format(“delta”).mode(“append”).save(silver\_path)

records\_count = df\_transformed.count()

log\_pipeline\_end(pipeline\_name, records\_count, “SUCCESS”)

except Exception as e:

log\_pipeline\_end(pipeline\_name, 0, “FAILED”, e)

raise

### Auditoria de Mudanças (Delta History)

**PYTHON**

# Consultar histórico de operações

delta\_history = spark.sql(“DESCRIBE HISTORY delta.`/mnt/datalake/silver/clientes/`”)

delta\_history.select(

“version”,

“timestamp”,

“operation”,

“operationParameters.mode”,

“operationMetrics.numOutputRows”

).show(20, truncate=False)

## 9.7 Monitoramento e Alertas

### Métricas de Pipeline

**PYTHON**

def collect\_metrics(pipeline\_name, df\_input, df\_output):

“““Coletar métricas de pipeline.”““

metrics = {

“pipeline\_name”: pipeline\_name,

“timestamp”: datetime.now().isoformat(),

“input\_records”: df\_input.count(),

“output\_records”: df\_output.count(),

“filtered\_records”: df\_input.count() - df\_output.count(),

“processing\_time\_sec”: None # Calcular com timer

}

# Gravar métricas

spark.createDataFrame([metrics]) \

.write.format(“delta”).mode(“append”).save(“/mnt/datalake/monitoring/metrics/”)

return metrics

# Exemplo

import time

start\_time = time.time()

df\_input = spark.read.format(“delta”).load(bronze\_path)

df\_output = df\_input.filter(“status = 'active'“)

df\_output.write.format(“delta”).mode(“append”).save(silver\_path)

end\_time = time.time()

metrics = collect\_metrics(“vendas\_etl”, df\_input, df\_output)

metrics[“processing\_time\_sec”] = end\_time - start\_time

print(f”Métricas: {metrics}”)

### Alertas (Integração com ADF)

**No notebook Databricks, retornar status:**

**PYTHON**

# Se erro, retornar status FAILED

if failed\_checks > 0:

dbutils.notebook.exit(“FAILED”)

else:

dbutils.notebook.exit(“SUCCESS”)

**No ADF, verificar output:**

**JSON**

// Pipeline ADF

{

“activities”: [

{

“type”: “DatabricksNotebook”,

“outputs”: [{

“name”: “notebookOutput”

}]

},

{

“type”: “IfCondition”,

“dependsOn”: [“DatabricksNotebook”],

“condition”: “@equals(activity('DatabricksNotebook').output.runOutput, 'FAILED')”,

“ifTrueActivities”: [

{

“type”: “WebActivity”,

“url”: “https://hooks.slack.com/...”, // Webhook Slack/Teams

“body”: {

“text”: “Pipeline vendas\_etl FALHOU!”

}

}

]

}

]

}

## 9.8 Best Practices de ETL

### 1. \*\*Separar Camadas (Bronze/Silver/Gold)\*\*

Bronze (Raw):

- Dados brutos, sem transformação

- Schema original

- Append-only (imutável)

Silver (Cleansed):

- Dados limpos, validados

- Deduplicados

- Schema padronizado

- Delta Lake (ACID)

Gold (Curated):

- Agregações, métricas

- Modelo dimensional (Star Schema)

- Otimizado para consumo (BI, ML)

### 2. \*\*Particionamento Inteligente\*\*

**PYTHON**

# Particionar por data (comum em time-series)

df.write \

.format(“delta”) \

.partitionBy(“year”, “month”, “day”) \

.save(“/mnt/datalake/silver/vendas/”)

# Evitar: Muitas partições pequenas (< 1 GB cada)

# Ideal: 1-10 GB por partição

### 3. \*\*Caching Estratégico\*\*

**PYTHON**

# Cache DataFrames reutilizados

df\_customers = spark.read.format(“delta”).load(customers\_path).cache()

# Usar múltiplas vezes

df\_filtered = df\_customers.filter(“status = 'active'“)

df\_aggregated = df\_customers.groupBy(“city”).count()

# Liberar cache após uso

df\_customers.unpersist()

### 4. \*\*Broadcast Joins\*\*

**PYTHON**

from pyspark.sql.functions import broadcast

# Se df\_small < 100 MB

df\_joined = df\_large.join(broadcast(df\_small), “customer\_id”)

### 5. \*\*Schema Enforcement\*\*

**PYTHON**

from pyspark.sql.types import StructType, StructField, IntegerType, StringType

# Definir schema explicitamente (não inferir)

schema = StructType([

StructField(“id”, IntegerType(), False), # NOT NULL

StructField(“nome”, StringType(), False),

StructField(“idade”, IntegerType(), True) # NULLABLE

])

df = spark.read.schema(schema).csv(input\_path)

### 6. \*\*Optimize e Vacuum Regularmente\*\*

**PYTHON**

# Executar OPTIMIZE semanalmente

spark.sql(“OPTIMIZE delta.`/mnt/datalake/silver/vendas/` ZORDER BY (customer\_id)”)

# VACUUM mensalmente

spark.sql(“VACUUM delta.`/mnt/datalake/silver/vendas/` RETAIN 168 HOURS”)

## 9.9 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar Incremental Load
* [ ] Diferenciar estratégias de deduplicação
* [ ] Entender idempotência

**Práticos:**

* [ ] Implementar Incremental Load (watermark)
* [ ] Remover duplicatas (dropDuplicates, window functions)
* [ ] Criar Data Quality Checks automatizados
* [ ] Implementar Dead Letter Queue (DLQ)
* [ ] Fazer logging de pipelines
* [ ] Coletar métricas de execução
* [ ] Gravar pipelines idempotentes (MERGE)

**Avançado:**

* [ ] Framework de validações reutilizável
* [ ] Integração com alertas (ADF + Webhook)
* [ ] Auditoria com Delta History

## 9.10 Projeto Final: Pipeline Completo

**Objetivo:** Criar pipeline ETL completo Bronze → Silver → Gold.

**Requisitos:**

1. **Bronze (Ingestão):**
   * Ler arquivos CSV de /landing/vendas/
   * Gravar em Delta Lake /bronze/vendas/ (append-only)
   * Logging de início/fim
2. **Silver (Limpeza):**
   * Incremental Load (watermark)
   * Remover duplicatas
   * Data Quality Checks (nulls, ranges, formats)
   * DLQ para registros inválidos
   * Gravar em Delta Lake /silver/vendas/
3. **Gold (Agregação):**
   * Agregação: Total de vendas por cliente, produto, dia
   * Star Schema (Dim + Fact)
   * Gravar em /gold/fato\_vendas/ e /gold/dim\_clientes/
4. **Orquestração ADF:**
   * Pipeline com 3 Notebook Activities (Bronze, Silver, Gold)
   * Schedule Trigger (diário)
   * Alertas em caso de falha
5. **Monitoramento:**
   * Dashboard no Power BI com métricas de pipelines

## 9.11 Recursos Complementares

**Documentação:**

* [Delta Lake Best Practices](https://docs.delta.io/latest/best-practices.html)
* [Databricks ETL Patterns](https://learn.microsoft.com/azure/databricks/delta/etl-patterns)

**Livros:**

* “Designing Data-Intensive Applications” - Martin Kleppmann
* “The Data Warehouse Toolkit” - Ralph Kimball

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Incremental Load:** Watermark, CDC, MERGE

**Deduplication:** dropDuplicates, Window Functions, MERGE

**Data Quality Checks:** Framework de validações automatizadas

**Error Handling:** Try-except, Dead Letter Queue

**Idempotência:** MERGE, Run ID tracking

**Logging/Auditoria:** Pipeline runs, Delta History

**Monitoramento:** Métricas, alertas via ADF

**Best Practices:** Particionamento, caching, schema enforcement, OPTIMIZE/VACUUM

## Parabéns! Você completou a Parte 2!

**Você agora domina:**

* Data Flows (transformações visuais)
* Databricks e PySpark (processamento distribuído)
* Delta Lake (ACID, Time Travel, SCD)
* Streaming (tempo real)
* Padrões de ETL e Data Quality

**Próxima parte (Caps 10-15):** Power BI, Synapse Analytics, Segurança, Monitoramento, Projeto Final!

# Capítulo 10: Power BI e Visualização de Dados

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Conectar Power BI em fontes Azure (SQL Database, Data Lake, Databricks)
* Criar modelos de dados (Star Schema, relacionamentos)
* Desenvolver dashboards interativos
* Aplicar DAX básico (medidas calculadas)
* Implementar Row-Level Security (RLS)
* Publicar e compartilhar relatórios
* Agendar refresh automático de dados
* Otimizar performance de dashboards
* Integrar Power BI com pipelines de dados

**Tempo estimado:** 5-6 horas

**Custo Azure:** Power BI Pro (~R$ 50/mês por usuário)

## 10.1 O que é Power BI?

**Power BI** é a plataforma de Business Intelligence da Microsoft para visualização de dados.

### Componentes Principais

**1. Power BI Desktop (FREE):**

* Aplicativo Windows para criar relatórios
* Conectar fontes de dados
* Modelar dados
* Criar visualizações

**2. Power BI Service (Cloud):**

* Portal web (app.powerbi.com)
* Publicar e compartilhar relatórios
* Agendamento de refresh
* Colaboração

**3. Power BI Mobile:**

* Apps iOS/Android
* Acesso mobile a dashboards

### Power BI vs Outras Ferramentas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Característica | Power BI | Tableau | Looker | Excel |
| **Curva aprendizado** | Baixa | Média | Média | Muito baixa |
| **Integração Azure** | Excelente | Boa | Boa | Limitada |
| **Custo** | Baixo-Médio | Alto | Alto | Muito baixo |
| **Performance** | Boa | Excelente | Boa | Limitada |
| **Modelagem dados** | Excelente | Boa | Média | Limitada |

## 10.2 Instalar Power BI Desktop

### Download

1. **Acesse:** https://powerbi.microsoft.com/downloads/
2. **Clique em “Download gratuito”** (Power BI Desktop)
3. **Instale** (executável ~800 MB)
4. **Abra Power BI Desktop**



**Nome do arquivo:** Junior\_Cap10\_01\_powerbi\_desktop\_home.png

*Figura 10.1 - Power BI Desktop (tela inicial)*

## 10.3 Conectar em Azure SQL Database

### Passo 1: Get Data

1. **Power BI Desktop → Home → Get data**
2. **Selecione “Azure” → “Azure SQL Database”**
3. **Clique “Connect”**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_02\_get\_data\_azure\_sql.png

*Figura 10.2 - Conectar em Azure SQL Database*

### Passo 2: Configurar Conexão

**Server:** sql-azuredejunior-<seunome>.database.windows.net

**Database:** sqldb-adventureworks

**Data Connectivity mode:**

* **Import** (recomendado): Importa dados para Power BI
* **DirectQuery**: Query direto no banco (dados sempre atualizados, performance menor)

**Clique “OK”**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_03\_sql\_connection\_config.png

*Figura 10.3 - Configuração de conexão SQL*

### Passo 3: Autenticação

**Database:**

* **User name:** sqladmin
* **Password:** <sua\_senha>

**Clique “Connect”**

### Passo 4: Selecionar Tabelas

**Navigator mostra tabelas disponíveis:**

Selecione:

* SalesLT.Customer
* SalesLT.Product
* SalesLT.SalesOrderHeader
* SalesLT.SalesOrderDetail

**Clique “Load”** (ou “Transform Data” para limpeza)

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_04\_navigator\_select\_tables.png

*Figura 10.4 - Seleção de tabelas no Navigator*

## 10.4 Modelagem de Dados (Star Schema)

### Relacionamentos

**Power BI detecta relacionamentos automaticamente (se houver chaves FK).**

**Verificar relacionamentos:**

1. **Clique em “Model” (ícone à esquerda)**
2. **Visualização do modelo de dados:**
   * Tabelas como boxes
   * Linhas conectando tabelas (relacionamentos)

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_05\_data\_model\_relationships.png

*Figura 10.5 - Modelo de dados (Star Schema)*

### Criar Relacionamento Manual

**Se Power BI não detectou:**

1. **Arraste coluna de uma tabela para outra**
   * Ex: SalesOrderDetail[ProductID] → Product[ProductID]
2. **Ou: Model → Manage relationships → New**
3. **Configurar:**
   * Table 1: SalesOrderDetail
   * Column: ProductID
   * Table 2: Product
   * Column: ProductID
   * **Cardinality:** Many to One (:1)
   * **Cross filter direction:** Single

**Clique “OK”**

### Tipos de Relacionamentos

**Cardinality:**

* **One to Many (1:\*)** - Mais comum (ex: Customer 1 → Pedidos )
* **Many to Many (\*:\*)** - Evitar (performance ruim)
* **One to One (1:1)** - Raro

**Cross filter direction:**

* **Single** - Filtro flui em uma direção
* **Both** - Filtro flui nas duas direções (cuidado com ambiguidade)

## 10.5 Power Query (Transformações)

**Power Query Editor** permite limpar/transformar dados antes de carregar.

### Abrir Power Query Editor

**Home → Transform data**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_06\_power\_query\_editor.png

*Figura 10.6 - Power Query Editor*

### Transformações Comuns

**1. Remover colunas desnecessárias:**

* Selecionar colunas → Right-click → Remove

**2. Filtrar linhas:**

* Clicar seta no header → Filter (ex: Status = “Active”)

**3. Alterar tipo de dados:**

* Clicar ícone de tipo (ex: 123 → ABC) → Alterar para Text/Number/Date

**4. Adicionar coluna calculada:**

* Add Column → Custom Column
* Exemplo: [ListPrice] \* 1.1 (preço com margem)

**5. Merge queries (Join):**

* Home → Merge Queries
* Selecionar tabelas e colunas de join

**6. Append queries (Union):**

* Home → Append Queries

### Aplicar e Carregar

**Close & Apply** (canto superior esquerdo)

## 10.6 Criar Visualizações

### Passo 1: Criar Página de Relatório

**Report view (ícone à esquerda) → Página em branco**

### Passo 2: Card (Métrica Simples)

**Visualizações → Card**

**Arrastar campo:**

* SalesOrderHeader[TotalDue] → Fields

**Power BI soma automaticamente!**

**Renomear:**

* Format → Title → “Total de Vendas”

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_07\_card\_total\_vendas.png

*Figura 10.7 - Card com total de vendas*

### Passo 3: Bar Chart (Gráfico de Barras)

**Visualizações → Clustered bar chart**

**Configurar:**

* **Y-axis:** Product[ProductCategory]
* **X-axis:** SalesOrderDetail[LineTotal] (soma automática)

**Resultado:** Vendas por categoria de produto.

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_08\_bar\_chart\_category.png

*Figura 10.8 - Vendas por categoria*

### Passo 4: Line Chart (Evolução Temporal)

**Visualizações → Line chart**

**Configurar:**

* **X-axis:** SalesOrderHeader[OrderDate] (hierarquia: Year → Quarter → Month)
* **Y-axis:** SalesOrderHeader[TotalDue]

**Resultado:** Evolução de vendas ao longo do tempo.

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_09\_line\_chart\_time.png

*Figura 10.9 - Evolução de vendas (temporal)*

### Passo 5: Table (Tabela Detalhada)

**Visualizações → Table**

**Arrastar campos:**

* Customer[FirstName]
* Customer[LastName]
* SalesOrderHeader[TotalDue]

**Ordenar:** Clicar em header (ex: TotalDue DESC)

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_10\_table\_customers.png

*Figura 10.10 - Tabela de clientes*

### Passo 6: Slicer (Filtro Interativo)

**Visualizações → Slicer**

**Arrastar campo:**

* SalesOrderHeader[OrderDate]

**Tipo de slicer:** Between (range de datas)

**Usuário pode filtrar dashboard por período!**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_11\_slicer\_date.png

*Figura 10.11 - Slicer de data (filtro interativo)*

## 10.7 DAX (Data Analysis Expressions)

**DAX** é linguagem de fórmulas para criar medidas calculadas.

### Criar Medida Calculada

**Exemplo: Margem de Lucro**

1. **Home → New measure** (ou clicar direito em tabela → New measure)
2. **Escrever DAX:**

**DAX**

Margem Bruta =

SUM(SalesOrderDetail[LineTotal]) - SUM(Product[StandardCost] \* SalesOrderDetail[OrderQty])

1. **Formatar:** Home → Format → Currency → R$
2. **Usar em Card ou gráfico**

### DAX Básico - Funções Comuns

**Agregações:**

* SUM(coluna)
* AVERAGE(coluna)
* COUNT(coluna)
* DISTINCTCOUNT(coluna) - Contar únicos
* MIN(coluna), MAX(coluna)

**Lógica:**

* IF(condição, valor\_se\_verdadeiro, valor\_se\_falso)
* SWITCH(expressão, valor1, resultado1, valor2, resultado2, ...)

**Contexto:**

* CALCULATE(medida, filtro1, filtro2, ...) - Modificar contexto
* FILTER(tabela, condição) - Filtrar tabela
* ALL(tabela) - Remover filtros

### Exemplo: Total de Vendas do Ano Anterior

**DAX**

Vendas Ano Anterior =

CALCULATE(

SUM(SalesOrderHeader[TotalDue]),

SAMEPERIODLASTYEAR(SalesOrderHeader[OrderDate])

)

### Exemplo: % de Crescimento

**DAX**

% Crescimento =

DIVIDE(

[Total Vendas] - [Vendas Ano Anterior],

[Vendas Ano Anterior],

0

)

**Formatar como Percentage**

## 10.8 Row-Level Security (RLS)

**RLS** restringe acesso a dados baseado no usuário.

### Cenário

**Gerentes de região só veem vendas de sua região.**

### Passo 1: Criar Role

1. **Modeling → Manage roles**
2. **Create role:** “Gerente Região Sul”
3. **Selecionar tabela:** SalesOrderHeader
4. **DAX Filter:**

**DAX**

[TerritoryName] = “Sul”

1. **Save**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_12\_rls\_role\_creation.png

*Figura 10.12 - Criação de Role (RLS)*

### Passo 2: Testar RLS

**Modeling → View as Roles → Selecionar “Gerente Região Sul”**

**Dashboard agora mostra apenas dados da região Sul!**

### Passo 3: Atribuir Usuários (no Power BI Service)

**Após publicar:**

1. **Power BI Service → Workspace → Dataset → Security**
2. **Adicionar usuários ao role**

## 10.9 Publicar Relatório

### Passo 1: Salvar Arquivo

**File → Save → `Relatorio\_Vendas.pbix`**

### Passo 2: Publicar no Power BI Service

1. **Home → Publish**
2. **Login com conta Microsoft** (precisa licença Pro ou Premium)
3. **Selecionar Workspace** (ex: My workspace)
4. **Clique “Select”**
5. **Aguardar upload**
6. **“Success! Open 'Relatorio\_Vendas' in Power BI”**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_13\_publish\_success.png

*Figura 10.13 - Publicação bem-sucedida*

### Passo 3: Visualizar no Power BI Service

**Clicar no link → Abre app.powerbi.com**

**Relatório disponível no workspace!**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_14\_powerbi\_service\_report.png

*Figura 10.14 - Relatório no Power BI Service*

## 10.10 Agendar Refresh de Dados

**Dados no Power BI ficam desatualizados após publicação.**

### Configurar Refresh Automático

**Pré-requisito:** Gateway instalado (se fonte on-premises) ou credenciais armazenadas.

### Passo 1: Configurar Credenciais

1. **Power BI Service → Workspace → Dataset (não o relatório)**
2. **Settings → Data source credentials**
3. **Edit credentials → Inserir usuário/senha do SQL Database**
4. **Sign in**

### Passo 2: Configurar Schedule

1. **Settings → Scheduled refresh**
2. **Keep your data up to date:** ON
3. **Refresh frequency:** Daily
4. **Time:** 08:00
5. **Time zone:** (UTC-03:00) Brasília
6. **Apply**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_15\_schedule\_refresh.png

*Figura 10.15 - Agendamento de refresh automático*

### Refresh Manual

**Dataset → Refresh now** (ícone de refresh)

## 10.11 Conectar em Data Lake (Delta Lake)

**Power BI pode ler Delta Lake via Databricks SQL Endpoint.**

### Passo 1: Habilitar SQL Endpoint no Databricks

1. **Databricks Workspace → SQL → SQL Warehouses**
2. **Create SQL Warehouse** (ou usar existente)
3. **Copiar Server hostname e HTTP path**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap10\_16\_databricks\_sql\_endpoint.png

*Figura 10.16 - Databricks SQL Endpoint*

### Passo 2: Conectar no Power BI

1. **Power BI Desktop → Get data → More → Azure Databricks**
2. **Server hostname:** adb-123456789.azuredatabricks.net
3. **HTTP path:** /sql/1.0/warehouses/abc123
4. **Data Connectivity mode:** Import
5. **Autenticação:** Personal Access Token (gerar no Databricks)
6. **Selecionar tabelas Delta**
7. **Load**

## 10.12 Otimização de Performance

### 1. Usar Import ao Invés de DirectQuery

**Import:**

* Dados em memória (rápido)
* Precisa refresh

**DirectQuery:**

* Dados sempre atualizados
* Lento (query no banco a cada interação)

**Recomendação:** Import + Refresh agendado

### 2. Reduzir Colunas/Linhas

**Power Query:**

* Remover colunas não usadas
* Filtrar linhas (ex: últimos 2 anos apenas)

**Menos dados = melhor performance**

### 3. Otimizar Relacionamentos

* Evitar Many-to-Many
* Usar Integer keys (não String)
* Minimizar relacionamentos bidirecionais

### 4. Agregações

**Para datasets grandes (> 1 GB):**

**Criar tabela agregada (ex: vendas mensais) e usar como fonte principal.**

## 10.13 Integração com Pipelines de Dados

### Fluxo Completo

1. Azure Data Factory:

- Ingest dados (bronze)

- Transform (silver/gold)

2. Databricks:

- Processar Big Data

- Gravar Delta Lake (gold layer)

3. Power BI:

- Conectar em Databricks SQL Endpoint

- Ler tabelas gold (agregadas)

- Dashboards para negócio

4. Refresh automático:

- ADF roda diariamente → atualiza gold

- Power BI refresh lê novos dados

## 10.14 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar diferença Import vs DirectQuery
* [ ] Entender Star Schema
* [ ] Diferenciar medidas vs colunas calculadas

**Práticos:**

* [ ] Conectar Power BI em Azure SQL Database
* [ ] Conectar Power BI em Databricks (Delta Lake)
* [ ] Criar relacionamentos entre tabelas
* [ ] Transformar dados com Power Query
* [ ] Criar visualizações (Card, Bar Chart, Line Chart, Table, Slicer)
* [ ] Escrever medidas DAX básicas
* [ ] Implementar Row-Level Security (RLS)
* [ ] Publicar relatório no Power BI Service
* [ ] Agendar refresh automático

## 10.15 Recursos Complementares

**Documentação Oficial:**

* [Power BI Documentation](https://learn.microsoft.com/power-bi/)
* [DAX Reference](https://dax.guide/)
* [Power BI Best Practices](https://learn.microsoft.com/power-bi/guidance/power-bi-optimization)

**Tutoriais:**

* [Power BI Guided Learning](https://learn.microsoft.com/training/powerplatform/power-bi)
* [DAX Patterns](https://www.daxpatterns.com/)

**Community:**

* [Power BI Community](https://community.powerbi.com/)

## 10.16 Exercícios Práticos

**Exercício 1: Dashboard de Vendas**

Criar dashboard com:

1. Card: Total de Vendas, Total de Pedidos
2. Bar Chart: Vendas por categoria de produto
3. Line Chart: Evolução mensal de vendas
4. Table: Top 10 clientes
5. Slicer: Filtro por data

**Exercício 2: DAX Avançado**

Criar medidas:

1. Margem Bruta (Vendas - Custo)
2. % Margem (Margem / Vendas)
3. Vendas Acumuladas no Ano (YTD)
4. Ticket Médio (Vendas / Pedidos)

**Exercício 3: RLS**

Implementar RLS:

1. Criar role “Vendedor” que vê apenas seus clientes
2. Testar com View as Roles
3. Publicar e atribuir usuários

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Power BI Desktop:** Ferramenta FREE para criar relatórios

**Conexões Azure:**

* SQL Database
* Databricks (Delta Lake)
* Data Lake

**Modelagem:** Star Schema, relacionamentos

**Power Query:** Transformações (filter, remove columns, merge)

**Visualizações:** Card, Bar/Line Chart, Table, Slicer

**DAX:** Medidas calculadas (SUM, CALCULATE, time intelligence)

**RLS:** Row-Level Security para controle de acesso

**Publicação:** Power BI Service, refresh agendado

**Integração:** Pipelines de dados → Power BI

**Próximo capítulo:** Azure Synapse Analytics - Data Warehouse!

# Capítulo 11: Azure Synapse Analytics - Fundamentos

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender o que é Azure Synapse Analytics
* Diferenciar Synapse SQL Pool vs Serverless
* Provisionar Synapse Workspace
* Criar e consultar tabelas (SQL Dedicated Pool)
* Usar Serverless SQL para query em Data Lake
* Integrar Synapse com Data Lake e Delta Lake
* Criar pipelines de integração no Synapse
* Otimizar queries (distribuição, particionamento)
* Implementar Star Schema em Synapse
* Conectar Power BI em Synapse

**Tempo estimado:** 5-6 horas

**Custo Azure:** R$ 300-800/mês (Dedicated Pool DW100c pausado ~R$ 0)

## 11.1 O que é Azure Synapse Analytics?

**Azure Synapse** é plataforma de analytics unificada que combina:

* **Data Warehousing** (SQL Dedicated Pool)
* **Big Data** (Spark Pools)
* **Data Integration** (Pipelines - similar ADF)
* **Visualization** (Power BI integrado)

**Em uma única interface!**

### Synapse vs Outras Ferramentas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Característica | Synapse | Databricks | Snowflake | SQL Database |
| **Data Warehouse** | Sim (Dedicated Pool) | Não | Sim | Limitado |
| **Big Data (Spark)** | Sim (Spark Pools) | Sim | Não | Não |
| **Serverless SQL** | Sim | Não | Sim | Não |
| **Pipelines ETL** | Sim | Não | Limitado | Não |
| **Melhor para** | DW + Big Data | Big Data + ML | DW Cloud-native | OLTP |

### Componentes do Synapse

**1. SQL Pools:**

* **Dedicated Pool (antes SQL DW):** Data Warehouse provisionado
* **Serverless Pool:** Query on-demand em Data Lake (sem provisionar)

**2. Spark Pools:**

* Apache Spark gerenciado (similar Databricks)

**3. Pipelines:**

* ETL/ELT (mesma engine do ADF)

**4. Synapse Studio:**

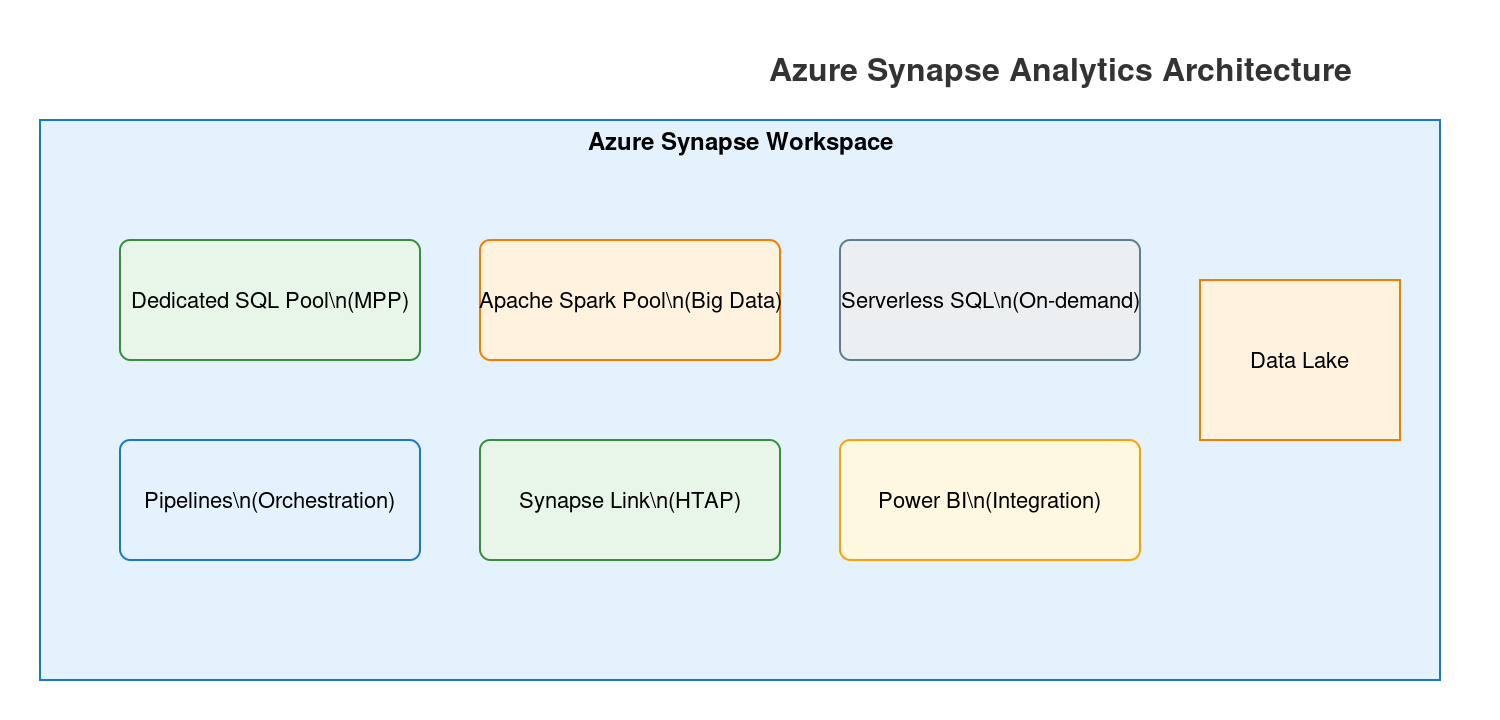
* Interface web unificada

## 11.2 Provisionar Synapse Workspace

### Passo 1: Criar Workspace

**Azure Portal:**

1. **Barra de pesquisa:** synapse
2. **Azure Synapse Analytics → + Create**



**Nome do arquivo:** Junior\_Cap11\_01\_synapse\_create.png

*Figura 11.1 - Criar Synapse Workspace*

1. **Aba “Basics”:**
   * **Subscription:** Free Trial
   * **Resource group:** rg-azure-de-junior
   * **Workspace name:** synapse-azuredejunior-<seunome>
   * **Region:** Brazil South
   * **Data Lake Storage Gen2:**
     + **Account name:** Selecione storage account existente
     + **File system name:** synapse (novo container)
2. **Aba “Security”:**
   * **SQL admin login:** sqladmin
   * **Password:** <senha\_forte>
3. **Review + create → Create**
4. **Aguardar deployment (5-7 minutos)**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap11\_02\_synapse\_workspace\_created.png

*Figura 11.2 - Synapse Workspace criado*

### Passo 2: Abrir Synapse Studio

**Clique em “Open Synapse Studio”**

**Nova aba abre:** Synapse Studio (interface unificada)

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap11\_03\_synapse\_studio\_home.png

*Figura 11.3 - Synapse Studio (home)*

## 11.3 Serverless SQL Pool (Query em Data Lake)

**Serverless SQL Pool** permite fazer queries SQL em arquivos no Data Lake **sem provisionar infraestrutura.**

### Vantagens

* **Pay-per-query** (cobra por TB processado, não por hora)
* **Sem gerenciamento** (serverless)
* **Acesso direto** a Parquet, CSV, JSON, Delta Lake

### Use Cases

* Exploração de dados (ad-hoc queries)
* Criar views sobre Data Lake
* ETL leve (transformações SQL)

### Passo 1: Query em Arquivo Parquet

**Synapse Studio:**

1. **Develop → + SQL script**
2. **Connect to:** Built-in (Serverless Pool)
3. **Database:** master (ou criar database)

**Query:**

**SQL**

-- Query em arquivo Parquet no Data Lake

SELECT TOP 100 \*

FROM OPENROWSET(

BULK 'https://stadejuniorronaldo.dfs.core.windows.net/datalake/bronze/clientes/\*.parquet',

FORMAT = 'PARQUET'

) AS clientes

1. **Run**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap11\_04\_serverless\_query\_parquet.png

*Figura 11.4 - Serverless SQL query em Parquet*

### Passo 2: Criar External Table

**Facilita consultas (não precisa escrever OPENROWSET toda vez).**

**SQL**

-- 1. Criar Master Key (uma vez por database)

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'senha\_forte\_123!';

-- 2. Criar Database Scoped Credential (se Data Lake privado)

CREATE DATABASE SCOPED CREDENTIAL DataLakeCredential

WITH IDENTITY = 'SHARED ACCESS SIGNATURE',

SECRET = '<SAS\_TOKEN>';

-- 3. Criar External Data Source

CREATE EXTERNAL DATA SOURCE DataLakeBronze

WITH (

LOCATION = 'https://stadejuniorronaldo.dfs.core.windows.net/datalake/bronze',

CREDENTIAL = DataLakeCredential

);

-- 4. Criar External File Format

CREATE EXTERNAL FILE FORMAT ParquetFormat

WITH (

FORMAT\_TYPE = PARQUET,

DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'

);

-- 5. Criar External Table

CREATE EXTERNAL TABLE clientes\_external (

CustomerID INT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

EmailAddress VARCHAR(100)

)

WITH (

LOCATION = 'clientes/\*.parquet',

DATA\_SOURCE = DataLakeBronze,

FILE\_FORMAT = ParquetFormat

);

-- 6. Query simples!

SELECT \* FROM clientes\_external;

### Passo 3: Query em Delta Lake

**Synapse Serverless suporta Delta Lake nativamente!**

**SQL**

SELECT TOP 100 \*

FROM OPENROWSET(

BULK 'https://stadejuniorronaldo.dfs.core.windows.net/datalake/silver/clientes\_delta/',

FORMAT = 'DELTA'

) AS clientes\_delta

**Synapse lê transaction log automaticamente!**

## 11.4 Dedicated SQL Pool (Data Warehouse)

**Dedicated Pool** é Data Warehouse tradicional (provisionado).

### Vantagens

* Performance previsível (recursos dedicados)
* Índices, estatísticas, otimizações
* Workload Management (priorização de queries)

### Quando Usar

* Data Warehouse corporativo
* Queries complexas, high concurrency
* Dados estruturados (Star Schema)

### Passo 1: Criar Dedicated SQL Pool

**Synapse Studio:**

1. **Manage → SQL pools → + New**
2. **Configurar:**
   * **Pool name:** sqldw\_junior
   * **Performance level:** DW100c (mínimo, ~R$ 10/hora)
     + Escala: DW100c → DW30000c
   * **Collation:** Default
3. **Create**
4. **Aguardar provisionamento (5-10 minutos)**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap11\_05\_dedicated\_pool\_created.png

*Figura 11.5 - Dedicated SQL Pool criado*

### Passo 2: Pausar/Resumir Pool

**Importante:** Dedicated Pool cobra por hora, mesmo se não estiver usando!

**Pausar quando não estiver usando:**

**Manage → SQL pools → sqldw\_junior → Pause**

**Resumir para usar:**

**Resume**

**Economia:** Pool pausado não cobra compute (só storage)!

### Passo 3: Criar Tabela

**Synapse Studio → Develop → + SQL script**

**Connect to:** sqldw\_junior (Dedicated Pool)

**Criar Star Schema:**

**SQL**

-- Dimension: Customers

CREATE TABLE dim\_customer (

customer\_key INT IDENTITY(1,1) PRIMARY KEY NONCLUSTERED NOT ENFORCED,

customer\_id INT NOT NULL,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100),

city VARCHAR(50),

country VARCHAR(50)

)

WITH (

DISTRIBUTION = REPLICATE, -- Replica em todos nós (small dimension)

CLUSTERED COLUMNSTORE INDEX

);

-- Dimension: Products

CREATE TABLE dim\_product (

product\_key INT IDENTITY(1,1) PRIMARY KEY NONCLUSTERED NOT ENFORCED,

product\_id INT NOT NULL,

product\_name VARCHAR(100),

category VARCHAR(50),

list\_price DECIMAL(10,2)

)

WITH (

DISTRIBUTION = REPLICATE,

CLUSTERED COLUMNSTORE INDEX

);

-- Fact: Sales

CREATE TABLE fato\_vendas (

venda\_key INT IDENTITY(1,1),

customer\_key INT,

product\_key INT,

order\_date DATE,

quantity INT,

unit\_price DECIMAL(10,2),

total\_amount DECIMAL(10,2)

)

WITH (

DISTRIBUTION = HASH(customer\_key), -- Distribui por hash de customer\_key

CLUSTERED COLUMNSTORE INDEX,

PARTITION (order\_date RANGE RIGHT FOR VALUES (

'2023-01-01', '2024-01-01', '2025-01-01'

))

);

**Execute!**

### Entender Distribuições

**DISTRIBUTION:** Como dados são distribuídos entre nós (nodes).

**Opções:**

**1. HASH (distribuição por hash):**

* Melhor para **fact tables grandes**
* Escolher coluna de join frequente
* Exemplo: HASH(customer\_key)

**2. REPLICATE (replicar em todos nós):**

* Melhor para **dimension tables pequenas (< 2 GB)**
* Elimina data movement em joins
* Exemplo: dimensões

**3. ROUND\_ROBIN (distribuição aleatória):**

* Default (não recomendado)
* Usa quando não há coluna boa para hash

## 11.5 Carregar Dados no Dedicated Pool

### Método 1: COPY Statement (Recomendado)

**COPY** é método mais rápido e flexível.

**SQL**

-- Carregar de Data Lake para dim\_customer

COPY INTO dim\_customer (customer\_id, first\_name, last\_name, email, city, country)

FROM 'https://stadejuniorronaldo.dfs.core.windows.net/datalake/silver/clientes/\*.parquet'

WITH (

FILE\_TYPE = 'PARQUET',

CREDENTIAL = (IDENTITY = 'SHARED ACCESS SIGNATURE', SECRET = '<SAS\_TOKEN>')

);

### Método 2: Synapse Pipeline (Orquestração)

**Similar ADF:**

1. **Integrate → + Pipeline**
2. **Adicionar “Copy data” activity**
3. **Source:** Data Lake (Parquet)
4. **Sink:** Dedicated Pool (dim\_customer)
5. **Run pipeline**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap11\_06\_synapse\_pipeline\_copy.png

*Figura 11.6 - Pipeline Synapse (carga de dados)*

### Método 3: PolyBase (Legado)

**PolyBase** é predecessor do COPY (ainda suportado).

**SQL**

-- 1. Criar External Data Source

CREATE EXTERNAL DATA SOURCE DataLakeSource

WITH (

TYPE = HADOOP,

LOCATION = 'wasbs://datalake@stadejuniorronaldo.blob.core.windows.net'

);

-- 2. Criar External File Format

CREATE EXTERNAL FILE FORMAT ParquetFormat

WITH (FORMAT\_TYPE = PARQUET);

-- 3. Criar External Table

CREATE EXTERNAL TABLE ext\_clientes (...)

WITH (

LOCATION = 'silver/clientes/',

DATA\_SOURCE = DataLakeSource,

FILE\_FORMAT = ParquetFormat

);

-- 4. INSERT INTO (carrega dados)

INSERT INTO dim\_customer

SELECT \* FROM ext\_clientes;

**Recomendação:** Use COPY (mais simples e rápido).

## 11.6 Otimizações

### 1. Columnstore Index

**Synapse usa Columnstore por padrão (compressão + performance).**

**Verificar:**

**SQL**

SELECT object\_name(object\_id) AS table\_name,

name AS index\_name,

type\_desc

FROM sys.indexes

WHERE object\_name(object\_id) = 'fato\_vendas';

### 2. Estatísticas

**Estatísticas ajudam otimizador de queries.**

**Criar estatísticas:**

**SQL**

CREATE STATISTICS stat\_customer\_key ON fato\_vendas(customer\_key);

CREATE STATISTICS stat\_order\_date ON fato\_vendas(order\_date);

**Update automático:**

**SQL**

UPDATE STATISTICS fato\_vendas;

### 3. Particionamento

**Tabela fato\_vendas já tem partition por order\_date.**

**Benefícios:**

* Elimina partições não necessárias (partition elimination)
* Manutenção mais rápida (SWITCH partitions)

**Verificar partitions:**

**SQL**

SELECT SCHEMA\_NAME(t.schema\_id) AS schema\_name,

t.name AS table\_name,

p.partition\_number,

p.rows

FROM sys.tables t

JOIN sys.partitions p ON t.object\_id = p.object\_id

WHERE t.name = 'fato\_vendas'

ORDER BY p.partition\_number;

### 4. Workload Management

**Classificar queries em grupos (importância).**

**Exemplo:**

**SQL**

-- Criar Workload Classifier

CREATE WORKLOAD CLASSIFIER HighPriorityQueries

WITH (

WORKLOAD\_GROUP = 'xlargerc', -- Mais recursos

MEMBERNAME = 'user@domain.com',

IMPORTANCE = HIGH

);

## 11.7 Queries Analíticas

### Exemplo: Total de Vendas por Produto

**SQL**

SELECT

p.product\_name,

p.category,

SUM(f.total\_amount) AS total\_vendas,

SUM(f.quantity) AS total\_quantidade

FROM fato\_vendas f

INNER JOIN dim\_product p ON f.product\_key = p.product\_key

GROUP BY p.product\_name, p.category

ORDER BY total\_vendas DESC;

### Exemplo: Vendas Mensais (Evolução)

**SQL**

SELECT

YEAR(order\_date) AS ano,

MONTH(order\_date) AS mes,

SUM(total\_amount) AS total\_vendas

FROM fato\_vendas

GROUP BY YEAR(order\_date), MONTH(order\_date)

ORDER BY ano, mes;

### Exemplo: Top 10 Clientes

**SQL**

SELECT TOP 10

c.first\_name + ' ' + c.last\_name AS cliente,

SUM(f.total\_amount) AS total\_gasto

FROM fato\_vendas f

INNER JOIN dim\_customer c ON f.customer\_key = c.customer\_key

GROUP BY c.first\_name, c.last\_name

ORDER BY total\_gasto DESC;

## 11.8 Integração com Power BI

### Conectar Power BI em Synapse

**Power BI Desktop:**

1. **Get data → Azure → Azure Synapse Analytics (SQL DW)**
2. **Server:** synapse-azuredejunior-<nome>.sql.azuresynapse.net
3. **Database:** sqldw\_junior
4. **Data Connectivity mode:** Import (ou DirectQuery)
5. **Autenticar**
6. **Selecionar tabelas:** dim\_customer, dim\_product, fato\_vendas
7. **Load**

**Criar dashboard normalmente!**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap11\_07\_powerbi\_synapse\_connection.png

*Figura 11.7 - Power BI conectado em Synapse*

## 11.9 Serverless vs Dedicated - Quando Usar?

|  |  |  |
| --- | --- | --- |
| Cenário | Serverless SQL | Dedicated SQL Pool |
| **Exploração de dados** | Ideal | Overkill |
| **Ad-hoc queries** | Pay-per-query | Paga mesmo sem usar |
| **Data Warehouse corporativo** | Performance variável | Ideal |
| **Queries complexas recorrentes** | OK (pode ser lento) | Otimizado |
| **Custo** | Baixo (< R$ 50/mês) | Alto (R$ 300-800/mês) |
| **Setup** | Zero | Provisionar + tuning |

**Regra geral:**

* **Serverless:** Exploração, ETL leve, dados em Data Lake
* **Dedicated:** DW corporativo, high performance, workloads previsíveis

## 11.10 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar Synapse Analytics (componentes)
* [ ] Diferenciar Serverless vs Dedicated Pool
* [ ] Entender distribuições (HASH, REPLICATE, ROUND\_ROBIN)

**Práticos:**

* [ ] Provisionar Synapse Workspace
* [ ] Fazer query em Data Lake com Serverless SQL (OPENROWSET)
* [ ] Criar External Table
* [ ] Query em Delta Lake
* [ ] Criar Dedicated SQL Pool
* [ ] Criar Star Schema (dimensions + fact)
* [ ] Carregar dados (COPY statement)
* [ ] Otimizar queries (statisticas, partitions)
* [ ] Conectar Power BI em Synapse

## 11.11 Recursos Complementares

**Documentação Oficial:**

* [Azure Synapse Analytics](https://learn.microsoft.com/azure/synapse-analytics/)
* [Serverless SQL Pool](https://learn.microsoft.com/azure/synapse-analytics/sql/on-demand-workspace-overview)
* [Dedicated SQL Pool Best Practices](https://learn.microsoft.com/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-best-practices)

**Tutoriais:**

* [Synapse Quickstart](https://learn.microsoft.com/azure/synapse-analytics/quickstart-create-workspace)

## 11.12 Exercícios Práticos

**Exercício 1: Serverless SQL**

1. Criar database no Serverless Pool
2. Criar External Table apontando para Data Lake (Parquet)
3. Query top 100 registros
4. Criar view materializada

**Exercício 2: Dedicated Pool Star Schema**

1. Criar Dedicated Pool (DW100c)
2. Criar Star Schema:
   * dim\_cliente
   * dim\_produto
   * dim\_data (calendar table)
   * fato\_vendas
3. Carregar dados com COPY
4. Criar query agregada (vendas por mês/categoria)

**Exercício 3: Integração completa**

1. Pipeline Synapse:
   * Source: Data Lake (bronze)
   * Transform: Limpeza (Serverless SQL ou Spark)
   * Sink: Dedicated Pool (gold)
2. Agendar pipeline (diário)
3. Conectar Power BI e criar dashboard

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Synapse Analytics:** Plataforma unificada (DW + Big Data + Pipelines)

**Serverless SQL Pool:**

* Query em Data Lake (Parquet, Delta Lake)
* Pay-per-query (R$ por TB processado)
* External Tables

**Dedicated SQL Pool:**

* Data Warehouse provisionado (MPP)
* Star Schema (dimensions + facts)
* Distribuições (HASH, REPLICATE)
* Otimizações (Columnstore, Statistics, Partitions)

**Carregamento:** COPY statement, Pipelines

**Integração:** Power BI, Pipelines (similar ADF)

**Quando usar:** Serverless (exploração) vs Dedicated (DW corporativo)

**Próximo capítulo:** Segurança e Governança de Dados!

# Capítulo 12: Segurança e Governança de Dados

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Implementar Azure Active Directory (Azure AD) para autenticação
* Configurar Role-Based Access Control (RBAC)
* Usar Azure Key Vault para gerenciar secrets
* Implementar Private Endpoints (rede privada)
* Configurar Data Lake ACLs (Access Control Lists)
* Habilitar auditing e logging
* Implementar data masking e encriptação
* Aplicar Microsoft Purview para data governance
* Implementar data lineage e data catalog
* Criar políticas de retenção e compliance

**Tempo estimado:** 4-5 horas

**Custo Azure:** Key Vault (R$ 5/mês), Purview (R$ 500+/mês - opcional)

## 12.1 Fundamentos de Segurança Azure

### Camadas de Segurança

**1. Identity & Access (Quem pode acessar?)**

* Azure Active Directory (Azure AD)
* RBAC (Role-Based Access Control)
* Managed Identities

**2. Network Security (Como acessar?)**

* Private Endpoints
* Virtual Networks (VNets)
* Firewall Rules

**3. Data Protection (Proteger dados)**

* Encryption at rest / in transit
* Azure Key Vault (secrets)
* Data Masking

**4. Monitoring & Compliance (Auditar)**

* Azure Monitor
* Log Analytics
* Microsoft Purview

## 12.2 Azure Active Directory (Azure AD)

**Azure AD** é serviço de identidade e acesso da Microsoft.

### Conceitos Principais

**Users:**

* Usuários individuais (ex: joao@empresa.com)

**Groups:**

* Conjunto de usuários (ex: “Equipe Data Engineering”)

**Service Principals:**

* Identidade para aplicações (ex: ADF, Databricks)

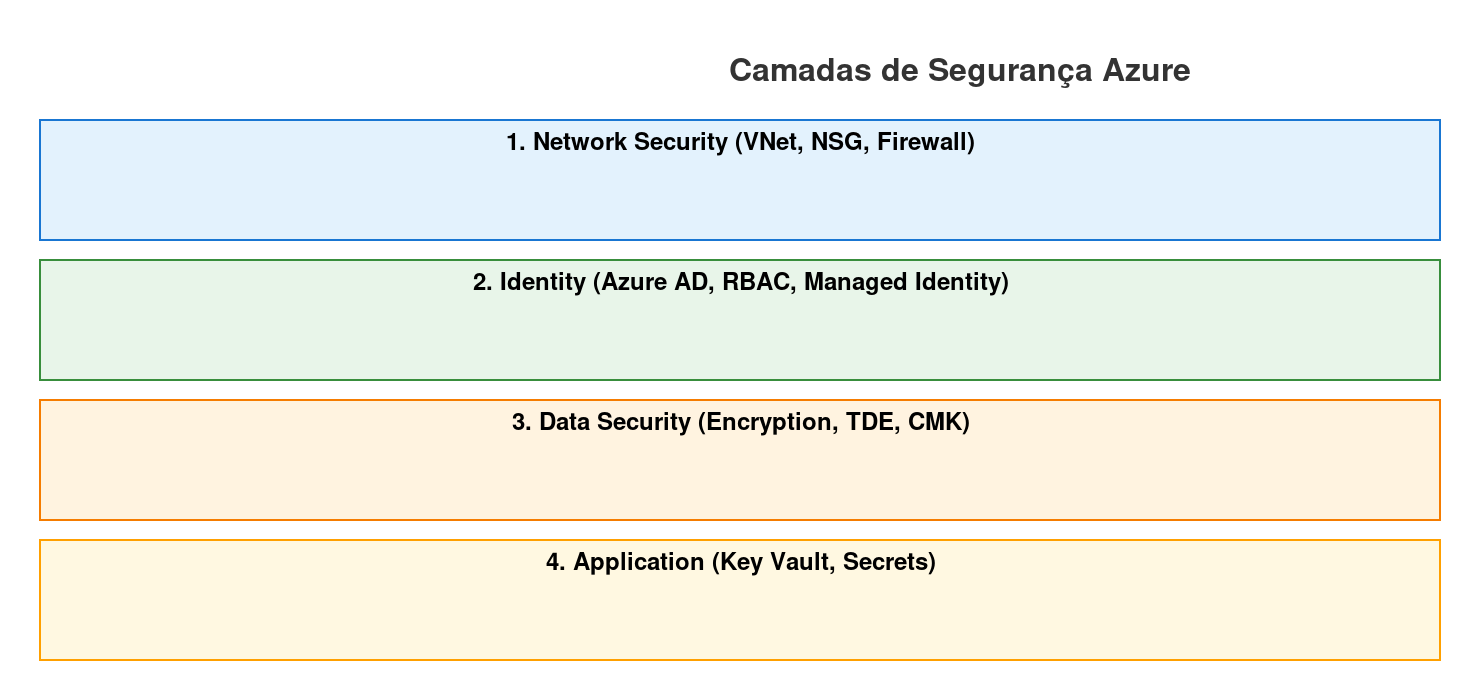
**Managed Identities:**

* Identidade automática para recursos Azure (sem senha!)

### Criar Usuário no Azure AD

**Azure Portal:**

1. **Azure Active Directory → Users → + New user**
2. **User principal name:** maria.silva@seudominio.onmicrosoft.com
3. **Name:** Maria Silva
4. **Password:** Auto-generate (copiar!)
5. **Groups:** Adicionar em grupo (ex: “Data Engineers”)
6. **Create**



**Nome do arquivo:** Junior\_Cap12\_01\_azure\_ad\_user.png

*Figura 12.1 - Criar usuário no Azure AD*

## 12.3 Role-Based Access Control (RBAC)

**RBAC** controla **quem** pode fazer **o quê** em **qual recurso**.

### Principais Built-in Roles

**Nível de Subscription/Resource Group:**

* **Owner:** Controle total (incluindo atribuir permissões)
* **Contributor:** Criar/modificar recursos (sem atribuir permissões)
* **Reader:** Apenas leitura

**Específicos para Data:**

* **Storage Blob Data Contributor:** Ler/gravar blobs
* **Storage Blob Data Reader:** Apenas ler blobs
* **SQL DB Contributor:** Gerenciar databases
* **Data Factory Contributor:** Gerenciar pipelines ADF

### Atribuir Role (RBAC)

**Exemplo: Dar acesso de leitura ao Data Lake**

**Azure Portal:**

1. **Storage Account → Access Control (IAM)**
2. **+ Add → Add role assignment**
3. **Role:** Storage Blob Data Reader
4. **Assign access to:** User, group, or service principal
5. **Select:** Maria Silva
6. **Save**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_02\_rbac\_assignment.png

*Figura 12.2 - Atribuir RBAC role*

**Agora Maria pode ler blobs, mas não modificar!**

### Managed Identity

**Managed Identity** elimina necessidade de armazenar credenciais.

**Tipos:**

**System-assigned:**

* Criada automaticamente com recurso (ex: ADF)
* Deletada quando recurso é deletado

**User-assigned:**

* Criada manualmente
* Pode ser atribuída a múltiplos recursos

**Habilitar Managed Identity no ADF:**

**Azure Portal:**

1. **Data Factory → Identity**
2. **System assigned → Status: On**
3. **Save**
4. **Copiar Object (principal) ID**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_03\_adf\_managed\_identity.png

*Figura 12.3 - Managed Identity do ADF*

**Dar permissão ao ADF para acessar Data Lake:**

**Storage Account → Access Control (IAM) → Add role assignment:**

* **Role:** Storage Blob Data Contributor
* **Assign access to:** Managed identity
* **Select:** data-factory-azuredejunior

**Agora ADF pode acessar Data Lake sem senha!**

## 12.4 Azure Key Vault

**Key Vault** armazena secrets de forma segura (senhas, chaves, certificados).

### Vantagens

* Centralizado (um lugar para todos secrets)
* Controle de acesso (RBAC)
* Auditoria (quem acessou qual secret)
* Rotação automática de secrets

### Criar Key Vault

**Azure Portal:**

1. **Buscar “Key vaults” → + Create**
2. **Configurar:**
   * **Resource group:** rg-azure-de-junior
   * **Key vault name:** kv-azuredejunior-<seunome>
   * **Region:** Brazil South
   * **Pricing tier:** Standard
3. **Access configuration:**
   * **Permission model:** Azure role-based access control (RBAC)
4. **Review + create → Create**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_04\_key\_vault\_created.png

*Figura 12.4 - Key Vault criado*

### Adicionar Secret

**Key Vault → Secrets → + Generate/Import**

**Configurar:**

* **Name:** sql-admin-password
* **Value:** <sua\_senha\_sql>
* **Enabled:** Yes

**Create**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_05\_key\_vault\_secret.png

*Figura 12.5 - Adicionar secret ao Key Vault*

### Usar Secret no ADF

**ADF Linked Service (SQL Database):**

1. **Password:** Selecionar **Azure Key Vault**
2. **AKV linked service:** Criar novo:
   * **Base URL:** https://kv-azuredejunior-<nome>.vault.azure.net/
   * **Authentication:** Managed Identity (ADF)
3. **Secret name:** sql-admin-password
4. **Test connection → Success!**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_06\_adf\_key\_vault\_linked\_service.png

*Figura 12.6 - ADF usando Key Vault secret*

**Dar permissão ADF para ler secrets:**

**Key Vault → Access Control (IAM) → Add role assignment:**

* **Role:** Key Vault Secrets User
* **Assign to:** Managed identity → data-factory-azuredejunior

## 12.5 Network Security

### Private Endpoints

**Private Endpoint** cria conexão privada entre VNet e recurso Azure.

**Vantagens:**

* Dados não trafegam pela internet pública
* Acesso apenas de dentro da VNet
* Compliance (LGPD, HIPAA, etc)

**Criar Private Endpoint (Storage Account):**

**Azure Portal:**

1. **Storage Account → Networking → Private endpoint connections**
2. **+ Private endpoint**
3. **Configurar:**
   * **Name:** pe-datalake
   * **Region:** Brazil South
   * **Virtual network:** Criar ou selecionar VNet existente
   * **Subnet:** default
   * **Target sub-resource:** blob
4. **Create**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_07\_private\_endpoint.png

*Figura 12.7 - Private Endpoint (rede privada)*

**Desabilitar acesso público:**

**Storage Account → Networking → Firewalls and virtual networks:**

* **Public network access:** Disabled

**Agora só acessa via Private Endpoint!**

### Firewall Rules (IP Allowlist)

**Alternativa mais simples (sem VNet):**

**Storage Account → Networking → Firewalls and virtual networks:**

1. **Public network access:** Enabled from selected virtual networks and IP addresses
2. **Firewall → Add your client IP address** (seu IP atual)
3. **Save**

**Apenas seu IP pode acessar!**

## 12.6 Data Lake ACLs (Access Control Lists)

**ACLs** controlam acesso nível de arquivo/pasta no Data Lake.

### Tipos de Permissões

**rwx:**

* **r** (read): Ler arquivo/listar pasta
* **w** (write): Gravar arquivo/criar subpasta
* **x** (execute): Acessar pasta (travessia)

### Configurar ACL

**Azure Portal:**

1. **Storage Account → Containers → datalake → bronze/**
2. **Right-click pasta → Manage ACL**
3. **Add principal:**
   * **User:** maria.silva@empresa.onmicrosoft.com
   * **Permissions:** Read, Execute (r-x)
   * **Apply to:** This folder and all children
4. **Save**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_08\_data\_lake\_acl.png

*Figura 12.8 - Data Lake ACL (permissões granulares)*

**Maria agora pode:**

* Ler arquivos em bronze/
* Não pode modificar/deletar

## 12.7 Encryption (Criptografia)

### Encryption at Rest

**Azure encrypta dados automaticamente (padrão)!**

**Storage Account:**

* **Encryption:** Microsoft-managed keys (default)
* Ou: **Customer-managed keys** (Key Vault)

**Verificar:**

**Storage Account → Encryption → Encryption type: Microsoft-managed keys**

### Encryption in Transit

**TLS (HTTPS) obrigatório:**

**Storage Account → Configuration:**

* **Secure transfer required:** Enabled

**Bloqueia conexões HTTP (inseguras).**

### Transparent Data Encryption (TDE) - SQL Database

**TDE encrypta banco de dados automaticamente.**

**SQL Database → Transparent data encryption:**

* **Status:** Enabled (padrão)

**Dados em disco são encriptados!**

## 12.8 Data Masking

**Data Masking** oculta dados sensíveis em queries.

### Dynamic Data Masking (SQL Database)

**Cenário:** Ocultar parte do email para usuários não-admin.

**SQL Database → Dynamic Data Masking:**

1. **+ Add mask**
2. **Configurar:**
   * **Schema:** SalesLT
   * **Table:** Customer
   * **Column:** EmailAddress
   * **Masking function:** Email (xxx@xxxx.com)
3. **Add**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_09\_data\_masking.png

*Figura 12.9 - Dynamic Data Masking (SQL)*

**Resultado:**

**Usuário admin:**

**SQL**

SELECT EmailAddress FROM SalesLT.Customer;

-- joao.silva@exemplo.com

**Usuário comum:**

**SQL**

SELECT EmailAddress FROM SalesLT.Customer;

-- jXXX@XXXX.com

## 12.9 Auditing e Logging

### Azure Monitor

**Azure Monitor** centraliza logs de todos recursos.

**Habilitar Diagnostic Settings (Storage Account):**

**Azure Portal:**

1. **Storage Account → Diagnostic settings → + Add diagnostic setting**
2. **Configurar:**
   * **Name:** storage-audit-logs
   * **Logs:** Select all (StorageRead, StorageWrite, StorageDelete)
   * **Destination:** Send to Log Analytics workspace
3. **Save**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_10\_diagnostic\_settings.png

*Figura 12.10 - Diagnostic Settings (auditoria)*

### Consultar Logs (Log Analytics)

**Log Analytics Workspace → Logs:**

**Query KQL (Kusto Query Language):**

**KQL**

StorageBlobLogs

| where TimeGenerated > ago(24h)

| where OperationName == “PutBlob” or OperationName == “GetBlob”

| project TimeGenerated, OperationName, CallerIpAddress, Uri

| order by TimeGenerated desc

**Resultado:** Todas operações de leitura/gravação nas últimas 24h.

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_11\_log\_analytics\_query.png

*Figura 12.11 - Log Analytics (consulta de logs)*

### SQL Database Auditing

**SQL Database → Auditing:**

1. **Enable Azure SQL Auditing:** ON
2. **Audit log destination:** Log Analytics
3. **Log Analytics workspace:** Selecionar workspace
4. **Save**

**Audita todas queries, logins, mudanças de schema!**

## 12.10 Microsoft Purview (Data Governance)

**Microsoft Purview** é plataforma de governança de dados.

**Funcionalidades:**

* **Data Catalog:** Catálogo searchable de todos datasets
* **Data Lineage:** Rastrear origem/destino de dados
* **Data Classification:** Identificar dados sensíveis (PII, LGPD)
* **Data Quality:** Monitorar qualidade

### Criar Purview Account

**Azure Portal:**

1. **Buscar “Microsoft Purview” → + Create**
2. **Configurar:**
   * **Resource group:** rg-azure-de-junior
   * **Purview account name:** purview-azuredejunior
   * **Region:** Brazil South
3. **Review + create → Create**

**Custo:** ~R$ 500/mês (considerar para produção)

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_12\_purview\_account.png

*Figura 12.12 - Microsoft Purview (data governance)*

### Registrar Data Source

**Purview Studio (purview.azure.com):**

1. **Data Map → Sources → + Register**
2. **Selecionar tipo:** Azure Data Lake Storage Gen2
3. **Configurar:**
   * **Name:** datalake-bronze
   * **Storage account:** stadejuniorronaldo
   * **Collection:** Root
4. **Register**

### Scan Data Source

**Scan descobre automaticamente datasets e schemas.**

**Purview Studio:**

1. **Source registrado → + New scan**
2. **Configurar:**
   * **Name:** scan-bronze
   * **Credential:** Managed Identity (Purview)
   * **Scope:** Selecionar containers
3. **Run scan**
4. **Aguardar conclusão (5-10 min)**

**Purview cataloga automaticamente todos arquivos!**

### Data Lineage

**Purview rastreia fluxo de dados automaticamente (se integrado com ADF/Databricks).**

**Exemplo de lineage:**

Data Lake (bronze/clientes.csv)

↓

ADF Pipeline (Copy Activity)

↓

SQL Database (clientes table)

↓

Power BI (Dashboard Vendas)

**Visualização gráfica no Purview Studio!**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap12\_13\_purview\_lineage.png

*Figura 12.13 - Data Lineage (rastreabilidade)*

## 12.11 Compliance e Políticas

### Azure Policy

**Azure Policy** garante compliance automático.

**Exemplo: Forçar encryption em todos Storage Accounts**

**Azure Portal:**

1. **Policy → Definitions → Search “storage encryption”**
2. **Selecionar:** “Storage accounts should use customer-managed key for encryption”
3. **Assign → Scope:** Subscription
4. **Assign**

**Agora qualquer Storage Account sem encryption será flagged!**

### Retention Policies (Data Lake)

**Configurar retenção de dados (LGPD compliance):**

**Storage Account → Data management → Lifecycle management:**

1. **+ Add rule**
2. **Configurar:**
   * **Name:** delete-old-bronze-data
   * **Rule scope:** Limit blobs with filters
   * **Blob type:** Block blobs
   * **Blob subtype:** Base blobs
   * **Prefix:** bronze/
   * **Action:** Delete blob
   * **Days after last modification:** 365
3. **Add**

**Dados em bronze/ mais antigos que 1 ano são deletados automaticamente!**

## 12.12 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar camadas de segurança Azure
* [ ] Diferenciar RBAC vs ACL
* [ ] Entender Managed Identity

**Práticos:**

* [ ] Criar usuário no Azure AD
* [ ] Atribuir RBAC roles
* [ ] Habilitar Managed Identity (ADF)
* [ ] Criar Key Vault e armazenar secrets
* [ ] Usar Key Vault no ADF
* [ ] Configurar Private Endpoint
* [ ] Configurar Data Lake ACLs
* [ ] Habilitar Data Masking (SQL)
* [ ] Configurar Diagnostic Settings
* [ ] Consultar logs (Log Analytics)
* [ ] Usar Microsoft Purview (opcional)

## 12.13 Recursos Complementares

**Documentação Oficial:**

* [Azure Security Best Practices](https://learn.microsoft.com/azure/security/fundamentals/best-practices-and-patterns)
* [Azure Key Vault](https://learn.microsoft.com/azure/key-vault/)
* [Microsoft Purview](https://learn.microsoft.com/purview/)

**Compliance:**

* [Azure LGPD Compliance](https://learn.microsoft.com/compliance/regulatory/offering-lgpd)

## 12.14 Exercícios Práticos

**Exercício 1: RBAC Completo**

1. Criar 3 usuários:
   * Admin: Owner role
   * Developer: Contributor role
   * Analyst: Reader role
2. Testar acessos com cada usuário

**Exercício 2: Key Vault Integration**

1. Criar Key Vault
2. Adicionar secrets:
   * SQL admin password
   * Storage account key
3. Configurar ADF para usar ambos secrets

**Exercício 3: Auditoria**

1. Habilitar Diagnostic Settings (Storage, SQL, ADF)
2. Gerar atividade (upload files, queries)
3. Consultar logs no Log Analytics
4. Criar alerta para atividade suspeita

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Identity & Access:**

* Azure AD (users, groups)
* RBAC (roles, assignments)
* Managed Identities (sem senha!)

**Secrets Management:**

* Azure Key Vault (armazenar secrets)
* Integração ADF/Databricks

**Network Security:**

* Private Endpoints (rede privada)
* Firewall rules

**Data Protection:**

* Encryption (at rest, in transit)
* Data Masking (SQL)
* Data Lake ACLs (permissões granulares)

**Auditing:**

* Diagnostic Settings
* Log Analytics (KQL queries)
* SQL Auditing

**Governance:**

* Microsoft Purview (data catalog, lineage)
* Azure Policy (compliance)
* Retention policies

**Próximo capítulo:** Monitoramento e Alertas!

# Capítulo 13: Monitoramento e Alertas

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Configurar Azure Monitor para recursos de dados
* Criar dashboards de monitoramento
* Configurar alertas automatizados
* Monitorar pipelines ADF/Synapse
* Usar Application Insights para debugging
* Criar métricas customizadas
* Implementar health checks automatizados
* Configurar notificações (email, Teams, Slack)
* Analisar performance e custos
* Criar runbooks de incident response

**Tempo estimado:** 4-5 horas

**Custo Azure:** Azure Monitor (R$ 20-100/mês, depende de volume de logs)

## 13.1 Fundamentos de Monitoramento

### Por Que Monitorar?

**Objetivos:**

* **Detecção de falhas:** Identificar problemas antes de impactar negócio
* **Performance:** Otimizar queries e pipelines lentos
* **Custos:** Identificar desperdícios
* **Capacidade:** Planejar escalabilidade
* **Debugging:** Investigar erros

### Tipos de Monitoramento

**1. Metrics (Métricas):**

* Valores numéricos ao longo do tempo
* Ex: CPU%, memória, latência, throughput

**2. Logs:**

* Eventos textuais
* Ex: “Pipeline failed at 10:23”, “Query executed in 2.5s”

**3. Traces (Rastreamento):**

* Fluxo de execução distribuído
* Ex: Request ID atravessando múltiplos serviços

## 13.2 Azure Monitor

**Azure Monitor** é plataforma centralizada de monitoramento.

**Componentes:**

**1. Metrics Explorer:**

* Visualizar métricas em gráficos

**2. Logs (Log Analytics):**

* Consultar logs com KQL

**3. Alerts:**

* Dispara ações quando condições são atendidas

**4. Dashboards:**

* Painéis visuais customizados

**5. Workbooks:**

* Relatórios interativos

## 13.3 Monitorar Azure Data Factory

### Habilitar Diagnostic Settings

**Azure Portal:**

1. **Data Factory → Diagnostic settings → + Add diagnostic setting**
2. **Configurar:**
   * **Name:** adf-monitoring
   * **Logs:** Select all:
     + PipelineRuns
     + ActivityRuns
     + TriggerRuns
   * **Metrics:** AllMetrics
   * **Destination:** Send to Log Analytics workspace
3. **Save**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_01\_adf\_diagnostic\_settings.png

*Figura 13.1 - ADF Diagnostic Settings*

### Métricas do ADF

**Azure Monitor → Metrics:**

**Resource:** data-factory-azuredejunior

**Métricas disponíveis:**

* **PipelineSucceededRuns:** Pipelines bem-sucedidos
* **PipelineFailedRuns:** Pipelines falhados
* **ActivitySucceededRuns:** Activities bem-sucedidas
* **ActivityFailedRuns:** Activities falhadas
* **TriggerSucceededRuns:** Triggers executados

**Criar gráfico:**

1. **Metric:** PipelineFailedRuns
2. **Aggregation:** Count
3. **Time range:** Last 7 days

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_02\_adf\_metrics\_chart.png

*Figura 13.2 - ADF Metrics (falhas de pipelines)*

### Consultar Logs do ADF (KQL)

**Log Analytics Workspace → Logs:**

**Query 1: Pipelines falhados (últimas 24h)**

**KQL**

ADFPipelineRun

| where TimeGenerated > ago(24h)

| where Status == “Failed”

| project TimeGenerated, PipelineName, Status, FailureType, ErrorMessage

| order by TimeGenerated desc

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_03\_adf\_failed\_pipelines\_query.png

*Figura 13.3 - Query: Pipelines falhados*

**Query 2: Top 10 pipelines mais lentos**

**KQL**

ADFPipelineRun

| where TimeGenerated > ago(7d)

| where Status == “Succeeded”

| extend DurationMinutes = (End - Start) / 1m

| summarize AvgDuration = avg(DurationMinutes) by PipelineName

| top 10 by AvgDuration desc

**Query 3: Falhas por erro**

**KQL**

ADFActivityRun

| where TimeGenerated > ago(7d)

| where Status == “Failed”

| summarize FailureCount = count() by ErrorCode

| order by FailureCount desc

## 13.4 Criar Alertas

### Alert Rule (Pipelines Falhados)

**Azure Monitor → Alerts → + Create → Alert rule:**

**1. Scope:**

* **Resource:** data-factory-azuredejunior

**2. Condition:**

* **Signal name:** PipelineFailedRuns
* **Aggregation type:** Total
* **Operator:** Greater than
* **Threshold:** 0
* **Evaluation frequency:** 5 minutes

**3. Actions:**

* **Action group:** Criar novo

**Action group:**

* **Name:** notify-data-team
* **Notification type:** Email/SMS
* **Email:** seu-email@empresa.com

**4. Details:**

* **Alert rule name:** ADF Pipeline Failed
* **Severity:** 2 - Warning

**5. Create**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_04\_alert\_rule\_adf.png

*Figura 13.4 - Alert Rule (pipelines falhados)*

**Quando pipeline falhar → Email enviado automaticamente!**

### Alert com Webhook (Slack/Teams)

**Action group → + Add action:**

**Action type:** Webhook

**Webhook URL:** https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX

**Payload (JSON):**

**JSON**

{

“text”: “ \*ADF Pipeline Failed!\*\nPipeline: #alertContext.condition.allOf[0].metricName\nTime: #alertContext.condition.windowStartTime”

}

**Agora alertas aparecem no Slack!**

## 13.5 Monitorar Databricks

### Cluster Metrics

**Databricks Workspace → Compute → Cluster → Metrics:**

**Métricas disponíveis:**

* CPU Utilization
* Memory Usage
* Network I/O
* Disk I/O

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_05\_databricks\_cluster\_metrics.png

*Figura 13.5 - Databricks Cluster Metrics*

### Spark UI (Job Monitoring)

**Notebook → Após executar célula → Spark Jobs → View (abre Spark UI)**

**Informações disponíveis:**

* **Jobs:** Lista de jobs executados
* **Stages:** Divisões de cada job
* **Tasks:** Paralelização (executors)
* **Storage:** Dados cacheados
* **Executors:** Workers ativos

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_06\_spark\_ui\_jobs.png

*Figura 13.6 - Spark UI (monitoramento de jobs)*

### Databricks Logs (Log Analytics)

**Databricks Workspace → Admin Console → Logging:**

**Configurar:**

* **Destination:** Log Analytics
* **Log Analytics workspace:** Selecionar workspace
* **Enable**

**Logs disponíveis:**

* Cluster events (start, stop, errors)
* Notebook executions
* Job runs

## 13.6 Monitorar SQL Database

### Métricas de Performance

**SQL Database → Metrics:**

**Métricas importantes:**

* **DTU percentage:** Uso de recursos (Database Transaction Units)
* **CPU percentage**
* **Data IO percentage**
* **Log IO percentage**
* **Deadlocks:** Bloqueios
* **Failed connections**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_07\_sql\_database\_metrics.png

*Figura 13.7 - SQL Database Metrics*

### Query Performance Insights

**SQL Database → Query Performance Insight:**

**Mostra:**

* Top queries por CPU
* Top queries por duração
* Top queries por execuções

**Identificar queries problemáticas!**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_08\_query\_performance\_insight.png

*Figura 13.8 - Query Performance Insight*

### Automatic Tuning

**SQL Database → Automatic tuning:**

**Habilitar:**

* Create index (cria índices automaticamente)
* Drop index (remove índices não utilizados)
* Force last good plan (usa plano de execução otimizado)

**Azure otimiza automaticamente performance!**

## 13.7 Monitorar Data Lake (Storage)

### Métricas de Storage

**Storage Account → Metrics:**

**Métricas:**

* **Transactions:** Operações (GET, PUT, DELETE)
* **Ingress:** Bytes enviados ao storage
* **Egress:** Bytes lidos do storage
* **Availability:** % uptime
* **Success E2E Latency:** Latência ponta-a-ponta

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_09\_storage\_metrics.png

*Figura 13.9 - Storage Account Metrics*

### Storage Logs (Auditoria)

**Visto no Cap 12:** Diagnostic Settings → Log Analytics

**Query: Top arquivos acessados**

**KQL**

StorageBlobLogs

| where TimeGenerated > ago(24h)

| where OperationName == “GetBlob”

| summarize AccessCount = count() by Uri

| top 10 by AccessCount desc

## 13.8 Dashboards Customizados

### Criar Dashboard no Azure Portal

**Azure Portal → Dashboard → + New dashboard:**

1. **Name:** Data Platform Monitoring
2. **Adicionar tiles:**

**Tile 1: Metrics Chart (ADF Pipelines)**

* Resource: data-factory-azuredejunior
* Metric: PipelineFailedRuns
* Aggregation: Count
* Time range: Last 24 hours

**Tile 2: Metrics Chart (SQL Database)**

* Resource: sql-azuredejunior
* Metric: DTU percentage
* Time range: Last 1 hour

**Tile 3: Metrics Chart (Storage)**

* Resource: stadejuniorronaldo
* Metric: Transactions
* Time range: Last 6 hours

**Tile 4: Query Results (Log Analytics)**

* Query: ADFPipelineRun | where Status == “Failed” | count

**3. Save dashboard**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_10\_azure\_dashboard.png

*Figura 13.10 - Dashboard customizado*

### Compartilhar Dashboard

**Dashboard → Share:**

* **Publish to Dashboard:** Yes
* **Access control:** Adicionar usuários/grupos

**Agora equipe toda vê dashboard!**

## 13.9 Application Insights (Debugging Avançado)

**Application Insights** monitora aplicações (telemetria detalhada).

### Integrar ADF com Application Insights

**Data Factory → Diagnostic settings → Destination:**

* Send to Log Analytics
* Stream to Event Hub → Application Insights

**Alternativa:** Usar Azure Functions para processamento customizado.

### Distributed Tracing

**Application Insights rastreia requests distribuídos:**

User Request (Power BI)

↓ [request\_id: abc123]

ADF Pipeline

↓ [request\_id: abc123]

Databricks Notebook

↓ [request\_id: abc123]

SQL Database

**Application Insights → Transaction search → Filter by request\_id**

**Visualização completa da execução end-to-end!**

## 13.10 Cost Monitoring (Monitoramento de Custos)

### Cost Management + Billing

**Azure Portal → Cost Management + Billing → Cost analysis:**

**Filtros:**

* **Resource group:** rg-azure-de-junior
* **Time range:** Last 30 days

**Visualizações:**

* **By service:** Quanto gasto por serviço (ADF, Databricks, SQL, Storage)
* **By resource:** Detalhamento por recurso específico
* **Forecast:** Projeção de gastos futuros

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap13\_11\_cost\_analysis.png

*Figura 13.11 - Análise de custos*

### Budget Alerts

**Cost Management → Budgets → + Add:**

**Configurar:**

* **Name:** data-platform-budget-monthly
* **Amount:** R$ 500
* **Alert conditions:**
  + 50% of budget → Email
  + 80% of budget → Email
  + 100% of budget → Email + webhook

**Previne surpresas na fatura!**

## 13.11 Health Checks Automatizados

### Criar Health Check Pipeline (ADF)

**Pipeline que testa conectividade de todas fontes:**

**JSON**

{

“name”: “PL\_HealthCheck\_Daily”,

“activities”: [

{

“name”: “Test SQL Connection”,

“type”: “Lookup”,

“linkedServiceName”: “LS\_SQL\_Database”,

“typeProperties”: {

“source”: {

“query”: “SELECT 1 AS HealthCheck”

}

}

},

{

“name”: “Test Data Lake Connection”,

“type”: “GetMetadata”,

“linkedServiceName”: “LS\_DataLake”,

“typeProperties”: {

“dataset”: “DS\_DataLake\_Root”,

“fieldList”: [“exists”]

}

},

{

“name”: “Test Databricks Connection”,

“type”: “DatabricksNotebook”,

“linkedServiceName”: “LS\_Databricks”,

“typeProperties”: {

“notebookPath”: “/health\_check”,

“baseParameters”: {

“test”: “connectivity”

}

}

}

]

}

**Agendar:** Daily 7:00 AM

**Se alguma activity falhar → Alert disparado!**

## 13.12 Incident Response Runbooks

### Runbook Exemplo: “Pipeline Failed”

**Documento de procedimentos:**

**MARKDOWN**

# Runbook: ADF Pipeline Failed

## Trigger

Alert: ADF Pipeline Failed (Severity 2)

## Steps

1. \*\*Verificar erro:\*\*

- Azure Portal → Data Factory → Monitor → Pipeline runs

- Identificar pipeline falhado

- Clicar em pipeline → Ver detalhes do erro

2. \*\*Classificar erro:\*\*

- Transient (temporário): Timeout, network issue

- Persistent (persistente): Dados inválidos, schema mismatch

3. \*\*Ação imediata:\*\*

- Transient: Re-run pipeline

- Persistent: Investigar root cause

4. \*\*Investigação (Persistent):\*\*

- Log Analytics: Consultar logs detalhados

- Spark UI (se Databricks): Ver job failures

- SQL Query History: Ver queries que falharam

5. \*\*Resolver:\*\*

- Corrigir código/configuração

- Re-deploy pipeline

- Re-run

6. \*\*Comunicar:\*\*

- Teams/Slack: Notificar resolução

- Jira/ADO: Criar ticket (se necessário)

7. \*\*Post-mortem (se grave):\*\*

- Documentar causa raiz

- Implementar prevenção

## 13.13 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar metrics vs logs vs traces
* [ ] Diferenciar alertas reativos vs proativos

**Práticos:**

* [ ] Habilitar Diagnostic Settings (ADF, SQL, Storage)
* [ ] Consultar logs com KQL (Log Analytics)
* [ ] Criar gráficos de métricas (Azure Monitor)
* [ ] Configurar alertas (email, webhook)
* [ ] Criar action groups
* [ ] Monitorar Databricks (Cluster Metrics, Spark UI)
* [ ] Usar Query Performance Insight (SQL)
* [ ] Criar dashboard customizado
* [ ] Analisar custos (Cost Management)
* [ ] Criar health check pipeline

## 13.14 Recursos Complementares

**Documentação Oficial:**

* [Azure Monitor](https://learn.microsoft.com/azure/azure-monitor/)
* [KQL Reference](https://learn.microsoft.com/azure/data-explorer/kusto/query/)
* [Cost Management](https://learn.microsoft.com/azure/cost-management-billing/)

**Tutoriais:**

* [Monitoring ADF Pipelines](https://learn.microsoft.com/azure/data-factory/monitor-pipelines)

## 13.15 Exercícios Práticos

**Exercício 1: Monitoramento Completo**

1. Habilitar Diagnostic Settings para:
   * ADF
   * SQL Database
   * Storage Account
   * Databricks
2. Criar 3 queries KQL úteis
3. Criar dashboard com 6 tiles

**Exercício 2: Alertas**

1. Criar alertas para:
   * ADF Pipeline Failed
   * SQL DTU > 80%
   * Storage Availability < 99%
2. Configurar action group com email + webhook (Slack)
3. Testar disparando alerta manualmente

**Exercício 3: Cost Optimization**

1. Analisar custos dos últimos 30 dias
2. Identificar top 3 recursos mais caros
3. Criar budget alert (R$ 300/mês)
4. Implementar otimização (pausar Databricks, reduzir SQL tier)

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Azure Monitor:** Plataforma centralizada (metrics, logs, alerts)

**Monitoramento por serviço:**

* ADF: Pipeline runs, activity runs, trigger runs
* Databricks: Cluster metrics, Spark UI
* SQL: DTU, Query Performance Insight
* Storage: Transactions, ingress, egress

**Logs:**

* KQL queries (Log Analytics)
* Consultas úteis (falhas, lentidão, top errors)

**Alertas:**

* Alert rules (conditions, thresholds)
* Action groups (email, webhook, SMS)
* Integração Slack/Teams

**Dashboards:** Visualização customizada

**Custos:** Cost Management, budgets

**Health Checks:** Pipelines automatizados

**Incident Response:** Runbooks documentados

**Próximo capítulo:** CI/CD para Pipelines de Dados!

# Capítulo 14: CI/CD para Pipelines de Dados

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender CI/CD para engenharia de dados
* Configurar Git (Azure DevOps / GitHub) para ADF
* Criar branches e pull requests
* Implementar CI (Continuous Integration) com validações
* Implementar CD (Continuous Deployment) automático
* Usar ARM Templates para deploy de infraestrutura
* Versionar notebooks Databricks
* Criar ambientes (Dev, Staging, Prod)
* Implementar testes automatizados
* Rollback de deployments

**Tempo estimado:** 5-6 horas

**Custo Azure:** Azure DevOps FREE (até 5 usuários)

## 14.1 O que é CI/CD?

### Continuous Integration (CI)

**CI:** Integrar código frequentemente, validar automaticamente.

**Fluxo:**

1. Developer faz mudança
2. Commit + Push para Git
3. Build automático
4. Testes automáticos
5. Se passar → Merge

**Benefícios:**

* Detectar erros cedo
* Código sempre funcional
* Reduz conflitos de merge

### Continuous Deployment (CD)

**CD:** Deploy automático para ambientes após CI passar.

**Fluxo:**

1. CI passa
2. Deploy automático para Dev
3. Testes de integração (Dev)
4. Deploy para Staging (manual approval)
5. Deploy para Prod (manual approval)

**Benefícios:**

* Deploy rápido e consistente
* Menos erros manuais
* Rollback fácil

### CI/CD para Data Pipelines

**Diferenças vs software tradicional:**

**Software tradicional:**

* Código (C#, Java, Python)
* Unit tests
* Deploy de aplicação

**Data Pipelines:**

* Código + Configuração (JSON, ARM Templates)
* Data quality tests
* Deploy de infraestrutura + pipelines

## 14.2 Git Basics

### Conceitos Fundamentais

**Repository (Repo):**

* Projeto versionado (histórico completo)

**Branch:**

* Linha independente de desenvolvimento
* Ex: main, develop, feature/add-pipeline

**Commit:**

* Snapshot do código em um momento

**Pull Request (PR):**

* Proposta de merge de branch

**Merge:**

* Combinar mudanças de uma branch em outra

### Workflow Git Flow

main (produção)

↓

develop (desenvolvimento)

↓

feature/add-customer-pipeline (nova feature)

**Fluxo:**

1. Criar branch: feature/add-customer-pipeline
2. Desenvolver mudanças
3. Commit + Push
4. Criar Pull Request → develop
5. Code Review
6. Merge em develop
7. Deploy Dev
8. Merge develop → main (após testes)
9. Deploy Prod

## 14.3 Configurar Git no Azure Data Factory

### Opção 1: Azure DevOps Repos

**Azure DevOps (dev.azure.com):**

1. **Criar organização:** (se não tiver)
   * Nome: azuredejunior
2. **Criar projeto:**
   * Nome: data-platform
   * Visibility: Private
3. **Repos → Initialize repository**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap14\_01\_azure\_devops\_repo.png

*Figura 14.1 - Azure DevOps Repo*

### Conectar ADF ao Git

**ADF Studio:**

1. **Manage → Git configuration → Configure**
2. **Repository type:** Azure DevOps Git
3. **Azure DevOps Account:** Selecionar organização
4. **Project name:** data-platform
5. **Repository name:** adf-pipelines (criar novo)
6. **Collaboration branch:** main
7. **Publish branch:** adf\_publish
8. **Root folder:** /
9. **Import existing resources:** Yes (importar pipelines atuais)
10. **Save**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap14\_02\_adf\_git\_config.png

*Figura 14.2 - ADF Git Configuration*

**ADF agora salva mudanças no Git automaticamente!**

### Estrutura do Repositório ADF

adf-pipelines/

pipeline/

PL\_Ingest\_Customers.json

PL\_Transform\_Sales.json

...

dataset/

DS\_SQL\_Customers.json

DS\_DataLake\_Bronze.json

...

linkedService/

LS\_SQL\_Database.json

LS\_DataLake.json

...

trigger/

TR\_Daily\_Midnight.json

...

factory/

data-factory-azuredejunior.json

**Cada pipeline/dataset = arquivo JSON versionado!**

## 14.4 Workflow de Desenvolvimento

### Passo 1: Criar Branch de Feature

**ADF Studio:**

1. **Branch dropdown (canto superior esquerdo) → + New branch**
2. **Name:** feature/add-product-pipeline
3. **Base branch:** main
4. **Create**

**Agora trabalhando em branch isolada!**

### Passo 2: Desenvolver Pipeline

**Desenvolver normalmente:**

* Criar pipeline
* Criar datasets
* Testar (Debug)

**Mudanças salvas apenas em `feature/add-product-pipeline`!**

### Passo 3: Commit & Sync

**ADF salva automaticamente (auto-commit).**

**Para sincronizar com Azure DevOps:**

**ADF Studio → Save all (Ctrl+S)**

**Mudanças aparecem no Azure DevOps Repos!**

### Passo 4: Criar Pull Request

**Azure DevOps → Repos → Pull requests → + New pull request**

**Configurar:**

* **Source branch:** feature/add-product-pipeline
* **Target branch:** main
* **Title:** Add product ingestion pipeline
* **Description:** Pipeline para ingerir dados de produtos do SQL para Data Lake

**Reviewers:** Adicionar colegas (opcional)

**Create**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap14\_03\_pull\_request.png

*Figura 14.3 - Pull Request (code review)*

### Passo 5: Code Review

**Reviewer verifica:**

* Pipeline está correto?
* Nomenclatura seguiu padrão?
* Datasets corretos?

**Aprovar ou Solicitar mudanças**

### Passo 6: Merge

**Após aprovação:**

**Complete → Merge (squash commits)**

**Branch `feature/add-product-pipeline` mergeada em `main`!**

## 14.5 Deploy com ARM Templates

**ADF gera ARM Templates automaticamente ao publicar.**

### Publish (Gerar ARM Templates)

**ADF Studio:**

1. **Branch:** main (switch para main)
2. **Publish (botão superior)**
3. **Confirm**

**ADF gera ARM Templates em branch `adf\_publish`!**

**Estrutura:**

adf\_publish/

ARMTemplateForFactory.json (template principal)

ARMTemplateParametersForFactory.json (parâmetros)

linkedTemplates/

ArmTemplate\_0.json

...

### Deploy Manual (Azure Portal)

**Azure Portal → Create a resource → Template deployment (deploy a custom template):**

1. **Build your own template in the editor**
2. **Load file:** ARMTemplateForFactory.json
3. **Parameters:**
   * **factoryName:** data-factory-azuredejunior-prod
   * **linkedService\_SQL\_connectionString:** (prod connection string)
4. **Review + create → Create**

**Deploy em ambiente Prod!**

## 14.6 CI/CD Pipeline (Azure DevOps)

### Criar Build Pipeline (CI)

**Azure DevOps → Pipelines → + New pipeline:**

1. **Where is your code:** Azure Repos Git
2. **Select repository:** adf-pipelines
3. **Configure pipeline:** Starter pipeline
4. **YAML:**

**YAML**

# azure-pipelines.yml

trigger:

branches:

include:

- main

pool:

vmImage: 'ubuntu-latest'

stages:

- stage: Validate

displayName: 'Validate ADF Resources'

jobs:

- job: ValidateJSON

displayName: 'Validate JSON Files'

steps:

- task: PowerShell@2

inputs:

targetType: 'inline'

script: |

# Validate JSON syntax

Get-ChildItem -Path $(Build.SourcesDirectory) -Recurse -Filter \*.json | ForEach-Object {

Write-Host “Validating $($\_.FullName)”

Get-Content $\_.FullName | ConvertFrom-Json | Out-Null

}

Write-Host “All JSON files are valid!”

displayName: 'Validate JSON Syntax'

- task: AzureCLI@2

inputs:

azureSubscription: 'azure-service-connection'

scriptType: 'bash'

scriptLocation: 'inlineScript'

inlineScript: |

# Validate naming conventions

echo “Validating naming conventions...”

# Add custom validation logic here

displayName: 'Validate Naming Conventions'

1. **Save and run**

**Pipeline valida JSON automaticamente a cada commit!**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap14\_04\_ci\_pipeline.png

*Figura 14.4 - CI Pipeline (validação automática)*

### Criar Release Pipeline (CD)

**Azure DevOps → Pipelines → Releases → + New pipeline:**

**1. Artifact:**

* **Source:** Build pipeline (CI)
* **Branch:** main

**2. Stages:**

**Stage 1: Deploy to Dev**

* **Agent:** Azure Pipelines (ubuntu-latest)
* **Tasks:**
  + **Azure Resource Group Deployment**
    - Template: $(System.DefaultWorkingDirectory)/\_adf-pipelines/adf\_publish/ARMTemplateForFactory.json
    - Parameters: $(System.DefaultWorkingDirectory)/\_adf-pipelines/adf\_publish/ARMTemplateParametersForFactory.json
    - Override parameters:

-factoryName data-factory-azuredejunior-dev

-linkedService\_SQL\_connectionString $(SQL\_DEV\_CONNECTION\_STRING)

**Stage 2: Deploy to Prod**

* **Pre-deployment approval:** Enabled (requer aprovação manual)
* **Tasks:** (mesmos de Dev, mas parâmetros Prod)

**3. Save**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap14\_05\_cd\_pipeline.png

*Figura 14.5 - CD Pipeline (deploy automático)*

**Fluxo completo:**

Commit → CI Pipeline (validate) → Release Pipeline (deploy Dev) → Approval → Deploy Prod

## 14.7 Databricks + Git

### Repos (Databricks Git Integration)

**Databricks Workspace → Repos → Add Repo:**

1. **Git provider:** GitHub / Azure DevOps
2. **Git repository URL:** https://dev.azure.com/azuredejunior/data-platform/\_git/databricks-notebooks
3. **Clone**

**Estrutura:**

databricks-notebooks/

bronze\_to\_silver/

transform\_customers.py

transform\_sales.py

silver\_to\_gold/

aggregate\_metrics.py

create\_dimensions.py

tests/

test\_transformations.py

...

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap14\_06\_databricks\_repos.png

*Figura 14.6 - Databricks Repos (Git integration)*

### Workflow Databricks

**1. Criar branch:**

* Repos → Branch dropdown → Create branch: feature/improve-transformations

**2. Editar notebooks:**

* Mudanças salvas localmente (Databricks)

**3. Commit:**

* Repos → Git → Commit & Push

**4. Pull Request (Azure DevOps):**

* Mesmos passos que ADF

**5. Merge → Deploy:**

* CD Pipeline executa notebooks via Databricks CLI

## 14.8 Ambientes (Dev, Staging, Prod)

### Estratégia de Ambientes

**Dev (Desenvolvimento):**

* Developers testam mudanças
* Dados sample/sintéticos
* Custo baixo (recursos mínimos)

**Staging (Homologação):**

* Testes de integração
* Dados production-like
* Configuração idêntica a Prod

**Prod (Produção):**

* Dados reais
* SLA alto
* Monitoramento intenso

### Parametrização

**ARM Template parameters por ambiente:**

**Dev:**

**JSON**

{

“factoryName”: “adf-azuredejunior-dev”,

“sqlConnectionString”: “Server=sql-dev.database.windows.net;...”,

“storageAccountName”: “staazuredejuniordev”

}

**Prod:**

**JSON**

{

“factoryName”: “adf-azuredejunior-prod”,

“sqlConnectionString”: “Server=sql-prod.database.windows.net;...”,

“storageAccountName”: “staazuredejuniorprod”

}

**CD Pipeline usa parâmetros corretos por stage!**

## 14.9 Testes Automatizados

### Unit Tests (Databricks)

**Arquivo:** tests/test\_transformations.py

**PYTHON**

import pytest

from transformations import clean\_customer\_data

def test\_clean\_customer\_data():

# Arrange

data = [

(1, “ Alice “, None), # Nome com espaços, email null

(2, “Bob”, “bob@example.com”)

]

df = spark.createDataFrame(data, [“id”, “name”, “email”])

# Act

df\_cleaned = clean\_customer\_data(df)

# Assert

assert df\_cleaned.filter(“id = 1”).select(“name”).collect()[0][0] == “Alice” # Trim

assert df\_cleaned.filter(“id = 1 AND email IS NULL”).count() == 1 # Null mantido

**Executar tests:**

**BASH**

pytest tests/

### Integration Tests (ADF)

**Pipeline de teste:**

**`PL\_Test\_Integration`:**

1. **Ingest sample data** (SQL → Data Lake)
2. **Run transformation** (Databricks)
3. **Validate output** (row count, schema, data quality)

**CD Pipeline executa antes de deploy Prod:**

**YAML**

- task: AzureCLI@2

inputs:

scriptType: 'bash'

inlineScript: |

# Trigger test pipeline

az datafactory pipeline create-run \

--factory-name adf-azuredejunior-staging \

--name PL\_Test\_Integration

# Wait for completion

# Validate success

## 14.10 Rollback

### Estratégia de Rollback

**Cenário:** Deploy em Prod causou erro.

**Opção 1: Rollback via Git**

**BASH**

# Reverter commit

git revert <commit\_hash>

git push

# Trigger CD Pipeline novamente

**Opção 2: Rollback via ARM Template (versão anterior)**

**Azure DevOps → Releases → Previous release → Redeploy**

**Opção 3: Rollback manual (ADF Studio)**

**ADF Studio → Branch: adf\_publish → Git history → Checkout previous commit**

## 14.11 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar CI vs CD
* [ ] Entender Git workflow (branch, commit, PR, merge)
* [ ] Diferenciar ambientes (Dev, Staging, Prod)

**Práticos:**

* [ ] Configurar Git no ADF (Azure DevOps)
* [ ] Criar branches e commits
* [ ] Criar Pull Requests
* [ ] Gerar ARM Templates (Publish)
* [ ] Deploy manual via ARM Template
* [ ] Criar CI Pipeline (validação)
* [ ] Criar CD Pipeline (deploy automático)
* [ ] Versionar Databricks notebooks (Repos)
* [ ] Parametrizar pipelines (Dev vs Prod)
* [ ] Criar testes automatizados
* [ ] Fazer rollback de deploy

## 14.12 Recursos Complementares

**Documentação Oficial:**

* [ADF Source Control](https://learn.microsoft.com/azure/data-factory/source-control)
* [Azure DevOps Pipelines](https://learn.microsoft.com/azure/devops/pipelines/)
* [Databricks Repos](https://learn.microsoft.com/azure/databricks/repos/)

**Best Practices:**

* [CI/CD for Data Pipelines](https://learn.microsoft.com/azure/architecture/example-scenario/data/devops-data-platform)

## 14.13 Exercícios Práticos

**Exercício 1: Git Workflow Completo**

1. Configurar Git no ADF (Azure DevOps)
2. Criar branch feature/add-test-pipeline
3. Criar pipeline simples
4. Commit + Push
5. Criar Pull Request
6. Merge em main

**Exercício 2: CI/CD End-to-End**

1. Criar CI Pipeline (validação JSON)
2. Criar CD Pipeline (deploy Dev + Prod)
3. Fazer mudança em pipeline
4. Commit → Trigger CI → Deploy Dev → Aprovar → Deploy Prod
5. Validar pipeline rodando em Prod

**Exercício 3: Databricks Versioning**

1. Conectar Databricks Repos (Git)
2. Criar notebook transformação
3. Criar branch, editar, commit
4. Criar PR, merge
5. Deploy via CD Pipeline (Databricks CLI)

## Resumo do Capítulo

Neste capítulo você aprendeu:

**CI/CD:** Continuous Integration + Continuous Deployment

**Git:**

* Branches, commits, Pull Requests
* Code review workflow

**ADF + Git:**

* Configuração (Azure DevOps / GitHub)
* Estrutura de repo (pipelines, datasets, linkedServices)
* ARM Templates (deploy)

**CI Pipeline:**

* Validação automática (JSON syntax, naming conventions)

**CD Pipeline:**

* Deploy automático (Dev, Staging, Prod)
* Aprovações manuais
* Parametrização por ambiente

**Databricks:**

* Repos (Git integration)
* Versioning de notebooks

**Testes:**

* Unit tests (Databricks)
* Integration tests (ADF)

**Rollback:** Estratégias para reverter deploys

**Próximo capítulo:** Projeto Final Integrado!

# Capítulo 15: Projeto Final Integrado

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Integrar todos conhecimentos do livro em projeto completo
* Implementar pipeline end-to-end (Bronze → Silver → Gold)
* Aplicar arquitetura Medallion (camadas de dados)
* Criar Data Warehouse dimensional (Star Schema)
* Implementar monitoramento e alertas
* Configurar CI/CD completo
* Documentar arquitetura e processos
* Apresentar projeto (portfolio)

**Tempo estimado:** 12-15 horas

**Custo Azure:** R$ 100-200 (recursos temporários, deletar após projeto)

## 15.1 Visão Geral do Projeto

### Cenário de Negócio

**Empresa:** E-commerce “TechStore Brasil”

**Desafio:**

* Dados em múltiplas fontes (SQL Server transacional, arquivos CSV de fornecedores)
* Sem visibilidade de vendas em tempo real
* Relatórios manuais (Excel) demoram dias
* Decisões baseadas em dados desatualizados

**Objetivo:**

Construir plataforma de dados moderna no Azure para:

* Centralizar dados (Data Lake)
* Transformar e enriquecer (Databricks)
* Criar Data Warehouse (Synapse)
* Dashboards executivos (Power BI)
* Atualização automática (diária)

### Arquitetura Alvo

SOURCES

SQL Server CSV Files (Blob) External API

(Transacional) (Fornecedores) (Cotação USD)

Azure Data Factory

(Orchestration)

BRONZE LAYER (Raw Data)

Azure Data Lake Gen2

/bronze/vendas/ /bronze/produtos/ /bronze/clientes/

Parquet (append) CSV (original) Parquet

Databricks (Spark)

- Data Quality Checks

- Deduplication

- Enrichment

SILVER LAYER (Cleansed Data)

Azure Data Lake Gen2 (Delta Lake)

/silver/dim\_produto/ /silver/dim\_cliente/

/silver/fato\_vendas/ (ACID, SCD Type 2)

Databricks (Aggregations)

- Star Schema

- Business Metrics

GOLD LAYER (Curated Data)

Azure Synapse Dedicated SQL Pool

dim\_produto dim\_cliente dim\_data fato\_vendas

(Star Schema optimized for BI)

Power BI

- Executive Dashboard

- Sales Analytics

DASHBOARDS

Total Vendas Vendas/Tempo Top Produtos

Top Clientes Margem Lucro Vendas/Região

CROSS-CUTTING CONCERNS

Security: RBAC, Key Vault, Private Endpoints

Monitoring: Azure Monitor, Log Analytics, Alerts

CI/CD: Azure DevOps (Git, Pipelines)

Governance: Microsoft Purview, Data Lineage

## 15.2 Fase 1: Provisionar Infraestrutura

### Recursos Necessários

**Resource Group:**

* rg-techstore-prod

**Storage:**

* Storage Account: statechstoreprod
* Containers: bronze, silver, gold

**Processing:**

* Azure Data Factory: adf-techstore-prod
* Databricks Workspace: dbw-techstore-prod
* Synapse Workspace: synapse-techstore-prod

**Database:**

* SQL Database: sqldb-techstore-source (simula sistema transacional)

**Visualization:**

* Power BI (Desktop + Service)

**Security:**

* Key Vault: kv-techstore-prod

**Monitoring:**

* Log Analytics Workspace: law-techstore-prod

### Script de Provisionamento (ARM Template)

**Criar arquivo:** infrastructure.json

**JSON**

{

“$schema”: “https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#”,

“contentVersion”: “1.0.0.0”,

“parameters”: {

“projectName”: {

“type”: “string”,

“defaultValue”: “techstore”

},

“environment”: {

“type”: “string”,

“defaultValue”: “prod”

}

},

“variables”: {

“storageAccountName”: “[concat('sta', parameters('projectName'), parameters('environment'))]”,

“dataFactoryName”: “[concat('adf-', parameters('projectName'), '-', parameters('environment'))]”,

“synapseName”: “[concat('synapse-', parameters('projectName'), '-', parameters('environment'))]”

},

“resources”: [

{

“type”: “Microsoft.Storage/storageAccounts”,

“apiVersion”: “2021-04-01”,

“name”: “[variables('storageAccountName')]”,

“location”: “[resourceGroup().location]”,

“sku”: {

“name”: “Standard\_LRS”

},

“kind”: “StorageV2”,

“properties”: {

“isHnsEnabled”: true,

“minimumTlsVersion”: “TLS1\_2”

}

},

{

“type”: “Microsoft.DataFactory/factories”,

“apiVersion”: “2018-06-01”,

“name”: “[variables('dataFactoryName')]”,

“location”: “[resourceGroup().location]”,

“identity”: {

“type”: “SystemAssigned”

}

}

]

}

**Deploy:**

**BASH**

az deployment group create \

--resource-group rg-techstore-prod \

--template-file infrastructure.json

## 15.3 Fase 2: Criar Dados de Origem (Source)

### Simular Sistema Transacional

**SQL Database - Criar tabelas:**

**SQL**

-- Tabela de Clientes

CREATE TABLE clientes (

cliente\_id INT PRIMARY KEY,

nome VARCHAR(100),

email VARCHAR(100),

cidade VARCHAR(50),

estado VARCHAR(2),

data\_cadastro DATE

);

-- Tabela de Produtos

CREATE TABLE produtos (

produto\_id INT PRIMARY KEY,

nome\_produto VARCHAR(200),

categoria VARCHAR(50),

preco\_custo DECIMAL(10,2),

preco\_venda DECIMAL(10,2),

fornecedor VARCHAR(100)

);

-- Tabela de Vendas (transacional)

CREATE TABLE vendas (

venda\_id INT PRIMARY KEY,

cliente\_id INT,

produto\_id INT,

quantidade INT,

valor\_total DECIMAL(10,2),

data\_venda DATETIME,

status VARCHAR(20),

FOREIGN KEY (cliente\_id) REFERENCES clientes(cliente\_id),

FOREIGN KEY (produto\_id) REFERENCES produtos(produto\_id)

);

**Inserir dados sample:**

**SQL**

-- Script Python para gerar dados sample

INSERT INTO clientes VALUES

(1, 'João Silva', 'joao@email.com', 'São Paulo', 'SP', '2023-01-15'),

(2, 'Maria Santos', 'maria@email.com', 'Rio de Janeiro', 'RJ', '2023-02-20'),

(3, 'Pedro Oliveira', 'pedro@email.com', 'Belo Horizonte', 'MG', '2023-03-10');

-- ... (gerar 1000+ registros)

INSERT INTO produtos VALUES

(1, 'Notebook Dell Inspiron', 'Informática', 2500.00, 3500.00, 'Dell'),

(2, 'Mouse Logitech MX', 'Periféricos', 80.00, 150.00, 'Logitech'),

(3, 'Teclado Mecânico', 'Periféricos', 200.00, 350.00, 'Razer');

-- ... (gerar 100+ produtos)

INSERT INTO vendas VALUES

(1, 1, 1, 1, 3500.00, '2024-01-15 10:30:00', 'Concluída'),

(2, 2, 2, 2, 300.00, '2024-01-15 14:20:00', 'Concluída'),

(3, 1, 3, 1, 350.00, '2024-01-16 09:15:00', 'Concluída');

-- ... (gerar 10000+ vendas)

### Dados CSV (Fornecedores)

**Criar arquivo:** fornecedores.csv

**CSV**

fornecedor\_id,nome\_fornecedor,cnpj,cidade,estado

1,Tech Distribuidora,12345678000100,São Paulo,SP

2,Mega Atacado,98765432000111,Curitiba,PR

3,Global Tech,11122233000144,Fortaleza,CE

**Upload para Blob Storage (container `landing`):**

**BASH**

az storage blob upload \

--account-name statechstoreprod \

--container-name landing \

--name fornecedores/fornecedores.csv \

--file fornecedores.csv

## 15.4 Fase 3: Pipeline Bronze (Ingestão)

### Pipeline ADF: `PL\_Ingest\_Bronze`

**Activities:**

**1. Copy Activity: SQL → Bronze (Vendas)**

* Source: SQL Database (vendas table)
* Sink: Data Lake /bronze/vendas/vendas\_{date}.parquet
* Incremental: Watermark (data\_venda)

**2. Copy Activity: SQL → Bronze (Clientes)**

* Source: SQL Database (clientes table)
* Sink: Data Lake /bronze/clientes/clientes\_{date}.parquet
* Full load (pequeno)

**3. Copy Activity: SQL → Bronze (Produtos)**

* Source: SQL Database (produtos table)
* Sink: Data Lake /bronze/produtos/produtos\_{date}.parquet

**4. Copy Activity: Blob → Bronze (Fornecedores)**

* Source: Blob Storage (fornecedores.csv)
* Sink: Data Lake /bronze/fornecedores/fornecedores\_{date}.csv

**Parametrizar data:**

**Pipeline parameters:**

* execution\_date (tipo: String, valor: @formatDateTime(utcnow(), 'yyyy-MM-dd'))

**Usar em Sink path:**

/bronze/vendas/vendas\_@{pipeline().parameters.execution\_date}.parquet

**Trigger:**

**Schedule Trigger:**

* Name: TR\_Daily\_Bronze
* Recurrence: Daily, 01:00 AM (UTC-3)

## 15.5 Fase 4: Pipeline Silver (Transformação)

### Notebook Databricks: `bronze\_to\_silver.py`

**PYTHON**

# bronze\_to\_silver.py

from pyspark.sql.functions import col, current\_timestamp, sha2, concat\_ws, when

from delta.tables import DeltaTable

# Parâmetros

execution\_date = dbutils.widgets.get(“execution\_date”)

bronze\_path = f”/mnt/datalake/bronze”

silver\_path = f”/mnt/datalake/silver”

# ==============================================================================

# 1. CLIENTES (Dimension - SCD Type 2)

# ==============================================================================

print(“Processando dim\_cliente...”)

# Ler bronze

df\_clientes\_bronze = spark.read.parquet(f”{bronze\_path}/clientes/clientes\_{execution\_date}.parquet”)

# Data quality checks

df\_clientes = df\_clientes\_bronze.filter(

col(“cliente\_id”).isNotNull() &

col(“nome”).isNotNull() &

col(“email”).rlike(r”^[\w\.-]+@[\w\.-]+\.\w+$”)

)

# Adicionar colunas SCD Type 2

df\_clientes = df\_clientes \

.withColumn(“valid\_from”, current\_timestamp()) \

.withColumn(“valid\_to”, lit(None).cast(“timestamp”)) \

.withColumn(“is\_current”, lit(True)) \

.withColumn(“hash\_value”, sha2(concat\_ws(“||”, col(“nome”), col(“cidade”), col(“estado”)), 256))

# MERGE (SCD Type 2)

delta\_path\_cliente = f”{silver\_path}/dim\_cliente”

if DeltaTable.isDeltaTable(spark, delta\_path\_cliente):

delta\_table = DeltaTable.forPath(spark, delta\_path\_cliente)

# Expirar registros antigos se mudou

delta\_table.alias(“target”).merge(

df\_clientes.alias(“source”),

“target.cliente\_id = source.cliente\_id AND target.is\_current = true”

).whenMatchedUpdate(

condition=“target.hash\_value <> source.hash\_value”,

set={

“valid\_to”: current\_timestamp(),

“is\_current”: lit(False)

}

).execute()

# Inserir novos registros

df\_clientes.write.format(“delta”).mode(“append”).save(delta\_path\_cliente)

else:

df\_clientes.write.format(“delta”).mode(“overwrite”).save(delta\_path\_cliente)

print(f”dim\_cliente: {df\_clientes.count()} registros processados”)

# ==============================================================================

# 2. PRODUTOS (Dimension - SCD Type 1)

# ==============================================================================

print(“Processando dim\_produto...”)

df\_produtos\_bronze = spark.read.parquet(f”{bronze\_path}/produtos/produtos\_{execution\_date}.parquet”)

# Data quality

df\_produtos = df\_produtos\_bronze.filter(

col(“produto\_id”).isNotNull() &

col(“nome\_produto”).isNotNull() &

col(“preco\_venda”) > 0

)

# Calcular margem

df\_produtos = df\_produtos.withColumn(

“margem\_percentual”,

((col(“preco\_venda”) - col(“preco\_custo”)) / col(“preco\_venda”) \* 100).cast(“decimal(5,2)”)

)

# MERGE (SCD Type 1 - sobrescreve)

delta\_path\_produto = f”{silver\_path}/dim\_produto”

if DeltaTable.isDeltaTable(spark, delta\_path\_produto):

delta\_table = DeltaTable.forPath(spark, delta\_path\_produto)

delta\_table.alias(“target”).merge(

df\_produtos.alias(“source”),

“target.produto\_id = source.produto\_id”

).whenMatchedUpdateAll() \

.whenNotMatchedInsertAll() \

.execute()

else:

df\_produtos.write.format(“delta”).mode(“overwrite”).save(delta\_path\_produto)

print(f”dim\_produto: {df\_produtos.count()} registros processados”)

# ==============================================================================

# 3. VENDAS (Fact Table)

# ==============================================================================

print(“Processando fato\_vendas...”)

df\_vendas\_bronze = spark.read.parquet(f”{bronze\_path}/vendas/vendas\_{execution\_date}.parquet”)

# Data quality

df\_vendas = df\_vendas\_bronze.filter(

col(“venda\_id”).isNotNull() &

col(“cliente\_id”).isNotNull() &

col(“produto\_id”).isNotNull() &

col(“quantidade”) > 0 &

col(“valor\_total”) > 0

)

# Enriquecer (joins)

df\_vendas = df\_vendas.alias(“v”) \

.join(df\_clientes.alias(“c”), col(“v.cliente\_id”) == col(“c.cliente\_id”), “left”) \

.join(df\_produtos.alias(“p”), col(“v.produto\_id”) == col(“p.produto\_id”), “left”) \

.select(

col(“v.venda\_id”),

col(“v.cliente\_id”),

col(“v.produto\_id”),

col(“v.data\_venda”),

col(“v.quantidade”),

col(“v.valor\_total”),

col(“v.status”),

col(“c.cidade”).alias(“cliente\_cidade”),

col(“c.estado”).alias(“cliente\_estado”),

col(“p.categoria”).alias(“produto\_categoria”)

)

# Particionar por data

df\_vendas = df\_vendas.withColumn(“data\_particao”, col(“data\_venda”).cast(“date”))

# Gravar Delta (append)

delta\_path\_vendas = f”{silver\_path}/fato\_vendas”

df\_vendas.write \

.format(“delta”) \

.mode(“append”) \

.partitionBy(“data\_particao”) \

.save(delta\_path\_vendas)

print(f”fato\_vendas: {df\_vendas.count()} registros processados”)

# ==============================================================================

# Data Quality Report

# ==============================================================================

print(“\n=== Data Quality Summary ===“)

print(f”Clientes inválidos removidos: {df\_clientes\_bronze.count() - df\_clientes.count()}”)

print(f”Produtos inválidos removidos: {df\_produtos\_bronze.count() - df\_produtos.count()}”)

print(f”Vendas inválidas removidas: {df\_vendas\_bronze.count() - df\_vendas.count()}”)

dbutils.notebook.exit(“SUCCESS”)

**Pipeline ADF: `PL\_Transform\_Silver`**

**Activity:**

* **Databricks Notebook:**
  + Notebook path: /bronze\_to\_silver
  + Base parameters: {“execution\_date”: “@formatDateTime(utcnow(), 'yyyy-MM-dd')”}

**Dependency:**

* Runs after PL\_Ingest\_Bronze succeeds

## 15.6 Fase 5: Pipeline Gold (Data Warehouse)

### Synapse Dedicated Pool: Criar Star Schema

**SQL**

-- Dimension: Cliente

CREATE TABLE dim\_cliente (

cliente\_key INT IDENTITY(1,1) PRIMARY KEY NONCLUSTERED NOT ENFORCED,

cliente\_id INT NOT NULL,

nome VARCHAR(100),

email VARCHAR(100),

cidade VARCHAR(50),

estado VARCHAR(2),

valid\_from DATETIME2,

valid\_to DATETIME2,

is\_current BIT

)

WITH (

DISTRIBUTION = REPLICATE,

CLUSTERED COLUMNSTORE INDEX

);

-- Dimension: Produto

CREATE TABLE dim\_produto (

produto\_key INT IDENTITY(1,1) PRIMARY KEY NONCLUSTERED NOT ENFORCED,

produto\_id INT NOT NULL,

nome\_produto VARCHAR(200),

categoria VARCHAR(50),

preco\_custo DECIMAL(10,2),

preco\_venda DECIMAL(10,2),

margem\_percentual DECIMAL(5,2)

)

WITH (

DISTRIBUTION = REPLICATE,

CLUSTERED COLUMNSTORE INDEX

);

-- Dimension: Data (Calendar Table)

CREATE TABLE dim\_data (

data\_key INT PRIMARY KEY NONCLUSTERED NOT ENFORCED,

data DATE,

ano INT,

mes INT,

trimestre INT,

dia\_semana INT,

nome\_mes VARCHAR(20),

nome\_dia\_semana VARCHAR(20)

)

WITH (

DISTRIBUTION = REPLICATE,

CLUSTERED COLUMNSTORE INDEX

);

-- Fact: Vendas

CREATE TABLE fato\_vendas (

venda\_id INT,

cliente\_key INT,

produto\_key INT,

data\_key INT,

quantidade INT,

valor\_total DECIMAL(10,2),

status VARCHAR(20)

)

WITH (

DISTRIBUTION = HASH(cliente\_key),

CLUSTERED COLUMNSTORE INDEX,

PARTITION (data\_key RANGE RIGHT FOR VALUES (20230101, 20240101, 20250101))

);

### Pipeline ADF: `PL\_Load\_Gold`

**Activities:**

**1. Copy Data: Delta Lake → Synapse (dim\_cliente)**

* Source: Delta Lake /silver/dim\_cliente/ (filter: is\_current = true)
* Sink: Synapse dim\_cliente
* Pre-copy script: TRUNCATE TABLE dim\_cliente

**2. Copy Data: Delta Lake → Synapse (dim\_produto)**

* Source: Delta Lake /silver/dim\_produto/
* Sink: Synapse dim\_produto
* Pre-copy script: TRUNCATE TABLE dim\_produto

**3. Copy Data: Delta Lake → Synapse (fato\_vendas)**

* Source: Delta Lake /silver/fato\_vendas/ (incremental)
* Sink: Synapse fato\_vendas
* Mode: Append (incremental)

**4. Stored Procedure: Populate dim\_data**

**SQL**

CREATE PROCEDURE populate\_dim\_data

AS

BEGIN

TRUNCATE TABLE dim\_data;

DECLARE @start\_date DATE = '2023-01-01';

DECLARE @end\_date DATE = '2025-12-31';

WITH dates AS (

SELECT @start\_date AS data

UNION ALL

SELECT DATEADD(DAY, 1, data)

FROM dates

WHERE data < @end\_date

)

INSERT INTO dim\_data

SELECT

CAST(FORMAT(data, 'yyyyMMdd') AS INT) AS data\_key,

data,

YEAR(data) AS ano,

MONTH(data) AS mes,

DATEPART(QUARTER, data) AS trimestre,

DATEPART(WEEKDAY, data) AS dia\_semana,

FORMAT(data, 'MMMM', 'pt-BR') AS nome\_mes,

FORMAT(data, 'dddd', 'pt-BR') AS nome\_dia\_semana

FROM dates

OPTION (MAXRECURSION 0);

END;

## 15.7 Fase 6: Dashboard Power BI

### Conectar Power BI em Synapse

**Power BI Desktop:**

1. **Get data → Azure Synapse Analytics**
2. **Server:** synapse-techstore-prod.sql.azuresynapse.net
3. **Database:** sqldw\_techstore
4. **Import mode**
5. **Selecionar tabelas:** dim\_cliente, dim\_produto, dim\_data, fato\_vendas

### Criar Medidas DAX

**DAX**

Total Vendas = SUM(fato\_vendas[valor\_total])

Total Quantidade = SUM(fato\_vendas[quantidade])

Ticket Médio = DIVIDE([Total Vendas], DISTINCTCOUNT(fato\_vendas[venda\_id]), 0)

Vendas Mês Anterior =

CALCULATE(

[Total Vendas],

DATEADD(dim\_data[data], -1, MONTH)

)

% Crescimento MoM =

DIVIDE(

[Total Vendas] - [Vendas Mês Anterior],

[Vendas Mês Anterior],

0

)

Margem Bruta =

SUMX(

fato\_vendas,

fato\_vendas[quantidade] \*

(RELATED(dim\_produto[preco\_venda]) - RELATED(dim\_produto[preco\_custo]))

)

### Páginas do Dashboard

**Página 1: Overview Executivo**

* Card: Total Vendas
* Card: Total Pedidos
* Card: Ticket Médio
* Card: % Crescimento MoM
* Line Chart: Evolução de Vendas (mensal)
* Donut Chart: Vendas por Categoria
* Map: Vendas por Estado

**Página 2: Análise de Produtos**

* Table: Top 10 Produtos por Vendas
* Bar Chart: Vendas por Categoria
* Scatter Chart: Preço vs Quantidade Vendida
* Slicer: Filtro por Categoria

**Página 3: Análise de Clientes**

* Table: Top 20 Clientes por Valor Gasto
* Bar Chart: Vendas por Estado
* Bar Chart: Vendas por Cidade (Top 10)
* Slicer: Filtro por Estado

**Publicar dashboard:**

**File → Publish → Select workspace → Publish**

**Configurar refresh automático:**

**Power BI Service → Dataset → Settings → Scheduled refresh:**

* Daily, 08:00 AM

## 15.8 Fase 7: Segurança

### Implementar RBAC

**Resource Group `rg-techstore-prod`:**

**Roles:**

* **Owner:** Data Engineer Lead
* **Contributor:** Data Engineers
* **Reader:** Business Analysts, Data Scientists

### Key Vault (Secrets)

**Secrets:**

* sql-admin-password
* storage-account-key
* databricks-token

**ADF/Synapse usam Managed Identity para acessar Key Vault.**

### Private Endpoints

**Recursos com Private Endpoint:**

* Storage Account
* SQL Database
* Synapse Workspace

**Desabilitar acesso público.**

## 15.9 Fase 8: Monitoramento

### Alertas Configurados

**1. ADF Pipeline Failed**

* Metric: PipelineFailedRuns > 0
* Action: Email + Slack webhook

**2. Synapse DTU > 80%**

* Metric: DTU percentage > 80
* Action: Email (escalar recursos)

**3. Data Quality Degradation**

* Custom metric: % de registros inválidos > 5%
* Action: Email

### Dashboard de Monitoramento

**Azure Portal → Dashboard:**

**Tiles:**

* ADF: Pipeline runs (últimas 24h)
* Databricks: Cluster utilization
* Synapse: Query performance
* Storage: Transactions
* Cost: Daily spend

## 15.10 Fase 9: CI/CD

### Git Repository Structure

techstore-data-platform/

infrastructure/

arm-templates/

storage.json

adf.json

databricks.json

synapse.json

adf/

pipeline/

dataset/

linkedService/

databricks/

notebooks/

bronze\_to\_silver.py

silver\_to\_gold.py

tests/

test\_transformations.py

synapse/

scripts/

create\_schema.sql

populate\_dim\_data.sql

pipelines/

docs/

architecture.md

runbooks/

data\_dictionary.md

### CI Pipeline (Azure DevOps)

**YAML**

# azure-pipelines-ci.yml

trigger:

branches:

include:

- main

- develop

stages:

- stage: Validate

jobs:

- job: ValidateJSON

steps:

- task: PowerShell@2

displayName: 'Validate ADF JSON'

inputs:

targetType: 'inline'

script: |

Get-ChildItem -Path adf -Recurse -Filter \*.json | ForEach-Object {

Get-Content $\_.FullName | ConvertFrom-Json | Out-Null

}

- job: RunTests

steps:

- task: UsePythonVersion@0

inputs:

versionSpec: '3.9'

- script: |

pip install pytest pyspark

pytest databricks/tests/

displayName: 'Run Unit Tests'

### CD Pipeline (Releases)

**Stages:**

**1. Deploy to Dev:**

* Deploy ARM templates (dev parameters)
* Deploy ADF pipelines
* Deploy Databricks notebooks
* Run smoke tests

**2. Deploy to Prod (manual approval):**

* Deploy ARM templates (prod parameters)
* Deploy ADF pipelines
* Deploy Databricks notebooks
* Health check pipeline

## 15.11 Fase 10: Documentação

### Arquitetura (Diagrams)

**Criar diagrama no draw.io / Lucidchart:**

* Data flow diagram (sources → bronze → silver → gold → BI)
* Network diagram (VNets, Private Endpoints)
* Security diagram (RBAC, Key Vault)

**Exportar como PNG/PDF → Incluir no README.md**

### Data Dictionary

**Documento:** docs/data\_dictionary.md

**Estrutura:**

**MARKDOWN**

# Data Dictionary

## Dimension: dim\_cliente

| Column | Type | Description | Source |

|--------|------|-------------|--------|

| cliente\_key | INT | Surrogate key | Auto-generated |

| cliente\_id | INT | Business key | SQL.clientes.cliente\_id |

| nome | VARCHAR(100) | Customer name | SQL.clientes.nome |

| email | VARCHAR(100) | Email address | SQL.clientes.email |

| cidade | VARCHAR(50) | City | SQL.clientes.cidade |

| estado | VARCHAR(2) | State code | SQL.clientes.estado |

| valid\_from | DATETIME2 | SCD start date | Derived |

| valid\_to | DATETIME2 | SCD end date | Derived |

| is\_current | BIT | Current record flag | Derived |

## Fact: fato\_vendas

| Column | Type | Description | Source |

|--------|------|-------------|--------|

| venda\_id | INT | Sale ID | SQL.vendas.venda\_id |

| cliente\_key | INT | Customer FK | Lookup dim\_cliente |

| produto\_key | INT | Product FK | Lookup dim\_produto |

| data\_key | INT | Date FK | Lookup dim\_data |

| quantidade | INT | Quantity sold | SQL.vendas.quantidade |

| valor\_total | DECIMAL(10,2) | Total amount | SQL.vendas.valor\_total |

| status | VARCHAR(20) | Sale status | SQL.vendas.status |

### Runbooks

**Documento:** docs/runbooks/pipeline\_failed.md

(Visto no Cap 13 - Incident Response)

## 15.12 Apresentar Projeto (Portfolio)

### GitHub Repository

**Criar repositório público:** techstore-azure-data-platform

**README.md:**

**MARKDOWN**

# TechStore - Azure Data Platform

\*\*Modern Data Platform\*\* built on Azure for e-commerce analytics.

## Architecture

![Architecture Diagram](docs/architecture.png)

\*\*Stack:\*\*

- Azure Data Factory (orchestration)

- Azure Databricks (processing)

- Azure Synapse Analytics (data warehouse)

- Power BI (visualization)

- Azure DevOps (CI/CD)

## Features

- Medallion Architecture (Bronze → Silver → Gold)

- Incremental ETL with watermarking

- SCD Type 2 (customer dimension)

- Data Quality checks

- Automated monitoring & alerts

- CI/CD pipelines

- Row-Level Security (RLS)

## Business Impact

- \*\*80% reduction\*\* in reporting time (from days to hours)

- \*\*Real-time visibility\*\* into sales metrics

- \*\*Automated data quality\*\* (99.5% accuracy)

## Technologies

Azure Data Factory | Azure Databricks | Synapse Analytics | Power BI | Python | Spark | SQL | Delta Lake | Git | Azure DevOps

## Screenshots

### Power BI Dashboard

![Dashboard](docs/screenshots/dashboard.png)

### Data Lineage

![Lineage](docs/screenshots/lineage.png)

## Documentation

- [Architecture](docs/architecture.md)

- [Data Dictionary](docs/data\_dictionary.md)

- [Runbooks](docs/runbooks/)

## ‍ Author

\*\*Ronaldo Santana Ramires\*\*

Data Engineer | Azure Certified

[LinkedIn](https://linkedin.com/in/ronaldo-ramires) | [Portfolio](https://ronaldo-ramires.dev)

### LinkedIn Post

Concluí meu projeto de Engenharia de Dados no Azure!

Construí uma plataforma de dados moderna (end-to-end) para e-commerce, incluindo:

Pipeline ETL completo (Bronze → Silver → Gold)

Arquitetura Medallion com Delta Lake

Data Warehouse dimensional (Star Schema)

Dashboards executivos no Power BI

CI/CD com Azure DevOps

Monitoramento e alertas automatizados

Impacto:

• 80% de redução no tempo de geração de relatórios

• Visibilidade em tempo real de vendas

• Data Quality automatizado (99.5% de acurácia)

Stack: Azure Data Factory, Databricks, Synapse Analytics, Power BI, Python, Spark, SQL

Código completo no GitHub: [link]

#DataEngineering #Azure #Databricks #PowerBI #BigData

## 15.13 Checklist de Entrega

**Infraestrutura:**

* [ ] Todos recursos provisionados (ARM Template)
* [ ] RBAC configurado
* [ ] Key Vault com secrets
* [ ] Private Endpoints habilitados

**Pipelines:**

* [ ] Bronze: Ingestão de todas fontes
* [ ] Silver: Transformações + Data Quality
* [ ] Gold: Data Warehouse (Star Schema)
* [ ] Pipelines orquestrados (triggers)

**Data Quality:**

* [ ] Validações implementadas
* [ ] Dead Letter Queue configurado
* [ ] Logging de qualidade

**Monitoramento:**

* [ ] Diagnostic Settings habilitados
* [ ] Alertas configurados (email + webhook)
* [ ] Dashboard de monitoramento

**CI/CD:**

* [ ] Git configurado (ADF, Databricks)
* [ ] CI Pipeline (validação)
* [ ] CD Pipeline (deploy Dev/Prod)

**Documentação:**

* [ ] README.md completo
* [ ] Diagrama de arquitetura
* [ ] Data Dictionary
* [ ] Runbooks

**BI:**

* [ ] Dashboard Power BI publicado
* [ ] Refresh agendado
* [ ] RLS implementado

## 15.14 Próximos Passos (Além do Junior)

**Volume 2 (Pleno):**

* Streaming em tempo real (Event Hubs, Kafka)
* Machine Learning pipelines (MLflow, AzureML)
* Data Mesh architecture
* Advanced optimizations (Spark tuning)

**Volume 3 (Senior):**

* Multi-cloud strategies (Azure + AWS/GCP)
* Data Governance avançado (Compliance, LGPD)
* Cost optimization strategies
* Team leadership & architecture

## Resumo do Capítulo

Neste capítulo você **integrou todos conhecimentos do livro** em projeto completo:

**Arquitetura Medallion:** Bronze → Silver → Gold

**Pipeline End-to-End:**

* Ingestão (SQL, CSV, API)
* Transformação (Databricks, Delta Lake)
* Data Warehouse (Synapse, Star Schema)
* Visualização (Power BI)

**Data Quality:** Validações, deduplicação, DLQ

**Segurança:** RBAC, Key Vault, Private Endpoints

**Monitoramento:** Alertas, dashboards, logs

**CI/CD:** Git, pipelines, ambientes (Dev/Prod)

**Documentação:** Arquitetura, Data Dictionary, Runbooks

**Portfolio:** GitHub, LinkedIn, apresentação profissional

## Parabéns!

**Você concluiu o Volume 1 (Junior) do livro Engenharia de Dados com Azure!**

Você agora domina:

* Azure Data Factory
* Azure Databricks
* Delta Lake
* Azure Synapse Analytics
* Power BI
* Segurança Azure
* Monitoramento
* CI/CD

## Agradecimentos

Obrigado por completar esta jornada! Continue praticando, construindo projetos e compartilhando conhecimento com a comunidade.

**Sucesso na sua carreira de Data Engineering!**

**Próximo passo:** Volume 2 - Pleno!