**AZURE DATA ENGINEERING**

Serie Junior

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Do Zero ao Primeiro Pipeline de Dados*

[Capítulo 1: Introdução à Engenharia de Dados 26](#_Toc221982927)

[Objetivos do Capítulo 26](#_Toc221982928)

[1.1 O Papel do Engenheiro de Dados Moderno 26](#_Toc221982929)

[O que é Engenharia de Dados? 26](#_Toc221982930)

[O que faz um Engenheiro de Dados no dia a dia? 26](#_Toc221982931)

[Evolução da Carreira 28](#_Toc221982932)

[1.2 Engenharia de Dados vs Outras Carreiras 28](#_Toc221982933)

[Comparação: DE vs Data Analyst vs Data Scientist 28](#_Toc221982934)

[Transições de Carreira 29](#_Toc221982935)

[1.3 Pipeline de Dados: Conceitos Fundamentais 30](#_Toc221982936)

[O que é um Pipeline de Dados? 30](#_Toc221982937)

[ETL vs ELT: Abordagens Fundamentais 30](#_Toc221982938)

[1.4 Arquiteturas de Dados: Data Warehouse, Data Lake, Lakehouse 31](#_Toc221982939)

[Data Warehouse (DW) - A Casa Organizada 32](#_Toc221982940)

[Data Lake - O Lago de Dados Brutos 32](#_Toc221982941)

[Lakehouse - O Melhor dos Dois Mundos 33](#_Toc221982942)

[Comparação Resumida 34](#_Toc221982943)

[1.5 Stack de Tecnologias Azure para Dados 35](#_Toc221982944)

[Armazenamento 35](#_Toc221982945)

[Processamento e Transformação 35](#_Toc221982946)

[Análise e Consumo 36](#_Toc221982947)

[Governança e Segurança 36](#_Toc221982948)

[Streaming e Real-Time 37](#_Toc221982949)

[Diagrama da Stack 37](#_Toc221982950)

[1.6 Roadmap de Carreira: Júnior → Pleno → Sênior 37](#_Toc221982951)

[Júnior (0-2 anos) - ESTE LIVRO! 37](#_Toc221982952)

[Pleno (2-4 anos) - Volume 2 deste livro 38](#_Toc221982953)

[Sênior (4-7 anos) - Volume 3 deste livro 38](#_Toc221982954)

[Staff / Principal (7+ anos) 39](#_Toc221982955)

[1.7 Hands-On: Criando sua Conta Azure Gratuita 39](#_Toc221982956)

[Por que Azure? 40](#_Toc221982957)

[Pré-requisitos 40](#_Toc221982958)

[Passo 1: Acesse a Página de Cadastro 40](#_Toc221982959)

[Passo 2: Criar Conta Microsoft (se não tiver) 41](#_Toc221982960)

[Passo 3: Preencher Informações Pessoais 41](#_Toc221982961)

[Passo 4: Validação de Identidade (Cartão) 42](#_Toc221982962)

[Passo 5: Concordar com Termos 42](#_Toc221982963)

[Passo 6: Bem-vindo ao Azure Portal! 42](#_Toc221982964)

[1.8 Tour Guiado pelo Azure Portal 44](#_Toc221982965)

[Anatomia do Portal 44](#_Toc221982966)

[Ação opcional: Idioma 45](#_Toc221982967)

[Navegação: 3 Formas de Encontrar Serviços 45](#_Toc221982968)

[Conceitos Importantes: Subscription e Resource Group 46](#_Toc221982969)

[Hands-On: Criar seu Primeiro Resource Group 47](#_Toc221982970)

[Explorando Serviços de Dados 49](#_Toc221982971)

[1.9 Mapa Mental da Stack Azure 50](#_Toc221982972)

[1.10 Checklist de Habilidades 50](#_Toc221982973)

[1.11 Próximos Passos 51](#_Toc221982974)

[1.12 Recursos Complementares 51](#_Toc221982975)

[1.13 Exercícios Práticos (Opcional) 52](#_Toc221982976)

[Resumo do Capítulo 53](#_Toc221982977)

[Capítulo 2: Azure Storage e Data Lake Gen2 54](#_Toc221982978)

[Objetivos do Capítulo 54](#_Toc221982979)

[2.1 Azure Storage: Fundamentos 54](#_Toc221982980)

[O que é Azure Storage? 54](#_Toc221982981)

[Tipos de Storage no Azure 54](#_Toc221982982)

[2.2 Blob Storage vs Data Lake Gen2 55](#_Toc221982983)

[Blob Storage (Armazenamento de Blobs) 55](#_Toc221982984)

[Data Lake Storage Gen2 (ADLS Gen2) 56](#_Toc221982985)

[2.3 Arquitetura Medallion: Bronze, Silver, Gold 57](#_Toc221982986)

[Conceito das Três Camadas 57](#_Toc221982987)

[Fluxo Completo Medallion 59](#_Toc221982988)

[Vantagens da Arquitetura Medallion 59](#_Toc221982989)

[2.4 Hands-On: Criar Storage Account com Data Lake Gen2 59](#_Toc221982990)

[Passo 1: Acessar Criação de Storage Account 60](#_Toc221982991)

[Passo 2: Configurar Basics 60](#_Toc221982992)

[Passo 3: Habilitar Data Lake Gen2 61](#_Toc221982993)

[Passo 4: Networking (manter padrão) 62](#_Toc221982994)

[Passo 5: Data Protection (opcional para dev) 62](#_Toc221982995)

[Passo 6: Encryption (manter padrão) 63](#_Toc221982996)

[Passo 7: Tags (opcional) 63](#_Toc221982997)

[Passo 8: Revisar e Criar 63](#_Toc221982998)

[Passo 9: Explorar a Storage Account 64](#_Toc221982999)

[2.5 Criar Estrutura Medallion 64](#_Toc221983000)

[Passo 1: Criar Container (Filesystem) 64](#_Toc221983001)

[Passo 2: Criar Pastas Bronze, Silver, Gold 66](#_Toc221983002)

[Passo 3: Criar Subpastas (exemplo: vendas) 67](#_Toc221983003)

[2.6 Upload de Arquivos 68](#_Toc221983004)

[Preparar Arquivo de Exemplo 68](#_Toc221983005)

[Fazer Upload via Portal 69](#_Toc221983006)

[Visualizar Conteúdo do Arquivo 70](#_Toc221983007)

[2.7 Azure Storage Explorer (Desktop App) 70](#_Toc221983008)

[Instalar Azure Storage Explorer 70](#_Toc221983009)

[Conectar ao Azure 70](#_Toc221983010)

[Navegar no Data Lake 71](#_Toc221983011)

[Operações Úteis no Storage Explorer 72](#_Toc221983012)

[2.8 Formatos de Arquivo: CSV, JSON, Parquet 73](#_Toc221983013)

[CSV (Comma-Separated Values) 73](#_Toc221983014)

[JSON (JavaScript Object Notation) 73](#_Toc221983015)

[Parquet (Formato Colunar) 74](#_Toc221983016)

[Delta Lake (Parquet + ACID) 76](#_Toc221983017)

[Resumo de Formatos 76](#_Toc221983018)

[2.9 Segurança e Permissões 77](#_Toc221983019)

[Tipos de Autenticação 77](#_Toc221983020)

[RBAC: Roles Importantes 77](#_Toc221983021)

[Hands-On: Configurar RBAC (Permissão de Leitura) 78](#_Toc221983022)

[ACLs: Permissões Granulares 80](#_Toc221983023)

[2.10 Boas Práticas de Organização 80](#_Toc221983024)

[Convenções de Nomenclatura 80](#_Toc221983025)

[Governança de Dados 81](#_Toc221983026)

[2.11 Gerenciamento de Custos 82](#_Toc221983027)

[Precificação do Azure Storage 82](#_Toc221983028)

[Otimizar Custos 82](#_Toc221983029)

[2.12 Checklist de Habilidades 83](#_Toc221983030)

[2.13 Próximos Passos 84](#_Toc221983031)

[2.14 Recursos Complementares 84](#_Toc221983032)

[2.15 Exercícios Práticos 85](#_Toc221983033)

[Resumo do Capítulo 87](#_Toc221983034)

[Capítulo 3: Azure SQL Database - Fundamentos 88](#_Toc221983035)

[Objetivos do Capítulo 88](#_Toc221983036)

[3.1 Bancos Relacionais em Engenharia de Dados 88](#_Toc221983037)

[Por que ainda usar SQL Database? 88](#_Toc221983038)

[3.2 Azure SQL Database: Visão Geral 89](#_Toc221983039)

[O que é Azure SQL Database? 89](#_Toc221983040)

[Tiers de Serviço (Pricing) 90](#_Toc221983041)

[3.3 Hands-On: Criar Azure SQL Database 91](#_Toc221983042)

[Passo 1: Iniciar Criação 91](#_Toc221983043)

[Passo 2: Configurar Basics 92](#_Toc221983044)

[Passo 3: Escolher Tier Serverless 93](#_Toc221983045)

[Passo 4: Backup e Redundancy 94](#_Toc221983046)

[Passo 5: Networking (permitir acesso) 95](#_Toc221983047)

[Passo 6: Security (manter padrão) 96](#_Toc221983048)

[Passo 7: Additional Settings 96](#_Toc221983049)

[Passo 8: Revisar e Criar 97](#_Toc221983050)

[Passo 9: Explorar o Database 98](#_Toc221983051)

[3.4 Conectar ao Database: Azure Data Studio 99](#_Toc221983052)

[Instalar Azure Data Studio 100](#_Toc221983053)

[Conectar ao Database 101](#_Toc221983054)

[No caso do VSCode: 103](#_Toc221983055)

[Explorar o Schema AdventureWorksLT 103](#_Toc221983056)

[Primeira Query: Explorar Dados 104](#_Toc221983057)

[3.5 Modelagem Dimensional: Star Schema 105](#_Toc221983058)

[O que é Star Schema? 106](#_Toc221983059)

[Fato vs Dimensão 106](#_Toc221983060)

[Exemplo: E-commerce 107](#_Toc221983061)

[Por que Star Schema? 108](#_Toc221983062)

[Snowflake Schema (alternativa) 108](#_Toc221983063)

[3.6 SQL Essencial para Engenharia de Dados 109](#_Toc221983064)

[SELECT Básico 112](#_Toc221983065)

[JOINs (Unir Tabelas) 112](#_Toc221983066)

[Agregações (GROUP BY) 113](#_Toc221983067)

[HAVING (Filtrar Agregações) 113](#_Toc221983068)

[CTEs (Common Table Expressions) 114](#_Toc221983069)

[Window Functions (Funções de Janela) 114](#_Toc221983070)

[CASE WHEN (Lógica Condicional) 115](#_Toc221983071)

[Subqueries 115](#_Toc221983072)

[3.7 Criar Tabelas Customizadas 116](#_Toc221983073)

[Criar Dimensão Cliente 116](#_Toc221983074)

[Criar Dimensão Produto 116](#_Toc221983075)

[Criar Dimensão Tempo 117](#_Toc221983076)

[Criar Fato Vendas 117](#_Toc221983077)

[Inserir Dados de Exemplo 118](#_Toc221983078)

[Query Analítica: Vendas por Cliente 120](#_Toc221983079)

[3.8 Backup e Restore 120](#_Toc221983080)

[Backups Automáticos 120](#_Toc221983081)

[Restore Manual (Point-in-Time) 121](#_Toc221983082)

[Exportar Database (BACPAC) 122](#_Toc221983083)

[3.9 Monitoramento e Performance 122](#_Toc221983084)

[Query Performance Insight 122](#_Toc221983085)

[Índices (Básico) 123](#_Toc221983086)

[Estatísticas 124](#_Toc221983087)

[3.10 Checklist de Habilidades 124](#_Toc221983088)

[3.11 Próximos Passos 125](#_Toc221983089)

[3.12 Recursos Complementares 125](#_Toc221983090)

[3.13 Exercícios Práticos 126](#_Toc221983091)

[Resumo do Capítulo 127](#_Toc221983092)

[Capítulo 4: Azure Data Factory - Primeiros Passos 128](#_Toc221983093)

[Objetivos do Capítulo 128](#_Toc221983094)

[4.1 O que é Azure Data Factory? 128](#_Toc221983095)

[O que ADF faz? 129](#_Toc221983096)

[ADF vs Outras Ferramentas 130](#_Toc221983097)

[4.2 Arquitetura e Componentes do ADF 130](#_Toc221983098)

[Componentes Principais 130](#_Toc221983099)

[Fluxo de Criação de Pipeline 132](#_Toc221983100)

[4.3 Hands-On: Criar Azure Data Factory 132](#_Toc221983101)

[Passo 1: Iniciar Criação 132](#_Toc221983102)

[Passo 2: Configurar Basics 133](#_Toc221983103)

[Passo 3: Git Configuration (Opcional) 134](#_Toc221983104)

[Passo 4: Networking (Padrão) 134](#_Toc221983105)

[Passo 5: Review + Create 134](#_Toc221983106)

[Passo 6: Abrir Data Factory Studio 135](#_Toc221983107)

[4.4 Criar Linked Services (Conexões) 136](#_Toc221983108)

[Linked Service 1: Azure SQL Database 136](#_Toc221983109)

[Linked Service 2: Azure Data Lake Gen2 139](#_Toc221983110)

[4.5 Criar Datasets 141](#_Toc221983111)

[Dataset 1: Tabela SQL (Source) 142](#_Toc221983112)

[Dataset 2: Arquivo Parquet no Data Lake (Sink/Destino) 143](#_Toc221983113)

[4.6 Criar Pipeline com Copy Activity 145](#_Toc221983114)

[Passo 1: Criar Pipeline 145](#_Toc221983115)

[Passo 2: Adicionar Copy Activity 146](#_Toc221983116)

[Passo 3: Configurar Source (Origem) 147](#_Toc221983117)

[Passo 4: Configurar Sink (Destino) 147](#_Toc221983118)

[Passo 5: Configurar Mapping (Opcional) 148](#_Toc221983119)

[Passo 6: Configurar Settings 148](#_Toc221983120)

[Passo 7: Salvar e Validar 149](#_Toc221983121)

[4.7 Debug e Executar Pipeline 149](#_Toc221983122)

[Debug Manual 149](#_Toc221983123)

[Verificar Dados no Data Lake 150](#_Toc221983124)

[4.8 Monitoramento de Pipelines 151](#_Toc221983125)

[Monitor Page 151](#_Toc221983126)

[Filtros e Pesquisa 152](#_Toc221983127)

[Alertas (Opcional) 152](#_Toc221983128)

[4.9 Triggers: Agendar Pipelines 152](#_Toc221983129)

[Criar Schedule Trigger 152](#_Toc221983130)

[Gerenciar Triggers 154](#_Toc221983131)

[Outros Tipos de Trigger 155](#_Toc221983132)

[4.10 Parametrização de Pipelines 155](#_Toc221983133)

[Adicionar Parâmetros ao Pipeline 155](#_Toc221983134)

[Usar Parâmetro em Dataset 156](#_Toc221983135)

[Passar Parâmetro no Pipeline 157](#_Toc221983136)

[Testar com Debug 157](#_Toc221983137)

[4.11 Loops e Condicionais 157](#_Toc221983138)

[ForEach Activity 157](#_Toc221983139)

[If Condition Activity 158](#_Toc221983140)

[4.12 Integração com Databricks 158](#_Toc221983141)

[4.13 Checklist de Habilidades 159](#_Toc221983142)

[4.14 Próximos Passos 160](#_Toc221983143)

[4.15 Recursos Complementares 160](#_Toc221983144)

[4.16 Exercícios Práticos 161](#_Toc221983145)

[Resumo do Capítulo 162](#_Toc221983146)

[Capítulo 5: Data Flows no Azure Data Factory 163](#_Toc221983147)

[Objetivos do Capítulo 163](#_Toc221983148)

[5.1 O que são Data Flows? 163](#_Toc221983149)

[Copy Activity vs Data Flow 163](#_Toc221983150)

[Quando usar Data Flow? 164](#_Toc221983151)

[Arquitetura de Data Flow 164](#_Toc221983152)

[5.2 Hands-On: Criar Primeiro Data Flow 165](#_Toc221983153)

[Cenário 165](#_Toc221983154)

[Passo 1: Criar Data Flow 165](#_Toc221983155)

[Passo 2: Configurar Source 166](#_Toc221983156)

[Passo 3: Adicionar Filter (Filtrar Linhas) 168](#_Toc221983157)

[Passo 4: Adicionar Derived Column (Coluna Calculada) 170](#_Toc221983158)

[Passo 5: Adicionar Select (Selecionar Colunas) 171](#_Toc221983159)

[Passo 6: Adicionar Sink (Destino) 172](#_Toc221983160)

[Passo 7: Validar e Salvar 173](#_Toc221983161)

[5.3 Executar Data Flow em Pipeline 173](#_Toc221983162)

[Criar Pipeline com Data Flow Activity 173](#_Toc221983163)

[Executar Pipeline 175](#_Toc221983164)

[Verificar Dados no Data Lake 175](#_Toc221983165)

[5.4 Transformações Avançadas 176](#_Toc221983166)

[Join (Combinar Datasets) 176](#_Toc221983167)

[Aggregate (Agrupar e Sumarizar) 177](#_Toc221983168)

[Lookup (Enriquecer Dados) 179](#_Toc221983169)

[Conditional Split (Dividir Fluxo) 179](#_Toc221983170)

[Window (Funções de Janela) 180](#_Toc221983171)

[5.5 Otimização de Data Flows 180](#_Toc221983172)

[Partitioning (Paralelização) 181](#_Toc221983173)

[Broadcast (Otimização de Join) 182](#_Toc221983174)

[Caching (Reutilizar Datasets) 183](#_Toc221983175)

[Cluster Sizing (Dimensionar Cluster) 183](#_Toc221983176)

[Time to Live (TTL) 183](#_Toc221983177)

[5.6 Monitoramento de Data Flows 184](#_Toc221983178)

[Pipeline Monitor 184](#_Toc221983179)

[Performance Tuning 185](#_Toc221983180)

[5.7 Implementar SCD Type 1 185](#_Toc221983181)

[Fluxo SCD Type 1 186](#_Toc221983182)

[Hands-On SCD Type 1 186](#_Toc221983183)

[5.8 Checklist de Habilidades 187](#_Toc221983184)

[5.9 Próximos Passos 187](#_Toc221983185)

[5.10 Recursos Complementares 188](#_Toc221983186)

[5.11 Exercícios Práticos 188](#_Toc221983187)

[Resumo do Capítulo 189](#_Toc221983188)

[Capítulo 6: Azure Databricks e PySpark - Fundamentos 191](#_Toc221983189)

[Objetivos do Capítulo 191](#_Toc221983190)

[6.1 O que é Azure Databricks? 191](#_Toc221983191)

[Componentes Principais 191](#_Toc221983192)

[Databricks vs Outras Ferramentas 192](#_Toc221983193)

[Quando usar Databricks? 192](#_Toc221983194)

[6.2 Arquitetura do Apache Spark 193](#_Toc221983195)

[Conceitos Fundamentais 193](#_Toc221983196)

[Fluxo de Execução Spark 193](#_Toc221983197)

[6.3 Hands-On: Provisionar Azure Databricks 194](#_Toc221983198)

[Passo 1: Criar Workspace Databricks 194](#_Toc221983199)

[Passo 2: Abrir Databricks Workspace 196](#_Toc221983200)

[6.4 Criar Cluster Spark 197](#_Toc221983201)

[Passo 1: Configurar Cluster 197](#_Toc221983202)

[6.5 Criar Primeiro Notebook 199](#_Toc221983203)

[Passo 1: Criar Notebook 199](#_Toc221983204)

[Passo 2: Primeira Execução 200](#_Toc221983205)

[Estrutura de Notebooks 200](#_Toc221983206)

[6.6 PySpark Fundamentos 202](#_Toc221983207)

[Criar DataFrame Manualmente 202](#_Toc221983208)

[Ler Dados do Data Lake 202](#_Toc221983209)

[Transformações Básicas 204](#_Toc221983210)

[Join (combinar DataFrames) 205](#_Toc221983211)

[GroupBy (agrupar) 206](#_Toc221983212)

[6.7 Spark SQL 206](#_Toc221983213)

[Registrar DataFrame como Tabela Temporária 207](#_Toc221983214)

[Query SQL Complexa 207](#_Toc221983215)

[Magic Command %sql 207](#_Toc221983216)

[6.8 Gravar Dados no Data Lake 208](#_Toc221983217)

[Escrever Parquet 208](#_Toc221983218)

[Escrever Delta Lake 208](#_Toc221983219)

[Capítulo 7: Azure Synapse Analytics - Fundamentos 210](#_Toc221983220)

[Objetivos do Capítulo 210](#_Toc221983221)

[7.1 O que é Azure Synapse Analytics? 210](#_Toc221983222)

[Visão Geral 210](#_Toc221983223)

[O Problema que o Synapse Resolve 210](#_Toc221983224)

[Componentes Principais 211](#_Toc221983225)

[7.2 Criando seu Primeiro Workspace Synapse 211](#_Toc221983226)

[Pré-requisitos 211](#_Toc221983227)

[Passo 1: Acessar o Portal Azure 211](#_Toc221983228)

[Passo 2: Criar Workspace 212](#_Toc221983229)

[Passo 3: Revisar e Criar 212](#_Toc221983230)

[Passo 4: Abrir o Synapse Studio 213](#_Toc221983231)

[7.3 SQL Pool Serverless - Queries sob Demanda 213](#_Toc221983232)

[O que é SQL Pool Serverless? 213](#_Toc221983233)

[Sua Primeira Query no Data Lake 213](#_Toc221983234)

[Consultando Parquet 214](#_Toc221983235)

[Consultando Delta Lake 214](#_Toc221983236)

[7.4 Criando Views e External Tables 214](#_Toc221983237)

[Views sobre o Data Lake 214](#_Toc221983238)

[External Tables 215](#_Toc221983239)

[7.5 SQL Pool Dedicado (Data Warehouse) 216](#_Toc221983240)

[Quando Usar SQL Pool Dedicado? 216](#_Toc221983241)

[Arquitetura MPP 216](#_Toc221983242)

[Criando um SQL Pool Dedicado 217](#_Toc221983243)

[Criando Tabelas no Pool Dedicado 218](#_Toc221983244)

[7.6 Apache Spark Pool 218](#_Toc221983245)

[O que é Spark Pool? 218](#_Toc221983246)

[Criando um Spark Pool 219](#_Toc221983247)

[Seu Primeiro Notebook Spark 219](#_Toc221983248)

[7.7 Pipelines de Integração 220](#_Toc221983249)

[Synapse Pipelines vs Azure Data Factory 220](#_Toc221983250)

[Criando um Pipeline Simples 220](#_Toc221983251)

[7.8 Monitoramento e Custos 221](#_Toc221983252)

[Monitor Hub 221](#_Toc221983253)

[Gerenciando Custos 221](#_Toc221983254)

[7.9 Hands-On: Projeto Prático 222](#_Toc221983255)

[Cenário 222](#_Toc221983256)

[Passo 1: Upload dos Dados 222](#_Toc221983257)

[Passo 2: Query Exploratória (Serverless) 222](#_Toc221983258)

[Passo 3: Transformação com Spark 222](#_Toc221983259)

[Passo 4: Criar Views Analíticas 223](#_Toc221983260)

[7.10 Checklist de Habilidades 224](#_Toc221983261)

[7.11 Resumo 224](#_Toc221983262)

[Próximos Passos 225](#_Toc221983263)

[Capítulo 8: Delta Lake - Fundamentos 226](#_Toc221983264)

[Objetivos do Capítulo 226](#_Toc221983265)

[8.1 O que é Delta Lake? 226](#_Toc221983266)

[Problema: Data Lakes Tradicionais 226](#_Toc221983267)

[Solução: Delta Lake 226](#_Toc221983268)

[Arquitetura Delta Lake 227](#_Toc221983269)

[8.2 Criar Tabela Delta Lake 227](#_Toc221983270)

[Opção 1: DataFrame API 227](#_Toc221983271)

[Opção 2: SQL 228](#_Toc221983272)

[Opção 3: CREATE TABLE (managed table) 228](#_Toc221983273)

[8.3 Operações ACID 229](#_Toc221983274)

[INSERT 229](#_Toc221983275)

[UPDATE 229](#_Toc221983276)

[DELETE 230](#_Toc221983277)

[MERGE (UPSERT) 230](#_Toc221983278)

[8.4 Time Travel 231](#_Toc221983279)

[Consultar Versão Específica 231](#_Toc221983280)

[Visualizar Histórico de Transações 232](#_Toc221983281)

[Restaurar Versão Anterior 233](#_Toc221983282)

[8.5 Slowly Changing Dimensions (SCD Type 2) 233](#_Toc221983283)

[Schema SCD Type 2 233](#_Toc221983284)

[Implementação SCD Type 2 com MERGE 233](#_Toc221983285)

[8.6 Schema Evolution 234](#_Toc221983286)

[Adicionar Coluna 235](#_Toc221983287)

[Schema Enforcement 235](#_Toc221983288)

[8.7 Otimizações Delta Lake 235](#_Toc221983289)

[OPTIMIZE (File Compaction) 235](#_Toc221983290)

[Z-ORDERING 236](#_Toc221983291)

[VACUUM (Limpeza de Arquivos) 236](#_Toc221983292)

[8.8 Monitoramento e Debugging 236](#_Toc221983293)

[Estatísticas da Tabela 237](#_Toc221983294)

[Histórico de Operações 237](#_Toc221983295)

[Metadados de Arquivos 237](#_Toc221983296)

[8.9 Integração Delta Lake 237](#_Toc221983297)

[Com Azure Data Factory 237](#_Toc221983298)

[Com Power BI 237](#_Toc221983299)

[Com Synapse Analytics 238](#_Toc221983300)

[8.10 Checklist de Habilidades 238](#_Toc221983301)

[8.11 Próximos Passos 239](#_Toc221983302)

[8.12 Recursos Complementares 239](#_Toc221983303)

[8.13 Exercícios Práticos 239](#_Toc221983304)

[Resumo do Capítulo 240](#_Toc221983305)

[Capítulo 9: Streaming e Orquestração com Delta Lake 241](#_Toc221983306)

[Objetivos do Capítulo 241](#_Toc221983307)

[9.1 O que é Structured Streaming? 241](#_Toc221983308)

[Streaming vs Batch 241](#_Toc221983309)

[Como Funciona 241](#_Toc221983310)

[9.2 Fontes de Streaming Suportadas 242](#_Toc221983311)

[9.3 Hands-On: Streaming Básico (File Source) 242](#_Toc221983312)

[Passo 1: Configurar Source (Auto Loader) 243](#_Toc221983313)

[Passo 2: Transformações (Mesmo que Batch) 243](#_Toc221983314)

[Passo 3: Configurar Sink (Delta Lake) 244](#_Toc221983315)

[Passo 4: Simular Dados Chegando 244](#_Toc221983316)

[Passo 5: Monitorar Query 245](#_Toc221983317)

[9.4 Delta Lake como Fonte de Streaming 245](#_Toc221983318)

[Use Case: Processar Mudanças em Tempo Real 245](#_Toc221983319)

[Filtrar por Versão (Incremental) 246](#_Toc221983320)

[9.5 Triggers (Modos de Execução) 246](#_Toc221983321)

[Default (Microbatch Contínuo) 246](#_Toc221983322)

[Fixed Interval (Intervalo Fixo) 247](#_Toc221983323)

[Once (Uma Vez) 247](#_Toc221983324)

[Available Now (Processar Tudo Disponível) 247](#_Toc221983325)

[9.6 Agregações em Streaming 248](#_Toc221983326)

[Agregações Simples (Append Mode) 248](#_Toc221983327)

[Window Aggregations (Janelas de Tempo) 248](#_Toc221983328)

[9.7 Checkpointing e Tolerância a Falhas 249](#_Toc221983329)

[Como Funciona 249](#_Toc221983330)

[Recovery Após Falha 249](#_Toc221983331)

[9.8 Monitoramento de Streaming 250](#_Toc221983332)

[Query Progress 250](#_Toc221983333)

[Monitorar Todas Queries Ativas 251](#_Toc221983334)

[9.9 Integração com Azure Data Factory 251](#_Toc221983335)

[Opção 1: Notebook com Trigger Once 251](#_Toc221983336)

[Opção 2: Streaming Contínuo Gerenciado 252](#_Toc221983337)

[9.10 Batch + Streaming na Mesma Tabela Delta 252](#_Toc221983338)

[9.11 Otimizações de Streaming 252](#_Toc221983339)

[Auto Loader vs readStream (File Source) 252](#_Toc221983340)

[Particionamento de Sink 253](#_Toc221983341)

[Tuning de Microbatch Size 253](#_Toc221983342)

[9.12 Change Data Capture (CDC) Completo 253](#_Toc221983343)

[Cenário: Replicar SQL Database → Delta Lake 253](#_Toc221983344)

[9.13 Checklist de Habilidades 255](#_Toc221983345)

[9.14 Próximos Passos 255](#_Toc221983346)

[9.15 Recursos Complementares 256](#_Toc221983347)

[9.16 Exercícios Práticos 256](#_Toc221983348)

[Resumo do Capítulo 257](#_Toc221983349)

[Volume 1: JUNIOR 259](#_Toc221983350)

[Capítulo 10: Padrões de ETL e Data Quality 259](#_Toc221983351)

[Objetivos do Capítulo 259](#_Toc221983352)

[10.1 Incremental Load (Carga Incremental) 259](#_Toc221983353)

[Estratégias de Incremental Load 259](#_Toc221983354)

[10.2 Deduplication (Remover Duplicatas) 261](#_Toc221983355)

[Estratégia 1: dropDuplicates (Simples) 261](#_Toc221983356)

[Estratégia 2: Window Function (Manter Mais Recente) 261](#_Toc221983357)

[Estratégia 3: Deduplicate ao Gravar (Delta Lake) 262](#_Toc221983358)

[10.3 Data Quality Checks 262](#_Toc221983359)

[Framework de Validações 262](#_Toc221983360)

[Salvar Resultados de Quality 264](#_Toc221983361)

[Fail Pipeline se Checks Falham 264](#_Toc221983362)

[10.4 Error Handling (Tratamento de Erros) 264](#_Toc221983363)

[Try-Except em PySpark 264](#_Toc221983364)

[Dead Letter Queue (DLQ) 265](#_Toc221983365)

[10.5 Idempotência 266](#_Toc221983366)

[Problema: Append Não É Idempotente 266](#_Toc221983367)

[Solução 1: Overwrite 266](#_Toc221983368)

[Solução 2: Merge com Deduplicação 266](#_Toc221983369)

[Solução 3: Run ID / Execution Tracking 267](#_Toc221983370)

[10.6 Logging e Auditoria 267](#_Toc221983371)

[Framework de Logging 268](#_Toc221983372)

[Auditoria de Mudanças (Delta History) 269](#_Toc221983373)

[10.7 Monitoramento e Alertas 269](#_Toc221983374)

[Métricas de Pipeline 269](#_Toc221983375)

[Alertas (Integração com ADF) 270](#_Toc221983376)

[10.8 Best Practices de ETL 271](#_Toc221983377)

[1. Separar Camadas (Bronze/Silver/Gold) 271](#_Toc221983378)

[2. Particionamento Inteligente 271](#_Toc221983379)

[3. Caching Estratégico 271](#_Toc221983380)

[4. Broadcast Joins 272](#_Toc221983381)

[5. Schema Enforcement 272](#_Toc221983382)

[6. Optimize e Vacuum Regularmente 272](#_Toc221983383)

[10.9 Checklist de Habilidades 272](#_Toc221983384)

[10.10 Projeto Final: Pipeline Completo 273](#_Toc221983385)

[10.11 Recursos Complementares 274](#_Toc221983386)

[Resumo do Capítulo 274](#_Toc221983387)

[Parabéns! Você completou a Parte 2! 274](#_Toc221983388)

[Volume 1: JUNIOR 276](#_Toc221983389)

[Capítulo 11: Power BI e Visualização de Dados 276](#_Toc221983390)

[Objetivos do Capítulo 276](#_Toc221983391)

[11.1 O que é Power BI? 276](#_Toc221983392)

[Componentes Principais 276](#_Toc221983393)

[Power BI vs Outras Ferramentas 277](#_Toc221983394)

[11.2 Instalar Power BI Desktop 277](#_Toc221983395)

[Download 277](#_Toc221983396)

[11.3 Conectar em Azure SQL Database 277](#_Toc221983397)

[Passo 1: Get Data 277](#_Toc221983398)

[Passo 2: Configurar Conexão 278](#_Toc221983399)

[Passo 3: Autenticação 278](#_Toc221983400)

[Passo 4: Selecionar Tabelas 278](#_Toc221983401)

[11.4 Modelagem de Dados (Star Schema) 279](#_Toc221983402)

[Relacionamentos 279](#_Toc221983403)

[Criar Relacionamento Manual 279](#_Toc221983404)

[Tipos de Relacionamentos 280](#_Toc221983405)

[11.5 Power Query (Transformações) 280](#_Toc221983406)

[Abrir Power Query Editor 280](#_Toc221983407)

[Transformações Comuns 280](#_Toc221983408)

[Aplicar e Carregar 281](#_Toc221983409)

[11.6 Criar Visualizações 281](#_Toc221983410)

[Passo 1: Criar Página de Relatório 281](#_Toc221983411)

[Passo 2: Card (Métrica Simples) 281](#_Toc221983412)

[Passo 3: Bar Chart (Gráfico de Barras) 281](#_Toc221983413)

[Passo 4: Line Chart (Evolução Temporal) 282](#_Toc221983414)

[Passo 5: Table (Tabela Detalhada) 282](#_Toc221983415)

[Passo 6: Slicer (Filtro Interativo) 283](#_Toc221983416)

[11.7 DAX (Data Analysis Expressions) 283](#_Toc221983417)

[Criar Medida Calculada 283](#_Toc221983418)

[DAX Básico - Funções Comuns 283](#_Toc221983419)

[Exemplo: Total de Vendas do Ano Anterior 284](#_Toc221983420)

[Exemplo: % de Crescimento 284](#_Toc221983421)

[11.8 Row-Level Security (RLS) 284](#_Toc221983422)

[Cenário 284](#_Toc221983423)

[Passo 1: Criar Role 284](#_Toc221983424)

[Passo 2: Testar RLS 285](#_Toc221983425)

[Passo 3: Atribuir Usuários (no Power BI Service) 285](#_Toc221983426)

[11.9 Publicar Relatório 285](#_Toc221983427)

[Passo 1: Salvar Arquivo 285](#_Toc221983428)

[Passo 2: Publicar no Power BI Service 285](#_Toc221983429)

[Passo 3: Visualizar no Power BI Service 286](#_Toc221983430)

[11.10 Agendar Refresh de Dados 286](#_Toc221983431)

[Configurar Refresh Automático 286](#_Toc221983432)

[Passo 1: Configurar Credenciais 286](#_Toc221983433)

[Passo 2: Configurar Schedule 286](#_Toc221983434)

[Refresh Manual 287](#_Toc221983435)

[11.11 Conectar em Data Lake (Delta Lake) 287](#_Toc221983436)

[Passo 1: Habilitar SQL Endpoint no Synapse 287](#_Toc221983437)

[Passo 2: Conectar no Power BI 287](#_Toc221983438)

[11.12 Otimização de Performance 287](#_Toc221983439)

[1. Usar Import ao Invés de DirectQuery 287](#_Toc221983440)

[2. Reduzir Colunas/Linhas 288](#_Toc221983441)

[3. Otimizar Relacionamentos 288](#_Toc221983442)

[4. Agregações 288](#_Toc221983443)

[11.13 Integração com Pipelines de Dados 288](#_Toc221983444)

[Fluxo Completo 288](#_Toc221983445)

[11.14 Checklist de Habilidades 289](#_Toc221983446)

[11.15 Recursos Complementares 289](#_Toc221983447)

[11.16 Exercícios Práticos 290](#_Toc221983448)

[Resumo do Capítulo 290](#_Toc221983449)

[Volume 1: JUNIOR 292](#_Toc221983450)

[Capítulo 12: Azure Synapse Analytics - Fundamentos 292](#_Toc221983451)

[Objetivos do Capítulo 292](#_Toc221983452)

[12.1 O que é Azure Synapse Analytics? 292](#_Toc221983453)

[Synapse vs Outras Ferramentas 292](#_Toc221983454)

[Componentes do Synapse 293](#_Toc221983455)

[12.2 Provisionar Synapse Workspace 293](#_Toc221983456)

[Passo 1: Criar Workspace 293](#_Toc221983457)

[Passo 2: Abrir Synapse Studio 294](#_Toc221983458)

[12.3 Serverless SQL Pool (Query em Data Lake) 294](#_Toc221983459)

[Vantagens 294](#_Toc221983460)

[Use Cases 295](#_Toc221983461)

[Passo 1: Query em Arquivo Parquet 295](#_Toc221983462)

[Passo 2: Criar External Table 295](#_Toc221983463)

[Passo 3: Query em Delta Lake 296](#_Toc221983464)

[12.4 Dedicated SQL Pool (Data Warehouse) 296](#_Toc221983465)

[Vantagens 297](#_Toc221983466)

[Quando Usar 297](#_Toc221983467)

[Passo 1: Criar Dedicated SQL Pool 297](#_Toc221983468)

[Passo 2: Pausar/Resumir Pool 297](#_Toc221983469)

[Passo 3: Criar Tabela 298](#_Toc221983470)

[Entender Distribuições 299](#_Toc221983471)

[12.5 Carregar Dados no Dedicated Pool 299](#_Toc221983472)

[Método 1: COPY Statement (Recomendado) 299](#_Toc221983473)

[Método 2: Synapse Pipeline (Orquestração) 300](#_Toc221983474)

[Método 3: PolyBase (Legado) 300](#_Toc221983475)

[12.6 Otimizações 301](#_Toc221983476)

[1. Columnstore Index 301](#_Toc221983477)

[2. Estatísticas 301](#_Toc221983478)

[3. Particionamento 301](#_Toc221983479)

[4. Workload Management 302](#_Toc221983480)

[12.7 Queries Analíticas 302](#_Toc221983481)

[Exemplo: Total de Vendas por Produto 302](#_Toc221983482)

[Exemplo: Vendas Mensais (Evolução) 302](#_Toc221983483)

[Exemplo: Top 10 Clientes 303](#_Toc221983484)

[12.8 Integração com Power BI 303](#_Toc221983485)

[Conectar Power BI em Synapse 303](#_Toc221983486)

[12.9 Serverless vs Dedicated - Quando Usar? 303](#_Toc221983487)

[12.10 Checklist de Habilidades 304](#_Toc221983488)

[12.11 Recursos Complementares 304](#_Toc221983489)

[12.12 Exercícios Práticos 305](#_Toc221983490)

[Resumo do Capítulo 305](#_Toc221983491)

[Volume 1: JUNIOR 307](#_Toc221983492)

[Capítulo 13: Segurança e Governança de Dados 307](#_Toc221983493)

[Objetivos do Capítulo 307](#_Toc221983494)

[13.1 Fundamentos de Segurança Azure 307](#_Toc221983495)

[Camadas de Segurança 307](#_Toc221983496)

[13.2 Azure Active Directory (Azure AD) 308](#_Toc221983497)

[Conceitos Principais 308](#_Toc221983498)

[Criar Usuário no Azure AD 308](#_Toc221983499)

[13.3 Role-Based Access Control (RBAC) 309](#_Toc221983500)

[Principais Built-in Roles 309](#_Toc221983501)

[Atribuir Role (RBAC) 309](#_Toc221983502)

[Managed Identity 309](#_Toc221983503)

[13.4 Azure Key Vault 310](#_Toc221983504)

[Vantagens 310](#_Toc221983505)

[Criar Key Vault 311](#_Toc221983506)

[Adicionar Secret 311](#_Toc221983507)

[Usar Secret no ADF 312](#_Toc221983508)

[13.5 Network Security 312](#_Toc221983509)

[Private Endpoints 312](#_Toc221983510)

[Firewall Rules (IP Allowlist) 313](#_Toc221983511)

[13.6 Data Lake ACLs (Access Control Lists) 313](#_Toc221983512)

[Tipos de Permissões 313](#_Toc221983513)

[Configurar ACL 314](#_Toc221983514)

[13.7 Encryption (Criptografia) 314](#_Toc221983515)

[Encryption at Rest 314](#_Toc221983516)

[Encryption in Transit 314](#_Toc221983517)

[Transparent Data Encryption (TDE) - SQL Database 315](#_Toc221983518)

[13.8 Data Masking 315](#_Toc221983519)

[Dynamic Data Masking (SQL Database) 315](#_Toc221983520)

[13.9 Auditing e Logging 316](#_Toc221983521)

[Azure Monitor 316](#_Toc221983522)

[Consultar Logs (Log Analytics) 316](#_Toc221983523)

[SQL Database Auditing 317](#_Toc221983524)

[13.10 Microsoft Purview (Data Governance) 317](#_Toc221983525)

[Criar Purview Account 317](#_Toc221983526)

[Registrar Data Source 318](#_Toc221983527)

[Scan Data Source 318](#_Toc221983528)

[Data Lineage 318](#_Toc221983529)

[13.11 Compliance e Políticas 319](#_Toc221983530)

[Azure Policy 319](#_Toc221983531)

[Retention Policies (Data Lake) 319](#_Toc221983532)

[13.12 Checklist de Habilidades 320](#_Toc221983533)

[13.13 Recursos Complementares 320](#_Toc221983534)

[13.14 Exercícios Práticos 321](#_Toc221983535)

[Resumo do Capítulo 321](#_Toc221983536)

[Volume 1: JUNIOR 323](#_Toc221983537)

[Capítulo 14: Monitoramento e Alertas 323](#_Toc221983538)

[Objetivos do Capítulo 323](#_Toc221983539)

[14.1 Fundamentos de Monitoramento 323](#_Toc221983540)

[Por Que Monitorar? 323](#_Toc221983541)

[Tipos de Monitoramento 323](#_Toc221983542)

[14.2 Azure Monitor 324](#_Toc221983543)

[14.3 Monitorar Azure Data Factory 324](#_Toc221983544)

[Habilitar Diagnostic Settings 324](#_Toc221983545)

[Métricas do ADF 325](#_Toc221983546)

[Consultar Logs do ADF (KQL) 325](#_Toc221983547)

[14.4 Criar Alertas 326](#_Toc221983548)

[Alert Rule (Pipelines Falhados) 326](#_Toc221983549)

[Alert com Webhook (Slack/Teams) 327](#_Toc221983550)

[14.5 Monitorar Synapse 328](#_Toc221983551)

[Cluster Metrics 328](#_Toc221983552)

[Spark UI (Job Monitoring) 328](#_Toc221983553)

[Synapse Logs (Log Analytics) 328](#_Toc221983554)

[14.6 Monitorar SQL Database 329](#_Toc221983555)

[Métricas de Performance 329](#_Toc221983556)

[Query Performance Insights 329](#_Toc221983557)

[Automatic Tuning 330](#_Toc221983558)

[14.7 Monitorar Data Lake (Storage) 330](#_Toc221983559)

[Métricas de Storage 330](#_Toc221983560)

[Storage Logs (Auditoria) 330](#_Toc221983561)

[14.8 Dashboards Customizados 331](#_Toc221983562)

[Criar Dashboard no Azure Portal 331](#_Toc221983563)

[Compartilhar Dashboard 332](#_Toc221983564)

[14.9 Application Insights (Debugging Avançado) 332](#_Toc221983565)

[Integrar ADF com Application Insights 332](#_Toc221983566)

[Distributed Tracing 332](#_Toc221983567)

[14.10 Cost Monitoring (Monitoramento de Custos) 332](#_Toc221983568)

[Cost Management + Billing 332](#_Toc221983569)

[Budget Alerts 333](#_Toc221983570)

[14.11 Health Checks Automatizados 333](#_Toc221983571)

[Criar Health Check Pipeline (ADF) 333](#_Toc221983572)

[14.12 Incident Response Runbooks 334](#_Toc221983573)

[Runbook Exemplo: "Pipeline Failed" 334](#_Toc221983574)

[14.13 Checklist de Habilidades 335](#_Toc221983575)

[14.14 Recursos Complementares 336](#_Toc221983576)

[14.15 Exercícios Práticos 336](#_Toc221983577)

[Resumo do Capítulo 337](#_Toc221983578)

[Volume 1: JUNIOR 338](#_Toc221983579)

[Capítulo 15: CI/CD para Pipelines de Dados 338](#_Toc221983580)

[Objetivos do Capítulo 338](#_Toc221983581)

[15.1 O que é CI/CD? 338](#_Toc221983582)

[Continuous Integration (CI) 338](#_Toc221983583)

[Continuous Deployment (CD) 339](#_Toc221983584)

[CI/CD para Data Pipelines 339](#_Toc221983585)

[15.2 Git Basics 339](#_Toc221983586)

[Conceitos Fundamentais 339](#_Toc221983587)

[Workflow Git Flow 340](#_Toc221983588)

[15.3 Configurar Git no Azure Data Factory 340](#_Toc221983589)

[Opção 1: Azure DevOps Repos 340](#_Toc221983590)

[Conectar ADF ao Git 341](#_Toc221983591)

[Estrutura do Repositório ADF 341](#_Toc221983592)

[15.4 Workflow de Desenvolvimento 342](#_Toc221983593)

[Passo 1: Criar Branch de Feature 342](#_Toc221983594)

[Passo 2: Desenvolver Pipeline 342](#_Toc221983595)

[Passo 3: Commit & Sync 342](#_Toc221983596)

[Passo 4: Criar Pull Request 343](#_Toc221983597)

[Passo 5: Code Review 343](#_Toc221983598)

[Passo 6: Merge 343](#_Toc221983599)

[15.5 Deploy com ARM Templates 343](#_Toc221983600)

[Publish (Gerar ARM Templates) 343](#_Toc221983601)

[Deploy Manual (Azure Portal) 344](#_Toc221983602)

[15.6 CI/CD Pipeline (Azure DevOps) 344](#_Toc221983603)

[Criar Build Pipeline (CI) 344](#_Toc221983604)

[Criar Release Pipeline (CD) 346](#_Toc221983605)

[15.7 Synapse + Git 347](#_Toc221983606)

[Repos (Synapse Git Integration) 347](#_Toc221983607)

[Workflow Synapse 347](#_Toc221983608)

[15.8 Ambientes (Dev, Staging, Prod) 348](#_Toc221983609)

[Estratégia de Ambientes 348](#_Toc221983610)

[Parametrização 348](#_Toc221983611)

[15.9 Testes Automatizados 349](#_Toc221983612)

[Unit Tests (Synapse) 349](#_Toc221983613)

[Integration Tests (ADF) 349](#_Toc221983614)

[15.10 Rollback 350](#_Toc221983615)

[Estratégia de Rollback 350](#_Toc221983616)

[15.11 Checklist de Habilidades 350](#_Toc221983617)

[15.12 Recursos Complementares 351](#_Toc221983618)

[15.13 Exercícios Práticos 351](#_Toc221983619)

[Resumo do Capítulo 352](#_Toc221983620)

[Volume 1: JUNIOR 354](#_Toc221983621)

[Capítulo 16: Projeto Final Integrado 354](#_Toc221983622)

[Objetivos do Capítulo 354](#_Toc221983623)

[16.1 Visão Geral do Projeto 354](#_Toc221983624)

[Cenário de Negócio 354](#_Toc221983625)

[Arquitetura Alvo 355](#_Toc221983626)

[16.2 Fase 1: Provisionar Infraestrutura 356](#_Toc221983627)

[Recursos Necessários 356](#_Toc221983628)

[Script de Provisionamento (ARM Template) 357](#_Toc221983629)

[16.3 Fase 2: Criar Dados de Origem (Source) 358](#_Toc221983630)

[Simular Sistema Transacional 358](#_Toc221983631)

[Dados CSV (Fornecedores) 359](#_Toc221983632)

[16.4 Fase 3: Pipeline Bronze (Ingestão) 359](#_Toc221983633)

[Pipeline ADF: `PL\_Ingest\_Bronze` 360](#_Toc221983634)

[16.5 Fase 4: Pipeline Silver (Transformação) 360](#_Toc221983635)

[Notebook Synapse: `bronze\_to\_silver.py` 360](#_Toc221983636)

[16.6 Fase 5: Pipeline Gold (Data Warehouse) 364](#_Toc221983637)

[Synapse Dedicated Pool: Criar Star Schema 365](#_Toc221983638)

[Pipeline ADF: `PL\_Load\_Gold` 366](#_Toc221983639)

[16.7 Fase 6: Dashboard Power BI 367](#_Toc221983640)

[Conectar Power BI em Synapse 367](#_Toc221983641)

[Criar Medidas DAX 367](#_Toc221983642)

[Páginas do Dashboard 368](#_Toc221983643)

[16.8 Fase 7: Segurança 369](#_Toc221983644)

[Implementar RBAC 369](#_Toc221983645)

[Key Vault (Secrets) 369](#_Toc221983646)

[Private Endpoints 369](#_Toc221983647)

[16.9 Fase 8: Monitoramento 369](#_Toc221983648)

[Alertas Configurados 369](#_Toc221983649)

[Dashboard de Monitoramento 370](#_Toc221983650)

[16.10 Fase 9: CI/CD 370](#_Toc221983651)

[Git Repository Structure 370](#_Toc221983652)

[CI Pipeline (Azure DevOps) 371](#_Toc221983653)

[CD Pipeline (Releases) 371](#_Toc221983654)

[16.11 Fase 10: Documentação 372](#_Toc221983655)

[Arquitetura (Diagrams) 372](#_Toc221983656)

[Data Dictionary 372](#_Toc221983657)

[Runbooks 373](#_Toc221983658)

[16.12 Apresentar Projeto (Portfolio) 373](#_Toc221983659)

[GitHub Repository 373](#_Toc221983660)

[LinkedIn Post 374](#_Toc221983661)

[16.13 Checklist de Entrega 375](#_Toc221983662)

[16.14 Próximos Passos (Além do Junior) 376](#_Toc221983663)

[Resumo do Capítulo 376](#_Toc221983664)

[Parabéns! 377](#_Toc221983665)

[Agradecimentos 377](#_Toc221983666)

# Capítulo 1: Introdução à Engenharia de Dados

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender o papel e responsabilidades de um Engenheiro de Dados
* Diferenciar Engenheiro de Dados, Data Analyst e Data Scientist
* Identificar os componentes de uma arquitetura moderna de dados
* Conhecer a stack de tecnologias Azure para dados
* Navegar no Azure Portal com confiança
* Criar sua conta gratuita no Azure

**Tempo estimado:** 2-3 horas

**Custo Azure:** R$ 0 (usaremos apenas conta gratuita)

## 1.1 O Papel do Engenheiro de Dados Moderno

### O que é Engenharia de Dados?

**Engenharia de Dados** é a disciplina responsável por **projetar, construir e manter a infraestrutura** que permite que dados brutos sejam transformados em informações úteis e acessíveis para análise e tomada de decisão.

Se você pensar em dados como petróleo, o Engenheiro de Dados é quem constrói os oleodutos, refinarias e sistemas de distribuição. Sem essa infraestrutura, o “petróleo” (dados brutos) não tem valor.

### O que faz um Engenheiro de Dados no dia a dia?

**Responsabilidades principais:**

1. **Ingestão de Dados**
   * Conectar-se a diversas fontes: bancos de dados, APIs, arquivos, streams
   * Extrair dados de sistemas transacionais, CRMs, ERPs, sensores IoT
   * Garantir que dados sejam coletados de forma confiável e eficiente
2. **Transformação de Dados**
   * Limpar dados (remover duplicatas, tratar valores nulos)
   * Padronizar formatos (datas, moedas, nomenclaturas)
   * Enriquecer dados (combinar múltiplas fontes)
   * Agregar e calcular métricas de negócio
3. **Armazenamento e Modelagem**
   * Escolher onde armazenar dados (Data Lake, Data Warehouse, bancos NoSQL)
   * Modelar dados para consumo analítico (Star Schema, modelagem dimensional)
   * Otimizar para performance de consultas
4. **Orquestração e Automação**
   * Construir pipelines automatizados (ETL/ELT)
   * Agendar execuções (diárias, horárias, real-time)
   * Monitorar falhas e configurar alertas
5. **Governança e Qualidade**
   * Implementar controles de acesso (quem pode ver o quê?)
   * Garantir qualidade dos dados (validações, testes)
   * Documentar pipelines e datasets
6. **Performance e Escalabilidade**
   * Otimizar consultas lentas
   * Escalar sistemas para lidar com crescimento de volume
   * Reduzir custos de infraestrutura

**Exemplo prático:**

Imagine um e-commerce. O Engenheiro de Dados:

* Coleta dados de vendas do banco de produção
* Integra com dados de marketing (campanhas, cliques)
* Transforma e limpa os dados
* Armazena em um Data Warehouse modelado
* Cria pipelines que atualizam dashboards toda manhã às 6h
* Garante que apenas o time de vendas vê dados de receita

### Evolução da Carreira

O mercado de Engenharia de Dados está crescendo exponencialmente. Empresas descobriram que não basta ter dados: é preciso **torná-los acessíveis e confiáveis**.

**Roadmap típico:**

Júnior (0-2 anos) → Implementa pipelines, aprende ferramentas

↓

Pleno (2-4 anos) → Projeta arquiteturas, otimiza performance

↓

Sênior (4-7 anos) → Lidera tecnicamente, define estratégia

↓

Staff/Principal (7+ anos) → Arquiteto, influencia organização

**Salários no Brasil (2026):**

* Júnior: R$ 5.000 - R$ 8.000
* Pleno: R$ 10.000 - R$ 15.000
* Sênior: R$ 18.000 - R$ 25.000
* Staff+: R$ 30.000 - R$ 50.000+

*(Valores variam por região e empresa)*

## 1.2 Engenharia de Dados vs Outras Carreiras

É comum confundir Engenheiro de Dados com outras funções. Vamos esclarecer:

### Comparação: DE vs Data Analyst vs Data Scientist

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspecto** | **Engenheiro de Dados** | **Data Analyst** | **Data Scientist** |
| **Foco principal** | Infraestrutura de dados | Análise de negócio | Modelagem preditiva |
| **Pergunta chave** | “Como disponibilizar os dados?” | “O que os dados dizem?” | “O que vai acontecer?” |
| **Ferramentas** | Python, Spark, SQL, ADF, Databricks | Excel, Power BI, SQL, Python (Pandas) | Python, R, ML frameworks |
| **Output** | Pipelines, tabelas, APIs de dados | Dashboards, relatórios, insights | Modelos de ML, previsões |
| **Skills técnicas** | Engenharia de software, cloud, DBs | Estatística básica, visualização | Estatística avançada, ML |
| **Quem consome** | Analysts, Scientists, negócio | Gestores, executivos | Produto, negócio |

**Exemplo prático:**

Em um projeto de churn (previsão de cancelamento de clientes):

* **Engenheiro de Dados:** Cria pipeline que coleta dados de clientes, histórico de compras e interações. Modela tabela fato\_historico\_cliente atualizada diariamente.
* **Data Analyst:** Usa a tabela para criar dashboard mostrando taxa de churn por região, produto e perfil de cliente.
* **Data Scientist:** Treina modelo de ML para prever quais clientes têm maior risco de cancelar nos próximos 30 dias.

**Complementaridade:**

Essas carreiras não competem — elas se complementam! Em times maduros de dados, todas coexistem e colaboram:

[Engenheiro de Dados]

↓ (fornece dados limpos e confiáveis)

[Data Analyst] + [Data Scientist]

↓ (geram insights e modelos)

[Negócio toma decisões melhores]

### Transições de Carreira

**De onde vêm Engenheiros de Dados?**

1. **Desenvolvedores de Software** → Já sabem programar e arquitetura
2. **DBAs (Database Administrators)** → Entendem bem bancos de dados
3. **Data Analysts** → Querem ir mais fundo na parte técnica
4. **Analistas de BI** → Evoluir de consumir para criar dados

**O que precisam aprender:**

* Programação (Python, SQL avançado)
* Cloud (Azure, AWS, GCP)
* Big Data (Spark, distributed systems)
* DevOps (CI/CD, Git, automação)

## 1.3 Pipeline de Dados: Conceitos Fundamentais

### O que é um Pipeline de Dados?

Um **pipeline de dados** é uma série de etapas automatizadas que movem e transformam dados de um ponto A (origem) para um ponto B (destino).

**Analogia:**

Pense em uma fábrica de suco:

1. **Coleta:** Laranjas chegam da fazenda (dados brutos de sistemas)
2. **Limpeza:** Lavas as laranjas (remover dados ruins)
3. **Transformação:** Espremer, filtrar polpa (transformar e enriquecer)
4. **Embalagem:** Colocar em garrafas (modelar para consumo)
5. **Distribuição:** Levar ao supermercado (disponibilizar para análise)

### ETL vs ELT: Abordagens Fundamentais

**ETL (Extract, Transform, Load)** - Abordagem tradicional

[Fonte] → [Extract] → [Transform] → [Load] → [Data Warehouse]

* **Extract:** Puxar dados da origem (SQL Server, API, arquivo)
* **Transform:** Transformar **fora** do destino (em ferramenta de ETL)
* **Load:** Carregar dados já prontos no destino

**Vantagens:**

* Dados chegam limpos e prontos
* Menos processamento no destino
* Funciona bem com DW tradicional (recursos limitados)

**Desvantagens:**

* Transformações são “black box” (difícil auditar)
* Dificulta análises exploratórias (dados brutos não disponíveis)

**ELT (Extract, Load, Transform)** - Abordagem moderna

[Fonte] → [Extract] → [Load] → [Data Lake/Warehouse] → [Transform]

* **Extract:** Puxar dados da origem
* **Load:** Carregar dados **brutos** no destino (Data Lake)
* **Transform:** Transformar **dentro** do destino (Spark, SQL)

**Vantagens:**

* Dados brutos preservados (rastreabilidade completa)
* Flexibilidade: transformar de formas diferentes conforme necessidade
* Aproveita poder de cloud (Spark, Synapse podem processar muito)

**Desvantagens:**

* Exige destino com capacidade de processar (não funciona em DB pequeno)
* Dados brutos ocupam espaço

**Qual usar?**

* **ETL:** Sistemas legados, destinos com recursos limitados, transformações complexas e estáveis
* **ELT:** Cloud-first, Data Lakes, análises exploratórias, múltiplos casos de uso

**Na prática moderna (Azure):** Usamos **ELT híbrido**:

* Ingestão → Data Lake (bruto)
* Transformações leves → Data Lake (limpo)
* Transformações pesadas → Databricks/Synapse
* Resultado → Data Warehouse ou Data Lake (camada Gold)

## 1.4 Arquiteturas de Dados: Data Warehouse, Data Lake, Lakehouse

### Data Warehouse (DW) - A Casa Organizada

**O que é:**

Banco de dados otimizado para **consultas analíticas**, com dados estruturados, limpos e modelados.

**Características:**

* Estrutura rígida (schema-on-write)
* Dados relacionais (tabelas com colunas definidas)
* Modelagem dimensional (fatos e dimensões)
* Otimizado para leitura (não para escrita)
* Alta performance em consultas agregadas

**Exemplo no Azure:**

* Azure Synapse Analytics (Dedicated SQL Pool)
* Azure SQL Database (para volumes menores)

**Quando usar:**

* BI tradicional (dashboards, relatórios)
* Dados estruturados bem definidos
* Consultas repetitivas e conhecidas

**Limitações:**

* Custo alto com grande volume
* Difícil adaptar schema (não flexível)
* Não suporta dados não estruturados (imagens, vídeos, JSON complexo)

### Data Lake - O Lago de Dados Brutos

**O que é:**

Repositório centralizado que armazena **qualquer tipo de dado** em **formato bruto** (estruturado, semiestruturado, não estruturado).

**Características:**

* Estrutura flexível (schema-on-read)
* Armazena qualquer formato: CSV, JSON, Parquet, imagens, vídeos, logs
* Escalável para petabytes
* Custo baixo de armazenamento
* Permite análises exploratórias

**Exemplo no Azure:**

* Azure Data Lake Storage Gen2

**Quando usar:**

* Big Data (volume massivo)
* Dados não estruturados (logs, IoT, imagens)
* Machine Learning (precisa dados brutos)
* Múltiplos casos de uso (não sabe ainda como vai usar)

**Limitações:**

* Pode virar “data swamp” (pântano de dados) sem governança
* Performance de consulta inferior ao DW (se não otimizado)
* Exige ferramentas de processamento (Spark) para transformar

### Lakehouse - O Melhor dos Dois Mundos

**O que é:**

Arquitetura que combina **flexibilidade do Data Lake** com **performance e governança do Data Warehouse**.

**Características:**

* Armazenamento barato (Data Lake)
* ACID transactions (confiabilidade de DW)
* Schema enforcement opcional
* Suporta SQL e Spark
* Time travel e versionamento

**Tecnologia chave:** **Delta Lake** (formato de tabela open-source)

**Exemplo no Azure:**

* Azure Databricks + Delta Lake
* Synapse Serverless SQL Pool + Delta/Parquet

**Quando usar:**

* Quer flexibilidade de Lake + confiabilidade de DW
* Casos de uso variados (BI + ML + análises ad-hoc)
* Modernização de DW tradicional

**Arquitetura Medallion (padrão moderno):**

Bronze (Raw) → Dados brutos como chegaram

↓

Silver (Cleaned) → Dados limpos, validados, enriquecidos

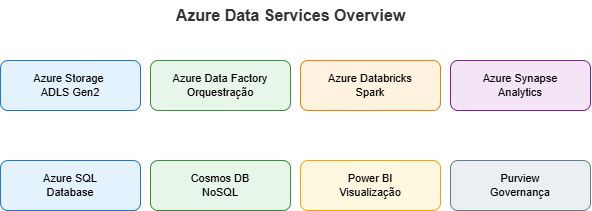
↓

Gold (Curated) → Dados agregados, modelados para negócio

Explicaremos essa arquitetura em detalhes no Capítulo 2.

### Comparação Resumida

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspecto** | **Data Warehouse** | **Data Lake** | **Lakehouse** |
| **Tipo de dados** | Estruturados | Todos os tipos | Todos os tipos |
| **Schema** | Rígido (on-write) | Flexível (on-read) | Híbrido |
| **Custo** | Alto | Baixo | Médio |
| **Performance consulta** | Excelente | Variável | Boa a excelente |
| **Casos de uso** | BI, relatórios | ML, exploração | BI + ML + exploração |
| **Governança** | Forte | Fraca (sem ferramentas) | Forte |
| **Tecnologia Azure** | Synapse Dedicated | ADLS Gen2 | Databricks Delta |



*Figura 1.1 - Evolução das arquiteturas de dados: do Data Warehouse tradicional ao Lakehouse moderno*

## 1.5 Stack de Tecnologias Azure para Dados

O ecossistema Azure oferece uma suíte completa de serviços para Engenharia de Dados. Vamos conhecer os principais:

### Armazenamento

**Azure Data Lake Storage Gen2 (ADLS Gen2)**

* Armazenamento de objetos massivo
* Base para Data Lakes
* Suporta hierarquia de pastas
* Integração nativa com ferramentas big data

**Azure Blob Storage**

* Armazenamento de objetos simples
* Mais barato que ADLS Gen2
* Ideal para backups, arquivos, imagens

**Azure SQL Database**

* Banco de dados relacional gerenciado
* Ideal para dados estruturados de volume médio
* Compatível com SQL Server

### Processamento e Transformação

**Azure Data Factory (ADF)**

* Orquestração de pipelines ETL/ELT
* Conecta 100+ fontes de dados
* Interface visual (low-code)
* Agendamento e monitoramento

**Azure Databricks**

* Plataforma de analytics baseada em Apache Spark
* Processamento de Big Data
* Notebooks colaborativos (Python, SQL, Scala)
* Suporta Delta Lake

**Azure Synapse Analytics**

* Plataforma analítica unificada
* **Dedicated SQL Pool:** Data Warehouse massivo
* **Serverless SQL Pool:** Consultar Data Lake sem provisionamento
* **Spark Pools:** Processamento big data integrado

### Análise e Consumo

**Power BI**

* Visualização e dashboards
* Conecta com todos os serviços Azure
* Self-service BI

**Azure Analysis Services**

* Modelagem semântica (cubos OLAP)
* Para casos complexos de BI

### Governança e Segurança

**Azure Purview**

* Catálogo de dados
* Data lineage (rastreamento de origem)
* Classificação de dados sensíveis

**Azure Key Vault**

* Gerenciamento de segredos (senhas, chaves)
* Integração com todos os serviços

**Azure Active Directory (AAD)**

* Identidade e controle de acesso (RBAC)

### Streaming e Real-Time

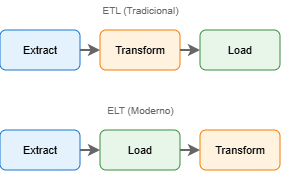
**Azure Event Hubs**

* Ingestão de eventos em streaming (milhões/segundo)
* Ideal para IoT, telemetria

**Azure Stream Analytics**

* Processamento de streaming em tempo real
* SQL-like para streams

### Diagrama da Stack



*Figura 1.2 - Stack completa de serviços Azure para Engenharia de Dados*

## 1.6 Roadmap de Carreira: Júnior → Pleno → Sênior

Vamos mapear o que você precisa dominar em cada nível:

### Júnior (0-2 anos) - ESTE LIVRO!

**Habilidades Técnicas:**

* SQL intermediário (JOINs, agregações, CTEs)
* Python básico (scripts, Pandas)
* Azure Data Factory (pipelines simples)
* Azure Databricks (notebooks, PySpark básico)
* Git (commit, push, pull)
* Modelagem dimensional básica (Star Schema)

**Responsabilidades:**

* Implementar pipelines seguindo especificação
* Manter pipelines existentes
* Debugar erros simples
* Documentar processos
* Aprender com seniores

**Mentoria:** Alta supervisão, pair programming frequente

### Pleno (2-4 anos) - Volume 2 deste livro

**Habilidades Técnicas:**

* SQL avançado (window functions, performance tuning)
* Python avançado (OOP, async, testes)
* Spark otimização (partitioning, caching)
* CI/CD (Azure DevOps, GitHub Actions)
* Modelagem avançada (Data Vault, SCD types)
* Observabilidade (logs, métricas, alertas)

**Responsabilidades:**

* Desenhar pipelines e arquiteturas
* Otimizar performance e custos
* Code review de juniores
* Resolver incidentes de produção
* Propor melhorias técnicas

**Mentoria:** Supervisão ocasional, autonomia na execução

### Sênior (4-7 anos) - Volume 3 deste livro

**Habilidades Técnicas:**

* Arquitetura de sistemas distribuídos
* Multi-cloud (Azure + AWS/GCP)
* Real-time streaming (Kafka, Flink)
* MLOps (feature stores, model serving)
* IaC avançado (Terraform, Bicep)
* Liderança técnica

**Responsabilidades:**

* Definir arquitetura de plataformas
* Tomar decisões técnicas críticas
* Mentorar plenos e juniores
* Influenciar roadmap técnico
* Comunicar com stakeholders executivos
* On-call para incidentes críticos

**Mentoria:** Autonomia completa, lidera tecnicamente

### Staff / Principal (7+ anos)

**Habilidades:**

* Visão estratégica de dados
* Influência organizacional
* Thought leadership (“Liderança de Pensamento”)
* Arquitetura enterprise

**Responsabilidades:**

* Definir estratégia de dados da empresa
* Avaliar tecnologias emergentes
* Liderar múltiplos times tecnicamente
* Falar em conferências
* Contribuir open-source

**Dica de Ouro:**

Não espere “estar pronto” para buscar promoção. Se você já faz 70% das responsabilidades do próximo nível, está na hora de conversar com seu gestor.

## 1.7 Hands-On: Criando sua Conta Azure Gratuita

Agora vamos colocar a mão na massa! Você vai criar sua conta Azure e explorar o portal.

### Por que Azure?

* R$ 1.200 em créditos gratuitos (válido 30 dias)
* 25+ serviços sempre gratuitos
* SQL Database, Data Lake, Functions gratuitos (com limites)
* Segundo maior provedor cloud (depois de AWS)
* Forte no mercado corporativo brasileiro

### Pré-requisitos

* Email válido (de preferência Outlook/Hotmail, mas Gmail funciona)
* Cartão de crédito (apenas para validação, não será cobrado)
* Telefone para verificação

### Passo 1: Acesse a Página de Cadastro

1. Abra o navegador e vá para: **https://azure.microsoft.com/free**
2. Clique em **“Iniciar gratuitamente”** ou **“Start free”**



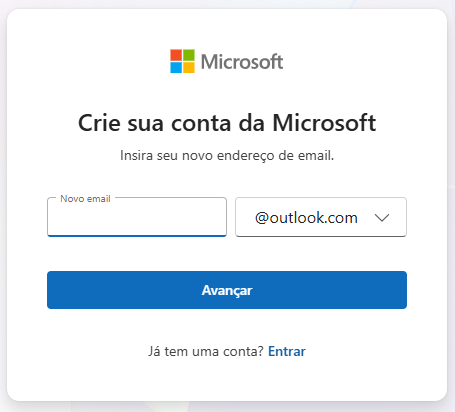
*Figura 1.3 - Página inicial do Azure Free*

### Passo 2: Criar Conta Microsoft (se não tiver)

1. Se você já tem conta Microsoft (Outlook, Hotmail, Xbox, Office 365), clique em **“Entrar”**
2. Se não tem, clique em **“Criar uma”**

**Criando nova conta:**

* Escolha um email (pode ser Gmail ou criar novo @outlook.com)
* Crie uma senha forte
* Confirme país e data de nascimento
* Complete verificação de segurança (CAPTCHA). Tenha paciência. Pode ser bem chato esse processo...



*Figura 1.4 - Criação de conta Microsoft*

### Passo 3: Preencher Informações Pessoais

1. **Dados pessoais:**
   * Nome completo
   * País/Região: Brasil
   * Telefone (receberá código SMS)
2. **Verificação por SMS:**
   * Digite seu número de celular
   * Clique em “Enviar código de texto”
   * Digite o código recebido

### Passo 4: Validação de Identidade (Cartão)

**IMPORTANTE:** O cartão é apenas para validação. Você **NÃO** será cobrado nos primeiros 30 dias, e depois disso, só se você manualmente atualizar para conta paga.

1. **Informações do cartão:**
   * Número do cartão
   * Data de validade
   * CVV
   * Nome no cartão
   * Endereço de cobrança
2. **Clique em “Avançar”**

Você verá uma cobrança temporária de R$ 1,00 ou similar (será estornado).

### Passo 5: Concordar com Termos

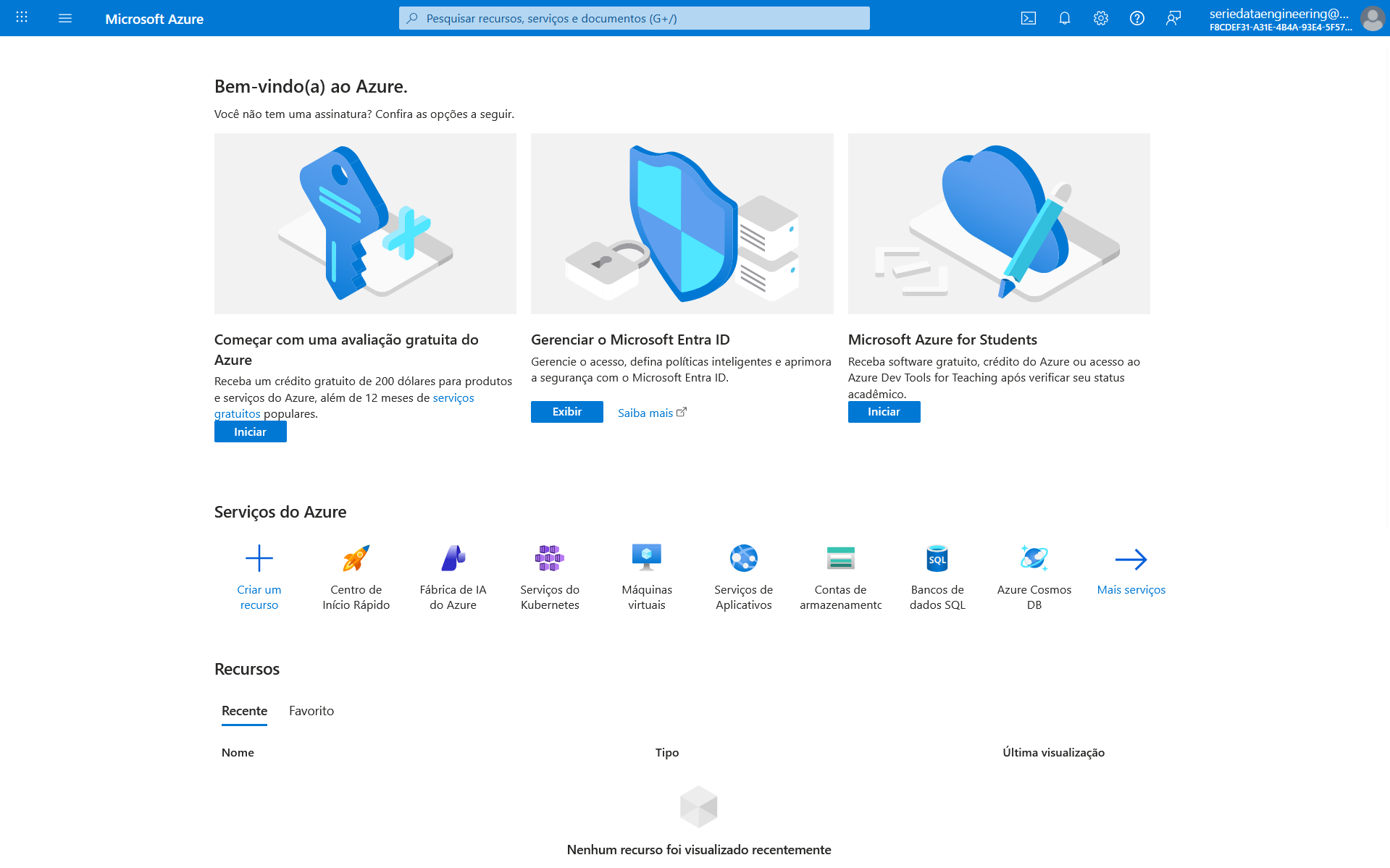
1. Leia (ou pelo menos role) os termos de serviço
2. Marque a caixa **“Concordo com os termos...”**
3. Clique em **“Inscrever-se”** ou **“Sign up”**

Aguarde alguns segundos enquanto sua conta é criada...

### Passo 6: Bem-vindo ao Azure Portal!

**Parabéns!** Sua conta foi criada com sucesso!

Você será redirecionado para o **Azure Portal** (https://portal.azure.com)

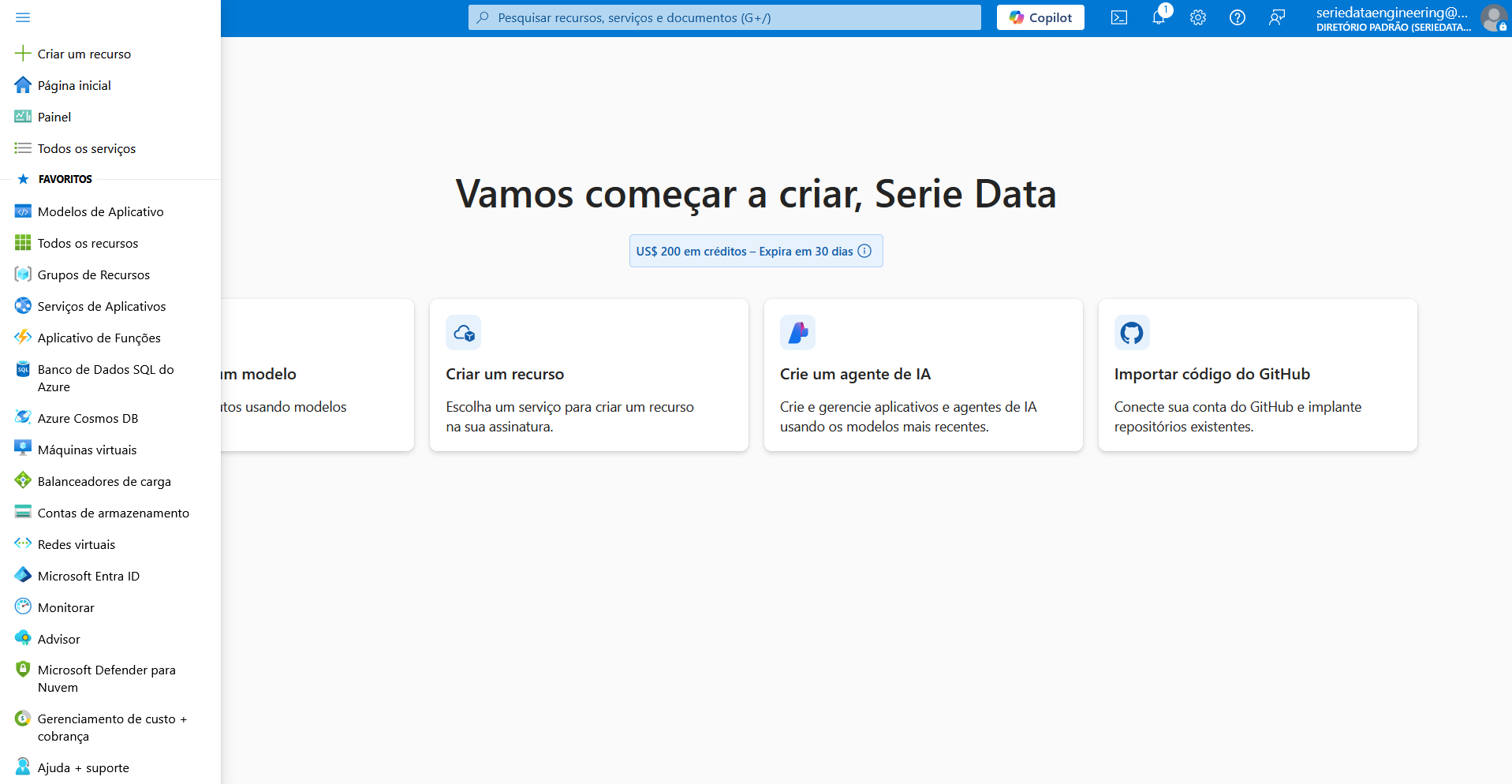
**

*Figura 1.7 - Página inicial do Azure Portal após login*

## 1.8 Tour Guiado pelo Azure Portal

Vamos nos familiarizar com a interface do Azure Portal, que será sua casa nos próximos meses.

### Anatomia do Portal



*Figura 1.8 - Anatomia do Azure Portal*

**Componentes principais:**

1. **Menu Hamburger (≡)** - Topo esquerdo
   * Expande/colapsa menu lateral
   * Clique para fixar menu aberto
2. **Barra de Pesquisa** - Topo central
   * Busca por serviços: “storage”, “databricks”, “sql”
   * Busca por recursos: nome do seu Data Lake
   * Atalho: tecla **/** (barra)
3. **Ícones Superiores Direita:**
   * **Cloud Shell** (ícone >\_): terminal bash/PowerShell no navegador
   * **Notificações** (ícone sino): alertas e status de deployments
   * **Configurações** (ícone engrenagem): tema, idioma
   * **Ajuda** (ícone ?)
   * **Perfil**: sua conta e assinaturas
4. **Menu Lateral Esquerdo:**
   * **Home**: volta ao dashboard inicial
   * **Criar um recurso**: provisionar novos serviços
   * **Todos os serviços**: catálogo completo
   * **Grupos de recursos**: containers lógicos de recursos
   * **Favoritos**: serviços que você mais usa (personalizável)
5. **Área Central:**
   * Conteúdo principal (dashboards, formulários, configurações)

### Ação opcional: Idioma

**Trocar o Idioma de exibição, clique no ícone da engrenagem na barra superior, e na sequência:**

* + **Idioma e Região: e**scolha o idioma inglês e o formato de região pode manter português (Brasil).
    - Clique imediatamente em aplicar e confirme

Navegue pelas demais opções para conhecer o portal. Escolha comportamento, aparência, complete informações etc.

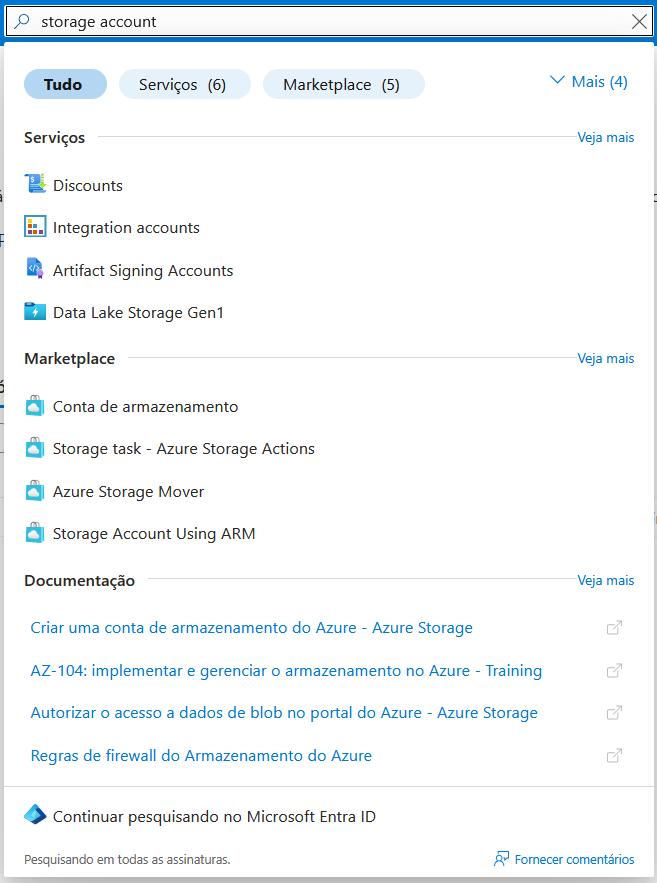
Por que alterar para inglês? Simples, é a língua com maior quantidade de referências escritas e na internet, e isso pode ser útil na sua carreira e aprendizado. Acostume-se com isso, vai se tornar natural com o tempo e uso.

No entanto, ao longo deste livro, vamos alternar entre comandos em português e inglês.

### Navegação: 3 Formas de Encontrar Serviços

**Forma 1: Pesquisa (MAIS RÁPIDA)**

1. Clique na barra de pesquisa (ou aperte G + /)
2. Digite o nome do serviço: storage accounts
3. Selecione na lista de sugestões. Aqui vai perceber que mesmo digitando em inglês, e caso tenha mantido a interface em português, você vai obter o mesmo resultado, aqui Contas de Armazenamento



*Figura 1.9 - Pesquisa de serviços na barra superior*

**Forma 2: Menu Lateral**

1. Clique em **“Todos os serviços”** no menu esquerdo
2. Navegue pelas categorias: Análise, Computação, Bancos de Dados, etc.
3. Ou use a barra de filtro

**Forma 3: “Criar um Recurso”**

1. Menu lateral > **“Criar um recurso”**
2. Ou botão **“+ Criar”** no topo
3. Navegue por categorias ou pesquise

### Conceitos Importantes: Subscription e Resource Group

Antes de criar qualquer recurso, entenda esses conceitos fundamentais:

**Subscription (Assinatura):**

* Container de cobrança
* Você tem pelo menos uma (Free Trial)
* Empresas grandes têm múltiplas (por departamento, projeto, ambiente)
* Cada recurso pertence a uma subscription

**Resource Group (Grupo de Recursos):**

* Container lógico para agrupar recursos relacionados
* Exemplo: rg-ecommerce-dev agrupa Data Lake, SQL, ADF do projeto e-commerce
* Facilita gerenciamento (deletar grupo = deletar tudo dentro)
* Todos os recursos vivem dentro de um RG

**Hierarquia:**

Azure Account

Subscription (Free Trial - R$ 1.200 créditos)

Resource Group: rg-projeto1

Storage Account

SQL Database

Data Factory

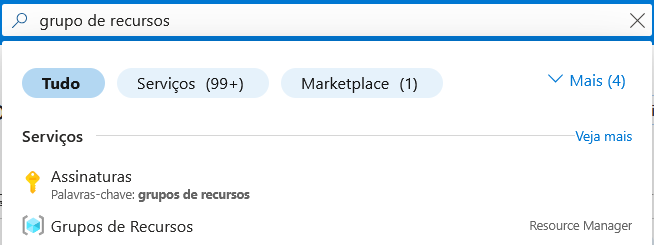
Resource Group: rg-projeto2

...

### Hands-On: Criar seu Primeiro Resource Group

Vamos criar um Resource Group para organizar os recursos deste livro:

1. **Na barra de pesquisa, digite:** resource groups
2. **Clique em “Resource groups”**

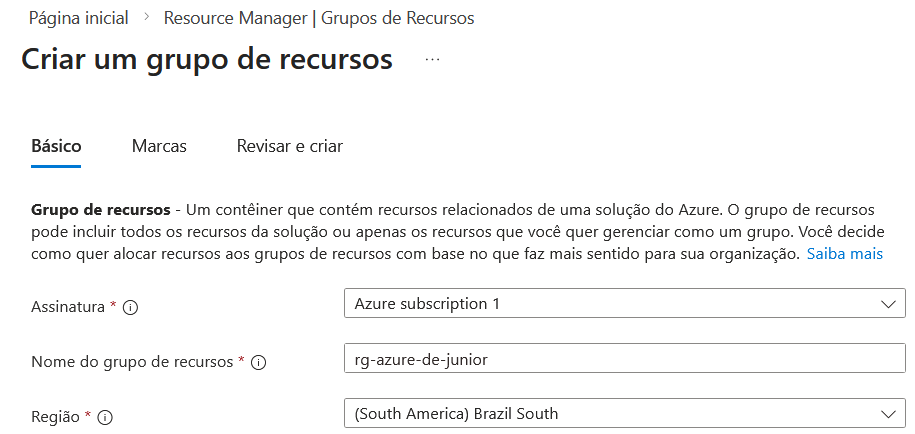
****

*Figura 1.10 - Página de Resource Groups*

1. **Clique em “+ Create”** (ou “+ Criar”)
2. **Preencha o formulário:**
   * **Subscription:** Selecione sua “Free Trial” ou “Azure subscription 1”
   * **Resource group:** rg-azure-de-junior (nome livre, sem espaços)
   * **Region:** Brazil South (São Paulo)

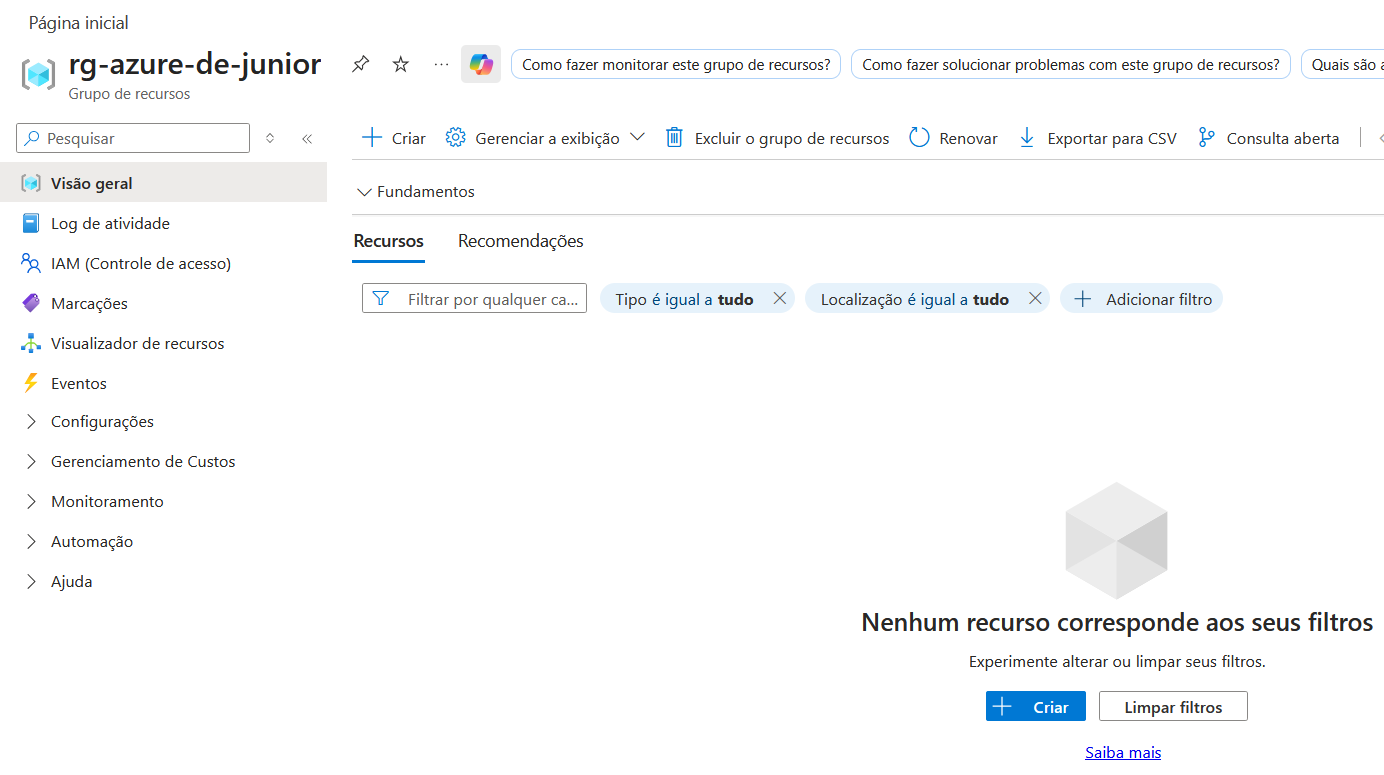
**Dica de Nomenclatura:**

* + Use prefixos descritivos: rg- para resource group, st- para storage, etc.
  + Inclua ambiente se necessário: rg-projeto-dev, rg-projeto-prod
  + Minúsculas e hifens (convenção)

****

*Figura 1.11 - Criação de Resource Group*

1. **Clique em “Review + create”** (revisar + criar)
2. **Aguarde validação** (alguns segundos)
3. **Clique em “Create”**
4. **Aguarde deployment** (< 10 segundos)
5. **Clique em “Go to resource group”**

* Figura 1.12 - Resource Group criado com sucesso (vazio)*

**Parabéns!** Você criou seu primeiro recurso no Azure!

### Explorando Serviços de Dados

Vamos dar uma olhada rápida nos serviços que usaremos nos próximos capítulos:

**1. Pesquise por cada serviço:**

* + Storage Accounts (Data Lake Gen2)
  + SQL databases (bancos relacionais)
  + Data Factory (orquestração ETL)
  + Databricks (big data processing)
  + Synapse Analytics (data warehouse + analytics)

**2. Para cada um, clique e explore:**

* + Descrição do serviço
  + Pricing (preços)
  + Documentation (documentação)
  + **NÃO CRIE AINDA** — faremos isso nos próximos capítulos

**Objetivo:** Se familiarizar com os nomes e ícones dos serviços.

## 1.9 Mapa Mental da Stack Azure

Para consolidar o aprendizado, vamos criar um mapa mental da stack completa:

ENGENHARIA DE DADOS NO AZURE

ARMAZENAMENTO

Data Lake Storage Gen2 (dados brutos, Big Data)

Blob Storage (arquivos, backups)

SQL Database (dados estruturados, transacional)

INGESTÃO E ORQUESTRAÇÃO

Data Factory (ETL/ELT visual, conectores)

Event Hubs (streaming, eventos real-time)

PROCESSAMENTO E TRANSFORMAÇÃO

Databricks (Spark, notebooks, Delta Lake)

Synapse Spark Pools (Spark integrado)

Stream Analytics (processamento de streams)

ANÁLISE E CONSUMO

Synapse SQL Pools (Data Warehouse)

Dedicated (provisionado, alto volume)

Serverless (on-demand, Data Lake)

Power BI (visualização, dashboards)

GOVERNANÇA E SEGURANÇA

Purview (catálogo, lineage, classificação)

Key Vault (segredos, chaves, certificados)

Active Directory (autenticação, RBAC)

**Dica:** Imprima ou salve este mapa. Ele será seu guia nos próximos capítulos!

## 1.10 Checklist de Habilidades

Ao final deste capítulo, você deve ser capaz de marcar em todos os itens:

**Conceitos:**

* [ ] Explicar o papel de um Engenheiro de Dados
* [ ] Diferenciar Engenheiro de Dados, Data Analyst e Data Scientist
* [ ] Descrever a diferença entre ETL e ELT
* [ ] Comparar Data Warehouse, Data Lake e Lakehouse
* [ ] Listar os principais serviços Azure para dados

**Práticos:**

* [ ] Criar conta gratuita no Azure
* [ ] Navegar no Azure Portal com confiança
* [ ] Criar um Resource Group
* [ ] Localizar serviços usando pesquisa
* [ ] Entender conceitos de Subscription e Resource Group

**Mindset:**

* [ ] Ter visão clara do roadmap de carreira (Junior → Senior)
* [ ] Entender que Engenharia de Dados é sobre infraestrutura, não análise
* [ ] Reconhecer que automação e confiabilidade são prioridades

## 1.11 Próximos Passos

No **Capítulo 2**, você vai:

Criar seu primeiro Data Lake Storage Gen2

Implementar a arquitetura Medallion (Bronze/Silver/Gold)

Fazer upload de arquivos (CSV, JSON, Parquet)

Configurar permissões e segurança básica

Usar Azure Storage Explorer para navegar dados

**Preparação:**

* Mantenha sua conta Azure ativa
* Revise conceitos de armazenamento de arquivos (pastas, hierarquia)
* Se possível, encontre um dataset CSV público para praticar (exemplo: vendas de e-commerce)

## 1.12 Recursos Complementares

**Documentação Oficial:**

* [Introdução ao Azure](https://learn.microsoft.com/azure/guides/)
* [Azure Data Factory Overview](https://learn.microsoft.com/azure/data-factory/)
* [What is a Data Engineer?](https://learn.microsoft.com/training/roles/data-engineer)

**Cursos Gratuitos Microsoft Learn:**

* [Azure Fundamentals](https://learn.microsoft.com/training/paths/azure-fundamentals/)
* [Explore data engineering fundamentals](https://learn.microsoft.com/training/paths/data-engineering/)

**Comunidades:**

* [r/dataengineering](https://reddit.com/r/dataengineering) (Reddit)
* [Data Engineering Brasil](https://www.linkedin.com/groups/) (LinkedIn - busque grupos)
* [Azure Meetups Brasil](https://www.meetup.com/pt-BR/) (eventos presenciais/online)

**Canais YouTube (inglês):**

* Seattle Data Guy
* Data Engineering Academy
* Microsoft Azure

## 1.13 Exercícios Práticos (Opcional)

**Exercício 1: Exploração do Portal**

Navegue pelo Azure Portal e encontre:

1. Página de Cost Management (gerenciamento de custos)
2. Página de Service Health (saúde dos serviços Azure)
3. Seu saldo de créditos gratuitos restantes

**Exercício 2: Pesquisa de Mercado**

1. Acesse LinkedIn e busque por “Engenheiro de Dados” + sua cidade
2. Leia 5 descrições de vagas
3. Liste as 5 tecnologias mais mencionadas
4. Compare com a stack que você aprenderá neste livro

**Exercício 3: Documentação**

Crie um arquivo learning-log.md onde você vai documentar:

* Data de início deste livro
* Principais aprendizados de cada capítulo
* Dúvidas e como resolveu
* Projetos que pretende fazer

## Resumo do Capítulo

Neste capítulo você aprendeu:

**O papel do Engenheiro de Dados:** Construir infraestrutura para tornar dados acessíveis e confiáveis

**Diferenças entre carreiras de dados:** DE (infraestrutura) vs Analyst (insights) vs Scientist (predição)

**ETL vs ELT:** ETL transforma antes de carregar, ELT carrega bruto e transforma depois

**Arquiteturas modernas:** Data Warehouse (estruturado), Data Lake (flexível), Lakehouse (híbrido)

**Stack Azure:** ADLS Gen2, SQL, ADF, Databricks, Synapse, Purview, Key Vault

**Navegação no Azure Portal:** Pesquisa, menu lateral, Resource Groups

**Hands-on:** Conta gratuita criada, primeiro Resource Group provisionado

**Próximo capítulo:** Criar Data Lake e implementar arquitetura Medallion!

# Capítulo 2: Azure Storage e Data Lake Gen2

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender a diferença entre Blob Storage e Data Lake Gen2
* Criar e configurar uma Storage Account com Data Lake Gen2
* Implementar a arquitetura Medallion (Bronze/Silver/Gold)
* Fazer upload e gerenciar arquivos no Data Lake
* Usar Azure Storage Explorer para navegação
* Configurar permissões básicas (RBAC e ACLs)
* Entender hierarquia de containers e pastas

**Tempo estimado:** 3-4 horas

**Custo Azure:** R$ 2-5 (armazenamento de poucos GB)

## 2.1 Azure Storage: Fundamentos

### O que é Azure Storage?

**Azure Storage** é o serviço de armazenamento de objetos da Microsoft Azure. Pense nele como um “HD infinito na nuvem” onde você pode guardar qualquer tipo de arquivo.

**Principais características:**

* **Escalável:** De alguns MB até petabytes
* **Durável:** Dados replicados automaticamente (99.999999999% de durabilidade)
* **Acessível:** Via REST API, SDKs (Python, .NET, Java), portal web
* **Seguro:** Encriptação em repouso e em trânsito
* **Econômico:** A partir de R$ 0,10/GB/mês (tier frio)

### Tipos de Storage no Azure

Azure oferece diferentes tipos de storage para diferentes necessidades:

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Descrição** | **Casos de Uso** |
| **Blob Storage** | Armazenamento de objetos (arquivos) | Imagens, vídeos, backups, logs |
| **Data Lake Gen2** | Blob + hierarquia de pastas + otimizações Big Data | Data Lakes, analytics, Spark |
| **File Storage** | File shares SMB (como pastas de rede) | Compartilhamento de arquivos entre VMs |
| **Queue Storage** | Filas de mensagens | Comunicação assíncrona entre sistemas |
| **Table Storage** | NoSQL key-value | Dados estruturados simples (telemetria, logs) |

**Foco deste capítulo:** **Data Lake Storage Gen2**

## 2.2 Blob Storage vs Data Lake Gen2

### Blob Storage (Armazenamento de Blobs)

**“Blob”** = Binary Large Object (qualquer arquivo binário)

**Características:**

* Estrutura plana (containers → blobs)
* Não tem conceito de “pastas” nativamente
* Otimizado para acesso via REST API
* Ideal para arquivos isolados (imagens, vídeos, backups)

**Estrutura:**

Storage Account

Container (ex: “imagens”)

foto1.jpg

foto2.jpg

video.mp4

**Limitações para dados:**

* Difícil organizar milhares de arquivos
* Sem suporte nativo para operações Big Data (Spark, Hadoop)
* Performance não otimizada para analytics

### Data Lake Storage Gen2 (ADLS Gen2)

**ADLS Gen2** é Blob Storage **+ hierarquia de diretórios + otimizações Big Data**.

**Diferenças-chave:**

|  |  |  |
| --- | --- | --- |
| **Aspecto** | **Blob Storage** | **Data Lake Gen2** |
| **Estrutura** | Plana (container → blobs) | Hierárquica (pastas como sistema de arquivos) |
| **Performance** | Boa | Otimizada para Big Data |
| **Permissões** | Container/blob level | Granular (nível de arquivo/pasta via ACLs) |
| **Big Data** | Suporte básico | Suporte nativo (Spark, Databricks, Synapse) |
| **Custo** | Padrão | Mesmo custo + recursos extras |

**Estrutura ADLS Gen2:**

Storage Account (com Data Lake Gen2 habilitado)

Container/Filesystem (ex: “datalake”)

bronze/

vendas/

2026-01-01.csv

2026-01-02.csv

2026-01-03.csv

silver/

vendas/

vendas\_limpo.parquet

gold/

vendas/

vendas\_agregado.parquet

**Por que usar ADLS Gen2 para Engenharia de Dados?**

1. **Organização:** Pastas lógicas (bronze/silver/gold, por data, por fonte)
2. **Performance:** Otimizado para leitura/escrita paralela (Spark)
3. **Segurança:** ACLs POSIX (permissões granulares por pasta/arquivo)
4. **Integração:** Funciona nativamente com Databricks, Synapse, ADF
5. **Custo-benefício:** Mesmo preço do Blob, mais recursos

**Decisão simples:**

* **Imagens, vídeos, backups?** → Blob Storage
* **Data Lake para analytics?** → Data Lake Gen2

## 2.3 Arquitetura Medallion: Bronze, Silver, Gold

A **arquitetura Medallion** (também chamada de **multi-hop architecture**) é um padrão moderno de organização de Data Lakes em camadas progressivas de qualidade.

### Conceito das Três Camadas

**Bronze (Bruto/Raw)**

* **Dados como chegam:** sem transformação, cópia exata da fonte
* **Formato:** Geralmente CSV, JSON, Avro (formato original)
* **Propósito:**
  + Histórico completo (auditoria)
  + Re-processamento se necessário
  + Dados imutáveis
* **Quem acessa:** Apenas engenheiros de dados

**Exemplo:**

bronze/

erp\_vendas/

2026-01-01.csv (dados brutos do ERP)

2026-01-02.csv

2026-01-03.csv

api\_clientes/

2026-01-01.json (resposta da API)

**Silver (Limpo/Refinado)**

* **Dados limpos e validados:** sem duplicatas, tipos corretos, enriched
* **Formato:** Parquet ou Delta Lake (colunar, comprimido)
* **Propósito:**
  + Base confiável para transformações
  + Dados prontos para análises exploratórias
  + Histórico completo limpo
* **Quem acessa:** Engenheiros + Data Scientists

**Transformações típicas Bronze → Silver:**

* Remover duplicatas
* Padronizar datas (YYYY-MM-DD)
* Corrigir tipos (string → int, string → date)
* Validar valores (remover nulos críticos, validar ranges)
* Enriquecer (adicionar campos calculados: idade = hoje - nascimento)

**Exemplo:**

silver/

vendas/

vendas\_limpo.parquet (dados validados e padronizados)

clientes/

clientes\_limpo.parquet

**Gold (Curado/Negócio)**

* **Dados agregados e modelados:** prontos para consumo de negócio
* **Formato:** Parquet, Delta Lake, ou tabelas em DW
* **Propósito:**
  + Dashboards e relatórios
  + Métricas de negócio pré-calculadas
  + Performance otimizada (já agregado)
* **Quem acessa:** Analistas, Power BI, executivos

**Transformações típicas Silver → Gold:**

* Agregações (vendas por mês, por produto, por região)
* Joins (vendas + clientes + produtos)
* Modelagem dimensional (fatos e dimensões)
* Métricas de negócio (receita total, ticket médio, churn rate)

**Exemplo:**

gold/

vendas\_mensais/

vendas\_por\_mes.parquet (agregado mensal)

kpis/

kpis\_executivos.parquet (métricas prontas)

### Fluxo Completo Medallion

[Sistema Origem (ERP, API)]

↓

[Ingestão]

↓

BRONZE (raw)

↓

[Limpeza, validação]

↓

SILVER (clean)

↓

[Agregação, modelagem]

↓

GOLD (curated)

↓

[Consumo: Power BI, SQL]

### Vantagens da Arquitetura Medallion

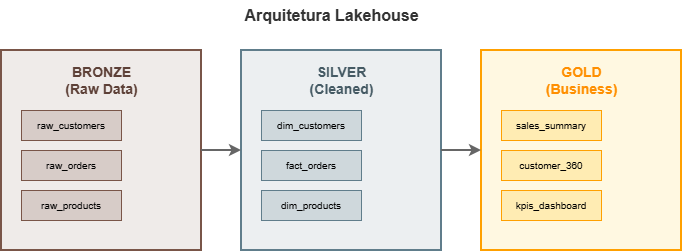
**Rastreabilidade:** Dados brutos sempre disponíveis (bronze)

**Re-processamento:** Se errar transformação, re-processa de bronze

**Governança:** Controle de acesso por camada (bronze restrito)

**Performance:** Gold já agregado = consultas rápidas

**Organização:** Clara separação de responsabilidades



**Nome do arquivo:** Junior\_Cap02\_01\_medallion\_architecture.png

*Figura 2.1 - Arquitetura Medallion: Bronze (raw), Silver (clean), Gold (curated)*

## 2.4 Hands-On: Criar Storage Account com Data Lake Gen2

Vamos colocar a mão na massa! Você vai criar seu primeiro Data Lake.

### Passo 1: Acessar Criação de Storage Account

1. **Abra o Azure Portal:** https://portal.azure.com
2. **Na barra de pesquisa, digite:** storage accounts
3. **Clique em “Storage accounts”**
4. **Clique em “+ Create”** (ou “+ Criar”)

*[INSERIR IMAGEM]*

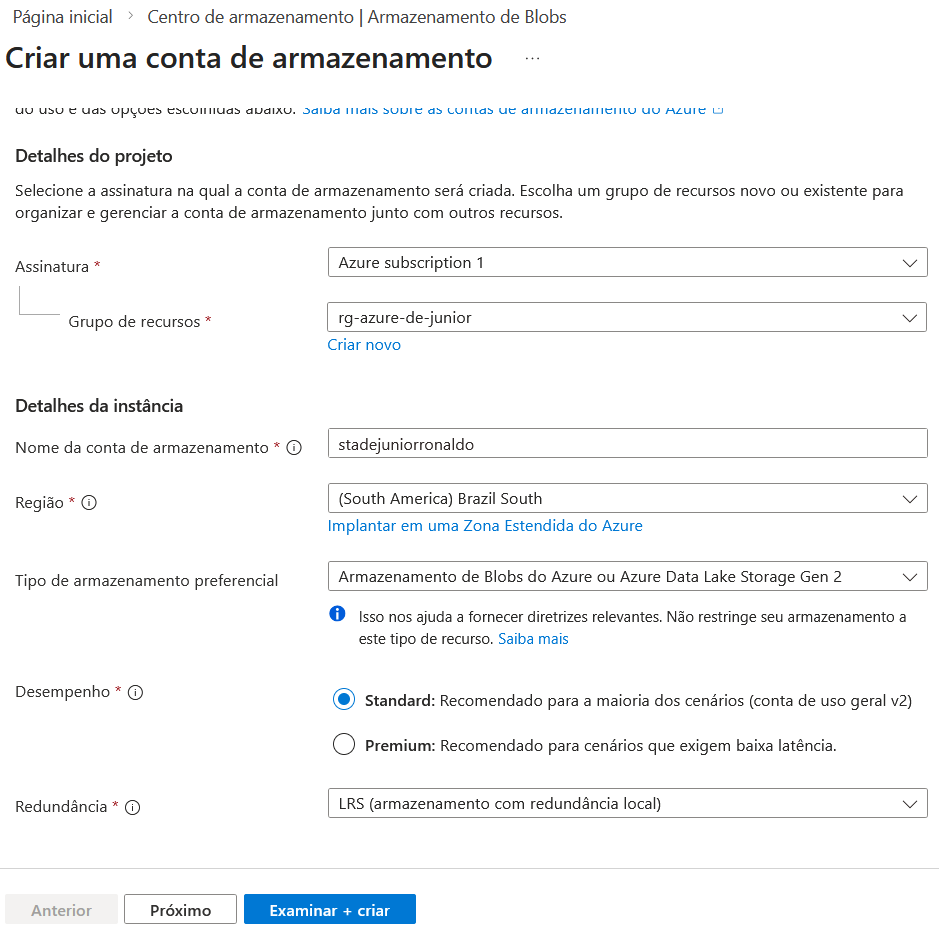
**Nome do arquivo:** Junior\_Cap02\_02\_storage\_accounts\_page.png

*Figura 2.2 - Página de Storage Accounts*

### Passo 2: Configurar Basics

**Aba “Basics”:**

1. **Subscription:** Selecione sua Free Trial
2. **Resource group:** Selecione rg-azure-de-junior (criado no Cap 1)
3. **Storage account name:** stadejunior<seu-nome>
   * Exemplo: stadejuniorronaldo
   * **Regras:**
     + 3-24 caracteres
     + Apenas letras minúsculas e números
     + Globalmente único (ninguém no mundo pode ter o mesmo nome)
   * **Dica:** Use prefixo st + projeto + seu nome/id
4. **Region:** Brazil South (São Paulo)
5. **Tipo de armazenamento preferencial:** Armazenamento de Blobs do Azure ou Azure Data Lake Storage Gen 2
6. **Performance:**
   * Standard (HDD) ← **Selecione este** (mais barato)
   * Premium (SSD) - para casos de alta performance
7. **Redundancy:** Locally-redundant storage (LRS)
   * **LRS:** 3 cópias no mesmo datacenter (mais barato)
   * **GRS:** Replicação para outra região (mais caro, maior durabilidade)
   * **Para desenvolvimento:** LRS é suficiente

**

*Figura 2.3 - Configuração básica da Storage Account*

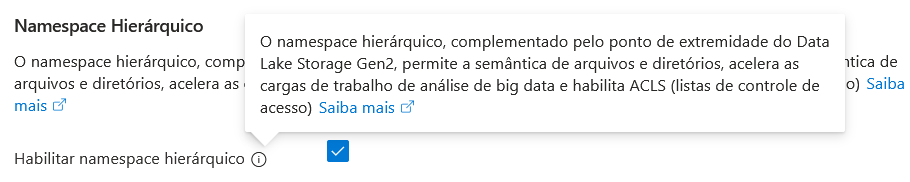
### Passo 3: Habilitar Data Lake Gen2

**Aba “Advanced”:**

1. **Clique na aba “Advanced”** (ao lado de Basics)
2. **Role até a seção “Data Lake Storage Gen2”**
3. **IMPORTANTE:** Marque a caixa:
   * **“Enable hierarchical namespace”**

**Esta é a única configuração que transforma Blob Storage em Data Lake Gen2!**

1. **Outras configurações podem deixar padrão:**
   * Secure transfer: Enabled ()
   * Blob public access: Disabled ()
   * Storage account key access: Enabled ()
   * Minimum TLS version: Version 1.2 ()

*Figura 2.4 - Habilitando Data Lake Gen2 (hierarchical namespace)*

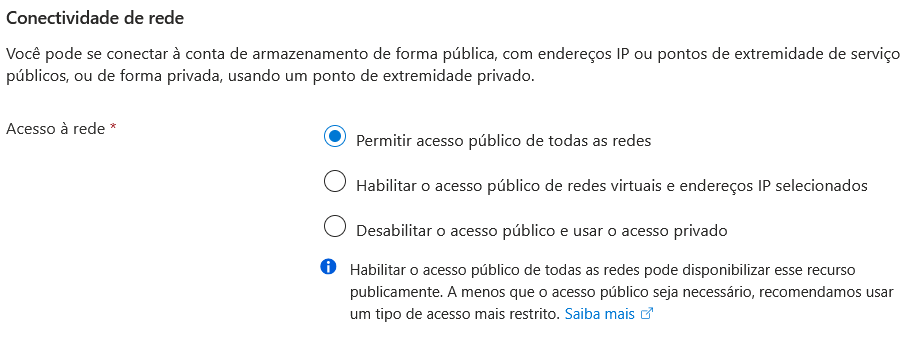
### Passo 4: Networking (manter padrão)

**Aba “Networking”:**

1. **Clique na aba “Networking”**
2. **Network access:**
   * Enable public access from all networks ← **Selecione este** (para aprendizado)
   * Enable public access from selected virtual networks and IP addresses
   * Disable public access

**Para produção:** Usaríamos redes privadas (VNets), mas para aprendizado, público é OK.

1. **Mantenha demais configurações padrão**

**

*Figura 2.5 - Configuração de rede (acesso público para desenvolvimento)*

### Passo 5: Data Protection (opcional para dev)

**Aba “Data protection”:**

Aqui você configura:

* **Soft delete:** Recuperar blobs deletados (até 7 dias)
* **Versioning:** Manter versões antigas de arquivos
* **Change feed:** Log de mudanças

**Para este exercício:** Deixe tudo desmarcado (economizar recursos).

**Para produção:** Habilite soft delete e versioning!

**Clique em “Next”**

### Passo 6: Encryption (manter padrão)

**Aba “Encryption”:**

* **Encryption type:** Microsoft-managed keys (padrão)
  + Azure gerencia chaves automaticamente
  + Opção avançada: Customer-managed keys (você controla)

**Deixe padrão e clique em “Next”**

### Passo 7: Tags (opcional)

**Aba “Tags”:**

Tags ajudam a organizar e rastrear custos.

**Exemplo:**

* **Key:** projeto | **Value:** azure-de-junior
* **Key:** ambiente | **Value:** desenvolvimento
* **Key:** capitulo | **Value:** 2

**Tags são opcionais, mas boas práticas!**

**Clique em “Next: Review + create”**

### Passo 8: Revisar e Criar

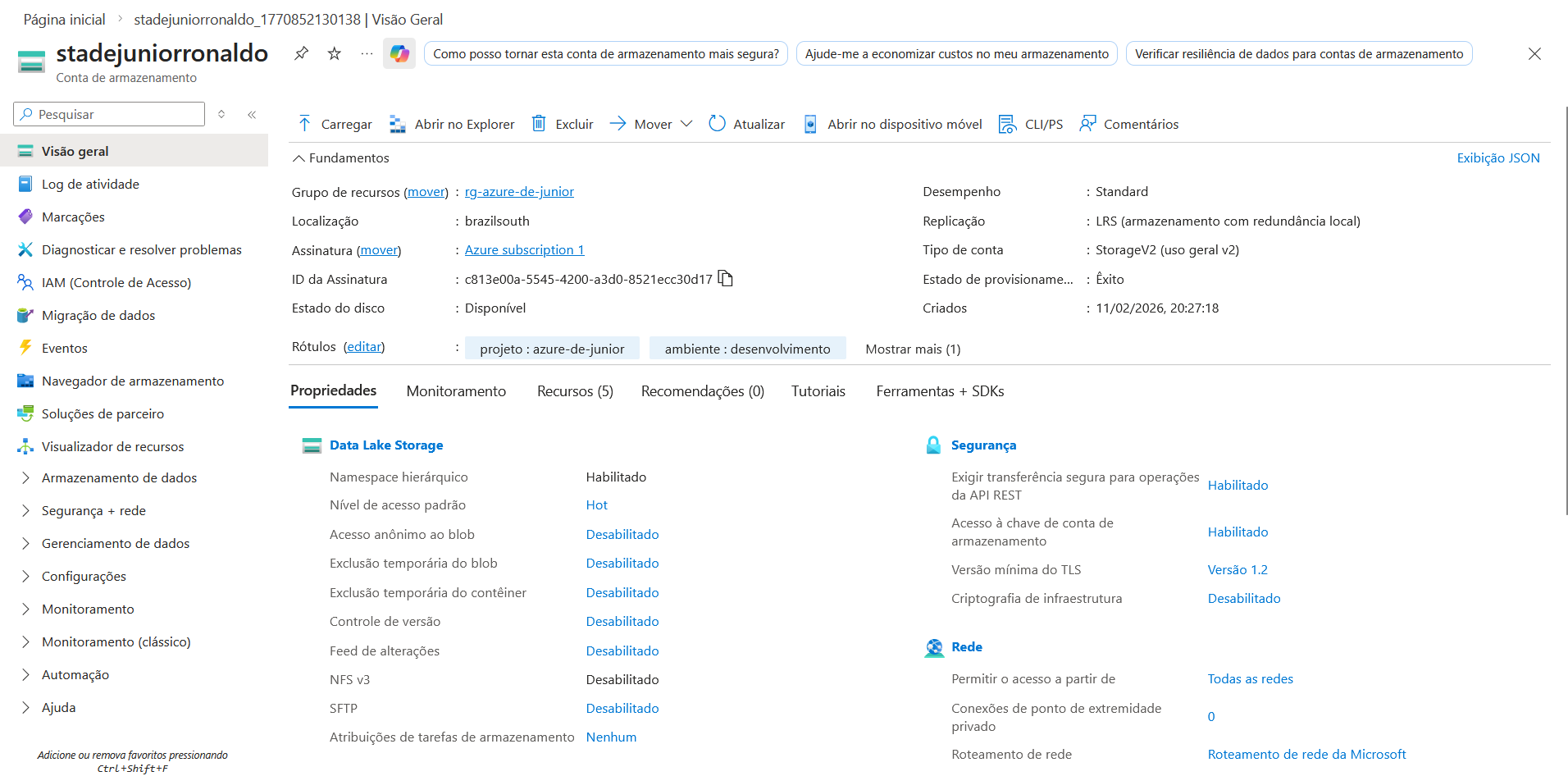
1. **Azure valida as configurações** (leva alguns segundos)
2. **Revise o resumo:**
   * Storage account name: stadejunior<seunome>
   * Region: Brazil South
   * Redundancy: LRS
   * **Hierarchical namespace: Enabled** ← Confirme que está habilitado!
3. **Custo estimado:** ~ R$ 0,10-0,50/mês (para poucos GB)
4. **Clique em “Create”**
5. **Aguarde o deployment** (30-60 segundos)
6. **Quando aparecer “Your deployment is complete”, clique em “Go to resource”**

### Passo 9: Explorar a Storage Account

Você está agora na página da sua Storage Account!

**Menu lateral esquerdo (principais seções):**

* **Overview:** Resumo, métricas, links rápidos
* **Containers:** Onde você cria filesystems (containers) do Data Lake
* **Storage browser:** Navegador web integrado (novo!)
* **Access keys:** Chaves de acesso (senhas para conectar aplicações)
* **Shared access signature (SAS):** Tokens temporários de acesso
* **Access control (IAM):** Permissões baseadas em roles

****

*Figura 2.6 - Overview da Storage Account criada*

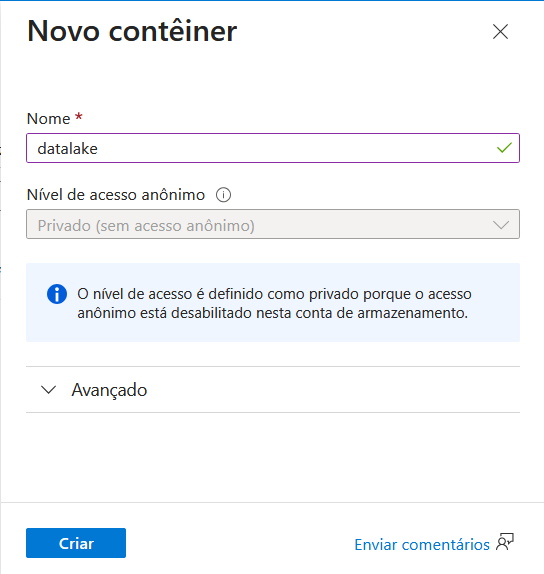
## 2.5 Criar Estrutura Medallion

Agora vamos criar a estrutura de pastas Bronze/Silver/Gold.

### Passo 1: Criar Container (Filesystem)

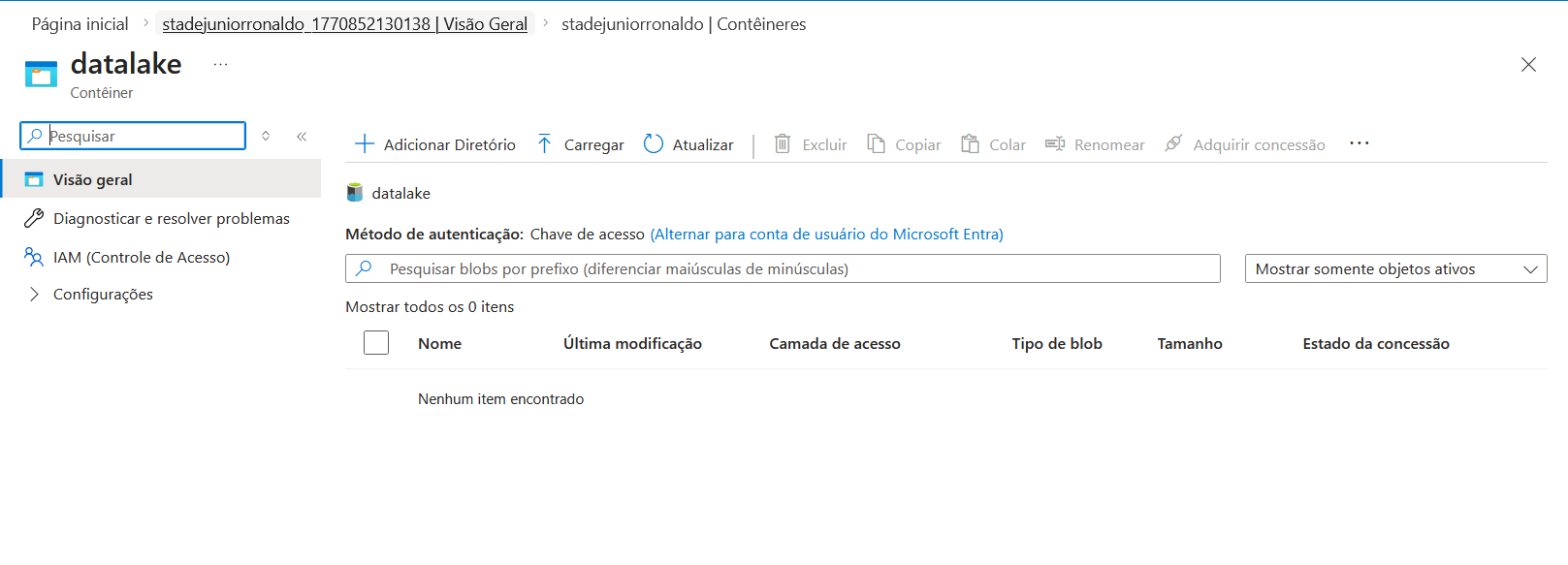
No Data Lake Gen2, **containers** são chamados de **filesystems**. São como “discos” ou “partições” dentro da Storage Account.

1. **No menu lateral esquerdo, clique em “Armazenamento de Dados” e depois em “Containers”**
2. **Clique em “+ Container”** (ou “+ Filesystem”)
3. **Preencha:**
   * **Name:** datalake
   * **Public access level:** Private (no anonymous access) ← **Mantenha privado**

****

*Figura 2.9 - Criação de container/filesystem “datalake”*

1. **Clique em “Create”**
2. **Container “datalake” aparecerá na lista**
3. **Clique no nome “datalake”** para abri-lo

*Figura 2.10 - Container “datalake” criado (vazio)*

### Passo 2: Criar Pastas Bronze, Silver, Gold

Agora vamos criar a hierarquia Medallion.

**No container “datalake”:**

1. **Clique em “Add Directory”** (ou “+ New directory”)
2. **Nome:** bronze
3. **Clique em “OK” ou “Create”**
4. **Repita para criar:**
   * silver
   * gold

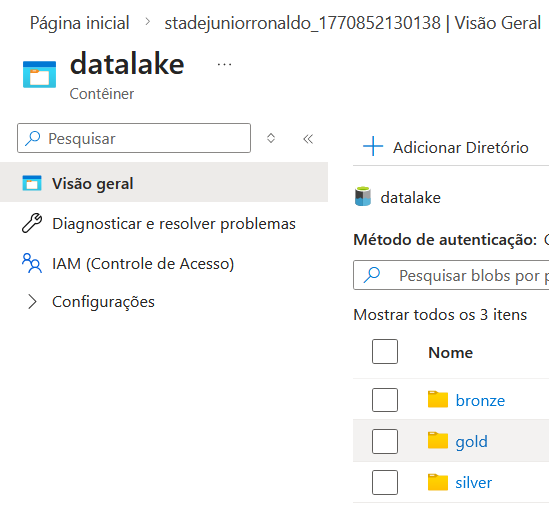
**Resultado:**

datalake/

bronze/

silver/

gold/

**

*Figura 2.11 - Estrutura Medallion criada: bronze, silver, gold*

### Passo 3: Criar Subpastas (exemplo: vendas)

Dentro de cada camada, organizamos por domínio/fonte de dados.

**Exemplo: dados de vendas**

1. **Clique na pasta “bronze”** para entrar nela
2. **Clique em “Add Directory”**
3. **Nome:** vendas
4. **Volte** (breadcrumb ou botão “Up one level”)
5. **Repita para “silver” e “gold”:** criar pasta vendas dentro de cada

**Resultado:**

datalake/

bronze/

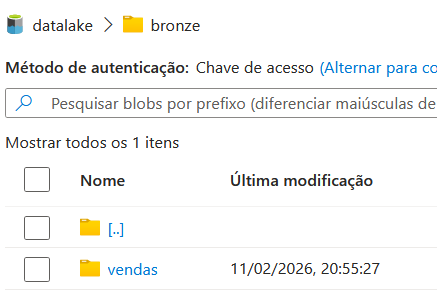
vendas/

silver/

vendas/

gold/

vendas/

**

*Figura 2.12 - Subpastas “vendas” criadas em cada camada*

**Boa prática:** Estruture por domínio de negócio:

datalake/

bronze/

vendas/

clientes/

produtos/

marketing/

silver/

vendas/

clientes/

...

gold/

...

## 2.6 Upload de Arquivos

Vamos fazer upload de um arquivo CSV de exemplo na camada Bronze.

### Preparar Arquivo de Exemplo

**Crie um arquivo CSV simples** chamado vendas\_2026-01-01.csv:

**CSV**

data,produto,quantidade,preco\_unitario,vendedor

2026-01-01,Notebook Dell,2,3500.00,João Silva

2026-01-01,Mouse Logitech,5,85.00,Maria Santos

2026-01-01,Teclado Mecânico,3,450.00,João Silva

2026-01-01,Monitor LG 27”,1,1200.00,Maria Santos

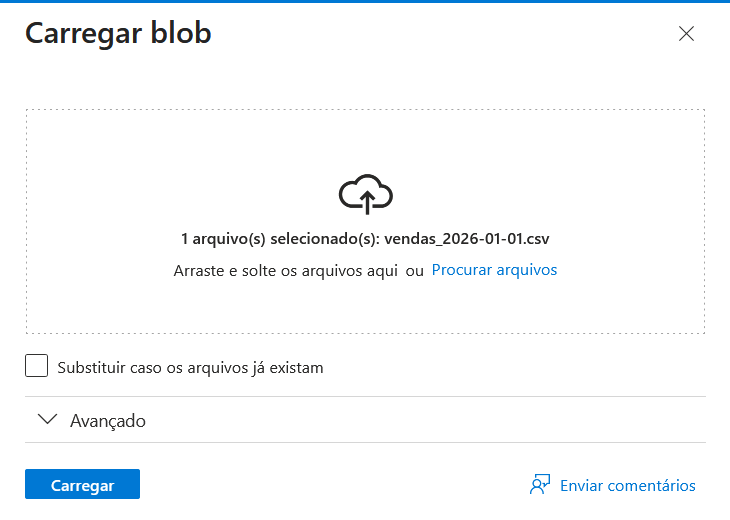
2026-01-01,Webcam HD,4,250.00,Pedro Oliveira

**Salve este arquivo no seu computador.**

**Ou use qualquer dataset CSV que você tenha!** (Kaggle, dados públicos, etc.)

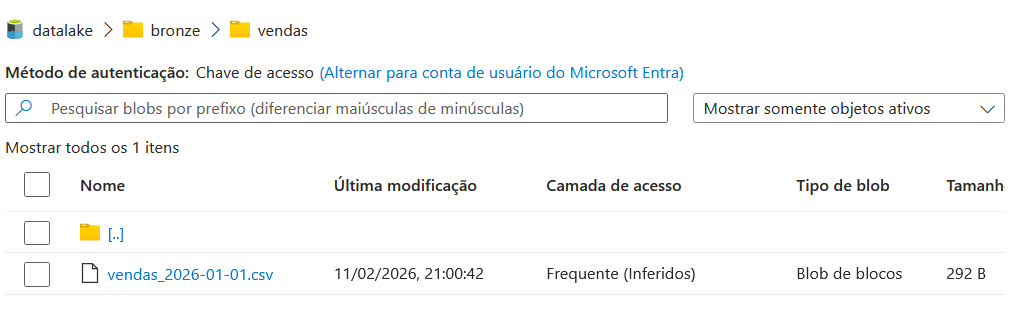
### Fazer Upload via Portal

1. **Navegue até:** datalake/bronze/vendas/
2. **Clique em “Upload”** (botão no topo)
3. **No painel lateral:**
   * **Clique em “Browse files”** ou arraste o arquivo
   * Selecione vendas\_2026-01-01.csv
   * **Overwrite if files already exist:** Deixe desmarcado (primeira vez)

**

*Figura 2.13 - Upload de arquivo CSV para camada Bronze*

1. **Clique em “Upload”**
2. **Aguarde conclusão** (alguns segundos para arquivo pequeno)
3. **Feche o painel de upload**
4. **Arquivo aparece na lista!**

*Figura 2.14 - Arquivo CSV carregado com sucesso na camada Bronze*

### Visualizar Conteúdo do Arquivo

1. **Clique no nome do arquivo:** vendas\_2026-01-01.csv
2. **No painel lateral, clique em “Download”** (ícone de download)
3. **Abra o arquivo baixado para confirmar conteúdo**

**Nota:** O portal não exibe preview de CSV diretamente, mas você pode baixar e conferir.

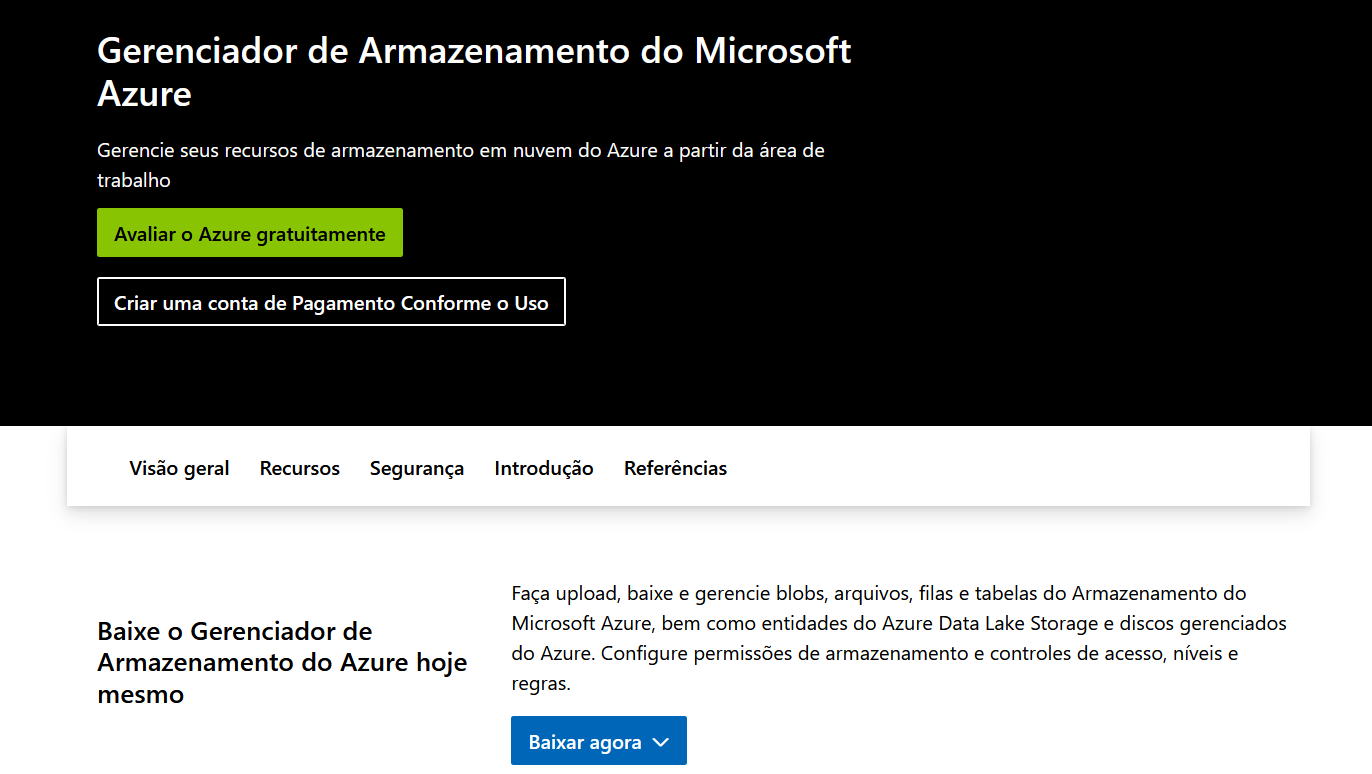
**Alternativa:** Usar **Storage Browser** (mais visual, veremos a seguir)

## 2.7 Azure Storage Explorer (Desktop App)

O **Azure Storage Explorer** é uma aplicação desktop gratuita que facilita muito a navegação e gerenciamento de Storage Accounts.

### Instalar Azure Storage Explorer

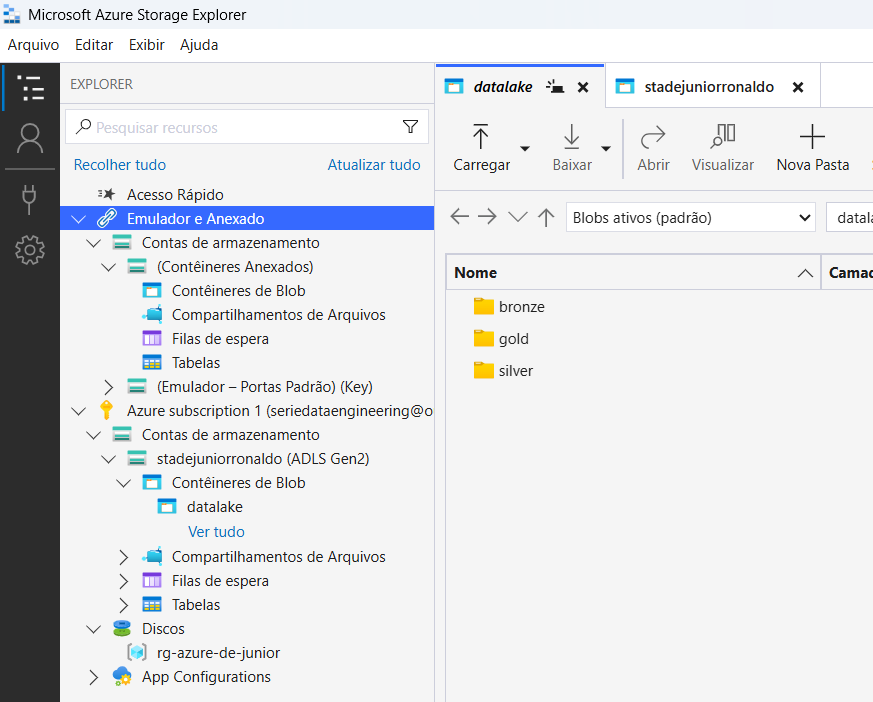
1. **Acesse:** https://azure.microsoft.com/features/storage-explorer/
2. **Clique em “Download now” ou “Baixar agora”**
3. **Escolha seu sistema operacional:** Windows, Mac ou Linux
4. **Instale o aplicativo** (processo padrão de instalação)

**

*Figura 2.15 - Página de download do Azure Storage Explorer*

### Conectar ao Azure

1. **Abra o Azure Storage Explorer**
2. **Primeira vez:** Wizard de boas-vindas aparece
3. **Clique em “Sign in to Azure...”** (ou ícone de pessoa no canto superior esquerdo)
4. **Escolha “Azure Cloud Environment”:** Azure (Global)
5. **Clique em “Next”**
6. **Faça login com sua conta Microsoft** (mesma do Azure Portal)
7. **Aguarde autenticação...**
8. **Marque sua subscription:** “Free Trial” ou “Azure subscription 1”
9. **Clique em “Open Explorer”** ou “Apply”

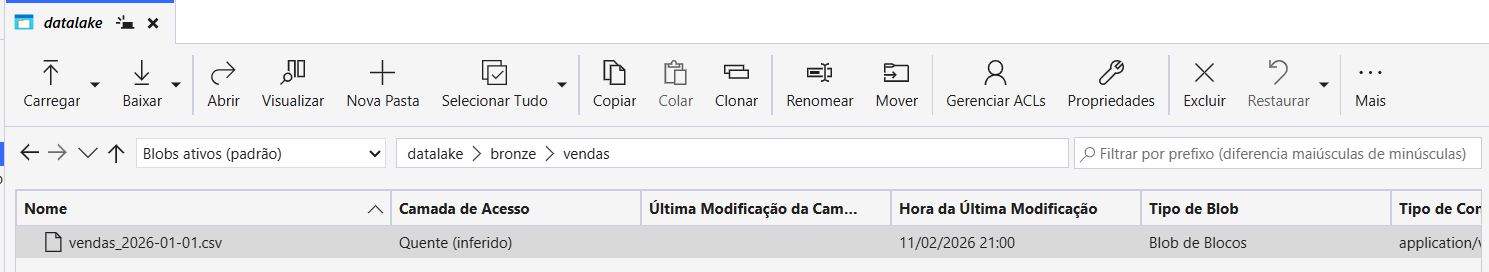
**

*Figura 2.16 - Azure Storage Explorer conectado à sua Storage Account*

Uma observação aqui. O Storage Explorer disponibiliza o recurso de emulador de Storage. Isso pode ser um pouco confuso no começo, e eventualmente talvez você tenha até clicado nele achando que entrou na sua conta, e assustou ao não ver o container e os diretórios criados anteriormente. É normal, não se preocupe.

### Navegar no Data Lake

1. **Na árvore à esquerda, expanda:**
   * **Sua subscription** (Free Trial)
   * **Storage Accounts**
   * **stadejunior[seunome]**
   * **Blob Containers**
   * **datalake**
2. **Clique em “datalake”**
3. **No painel central, você verá:** bronze, silver, gold
4. **Duplo-clique em “bronze” → “vendas”**
5. **Você verá o arquivo:** vendas\_2026-01-01.csv

**

*Figura 2.17 - Navegação no Data Lake via Storage Explorer*

### Operações Úteis no Storage Explorer

**Upload de múltiplos arquivos:**

1. Arraste e solte arquivos/pastas diretamente no painel
2. Ou clique em “Upload” → “Upload Files” ou “Upload Folder”

**Download:**

1. Clique com botão direito no arquivo
2. “Download”
3. Escolha local de destino

**Deletar:**

1. Clique com botão direito no arquivo/pasta
2. “Delete”

**Copiar/Mover:**

1. Botão direito → “Copy” ou “Cut”
2. Navegue para destino → “Paste”

**Criar pastas:**

1. Clique em “New Folder” no topo
2. Digite nome da pasta

**Propriedades e metadados:**

1. Botão direito no arquivo → “Properties”
2. Visualizar tamanho, tipo, data, metadados

## 2.8 Formatos de Arquivo: CSV, JSON, Parquet

Engenheiros de dados trabalham com diversos formatos de arquivo. Vamos entender os principais:

### CSV (Comma-Separated Values)

**Formato:** Texto plano, valores separados por vírgula (ou ponto-e-vírgula)

**Exemplo:**

**CSV**

id,nome,idade,cidade

1,João,30,São Paulo

2,Maria,25,Rio de Janeiro

**Vantagens:**

* Simples e legível por humanos
* Suportado por qualquer ferramenta (Excel, Python, SQL)
* Leve (texto puro)

**Desvantagens:**

* Sem tipos definidos (tudo é string, precisa converter)
* Sem compressão nativa (arquivos grandes)
* Lento para processar em Big Data (leitura sequencial)
* Problemas com dados contendo vírgulas ou quebras de linha

**Quando usar:**

* Dados pequenos (< 100 MB)
* Intercâmbio simples entre sistemas
* Camada Bronze (dados brutos)

### JSON (JavaScript Object Notation)

**Formato:** Texto estruturado com hierarquia (objetos e arrays)

**Exemplo:**

**JSON**

[

{

“id”: 1,

“nome”: “João”,

“idade”: 30,

“endereco”: {

“cidade”: “São Paulo”,

“estado”: “SP”

}

},

{

“id”: 2,

“nome”: “Maria”,

“idade”: 25,

“endereco”: {

“cidade”: “Rio de Janeiro”,

“estado”: “RJ”

}

}

]

**Vantagens:**

* Suporta hierarquia (objetos aninhados, arrays)
* Flexível (schema pode variar)
* Padrão de APIs REST

**Desvantagens:**

* Verboso (repete nomes de campos em cada registro)
* Sem tipos estritos
* Lento para Big Data (parsing complexo)
* Maior tamanho que formatos binários

**Quando usar:**

* Dados de APIs
* Dados semi-estruturados (hierarquia variável)
* Logs de aplicação
* Camada Bronze (dados brutos de APIs)

### Parquet (Formato Colunar)

**Formato:** Binário, otimizado para analytics, armazena dados por coluna

**Estrutura:**

Arquivo Parquet (binário, não legível)

Metadados (schema, estatísticas)

Coluna “id”: [1, 2, 3, ...]

Coluna “nome”: [“João”, “Maria”, ...]

Coluna “idade”: [30, 25, ...]

...

**Vantagens:**

* **Altamente comprimido** (10x menor que CSV)
* **Leitura colunar rápida** (ler apenas colunas necessárias)
* **Schema definido** (tipos, nullable)
* **Otimizado para Spark, Databricks, Synapse**
* Estatísticas embutidas (min/max por coluna) → query pushdown

**Desvantagens:**

* Não legível por humanos (binário)
* Não editável manualmente
* Precisa ferramentas específicas para ler (Spark, Pandas, DuckDB)

**Quando usar:**

* **Camadas Silver e Gold** (dados processados)
* Big Data (> 1 GB)
* Analytics (Spark, Synapse, Databricks)
* Armazenamento de longo prazo (compressão)

**Por que Parquet é rei em Data Lakes?**

Exemplo prático:

|  |  |  |  |
| --- | --- | --- | --- |
| **Formato** | **Tamanho** | **Tempo leitura (1 coluna de 10)** | **Tempo leitura (todas colunas)** |
| **CSV** | 1 GB | 10s | 10s |
| **Parquet** | 100 MB | 1s | 5s |

**Parquet lê apenas colunas necessárias!** Se sua query precisa de 2 de 50 colunas, Parquet lê só essas 2.

### Delta Lake (Parquet + ACID)

**Delta Lake** é um formato que adiciona camada de gerenciamento sobre Parquet:

**Recursos extras:**

* **ACID transactions** (atomicidade, consistência)
* **Time travel** (versões históricas: ler dados de 1 semana atrás)
* **Schema enforcement** (validação automática de tipos)
* **Schema evolution** (adicionar colunas sem quebrar)
* **Upserts e deletes** (modificar dados de forma eficiente)

**Quando usar:**

* Camadas Silver e Gold
* Dados que precisam ser atualizados (não apenas append)
* Histórico de versões importante
* Databricks, Synapse

**Abordaremos Delta Lake em detalhes no Capítulo 8!**

### Resumo de Formatos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Formato** | **Tipo** | **Compressão** | **Performance** | **Uso Recomendado** |
| **CSV** | Texto | Baixa | Lenta | Bronze (raw), intercâmbio |
| **JSON** | Texto | Baixa | Lenta | Bronze (APIs), semi-estruturado |
| **Parquet** | Binário colunar | Alta | Rápida | Silver, Gold, analytics |
| **Delta Lake** | Parquet + ACID | Alta | Rápida | Silver, Gold, tabelas gerenciadas |

**Regra de ouro:**

* **Bronze:** CSV/JSON (como chega)
* **Silver/Gold:** Parquet ou Delta Lake

## 2.9 Segurança e Permissões

Data Lakes armazenam dados sensíveis. Precisamos controlar quem acessa o quê.

### Tipos de Autenticação

**1. Access Keys (Chaves de Acesso)**

* Como uma “senha admin” da Storage Account
* Acesso total (read/write/delete em tudo)
* **Risco:** Se vazar, comprometeu tudo
* **Uso:** Apenas em desenvolvimento local (NUNCA comitar no Git!)

**2. Shared Access Signature (SAS)**

* Token temporário com permissões específicas
* Exemplo: “acesso read-only à pasta bronze/vendas/ por 24h”
* **Uso:** Compartilhar acesso limitado

**3. Azure Active Directory (AAD) + RBAC**

* Autenticação por usuário/service principal
* Roles: Storage Blob Data Reader, Storage Blob Data Contributor, Owner
* **Uso:** Produção, controle granular

**4. ACLs (Access Control Lists) - Nível de Arquivo/Pasta**

* POSIX-like permissions (read, write, execute) por arquivo/pasta
* Apenas no Data Lake Gen2 (não funciona em Blob puro)
* **Uso:** Controle fino (ex: “usuário X só lê bronze, usuário Y lê silver e gold”)

### RBAC: Roles Importantes

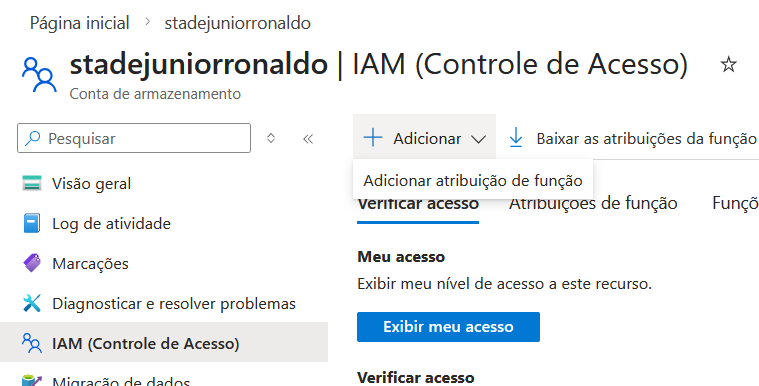
|  |  |  |
| --- | --- | --- |
| **Role** | **Permissões** | **Quando usar** |
| **Storage Blob Data Reader** | Ler blobs e listar containers | Analistas, cientistas (read-only) |
| **Storage Blob Data Contributor** | Ler, escrever, deletar blobs | Engenheiros de dados (read/write) |
| **Storage Blob Data Owner** | Tudo + gerenciar ACLs | Administradores |
| **Contributor (resource level)** | Gerenciar Storage Account (não dados) | DevOps, criar recursos |

### Hands-On: Configurar RBAC (Permissão de Leitura)

Vamos dar permissão de leitura para sua conta de usuário via RBAC.

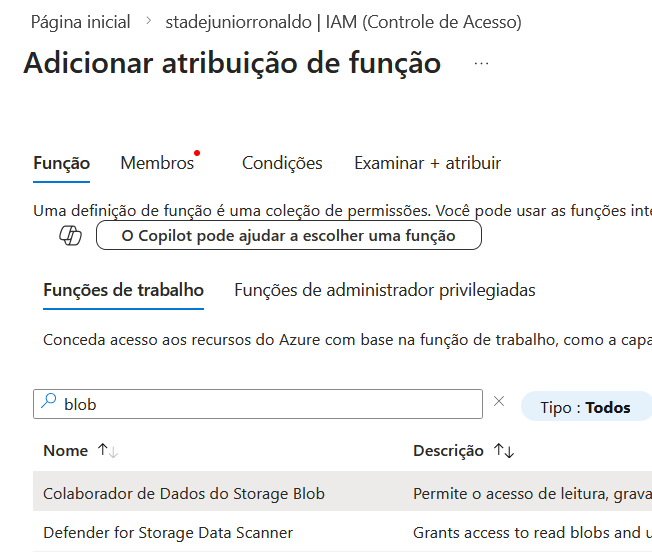
**Cenário:** Você quer acessar dados via Python/Databricks usando sua identidade AAD (sem access keys).

1. **No Azure Portal, vá para sua Storage Account:** stadejunior[seunome]
2. **Menu lateral esquerdo → “Access Control (IAM)”**
3. **Clique em “+ Add” → “Add role assignment”**

**

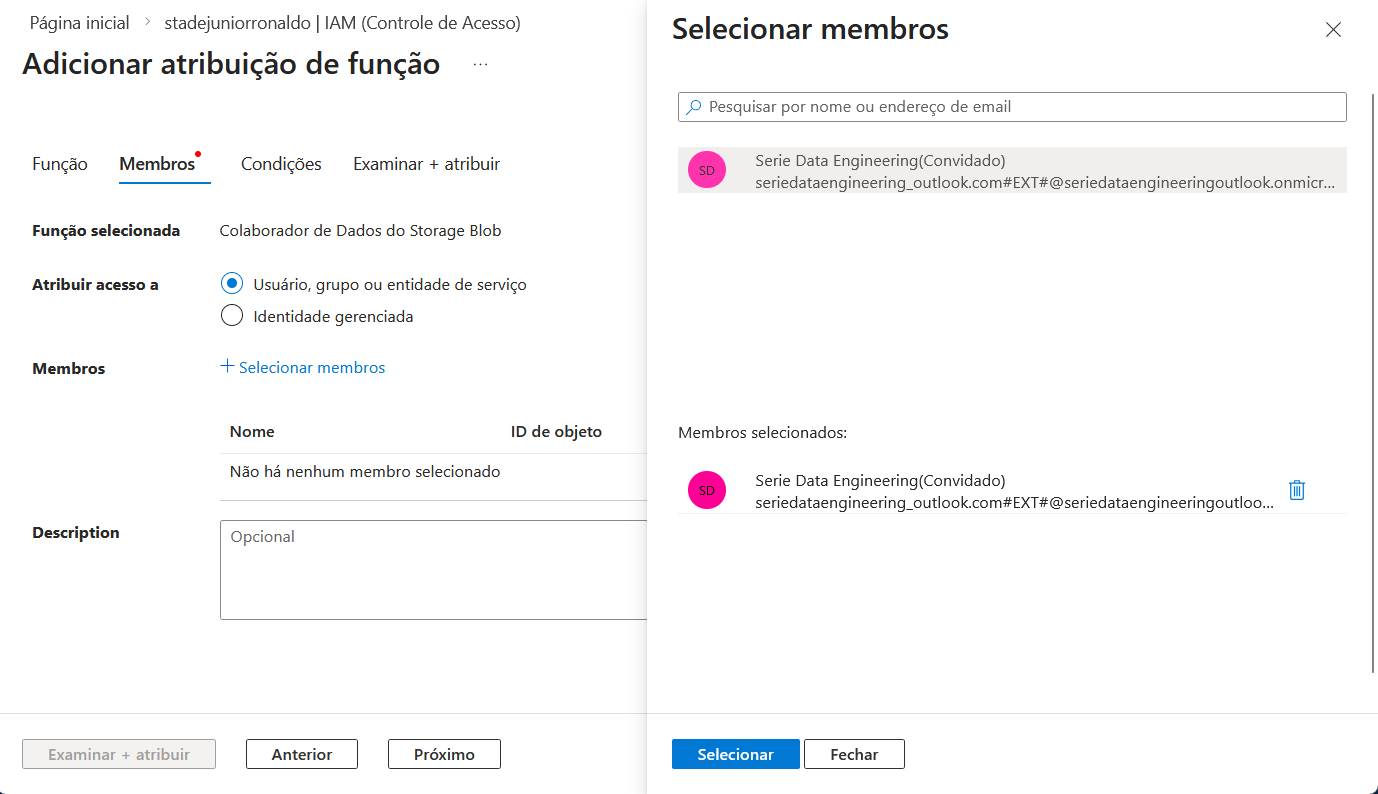
*Figura 2.18 - Adicionar role assignment em Access Control (IAM)*

1. **Aba “Role”:**
   * Busque: Storage Blob Data Contributor
   * Selecione o role
   * Clique em “Next”

****

*Figura 2.19 - Seleção do role “Storage Blob Data Contributor”*

1. **Aba “Members”:**
   * **Assign access to:** User, group, or service principal
   * Clique em “+ Select members”
   * No painel lateral, busque seu email (conta Azure)
   * Selecione seu usuário
   * Clique em “Select”

**

*Figura 2.20 - Seleção de membro (seu usuário)*

1. **Clique em “Next”**
2. **Aba “Review + assign”:**
   * Revise: Role = Storage Blob Data Contributor, Member = seu usuário
   * Clique em “Review + assign”
3. **Aguarde alguns segundos (propagação de permissões)**

**Pronto!** Agora você tem permissão de ler e escrever dados usando sua identidade AAD (sem precisar de access keys).

### ACLs: Permissões Granulares

**ACLs** (somente Data Lake Gen2) permitem definir permissões nível de pasta/arquivo.

**Exemplo de uso:**

* **Pasta bronze/:** Apenas engenheiros de dados podem ler/escrever
* **Pasta gold/:** Analistas podem ler, mas não escrever

**Permissões ACL:**

* **Read (r):** Ler conteúdo do arquivo / listar pasta
* **Write (w):** Modificar arquivo / adicionar arquivos na pasta
* **Execute (x):** Acessar pasta (necessário para navegar hierarquia)

**Como configurar ACL:**

1. **No Storage Explorer ou Portal, navegue até a pasta**
2. **Botão direito → “Manage Access”** (Storage Explorer)
   * Ou no Portal: Clique na pasta → “Manage ACL”
3. **Adicione usuário/grupo e defina permissões:** r, w, x

**Não vamos configurar ACLs agora (complexo para início), mas saiba que existe!**

## 2.10 Boas Práticas de Organização

### Convenções de Nomenclatura

**Pastas:**

* Minúsculas, separadas por underscore: vendas\_online, clientes\_ativos
* Ou hífens: vendas-online, clientes-ativos
* **Evite espaços:** “Vendas Online” → dificulta scripts

**Arquivos:**

* Incluir data no nome: vendas\_2026-01-01.csv
* Formato: <dominio>\_<YYYY-MM-DD>.<extensao>
* Ou timestamp: vendas\_20260101\_153045.csv

**Particionamento por data:**

Boa prática para grandes volumes:

bronze/vendas/

year=2026/

month=01/

day=01/

vendas.csv

day=02/

vendas.csv

month=02/

...

**Por que particionar?**

* Queries podem “pular” partições irrelevantes
* Exemplo: “vendas de janeiro 2026” → lê apenas year=2026/month=01/
* Performance 10-100x melhor em datasets grandes

### Governança de Dados

**Catálogo:**

* Documente o que cada pasta contém
* Use Azure Purview (veremos em capítulos futuros) ou README.md

**Retenção:**

* Defina políticas de retenção (quanto tempo manter bronze/silver/gold)
* Exemplo: Bronze 90 dias, Silver 1 ano, Gold permanente

**Versionamento:**

* Use Delta Lake para versionamento automático
* Ou nomeie arquivos com versão: vendas\_v1.parquet, vendas\_v2.parquet

**Backup:**

* Configure geo-replication para dados críticos (GRS)
* Ou faça cópia para outra Storage Account periodicamente

## 2.11 Gerenciamento de Custos

### Precificação do Azure Storage

**Componentes de custo:**

1. **Armazenamento (storage):**
   * Hot tier: ~ R$ 0,10/GB/mês (dados acessados frequentemente)
   * Cool tier: ~ R$ 0,05/GB/mês (acesso raro, > 30 dias)
   * Archive tier: ~ R$ 0,01/GB/mês (arquivamento, > 180 dias)
2. **Operações (transactions):**
   * Write: ~ R$ 0,000025 por 10.000 operações
   * Read: ~ R$ 0,000002 por 10.000 operações
3. **Transferência de dados:**
   * Egress (saída da região): ~ R$ 0,50/GB (primeiros 100 GB/mês grátis)
   * Ingress (entrada): Grátis

**Exemplo de custo mensal:**

* 10 GB de dados (hot tier)
* 100.000 writes + 500.000 reads
* Sem egress (dados ficam no Azure)

**Custo:** ~ R$ 1,50/mês

**Para aprendizado (< 50 GB):** < R$ 5/mês

### Otimizar Custos

Como falamos de alguns valores, agora vamos introduzir algo que vale um ponto de atenção. Apesar de diversos recursos terem custo baixo, ou serem até “free-tier”, é fundamental que você tenha plena consciência de que abusos não são tolerados. Usou pagou. A nuvem é fria e calculista, lembre-se sempre disso. Tanto em ambiente de aprendizado quanto no trabalho, cuide para não tomar sustos, ou causar sustos. Veremos ao longo da série a disciplina de Finops. Mas desde já, ficam a dicas abaixo.

**1. Use tiers apropriados:**

* Bronze (raramente acessado depois de processado) → Cool tier
* Silver/Gold (queries frequentes) → Hot tier

**2. Delete dados desnecessários:**

* Bronze após 90 dias (se Silver já processado)
* Arquivos temporários de testes

**3. Comprima arquivos:**

* Use Parquet com compressão (Snappy, Gzip)
* 10x menos espaço que CSV

**4. Lifecycle management:**

* Configure regras automáticas: “mover bronze para cool tier após 30 dias”

**Como configurar:**

1. Storage Account → “Lifecycle management”
2. “+ Add rule”
3. Condição: “Blob não modificado há 30 dias”
4. Ação: “Move to cool tier”

## 2.12 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar diferença entre Blob Storage e Data Lake Gen2
* [ ] Descrever arquitetura Medallion (Bronze/Silver/Gold)
* [ ] Comparar formatos CSV, JSON, Parquet
* [ ] Entender tipos de autenticação (Access Keys, SAS, AAD, ACLs)

**Práticos:**

* [ ] Criar Storage Account com Data Lake Gen2 habilitado
* [ ] Criar container/filesystem
* [ ] Criar estrutura de pastas (bronze/silver/gold)
* [ ] Fazer upload de arquivos via Portal
* [ ] Instalar e usar Azure Storage Explorer
* [ ] Configurar RBAC (role assignment)

**Organização:**

* [ ] Aplicar convenções de nomenclatura
* [ ] Particionar dados por data (conceito)
* [ ] Entender otimização de custos (tiers, lifecycle)

## 2.13 Próximos Passos

No **Capítulo 3**, você vai:

Provisionar Azure SQL Database

Importar dados de exemplo (Northwind ou AdventureWorks)

Modelar dimensões e fatos (Star Schema básico)

Escrever queries SQL essenciais (JOINs, agregações, CTEs)

Conectar Power BI ao SQL Database

**Preparação:**

* Mantenha sua Storage Account e dados de vendas (usaremos no Cap 4)
* Revise conceitos básicos de SQL (SELECT, WHERE, JOIN)
* Se possível, instale SQL Server Management Studio (SSMS) ou Azure Data Studio

## 2.14 Recursos Complementares

**Documentação Oficial:**

* [Azure Data Lake Storage Gen2](https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-introduction)
* [Hierarchical Namespace](https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-namespace)
* [Access Control in ADLS Gen2](https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-access-control)

**Azure Storage Explorer:**

* [Download](https://azure.microsoft.com/features/storage-explorer/)
* [Documentation](https://learn.microsoft.com/azure/vs-azure-tools-storage-manage-with-storage-explorer)

**Formatos de Arquivo:**

* [Parquet Documentation](https://parquet.apache.org/docs/)
* [Delta Lake](https://delta.io/)

**Artigos:**

* [Medallion Architecture](https://www.databricks.com/glossary/medallion-architecture)
* [Data Lake Best Practices](https://learn.microsoft.com/azure/architecture/data-guide/scenarios/data-lake)

## 2.15 Exercícios Práticos

**Exercício 1: Estrutura Multi-Domínio**

Expanda sua arquitetura Medallion para incluir 3 domínios:

1. vendas
2. clientes
3. produtos

Crie pastas para cada um em bronze/silver/gold.

**Exercício 2: Upload de Dataset Real**

1. Baixe um dataset CSV do Kaggle:
   * [Kaggle Datasets](https://www.kaggle.com/datasets)
   * Sugestão: E-commerce sales, Customer churn, etc.
2. Faça upload na pasta bronze/<dominio>/
3. Documente em um README.md:
   * Fonte do dataset
   * Descrição das colunas
   * Data de ingestão

**Exercício 3: Conversão CSV → Parquet**

Usando Python + Pandas localmente:

**PYTHON**

import pandas as pd

# Ler CSV

df = pd.read\_csv('vendas\_2026-01-01.csv')

# Salvar como Parquet

df.to\_parquet('vendas\_2026-01-01.parquet', compression='snappy')

# Comparar tamanhos

import os

csv\_size = os.path.getsize('vendas\_2026-01-01.csv')

parquet\_size = os.path.getsize('vendas\_2026-01-01.parquet')

print(f”CSV: {csv\_size} bytes”)

print(f”Parquet: {parquet\_size} bytes ({parquet\_size/csv\_size\*100:.1f}%)”)

Faça upload do Parquet na pasta silver/vendas/.

**Resultado esperado:** Parquet 50-90% menor que CSV.

**Exercício 4: Exploração via Storage Explorer**

1. Baixe 3 arquivos CSV diferentes
2. Faça upload via Storage Explorer (arrastar e soltar)
3. Renomeie um arquivo
4. Delete outro arquivo
5. Baixe o terceiro arquivo

**Exercício 5: Pesquisa de Custos**

No Azure Portal:

1. Vá para Cost Management + Billing
2. Encontre custo da sua Storage Account (últimos 7 dias)
3. Identifique quanto gastou em armazenamento vs operações

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Blob Storage vs Data Lake Gen2:** Hierarquia de pastas + otimizações Big Data

**Arquitetura Medallion:** Bronze (raw) → Silver (clean) → Gold (curated)

**Criação de Storage Account:** Com hierarchical namespace habilitado

**Upload e navegação:** Via Portal e Storage Explorer

**Formatos de arquivo:** CSV (texto) → Parquet (colunar, comprimido)

**Segurança:** RBAC (roles) e ACLs (permissões granulares)

**Organização:** Nomenclatura, particionamento, governança

**Custos:** Tiers (hot/cool/archive), lifecycle management

**Próximo capítulo:** Azure SQL Database - modelagem e queries fundamentais!

# Capítulo 3: Azure SQL Database - Fundamentos

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender quando usar SQL Database vs Data Lake
* Provisionar Azure SQL Database
* Conectar ao banco usando Azure Data Studio
* Modelar dados com Star Schema (fatos e dimensões)
* Importar dados de exemplo (AdventureWorks ou custom)
* Escrever queries SQL essenciais (JOINs, agregações, CTEs, window functions)
* Configurar backup e restore
* Monitorar performance básica

**Tempo estimado:** 3-4 horas

**Custo Azure:** R$ 20-50/mês (tier básico) ou R$ 0 com serverless pausado

## 3.1 Bancos Relacionais em Engenharia de Dados

### Por que ainda usar SQL Database?

Com Data Lakes e Big Data, você pode perguntar: **“Por que usar bancos relacionais tradicionais?”**

**Resposta:** Cada ferramenta tem seu papel.

|  |  |  |
| --- | --- | --- |
| **Aspecto** | **SQL Database** | **Data Lake** |
| **Estrutura** | Rígida (schema fixo) | Flexível (schema-on-read) |
| **Transações** | ACID (atomicidade, consistência) | Eventual consistency |
| **Consultas** | SQL otimizado, índices | Scan completo (sem índices) |
| **Volume** | Até alguns TB | Petabytes |
| **Custo** | Alto (R$ 500-5000/mês para DW) | Baixo (R$ 0,10/GB/mês) |
| **Uso típico** | DW final, apps transacionais | Raw data, Big Data analytics |

**Quando usar SQL Database:**

* Data Warehouse final (camada Gold servindo Power BI)
* Dados fortemente relacionais (muitos JOINs complexos)
* Aplicações transacionais (e-commerce, CRM)
* Queries conhecidas e repetitivas (dashboards)
* Necessidade de índices e performance previsível

**Quando usar Data Lake:**

* Armazenamento bruto (bronze/silver)
* Dados não estruturados (JSON, logs, imagens)
* Big Data (> 10 TB)
* Análises exploratórias (Data Science)
* Machine Learning

**Arquitetura moderna comum:**

[Fontes]

↓

Data Lake (bronze/silver) ← Armazenamento barato

↓

Transformações (Spark/Databricks)

↓

SQL Database (gold) ← Servir dashboards

↓

Power BI / Apps

## 3.2 Azure SQL Database: Visão Geral

### O que é Azure SQL Database?

**Azure SQL Database** é um banco de dados relacional **gerenciado** (PaaS - Platform as a Service) baseado no SQL Server.

**“Gerenciado” significa:**

* Azure cuida de: patches, backups, alta disponibilidade, segurança OS
* Você foca em: schema, queries, otimizações
* Sem necessidade de gerenciar VMs ou instalar SQL Server

**Compatibilidade:**

* 99% compatível com SQL Server on-premises
* Mesma T-SQL (Transact-SQL)
* Ferramentas: SSMS, Azure Data Studio, Visual Studio

### Tiers de Serviço (Pricing)

Azure SQL Database oferece diferentes tiers para diferentes necessidades:

**1. DTU-based (Database Transaction Units)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tier** | **Quando usar** | **Custo/mês** | **Performance** |
| **Basic** | Dev/test, baixíssimo uso | ~R$ 20 | 5 DTUs (muito lento) |
| **Standard** | Apps pequenos/médios | R$ 50-500 | 10-3000 DTUs |
| **Premium** | Apps críticos, alta performance | R$ 500-5000+ | 125-4000 DTUs |

**DTU = Database Transaction Unit** (abstração de CPU + memória + I/O)

**2. vCore-based (Mais flexível, recomendado)**

Você escolhe:

* **Compute:** vCores (2, 4, 8, 16...)
* **Memória:** Proporcional aos vCores
* **Storage:** Independente (1 GB até 4 TB)

**Tiers vCore:**

|  |  |  |
| --- | --- | --- |
| **Tier** | **Quando usar** | **Custo/mês (2 vCores)** |
| **General Purpose** | Maioria dos workloads | R$ 400-800 |
| **Business Critical** | Baixa latência, alta disponibilidade | R$ 1200-2000 |
| **Hyperscale** | Bancos > 100 GB, escala rápida | R$ 600-1500 |

**3. Serverless (Melhor para dev/test!)**

**Modelo:** Paga por segundo de uso + armazenamento

* **Auto-pausa:** Desliga se inativo (1h de inatividade)
* **Auto-resume:** Liga quando recebe query (alguns segundos de delay)
* **Custo:** R$ 0 quando pausado! Apenas storage (~R$ 5/mês para 10 GB)

**Ideal para:**

* Desenvolvimento e testes
* Ambientes usados esporadicamente
* Provas de conceito (POCs)
* Este livro!

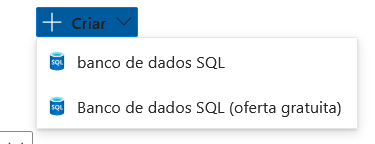
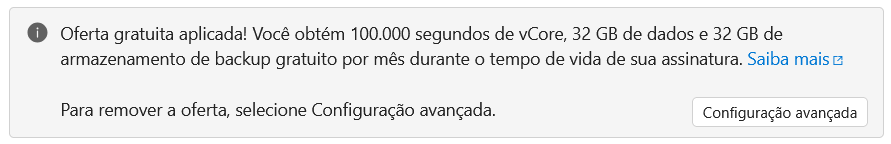
**Para este capítulo, usaremos: Serverless (vCore)** → Custo ~R$ 5-15/mês

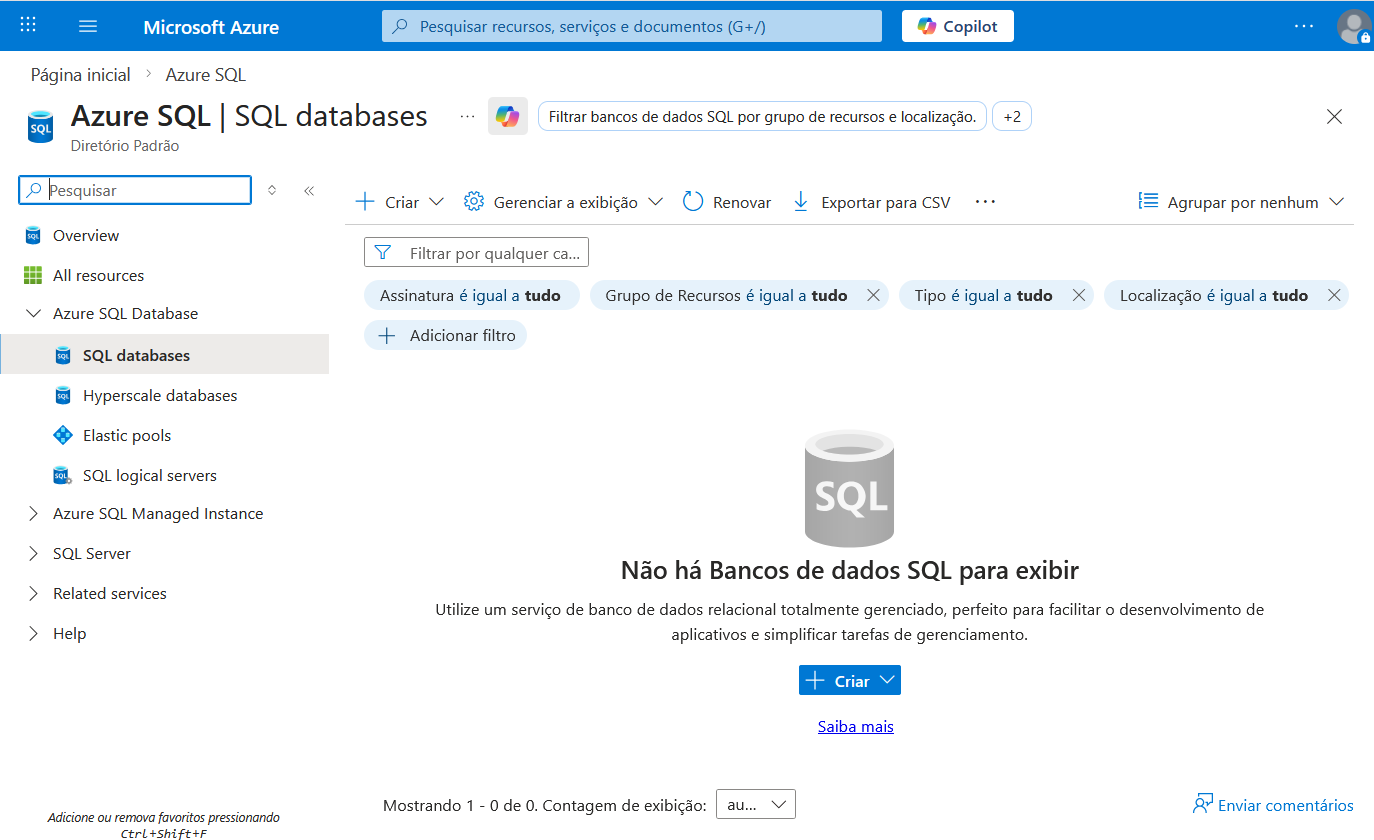
## 3.3 Hands-On: Criar Azure SQL Database

Vamos provisionar nosso primeiro banco de dados!

### Passo 1: Iniciar Criação

1. **Azure Portal:** https://portal.azure.com
2. **Barra de pesquisa:** sql databases
3. **Clique em “SQL databases”**
4. **Clique em “+ Create”**
   1. Aqui temos uma oferta, que eventualmente pode não estar mais disponível no momento da leitura ou do uso do livro. Se esta for a sua sorte, clique na opção com “oferta gratuita”:



*Figura 3.1 - Página de SQL Databases*

### Passo 2: Configurar Basics

**Aba “Basics”:**

1. **Subscription:** Free Trial
2. **Resource group:** rg-azure-de-junior
3. **Database name:** db-vendas-junior
   * Nome descritivo, minúsculas, hífens
4. **Server:** Clique em **“Create new”**

**Importante:** SQL Database precisa de um **SQL Server lógico** (container para databases)

**No popup “Create SQL Database Server”:**

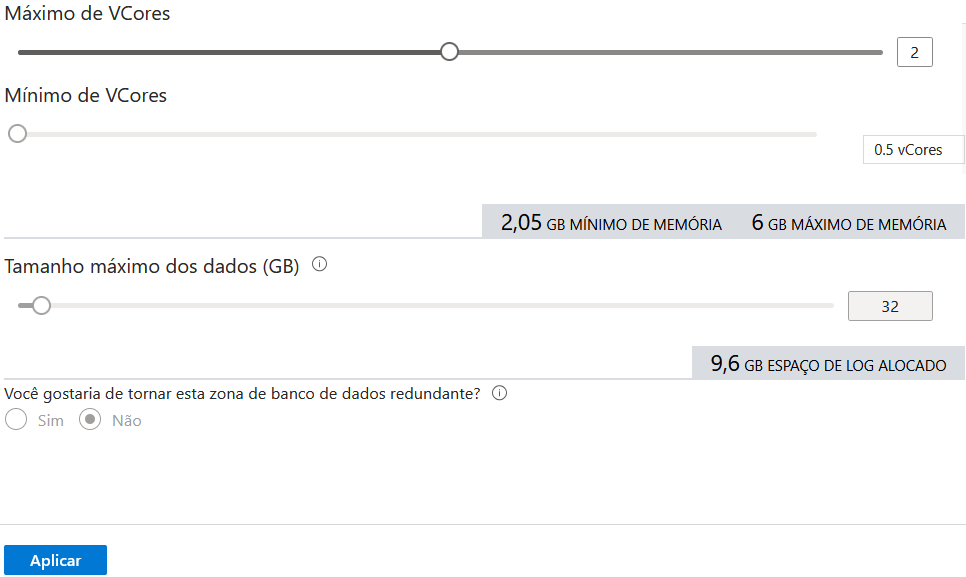
* + **Server name:** sql-azuredejunior-<seunome>
    - Exemplo: sql-azuredejunior-ronaldo
    - Globalmente único (como storage account)
    - Minúsculas, hífens, alfanumérico
  + **Location:** Brazil South
  + **Authentication method:** **Use SQL authentication**
  + **Server admin login:** sqladmin
    - Nome do usuário admin (pode ser qualquer um)
  + **Password:** Escolha senha forte
    - Mínimo 8 caracteres
    - Letras maiúsculas, minúsculas, números, símbolos
    - Exemplo: Azur3@2026!
    - **ANOTE ESSA SENHA!** Você vai precisar para conectar
  + **Confirm password:** Digite a senha novamente
  + **Clique em “OK”**

1. **Want to use SQL elastic pool?** → No (deixe desmarcado)
   * Elastic pools são para múltiplos databases compartilhando recursos
2. **Compute + storage:** Clique em **“Configure database”**

### Passo 3: Escolher Tier Serverless

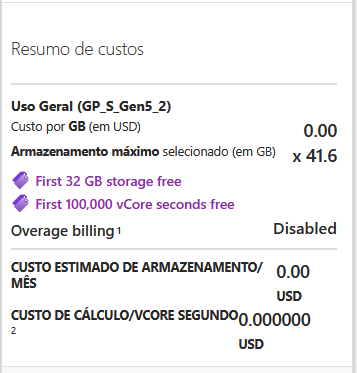
**Na tela “Configure”:**

1. **Service tier:** Aceite os padrões da oferta.

**

*Figura 3.3 – Caso a oferta esteja disponível aceite os padrões*

1. **Configurações Serverless (caso não tenha oferta disponível):**
   * **Compute Hardware:** Gen5
   * **vCores (min):** 0.5 (mínimo)
   * **vCores (max):** 2 (suficiente para aprendizado)
   * **Auto-pause delay:** 1 hour (pausa após 1h inativo)
   * **Data max size:** 10 GB (suficiente para datasets de exemplo)
2. **Custo estimado:**
   * Compute: ~R$ 10-15/mês (se usar 8h/dia)
   * Storage: ~R$ 5/mês (10 GB)
   * **Pausado:** R$ 0 de compute! Apenas Storage

**

*Figura 3.5 - Configuração da oferta padrão*

1. **Clique em “Apply”**

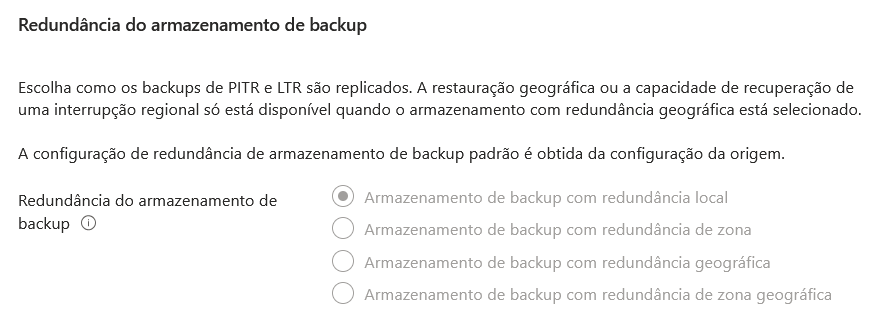
### Passo 4: Backup e Redundancy

**Aba “Backup storage redundancy”:**

1. **Backup storage redundancy:**
   * Selecione **“Locally-redundant backup storage”** (mais barato)
   * Opções:
     + **Locally-redundant (LRS):** Backups na mesma região (R$ 0,10/GB/mês)
     + **Geo-redundant (GRS):** Backups replicados para outra região (R$ 0,20/GB/mês)

**Para desenvolvimento:** LRS é OK

No entanto para a oferta, a configuração é um pouco diferente, você deve encontrar esse cenário abaixo:

****

*Figura 3.6 - Configuração de redundância de backup*

A explicação no (i): Não é possível configurar a redundância de armazenamento de backup quando a opção de pausar o banco de dados até o próximo mês está habilitada.

### Passo 5: Networking (permitir acesso)

**Aba “Networking”:**

1. **Connectivity method:**
   * Selecione **“Public endpoint”**
   * Para desenvolvimento, acesso público é OK
   * Produção: Private endpoint (VNet)
2. **Firewall rules:**

**Allow Azure services and resources to access this server**

* + Permite ADF, Databricks, Synapse acessarem

**Add current client IP address**

* + Adiciona SEU IP atual ao firewall (você pode conectar via SSMS/Azure Data Studio)

**

*Figura 3.7 - Configuração de rede e firewall*

### Passo 6: Security (manter padrão)

**Aba “Security”:**

Configurações avançadas de segurança (Defender, Ledger, Identity).

**Para este exercício:** Deixe tudo **desabilitado** (economizar recursos).

**Produção:** Habilite Microsoft Defender for SQL!

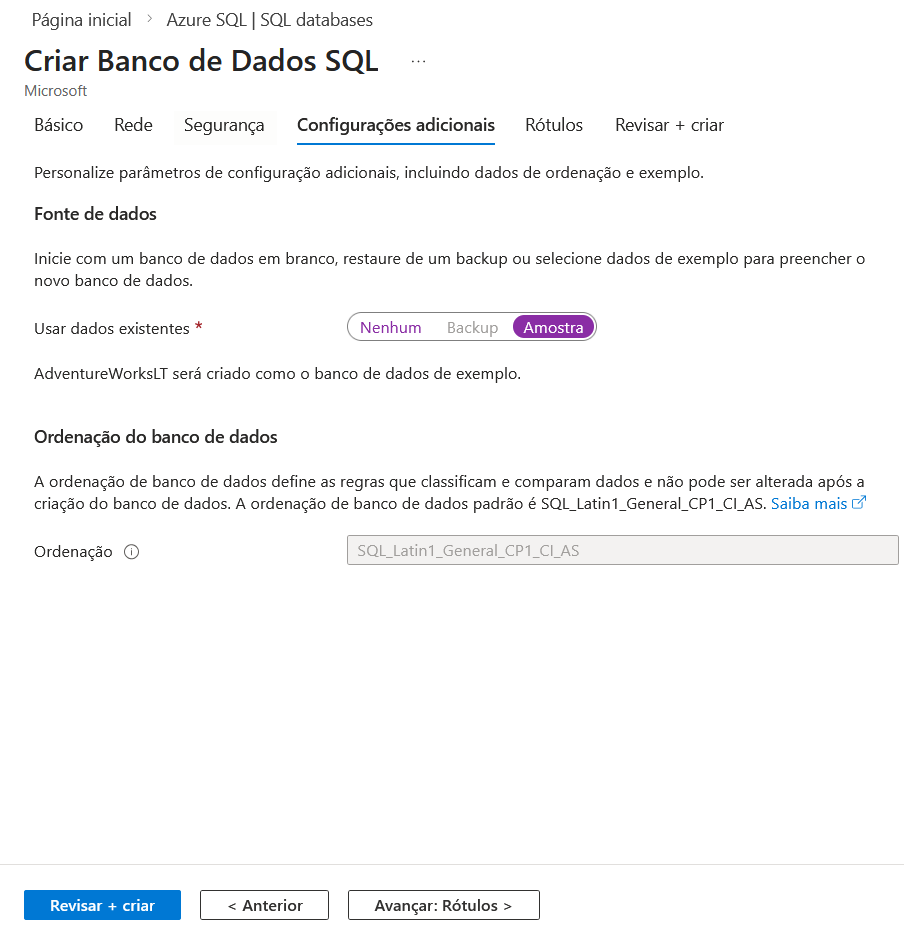
**Clique em “Next”**

### Passo 7: Additional Settings

**Aba “Additional settings”:**

1. **Use existing data:**
   * Selecione **“Sample”**
   * Azure cria banco com dataset de exemplo **AdventureWorksLT**
   * Contém tabelas de vendas, produtos, clientes (perfeito para aprender!)

**Alternativa:** Se quiser começar com banco vazio, selecione “None”

**

*Figura 3.8 - Carregar dataset de exemplo AdventureWorksLT*

1. **Database collation:** Deixe padrão (SQL\_Latin1\_General\_CP1\_CI\_AS)
2. **Maintenance window:** System default (Azure escolhe horário de manutenção)

**Clique em “Next: Review + create”**

### Passo 8: Revisar e Criar

1. **Revise o resumo:**
   * Database: db-vendas-junior
   * Server: sql-azuredejunior-[seunome]
   * Compute: Serverless, 0.5-2 vCores
   * Storage: 10 GB, LRS
   * Sample data: AdventureWorksLT
   * Custo estimado: ~R$ 10-20/mês (Ou R$ 0,00 no caso da oferta, respeitados os limites indicados)

**

*Figura 3.9 - Revisão antes de criar SQL Database*

1. **Clique em “Create”**
2. **Aguarde deployment** (2-5 minutos)
3. **Quando concluir, clique em “Go to resource”**

**

*Figura 3.10 - SQL Database criado com sucesso*

### Passo 9: Explorar o Database

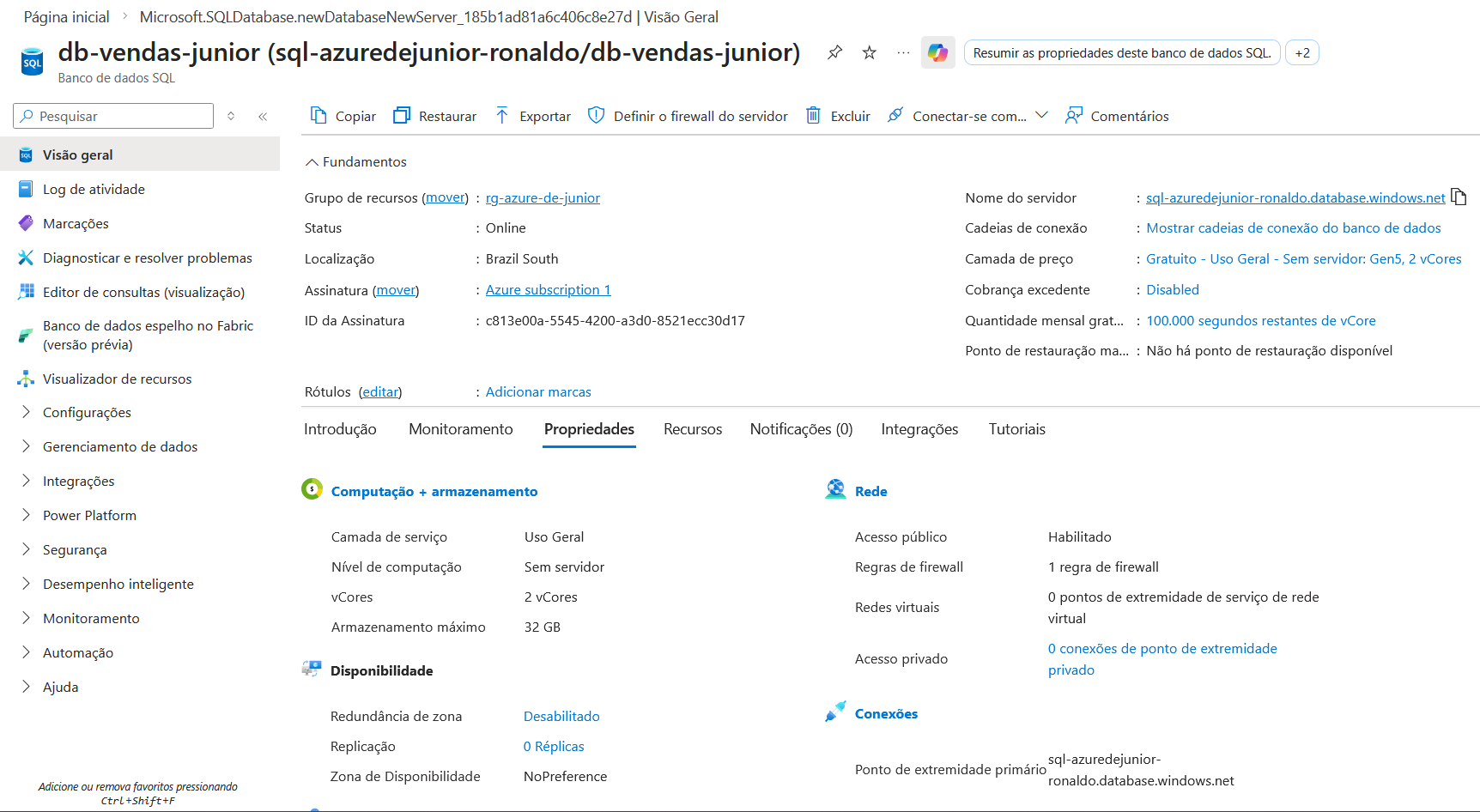
Você está na página Overview do database!

**Informações visíveis:**

* **Server name:** sql-azuredejunior-[seunome].database.windows.net
* **Status:** Online (ou “Paused” se inativo)
* **Pricing tier:** Serverless, General Purpose
* **Storage used:** ~3 MB (dados de exemplo)

**Menu lateral importante:**

* **Query editor:** Executar SQL direto no portal (básico)
* **Connection strings:** Para conectar aplicações
* **Backups:** Gerenciar backups e restore
* **Compute + storage:** Ajustar tier

**

*Figura 3.11 - Overview do SQL Database criado*

**Parabéns!** Você criou seu primeiro Azure SQL Database!

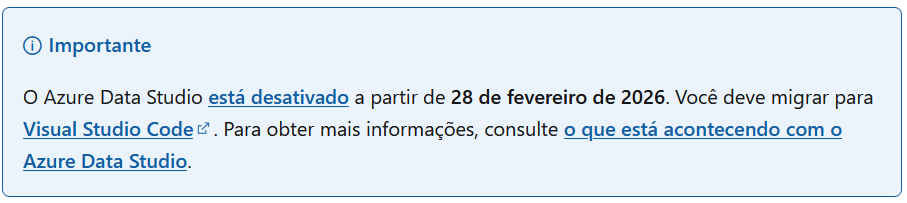
## 3.4 Conectar ao Database: Azure Data Studio

**Azure Data Studio** é uma ferramenta gratuita, moderna e cross-platform para gerenciar SQL Server/Azure SQL.

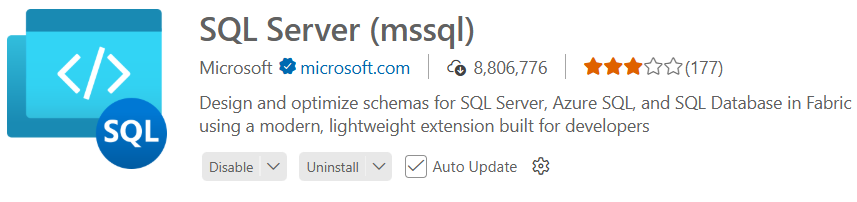
**Alternativas:**

* **SSMS (SQL Server Management Studio):** Mais pesado, só Windows, mais recursos avançados
* **Query Editor (portal):** Básico, limitado
* **DBeaver, DataGrip:** Ferramentas third-party

**Recomendação para este livro:** **Costumava ser o** **Azure Data Studio** (leve, multiplataforma, moderno). No entanto ele será descontinuado de atualizações a partir de Fevereiro/2026, com a Microsoft centrando esforços nas extensões para Visual Studio Code e no próprio SSMS.

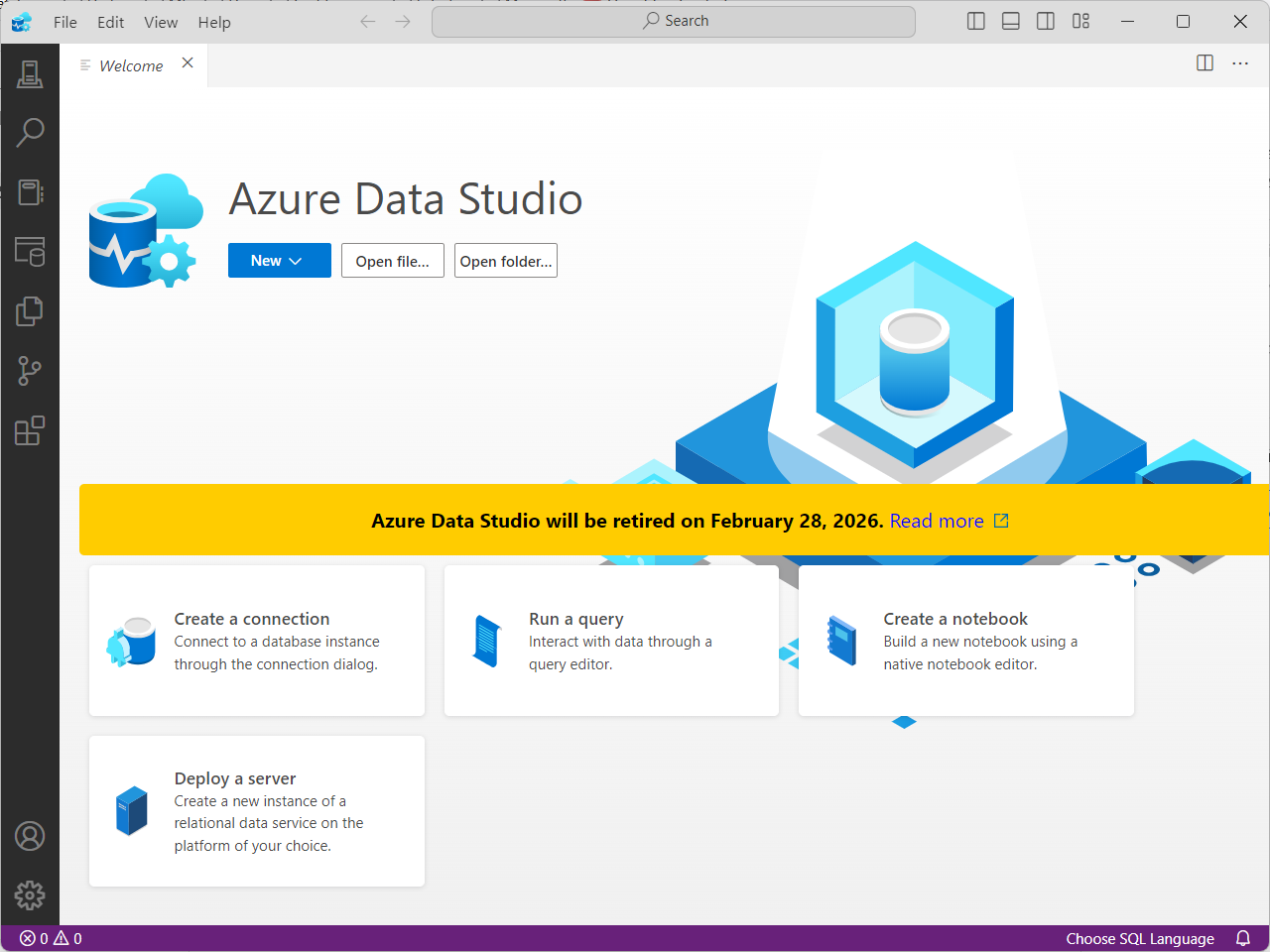


Apesar de não ser a mais bem pontuada entre as extensões, de preferência para a extensão suportada oficialmente pela Microsoft.



### Instalar Azure Data Studio

1. **Acesse:** <https://docs.microsoft.com/sql/azure-data-studio/download> ou <https://code.visualstudio.com/>
2. **Baixe para seu OS:** Windows, Mac ou Linux
3. **Instale** (processo padrão)
4. **Abra o Azure Data Studio/VSCode**

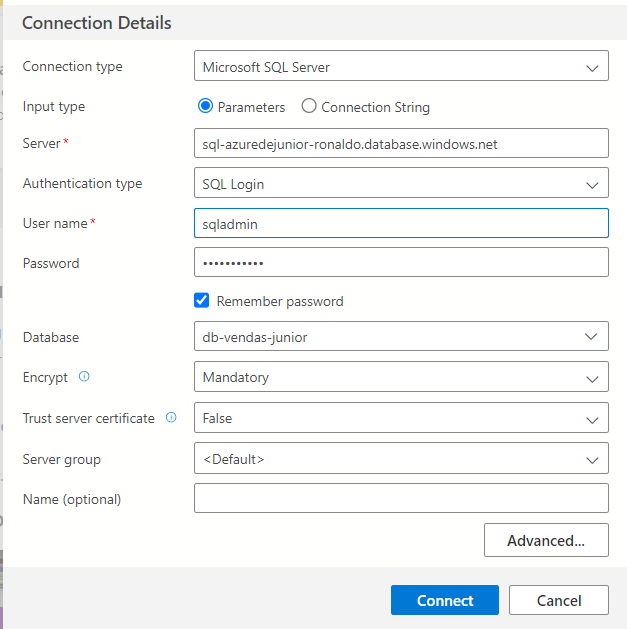
**

*Figura 3.12 - Download do Azure Data Studio*

### Conectar ao Database

**No Azure Data Studio (Parecido com as configurações pedidas na extensão do VSCode):**

1. **Clique em “New Connection”** (ou ícone de plug no canto superior esquerdo)
2. **Preencha os campos:**
   * **Connection type:** Microsoft SQL Server
   * **Server:** Cole o **Server name** do portal
     + Exemplo: sql-azuredejunior-ronaldo.database.windows.net
     + **Onde encontrar:** Azure Portal → SQL Database → Overview → “Server name”
   * **Authentication type:** SQL Login
   * **User name:** sqladmin (o que você definiu na criação)
   * **Password:** Sua senha (ex: Azur3@2026!)
   * **Remember password** (marque para não digitar sempre)
   * **Database:** db-vendas-junior (ou deixe <Default> e selecione depois)
   * **Encrypt:** True (padrão, segurança)
   * **Trust server certificate:** False (padrão)



*Figura 3.13 - Configuração de conexão no Azure Data Studio*

1. **Clique em “Connect”**

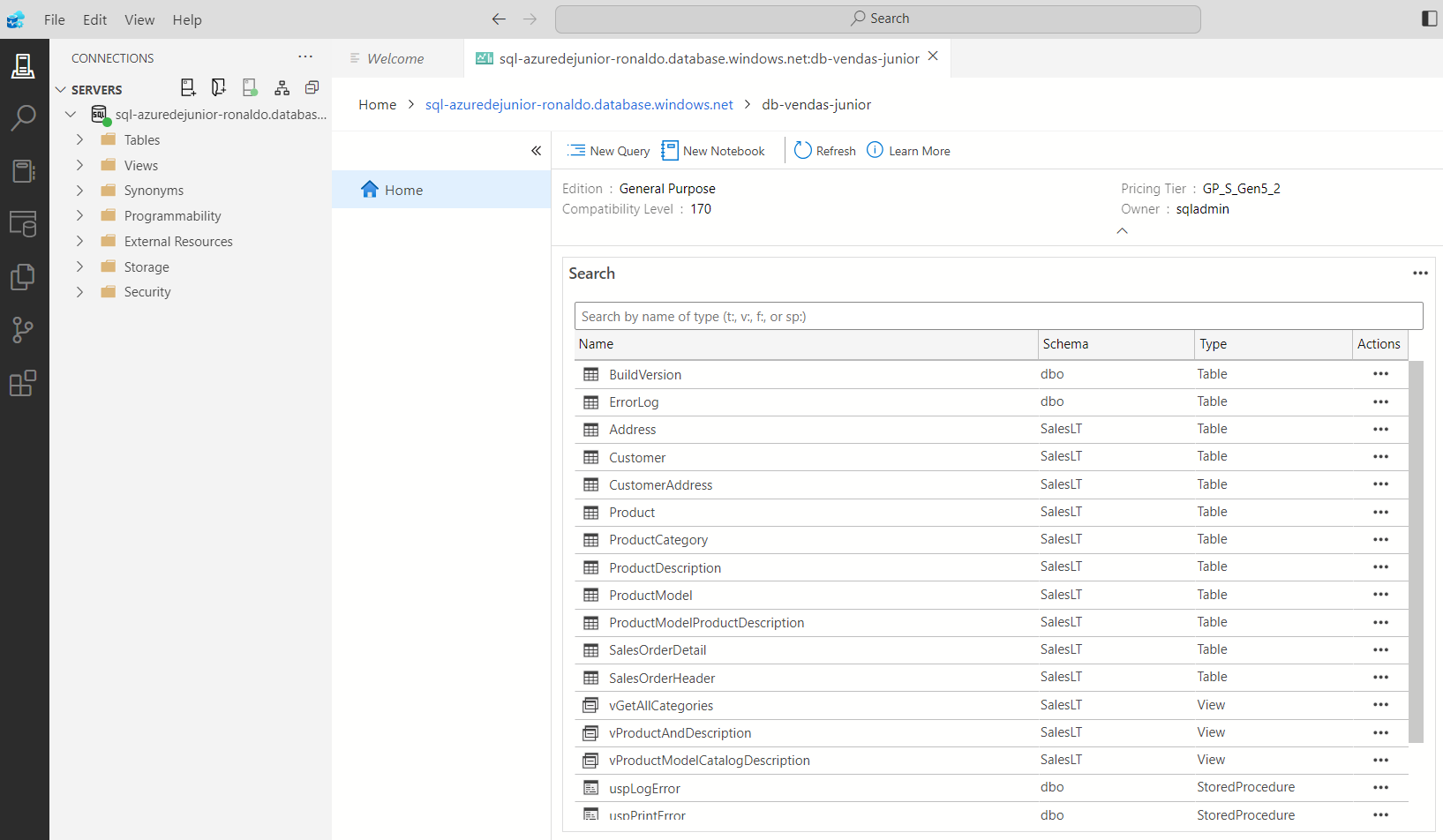
**Se der erro de firewall:**

* Volte ao portal: SQL Server → Networking → Firewall rules
* Adicione seu IP atual (botão “Add client IP”)
* Tente conectar novamente

1. **Conectado com sucesso!**

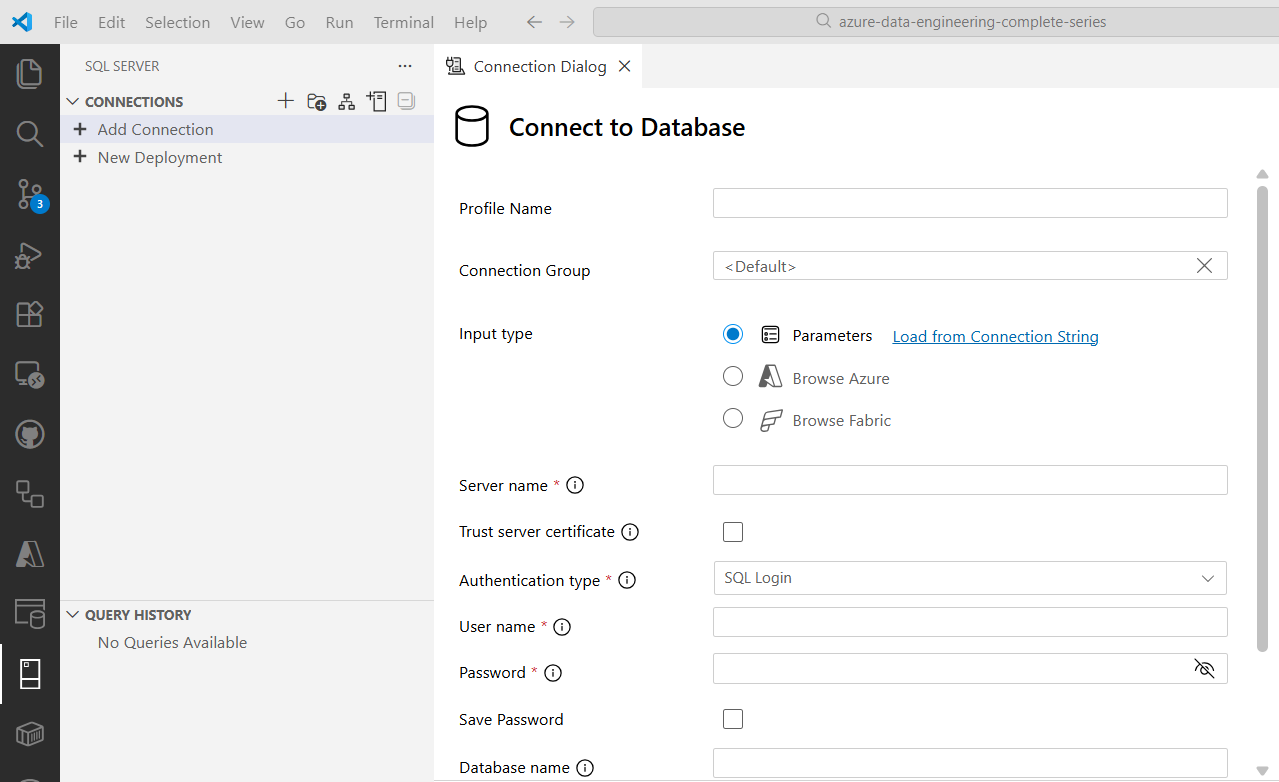
Você verá:

* **Servers** (à esquerda): Seu servidor listado
* **Databases** (expandir): db-vendas-junior
* **Tables** (expandir): Tabelas do AdventureWorksLT (SalesLT.Customer, SalesLT.Product, etc.)

**

*Figura 3.14 - Azure Data Studio conectado ao SQL Database*

### No caso do VSCode:



*Figura 3.14 - VSCode conectado ao SQL Database*

### Explorar o Schema AdventureWorksLT

**AdventureWorksLT** é um dataset de exemplo da Microsoft simulando uma empresa de vendas de bicicletas.

**Principais tabelas:**

|  |  |  |
| --- | --- | --- |
| **Schema** | **Tabela** | **Conteúdo** |
| **SalesLT** | Customer | Clientes (nome, email, telefone) |
| **SalesLT** | Product | Produtos (nome, preço, categoria) |
| **SalesLT** | ProductCategory | Categorias de produtos |
| **SalesLT** | SalesOrderHeader | Pedidos (cabeçalho: cliente, data, total) |
| **SalesLT** | SalesOrderDetail | Itens do pedido (produto, quantidade, preço) |
| **SalesLT** | Address | Endereços |
| **SalesLT** | CustomerAddress | Relacionamento cliente-endereço |

**Modelo relacional (simplified):**

Customer (1) (N) CustomerAddress (N) (1) Address

(1)

(N) SalesOrderHeader

(1)

(N) SalesOrderDetail (N) (1) Product

(N)

(1) ProductCategory

### Primeira Query: Explorar Dados

**Vamos executar nossa primeira query!**

1. **Clique em “New Query”** (ou Ctrl+N / Cmd+N)
2. **Digite o seguinte SQL:**

**SQL**

-- Listar primeiros 10 clientes

SELECT TOP 10

CustomerID,

FirstName,

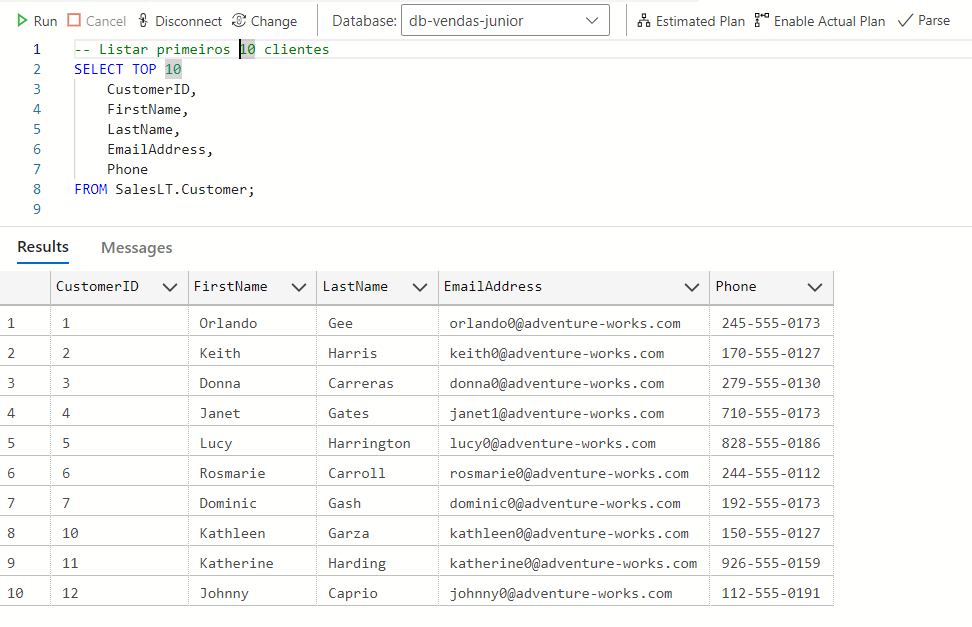
LastName,

EmailAddress,

Phone

FROM SalesLT.Customer;

1. **Clique em “Run”** (ou F5)
2. **Resultado aparece abaixo:** Tabela com 10 clientes

*****Figura 3.15 - Primeira query: listar clientes*

**Explore mais:**

**SQL**

-- Contar total de clientes

SELECT COUNT(\*) AS TotalClientes

FROM SalesLT.Customer;

-- Listar categorias de produtos

SELECT

ProductCategoryID,

Name,

ParentProductCategoryID

FROM SalesLT.ProductCategory

ORDER BY ProductCategoryID;

-- Ver produtos mais caros

SELECT TOP 5

Name,

ProductNumber,

ListPrice

FROM SalesLT.Product

ORDER BY ListPrice DESC;

## 3.5 Modelagem Dimensional: Star Schema

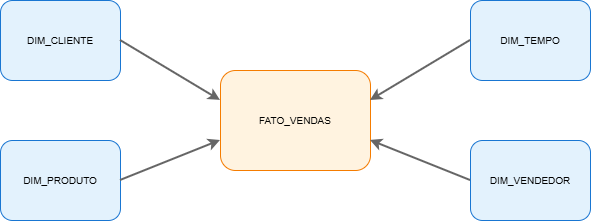
### O que é Star Schema?

**Star Schema** é um modelo de dados otimizado para **análises e relatórios**, comum em Data Warehouses.

**Estrutura:**

* **Fato (Fact):** Tabela central com métricas numéricas (vendas, quantidade, receita)
* **Dimensões (Dimensions):** Tabelas ao redor com atributos descritivos (produto, cliente, tempo, localização)

**Visualização:**



Parece uma **estrela** (fato no centro, dimensões nas pontas)

### Fato vs Dimensão

**Tabela Fato (Fact Table):**

* Contém **métricas/medidas** (valores numéricos agregáveis)
* Exemplos de colunas:
  + quantidade\_vendida (int)
  + valor\_total (decimal)
  + desconto (decimal)
  + custo (decimal)
* Contém **chaves estrangeiras** para dimensões
* **Granularidade:** Nível de detalhe (ex: uma linha = um item de venda)
* **Grande volume:** Milhões/bilhões de linhas

**Tabela Dimensão (Dimension Table):**

* Contém **atributos descritivos** (texto, categorias)
* Exemplos de colunas:
  + nome\_produto, categoria, marca
  + nome\_cliente, cidade, pais
  + data, mes, ano, trimestre
* Contém **chave primária** (DimClienteID, DimProdutoID)
* **Baixo volume:** Milhares de linhas (relativamente)

### Exemplo: E-commerce

**Fato\_Vendas:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **VendaID** | **DataID** | **ClienteID** | **ProdutoID** | **LojaID** | **Quantidade** | **Valor** | **Desconto** |
| **1** | 20260101 | 5 | 102 | 3 | 2 | 150.00 | 10.00 |
| **2** | 20260101 | 8 | 205 | 3 | 1 | 450.00 | 0.00 |

**Dim\_Cliente:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ClienteID** | **Nome** | **Cidade** | **Estado** | **Pais** |
| **5** | João Silva | São Paulo | SP | Brasil |
| **8** | Maria Santos | Rio de Janeiro | RJ | Brasil |

**Dim\_Produto:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ProdutoID** | **Nome** | **Categoria** | **Marca** | **Preco** |
| **102** | Mouse Logitech | Periféricos | Logitech | 85.00 |
| **205** | Teclado Mecânico | Periféricos | Razer | 450.00 |

**Dim\_Tempo:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DataID** | **Data** | **DiaSemana** | **Mes** | **Trimestre** | **Ano** |
| **20260101** | 2026-01-01 | Quarta | Janeiro | Q1 | 2026 |

**Dim\_Loja:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LojaID** | **Nome** | **Cidade** | **Gerente** |
| **3** | Loja Shopping SP | São Paulo | Pedro Costa |

### Por que Star Schema?

**Vantagens:**

1. **Queries rápidas:** JOINs simples (fato → dimensões)
2. **Intuitivo:** Fácil entender (negócio reconhece dimensões)
3. **Agregações eficientes:** Métricas já separadas
4. **BI-friendly:** Power BI, Tableau amam Star Schema

**Exemplo de query analítica:**

**SQL**

-- Vendas por categoria de produto e mês

SELECT

dt.Mes,

dp.Categoria,

SUM(fv.Quantidade) AS TotalVendido,

SUM(fv.Valor) AS ReceitaTotal

FROM Fato\_Vendas fv

INNER JOIN Dim\_Tempo dt ON fv.DataID = dt.DataID

INNER JOIN Dim\_Produto dp ON fv.ProdutoID = dp.ProdutoID

GROUP BY dt.Mes, dp.Categoria

ORDER BY dt.Mes, ReceitaTotal DESC;

**Simples e performático!**

### Snowflake Schema (alternativa)

**Snowflake** é uma variação onde dimensões são **normalizadas** (subdivididas).

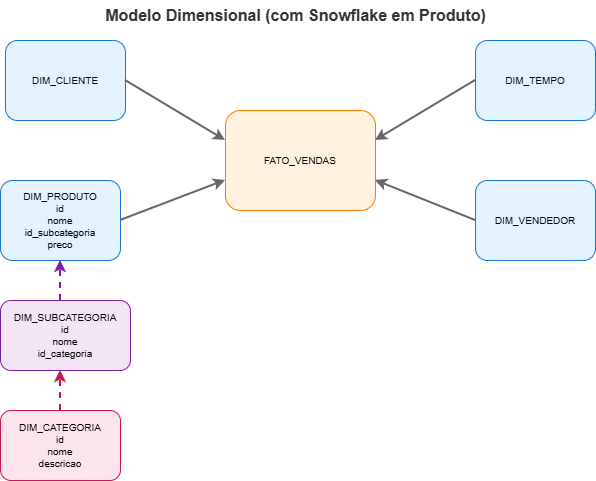
**Vantagens:**

* Menos redundância (normalização)

**Desvantagens:**

* Mais JOINs (mais lento)
* Mais complexo para usuários de negócio

**Recomendação:** Para DW analítico, **Star Schema é melhor** (desnormalizado, rápido).

****

*Figura 3.16 - Exemplo de Snow Flake: Fato\_Vendas com 4 dimensões + Subs*

## 3.6 SQL Essencial para Engenharia de Dados

Vamos revisar/aprender SQL fundamental para trabalhar com dados. SQL (Structured Query Language) permanece como a linguagem fundamental para engenharia de dados, mesmo em ambientes modernos de big data. Entretanto, cada data warehouse possui seu próprio dialeto SQL com extensões e otimizações específicas.

Cada dialeto possui particularidades em funções de janela, CTEs, merge/upsert operations e tipos de dados específicos. É fundamental conhecer as nuances do seu warehouse para otimização de queries.

**Dialetos SQL em Data Warehouses Modernos:**

Os principais data warehouses utilizam variações de SQL adaptadas para processamento massivo paralelo (MPP):

**Snowflake SQL**: baseado em ANSI SQL com extensões para semi-structured data (JSON, Parquet), suporte nativo a time travel e clone zero-copy

**BigQuery Standard SQL**: dialeto Google com forte foco em arrays e structs, funções de ML integradas e processamento Serverless.

**Redshift SQL:** baseado em PostgreSQL 8.0.2 com otimizações para queries analíticas e distribuição de dados.

**Databricks SQL:** utiliza Spark SQL (dialeto Hive) com suporte a Delta Lake, ACID transactions e merge operations.

**Synapse SQL:** Oferece T-SQL (SQL Server) com pools dedicados e serverless para diferentes workloads. Em nossa série vamos estudar especificamente essa tecnologia.

**Armazenamento Colunar vs. Row-Based:**

A arquitetura de armazenamento impacta diretamente a performance de queries analíticas:

**Row-Based Storage (OLTP tradicional):** Dados são armazenados linha por linha:

*[ID:1, Nome:João, Idade:30, Cidade:SP]*

Otimizado para INSERT, UPDATE, DELETE de registros completos

Eficiente para queries transacionais que retornam poucos registros completos.

Exemplo: SQL Server, MySQL, PostgreSQL (modo padrão)

**Columnar Storage (OLAP/Analytical):** Dados armazenados coluna por coluna:

[ID:1,2,3...], [Nome:João,Maria...], [Idade:30,25...]

Compressão superior (valores similares agrupados)

I/O reduzido: lê apenas colunas necessárias na query

Ideal para agregações: SUM, AVG, COUNT em milhões de registros

Exemplos: Parquet, ORC, columnar indexes no SQL Server

Em warehouses modernos, formatos colunares (Parquet no Databricks, arquivos colunares no Redshift) podem reduzir o volume de dados lidos em 90%+ comparado a row-based storage. E podemos estar falando sobre terabytes de leitura e escrita.

**Clusterização e Particionamento**

**Clusterização** organiza fisicamente os dados para otimizar queries específicas:

**Particionamento** divide tabelas em segmentos baseados em colunas (geralmente data: ano/mês/dia). Permite partition pruning: elimina partições irrelevantes antes da leitura exemplo:

WHERE data\_venda >= '2024-01-01' ignora partições de 2023 .

**Clustering**: Ordena dados dentro das partições por colunas frequentemente filtradas co-localiza dados relacionados para minimizar I/O:

**Como cada banco lida com isso**

**Synapse SQL DW**: Clustered Columnstore Index (CCI) com ordenação opcional por colunas específicas;

**Snowflake**: auto-clustering em micro-partitions

**Databricks**: Z-ordering para múltiplas colunas de filtro

**BigQuery**: clustering automático em até 4 colunas

**Exemplo prático:**

Tabela de vendas particionada por data\_venda e clusterizada por (regiao, categoria\_produto) permite queries rápidas como:

**SQL**

SELECT

SUM(valor\_venda)

FROM vendas

WHERE data\_venda

BETWEEN '2024-01-01'

AND '2024-01-31'

AND regiao = 'Sudeste'

AND categoria\_produto = 'Eletronicos'

A combinação de storage colunar + particionamento + clustering pode acelerar queries analíticas em 100x ou mais comparado a abordagens tradicionais row-based sem otimização.

Nos próximos tópicos, aplicaremos esses conceitos em queries práticas para pipelines de engenharia de dados.

### SELECT Básico

**SQL**

-- Selecionar colunas específicas

SELECT

FirstName,

LastName,

EmailAddress

FROM SalesLT.Customer;

-- Selecionar todas as colunas (evite em produção!)

SELECT \*

FROM SalesLT.Customer;

-- Limitar resultados

SELECT TOP 100 \*

FROM SalesLT.Customer;

-- Filtrar com WHERE

SELECT

FirstName,

LastName,

EmailAddress

FROM SalesLT.Customer

WHERE CompanyName IS NOT NULL;

### JOINs (Unir Tabelas)

**INNER JOIN:** Retorna apenas registros com match nas duas tabelas

**SQL**

-- Clientes e seus endereços

SELECT

c.CustomerID,

c.FirstName,

c.LastName,

a.AddressLine1,

a.City,

a.StateProvince

FROM SalesLT.Customer c

INNER JOIN SalesLT.CustomerAddress ca

ON c.CustomerID = ca.CustomerID

INNER JOIN SalesLT.Address a

ON ca.AddressID = a.AddressID;

**LEFT JOIN:** Retorna todos da tabela esquerda + matches da direita (NULL se sem match)

**SQL**

-- Todos clientes, com ou sem endereço

SELECT

c.CustomerID,

c.FirstName,

c.LastName,

a.City

FROM SalesLT.Customer c

LEFT JOIN SalesLT.CustomerAddress ca

ON c.CustomerID = ca.CustomerID

LEFT JOIN SalesLT.Address a

ON ca.AddressID = a.AddressID;

**RIGHT JOIN e FULL OUTER JOIN:** Menos comuns, mas existem.

### Agregações (GROUP BY)

**SQL**

-- Total de vendas por cliente

SELECT

c.CustomerID,

c.FirstName + ' ' + c.LastName AS NomeCompleto,

COUNT(soh.SalesOrderID) AS TotalPedidos,

SUM(soh.TotalDue) AS ReceitaTotal

FROM SalesLT.Customer c

INNER JOIN SalesLT.SalesOrderHeader soh

ON c.CustomerID = soh.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

ORDER BY ReceitaTotal DESC;

**Funções de agregação:**

* COUNT(\*) - contar linhas
* SUM(coluna) - somar valores
* AVG(coluna) - média
* MIN(coluna) - mínimo
* MAX(coluna) - máximo

**Regra:** Tudo que não está em função de agregação **deve estar no GROUP BY**.

### HAVING (Filtrar Agregações)

**SQL**

-- Clientes que gastaram mais de R$ 10.000

SELECT

c.CustomerID,

c.FirstName + ' ' + c.LastName AS NomeCompleto,

SUM(soh.TotalDue) AS ReceitaTotal

FROM SalesLT.Customer c

INNER JOIN SalesLT.SalesOrderHeader soh

ON c.CustomerID = soh.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

HAVING SUM(soh.TotalDue) > 10000

ORDER BY ReceitaTotal DESC;

**WHERE vs HAVING:**

* **WHERE:** Filtra linhas **antes** de agregar
* **HAVING:** Filtra grupos **depois** de agregar

### CTEs (Common Table Expressions)

CTEs são “queries temporárias” que tornam SQL mais legível.

**SQL**

-- Usar CTE para calcular vendas por categoria

WITH VendasPorCategoria AS (

SELECT

pc.Name AS Categoria,

SUM(sod.LineTotal) AS ReceitaTotal

FROM SalesLT.SalesOrderDetail sod

INNER JOIN SalesLT.Product p

ON sod.ProductID = p.ProductID

INNER JOIN SalesLT.ProductCategory pc

ON p.ProductCategoryID = pc.ProductCategoryID

GROUP BY pc.Name

)

SELECT

Categoria,

ReceitaTotal,

ROUND(ReceitaTotal / SUM(ReceitaTotal) OVER () \* 100, 2) AS PercentualTotal

FROM VendasPorCategoria

ORDER BY ReceitaTotal DESC;

**Vantagem:** Quebra query complexa em pedaços legíveis.

### Window Functions (Funções de Janela)

Permitem cálculos “por grupo” sem GROUP BY.

**SQL**

-- Ranking de produtos por receita em cada categoria

SELECT

pc.Name AS Categoria,

p.Name AS Produto,

SUM(sod.LineTotal) AS ReceitaProduto,

RANK() OVER (PARTITION BY pc.Name ORDER BY SUM(sod.LineTotal) DESC) AS RankingNaCategoria

FROM SalesLT.SalesOrderDetail sod

INNER JOIN SalesLT.Product p

ON sod.ProductID = p.ProductID

INNER JOIN SalesLT.ProductCategory pc

ON p.ProductCategoryID = pc.ProductCategoryID

GROUP BY pc.Name, p.Name

ORDER BY Categoria, RankingNaCategoria;

**Funções úteis:**

* ROW\_NUMBER() - numeração sequencial
* RANK() - ranking (empates = mesmo rank)
* DENSE\_RANK() - ranking sem pular números
* SUM() OVER (...) - soma acumulada
* AVG() OVER (...) - média móvel

### CASE WHEN (Lógica Condicional)

**SQL**

-- Classificar clientes por volume de compras

SELECT

c.CustomerID,

c.FirstName + ' ' + c.LastName AS NomeCompleto,

SUM(soh.TotalDue) AS ReceitaTotal,

CASE

WHEN SUM(soh.TotalDue) > 50000 THEN 'VIP'

WHEN SUM(soh.TotalDue) > 20000 THEN 'Premium'

WHEN SUM(soh.TotalDue) > 5000 THEN 'Regular'

ELSE 'Básico'

END AS SegmentoCliente

FROM SalesLT.Customer c

INNER JOIN SalesLT.SalesOrderHeader soh

ON c.CustomerID = soh.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

ORDER BY ReceitaTotal DESC;

### Subqueries

**SQL**

-- Produtos com preço acima da média

SELECT

Name,

ListPrice

FROM SalesLT.Product

WHERE ListPrice > (

SELECT AVG(ListPrice)

FROM SalesLT.Product

)

ORDER BY ListPrice DESC;

## 3.7 Criar Tabelas Customizadas

Vamos criar nossas próprias tabelas para um mini-projeto de vendas. Para isso, abra a conexão com Azure Studio/VSCode. No cold star podemos ter alguns segundos para o banco responder. Isso é normal no ambiente que estamos trabalhando. Em produção, uma vez ao dia também é admissível. O que devemos pesar aqui é o tempo para desligamento ou timeout.

### Criar Dimensão Cliente

**SQL**

-- Criar schema para organização

CREATE SCHEMA DW;

GO

-- Criar dimensão cliente

CREATE TABLE DW.DimCliente (

ClienteID INT IDENTITY(1,1) PRIMARY KEY,

NomeCompleto NVARCHAR(100) NOT NULL,

Email NVARCHAR(100),

Telefone NVARCHAR(20),

Cidade NVARCHAR(50),

Estado NVARCHAR(2),

Pais NVARCHAR(50),

DataCadastro DATE,

SegmentoCliente NVARCHAR(20)

);

### Criar Dimensão Produto

**SQL**

CREATE TABLE DW.DimProduto (

ProdutoID INT IDENTITY(1,1) PRIMARY KEY,

NomeProduto NVARCHAR(100) NOT NULL,

Categoria NVARCHAR(50),

Subcategoria NVARCHAR(50),

Marca NVARCHAR(50),

PrecoLista DECIMAL(10,2),

Ativo BIT DEFAULT 1

);

### Criar Dimensão Tempo

**SQL**

CREATE TABLE DW.DimTempo (

DataID INT PRIMARY KEY, -- formato YYYYMMDD (ex: 20260101)

Data DATE NOT NULL,

Ano INT,

Trimestre INT,

Mes INT,

MesNome NVARCHAR(20),

Semana INT,

DiaSemana INT,

DiaSemaNome NVARCHAR(20),

DiaUtil BIT

);

### Criar Fato Vendas

**SQL**

CREATE TABLE DW.FatoVendas (

VendaID INT IDENTITY(1,1) PRIMARY KEY,

DataID INT NOT NULL,

ClienteID INT NOT NULL,

ProdutoID INT NOT NULL,

Quantidade INT NOT NULL,

PrecoUnitario DECIMAL(10,2) NOT NULL,

Desconto DECIMAL(10,2) DEFAULT 0,

ValorTotal AS (Quantidade \* PrecoUnitario - Desconto) PERSISTED,

-- Foreign Keys

CONSTRAINT FK\_FatoVendas\_DimTempo

FOREIGN KEY (DataID) REFERENCES DW.DimTempo(DataID),

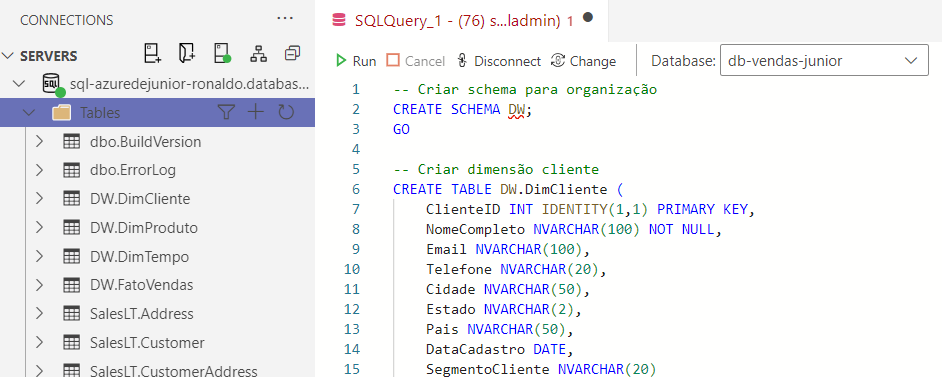
CONSTRAINT FK\_FatoVendas\_DimCliente

FOREIGN KEY (ClienteID) REFERENCES DW.DimCliente(ClienteID),

CONSTRAINT FK\_FatoVendas\_DimProduto

FOREIGN KEY (ProdutoID) REFERENCES DW.DimProduto(ProdutoID)

);

**

*Figura 3.17 - Criação de tabelas customizadas (Star Schema)*

### Inserir Dados de Exemplo

**Popular Dimensão Tempo (gerar datas de 2026):**

**SQL**

-- Popular DimTempo para o ano de 2026

DECLARE @DataInicio DATE = '2026-01-01';

DECLARE @DataFim DATE = '2026-12-31';

DECLARE @DataAtual DATE = @DataInicio;

WHILE @DataAtual <= @DataFim

BEGIN

INSERT INTO DW.DimTempo (DataID, Data, Ano, Trimestre, Mes, MesNome, Semana, DiaSemana, DiaSemaNome, DiaUtil)

VALUES (

CAST(FORMAT(@DataAtual, 'yyyyMMdd') AS INT), -- DataID

@DataAtual, -- Data

YEAR(@DataAtual), -- Ano

DATEPART(QUARTER, @DataAtual), -- Trimestre

MONTH(@DataAtual), -- Mes

FORMAT(@DataAtual, 'MMMM', 'pt-BR'), -- MesNome

DATEPART(WEEK, @DataAtual), -- Semana

DATEPART(WEEKDAY, @DataAtual), -- DiaSemana

FORMAT(@DataAtual, 'dddd', 'pt-BR'), -- DiaSemaNome

CASE WHEN DATEPART(WEEKDAY, @DataAtual) IN (1,7) THEN 0 ELSE 1 END -- DiaUtil (0=fds, 1=util)

);

SET @DataAtual = DATEADD(DAY, 1, @DataAtual);

END;

-- Verificar

SELECT TOP 10 \* FROM DW.DimTempo ORDER BY DataID;

**Popular Dimensão Cliente:**

**SQL**

INSERT INTO DW.DimCliente (NomeCompleto, Email, Telefone, Cidade, Estado, Pais, DataCadastro, SegmentoCliente)

VALUES

('João Silva', 'joao.silva@email.com', '11987654321', 'São Paulo', 'SP', 'Brasil', '2025-01-15', 'Premium'),

('Maria Santos', 'maria.santos@email.com', '21987654321', 'Rio de Janeiro', 'RJ', 'Brasil', '2025-03-22', 'Regular'),

('Pedro Oliveira', 'pedro.oliveira@email.com', '31987654321', 'Belo Horizonte', 'MG', 'Brasil', '2025-05-10', 'VIP'),

('Ana Costa', 'ana.costa@email.com', '11976543210', 'São Paulo', 'SP', 'Brasil', '2025-08-05', 'Regular'),

('Carlos Ferreira', 'carlos.ferreira@email.com', '41987654321', 'Curitiba', 'PR', 'Brasil', '2025-11-20', 'Básico');

SELECT \* FROM DW.DimCliente;

**Popular Dimensão Produto:**

**SQL**

INSERT INTO DW.DimProduto (NomeProduto, Categoria, Subcategoria, Marca, PrecoLista, Ativo)

VALUES

('Notebook Dell Inspiron 15', 'Informática', 'Notebooks', 'Dell', 3500.00, 1),

('Mouse Logitech MX Master 3', 'Informática', 'Periféricos', 'Logitech', 450.00, 1),

('Teclado Mecânico Razer', 'Informática', 'Periféricos', 'Razer', 650.00, 1),

('Monitor LG 27” 4K', 'Informática', 'Monitores', 'LG', 1800.00, 1),

('Webcam Logitech C920', 'Informática', 'Periféricos', 'Logitech', 350.00, 1);

SELECT \* FROM DW.DimProduto;

**Popular Fato Vendas:**

**SQL**

INSERT INTO DW.FatoVendas (DataID, ClienteID, ProdutoID, Quantidade, PrecoUnitario, Desconto)

VALUES

(20260101, 1, 1, 2, 3500.00, 200.00), -- João comprou 2 notebooks

(20260101, 2, 2, 3, 450.00, 0.00), -- Maria comprou 3 mouses

(20260102, 3, 4, 1, 1800.00, 100.00), -- Pedro comprou 1 monitor

(20260103, 1, 3, 1, 650.00, 0.00), -- João comprou 1 teclado

(20260103, 4, 5, 2, 350.00, 50.00), -- Ana comprou 2 webcams

(20260104, 2, 1, 1, 3500.00, 150.00), -- Maria comprou 1 notebook

(20260105, 5, 2, 5, 450.00, 100.00); -- Carlos comprou 5 mouses

SELECT \* FROM DW.FatoVendas;

### Query Analítica: Vendas por Cliente

**SQL**

SELECT

c.NomeCompleto,

c.Cidade,

c.SegmentoCliente,

COUNT(fv.VendaID) AS TotalCompras,

SUM(fv.Quantidade) AS TotalItens,

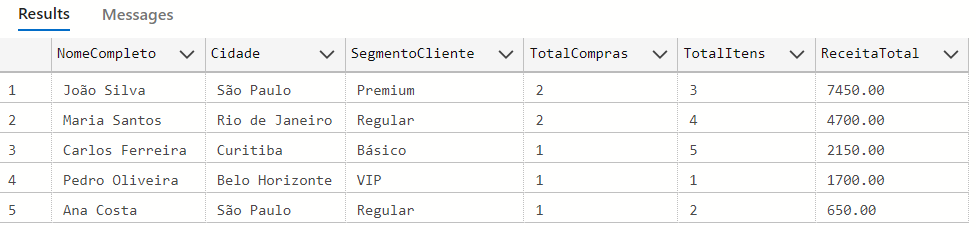
SUM(fv.ValorTotal) AS ReceitaTotal

FROM DW.FatoVendas fv

INNER JOIN DW.DimCliente c ON fv.ClienteID = c.ClienteID

GROUP BY c.NomeCompleto, c.Cidade, c.SegmentoCliente

ORDER BY ReceitaTotal DESC;



*Figura 3.18 - Query analítica: vendas por cliente*

## 3.8 Backup e Restore

Azure SQL Database faz backups automaticamente, mas é importante saber gerenciá-los.

### Backups Automáticos

**Azure SQL Database faz:**

* **Full backup:** Semanalmente
* **Differential backup:** A cada 12-24 horas
* **Log backup:** A cada 5-10 minutos

**Retenção padrão:**

* **7 dias** (Basic tier)
* **35 dias** (Standard/Premium)

**Tipo de backup:**

* **Locally-redundant (LRS):** 3 cópias na mesma região
* **Geo-redundant (GRS):** Replicado para região pareada

### Restore Manual (Point-in-Time)

**Cenário:** Você deletou dados por engano às 14:00. Quer restaurar banco como estava às 13:50.

**No Azure Portal:**

1. **Vá para sua SQL Database:** db-vendas-junior
2. **Clique em “Restore”** (barra superior)
3. **Preencha:**
   * **Database name:** db-vendas-junior-restore
   * **Restore point:** Selecione data/hora (últimos 7-35 dias)
   * **Server:** Mesmo servidor ou novo
4. **Clique em “Review + create” → “Create”**
5. **Aguarde restore** (alguns minutos)
6. **Novo database criado** com dados do ponto no tempo escolhido

Eventualmente como seu banco de dados é muito recente, é bem provável que você não tenha ainda nenhum ponto no tempo para restaurar. Como exercício, recomendo você criar um, efetuar alguma alteração, algumas horas depois retorne e efetue à restauração para um ponto anterior ao horário da alteração para ver os registros originais.

Importante entender que em um ambiente de produção essa é uma operação coordenada, em especial com os usuários, pois a restauração não é seletiva. Todos os usuários vão ver o banco no instante que foi restaurado.

Ou seja, não é algo para fazer sem critérios, e não se preocupe, não é tarefa do Junior da equipe. Mas é importante que você conheça.

**

*Figura 3.19 - Restore point-in-time de SQL Database*

### Exportar Database (BACPAC)

**BACPAC** é um arquivo que contém schema + dados (portável entre Azure SQL e SQL Server).

**Como exportar:**

1. **SQL Database → “Export”**
2. **Preencha:**
   * **Storage account:** Selecione ou crie
   * **Container:** Escolha ou crie
   * **File name:** db-vendas-junior.bacpac
   * **Server admin login / password:** Credenciais SQL
3. **Clique em “OK”**
4. **Aguarde exportação** (alguns minutos)
5. **Arquivo .bacpac salvo no Blob Storage**

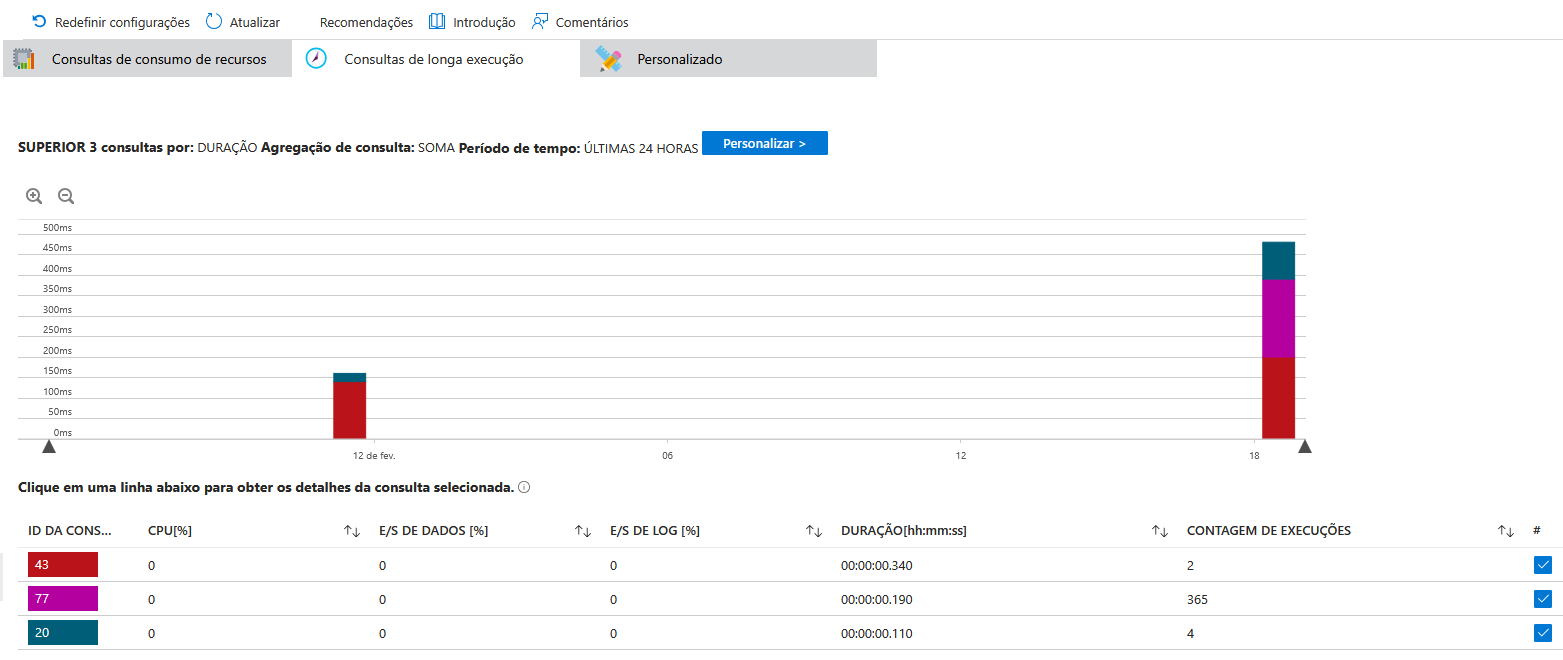
**Uso:** Migração, backup externo, mover entre ambientes.

## 3.9 Monitoramento e Performance

### Query Performance Insight

**No Azure Portal:**

1. **SQL Database → “Query Performance Insight”** (menu lateral)
2. **Veja:**
   * Top queries por duração
   * Queries mais frequentes
   * Queries que consomem mais CPU/IO
3. **Clique em uma query** para ver detalhes e recomendações

*Figura 3.20 - Query Performance Insight: monitorar queries lentas*

### Índices (Básico)

**Índices** aceleram consultas criando estruturas de busca otimizadas.

**Quando criar índice:**

* Colunas frequentemente usadas em WHERE
* Colunas usadas em JOINs
* Colunas usadas em ORDER BY

**Exemplo:**

**SQL**

-- Criar índice em coluna Email

CREATE INDEX IX\_DimCliente\_Email

ON DW.DimCliente(Email);

-- Índice composto (múltiplas colunas)

CREATE INDEX IX\_FatoVendas\_DataCliente

ON DW.FatoVendas(DataID, ClienteID);

**Cuidado:** Muitos índices = writes mais lentos (atualizar índice em cada INSERT/UPDATE).

**Dica:** Azure SQL recomenda índices automaticamente (Automatic Tuning).

### Estatísticas

SQL Server usa **estatísticas** para otimizar planos de execução.

**Atualizar estatísticas manualmente:**

**SQL**

UPDATE STATISTICS DW.FatoVendas;

**Azure SQL atualiza estatísticas automaticamente**, mas em datasets grandes, manual pode ajudar.

## 3.10 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar quando usar SQL Database vs Data Lake
* [ ] Descrever Star Schema (fato + dimensões)
* [ ] Diferenciar tier Serverless vs DTU vs vCore
* [ ] Entender backups automáticos (point-in-time recovery)

**Práticos:**

* [ ] Provisionar Azure SQL Database (Serverless)
* [ ] Conectar usando Azure Data Studio
* [ ] Escrever queries SQL: SELECT, JOIN, GROUP BY, CTE, Window Functions
* [ ] Criar tabelas customizadas (CREATE TABLE)
* [ ] Inserir dados (INSERT INTO)
* [ ] Restaurar database para ponto no tempo
* [ ] Monitorar performance de queries

**Modelagem:**

* [ ] Desenhar Star Schema para um caso de uso
* [ ] Identificar fatos vs dimensões
* [ ] Criar chaves estrangeiras (FOREIGN KEY)

## 3.11 Próximos Passos

No **Capítulo 4**, você vai:

Criar seu primeiro Azure Data Factory

Construir pipeline para copiar dados (SQL → Data Lake)

Usar Copy Activity e Data Flows

Agendar pipelines (triggers)

Monitorar execuções e debugar falhas

**Preparação:**

* Mantenha SQL Database ativo (usaremos como fonte no Cap 4)
* Mantenha Data Lake ativo (usaremos como destino)
* Revise conceitos de ETL/ELT do Cap 1

## 3.12 Recursos Complementares

**Documentação Oficial:**

* [Azure SQL Database Overview](https://learn.microsoft.com/azure/azure-sql/database/sql-database-paas-overview)
* [Serverless Compute Tier](https://learn.microsoft.com/azure/azure-sql/database/serverless-tier-overview)
* [Automated Backups](https://learn.microsoft.com/azure/azure-sql/database/automated-backups-overview)

**Ferramentas:**

* [Azure Data Studio Download](https://docs.microsoft.com/sql/azure-data-studio/download)
* [SSMS Download](https://docs.microsoft.com/sql/ssms/download-sql-server-management-studio-ssms)

**Modelagem:**

* [Star Schema Guide](https://learn.microsoft.com/power-bi/guidance/star-schema)
* [Kimball Dimensional Modeling](https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/)

**SQL Learning:**

* [Microsoft SQL Tutorial](https://learn.microsoft.com/sql/t-sql/tutorial-writing-transact-sql-statements)
* [SQLBolt (Interactive)](https://sqlbolt.com/)
* [W3Schools SQL](https://www.w3schools.com/sql/)

## 3.13 Exercícios Práticos

**Exercício 1: Star Schema E-commerce**

Desenhe um Star Schema para um e-commerce com:

* Fato: Pedidos
* Dimensões: Cliente, Produto, Tempo, MetodoPagamento, StatusPedido

Implemente no SQL Database.

**Exercício 2: Queries Analíticas**

Usando AdventureWorksLT, escreva queries para:

1. Total de vendas por categoria de produto
2. Top 10 clientes por receita
3. Produtos que nunca foram vendidos
4. Receita mensal (agregado por mês)
5. Percentual de cada produto na receita total (usar window function)

**Exercício 3: Importar CSV para SQL**

1. Baixe um dataset CSV (Kaggle)
2. Crie tabela no SQL Database compatível
3. Use Azure Data Studio para importar CSV:
   * Clique direito na tabela → “Import Data”
   * Ou use BULK INSERT (T-SQL)

**Exercício 4: Otimização de Query**

1. Execute uma query lenta (scan completo de tabela grande)
2. Use “Execution Plan” (Azure Data Studio) para analisar
3. Crie índice apropriado
4. Re-execute e compare tempo

## Resumo do Capítulo

Neste capítulo você aprendeu:

**SQL Database vs Data Lake:** Cada um tem seu papel (DW final vs raw storage)

**Provisionar SQL Database:** Tier Serverless (auto-pause, econômico)

**Azure Data Studio:** Conectar e gerenciar databases

**Star Schema:** Modelagem dimensional (fato + dimensões)

**SQL Essencial:** JOINs, GROUP BY, CTEs, Window Functions

**Criar tabelas:** Star Schema customizado

**Backup/Restore:** Point-in-time recovery automático

**Monitoramento:** Query Performance Insight, índices

**Próximo capítulo:** Azure Data Factory - construir pipelines ETL!

# Capítulo 4: Azure Data Factory - Primeiros Passos

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender o papel do Azure Data Factory em pipelines de dados
* Criar e configurar um Data Factory
* Construir seu primeiro pipeline (Copy Activity)
* Copiar dados: SQL Database → Data Lake
* Configurar Linked Services (conexões)
* Criar Datasets (esquemas de dados)
* Agendar pipelines com Triggers
* Monitorar execuções e debugar falhas
* Parametrizar pipelines (dinâmicos)

**Tempo estimado:** 4-5 horas

**Custo Azure:** R$ 5-15/mês (poucos pipelines, baixo volume)

## 4.1 O que é Azure Data Factory?

**Azure Data Factory (ADF)** é o serviço de **orquestração ETL/ELT** da Microsoft Azure. Pense nele como o “cérebro” que coordena a movimentação e transformação de dados.

ADF é uma ferramenta extremamente poderosas que o Engenheiro Azure precisa dominar no instante Zero da sua Carreira. Por excelencia o orquestrador da stack de engenharia de dados Azure. E não tem concorrente a sua altura, mesmo com quase 10 anos de estrada. Nem na AWS nem GCP. Sua evolução natural tem sido o seu irmão mais novo dentro do Microsoft Fabric, que herdou todo o know-how desses anos de estrada.

**Analogia:**

* **Data Lake / SQL Database:** Armazéns de dados
* **Databricks / Spark:** Fábrica que processa dados
* **Azure Data Factory:** Gerente logístico que coordena tudo (quando buscar, de onde, para onde, quando processar)

### O que ADF faz?

**1. Ingestão de Dados (Copy Data):**

* Conectar a 100+ fontes de dados (SQL Server, Oracle, APIs REST, arquivos FTP, SaaS...)
* Copiar dados para Azure (Data Lake, SQL Database, Synapse, Blob Storage)
* Suporta Big Data (paralelização, otimizações)

**2. Orquestração (Workflows):**

* Executar atividades em sequência ou paralelo
* Condicionais: “Se X falhar, execute Y”
* Loops: “Para cada arquivo na pasta, execute pipeline”

**3. Transformações Leves:**

* **Data Flows:** Transformações visuais (sem código) usando Spark
* **Lookup, Filter, Join:** Operações básicas
* Para transformações pesadas: chama Databricks, Synapse, Azure Functions

**4. Agendamento:**

* Triggers temporais: “Executar todo dia às 6h”
* Triggers de evento: “Executar quando arquivo chegar no Data Lake”
* Triggers manuais: On-demand

**5. Monitoramento:**

* Logs de execução detalhados
* Alertas de falhas
* Métricas de performance

### ADF vs Outras Ferramentas

|  |  |  |
| --- | --- | --- |
| **Ferramenta** | **Foco** | **Quando usar** |
| **Azure Data Factory** | Orquestração ETL/ELT, ingestão | Mover dados, agendar pipelines, transformações leves |
| **Databricks** | Transformações Big Data, ML | Processar dados complexos, feature engineering, ML |
| **Synapse Pipelines** | Similar ADF (integrado Synapse) | Se já usa Synapse Analytics |
| **Azure Functions** | Código customizado, eventos | Lógica complexa, APIs, microserviços |
| **Logic Apps** | Workflows low-code (não focado dados) | Automações gerais, integrações SaaS |

**Arquitetura típica:**

[Fontes diversas]

↓

Azure Data Factory (Copy Activity) → Data Lake (bronze)

↓

Azure Data Factory (Trigger) → Databricks (transformações) → Data Lake (silver/gold)

↓

Azure Data Factory (Copy Activity) → SQL Database (DW)

↓

Power BI (dashboards)

## 4.2 Arquitetura e Componentes do ADF

### Componentes Principais

**1. Pipeline:**

* Container de atividades (como um workflow)
* Define o que executar e em que ordem
* Exemplo: “Pipeline\_Ingestao\_Vendas”

**2. Activity (Atividade):**

* Ação individual dentro do pipeline
* Tipos:
  + **Copy Activity:** Copiar dados (fonte → destino)
  + **Data Flow:** Transformações visuais
  + **Lookup:** Ler valor de tabela/arquivo
  + **Execute Pipeline:** Chamar outro pipeline
  + **Databricks Notebook:** Executar notebook Databricks
  + **Stored Procedure:** Executar SP em SQL
  + **Web Activity:** Chamar API REST
  + **ForEach:** Loop sobre array
  + **If Condition:** Lógica condicional

**3. Linked Service (Serviço Vinculado):**

* Conexão a um sistema externo (como connection string)
* Exemplos:
  + Linked Service SQL Database (server, database, credenciais)
  + Linked Service Data Lake (storage account, credenciais)
  + Linked Service REST API (URL, autenticação)
* **Reutilizável:** Múltiplos datasets/pipelines usam mesma conexão

**4. Dataset:**

* Representação de dados em uma fonte/destino
* Aponta para Linked Service + localização específica
* Exemplos:
  + Dataset: Tabela “SalesLT.Customer” no SQL Database
  + Dataset: Arquivo “vendas.csv” na pasta “bronze/vendas/” do Data Lake
* Pode ser parametrizado (dataset genérico)

**5. Trigger (Gatilho):**

* Quando executar o pipeline
* Tipos:
  + **Schedule:** Recorrência (todo dia às 6h)
  + **Tumbling Window:** Janelas de tempo (processar últimas 24h)
  + **Event-based:** Quando arquivo chegar no Blob/Data Lake
  + **Manual:** Executar manualmente

**6. Integration Runtime (IR):**

* Infraestrutura de execução
* Tipos:
  + **Azure IR:** Cloud (padrão, grátis até certo limite)
  + **Self-hosted IR:** On-premises (conectar sistemas locais)
  + **Azure-SSIS IR:** Executar pacotes SSIS (legado)

### Fluxo de Criação de Pipeline

1. Criar Data Factory

↓

2. Criar Linked Services (conexões: SQL, Data Lake, etc)

↓

3. Criar Datasets (apontam para tabelas/arquivos)

↓

4. Criar Pipeline com Activities (Copy, Data Flow, etc)

↓

5. Debug/Test (executar manualmente)

↓

6. Criar Trigger (agendar)

↓

7. Publish (publicar mudanças)

↓

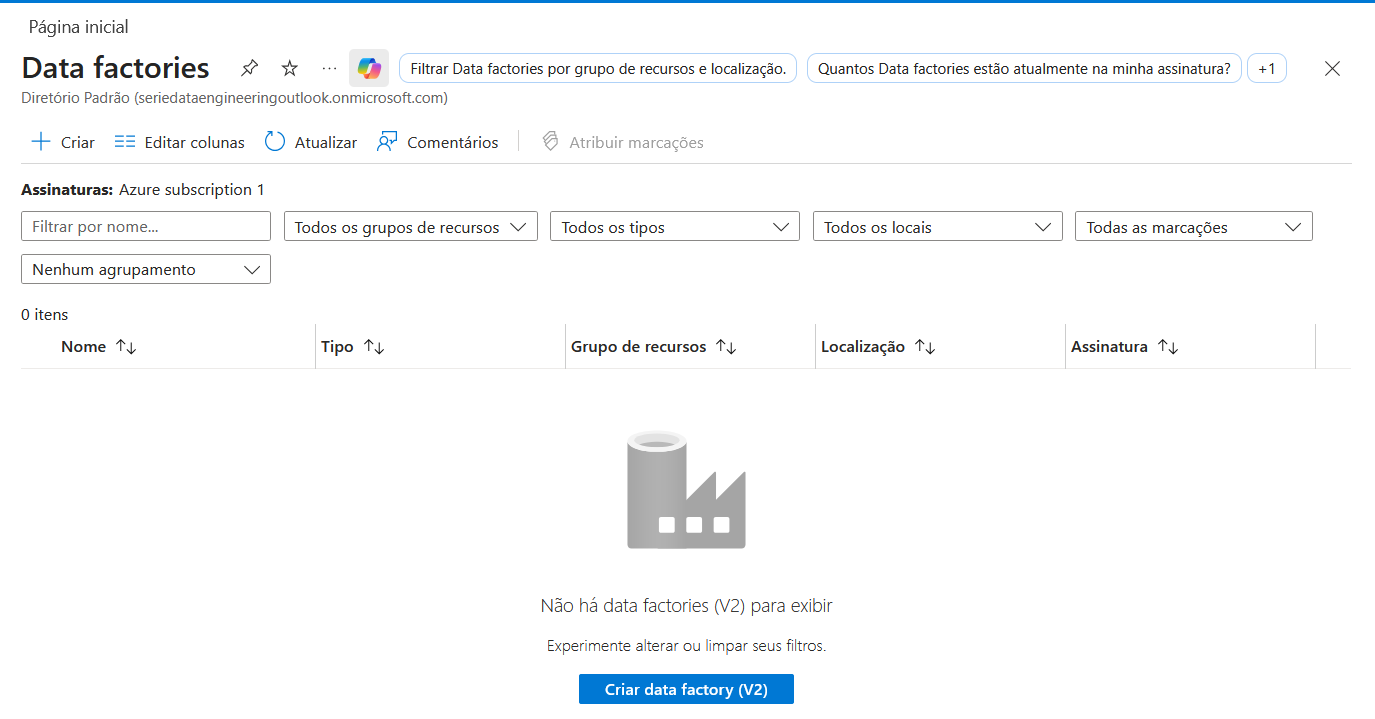
8. Monitor (acompanhar execuções)

## 4.3 Hands-On: Criar Azure Data Factory

Vamos criar nosso primeiro Data Factory!

### Passo 1: Iniciar Criação

1. **Azure Portal:** https://portal.azure.com
2. **Barra de pesquisa:** data factory
3. **Clique em “Data factories”**
4. **Clique em “+ Create”**

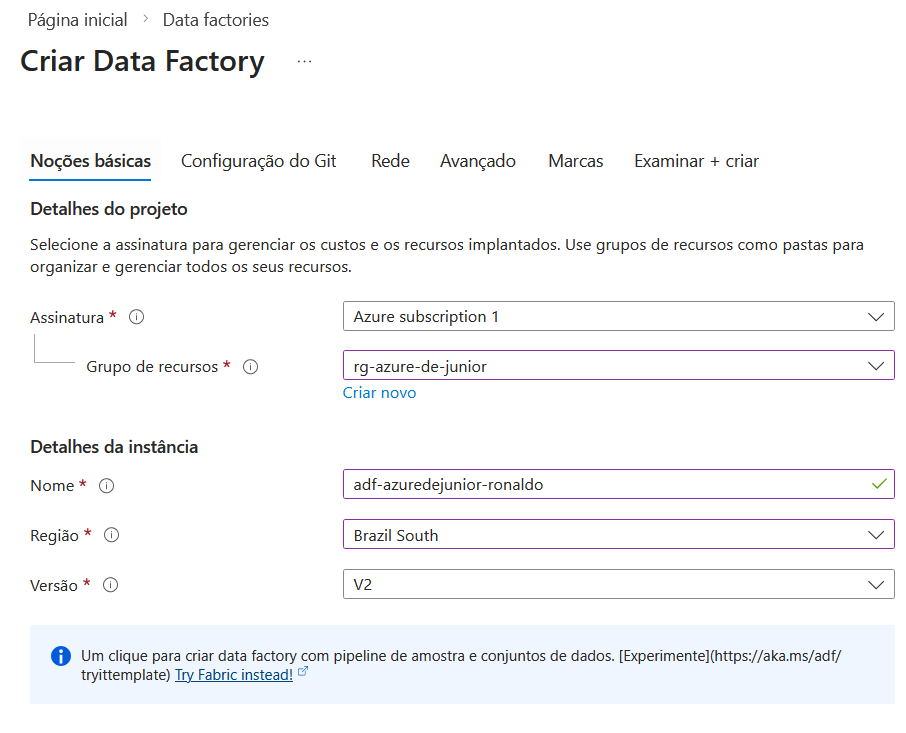


*Figura 4.1 - Página de Data Factories*

### Passo 2: Configurar Basics

**Aba “Basics”:**

1. **Subscription:** Free Trial
2. **Resource group:** rg-azure-de-junior
3. **Region:** Brazil South
4. **Name:** adf-azuredejunior-<seunome>
   * Exemplo: adf-azuredejunior-ronaldo
   * Globalmente único
   * Minúsculas, alfanumérico, hífens
   * Prefixo adf- (convenção)
5. **Version:** V2 (padrão, única opção atual)

****

*Figura 4.2 - Configuração básica do Data Factory*

### Passo 3: Git Configuration (Opcional)

**Aba “Git configuration”:**

Git integration permite versionar pipelines (recomendado para produção).

**Para este exercício:**

* Selecione **“Configure Git later”**
* Vamos pular integração Git por agora (simplificar)

**Produção:** Integre com Azure DevOps ou GitHub!

**Clique em “Next: Networking”**

### Passo 4: Networking (Padrão)

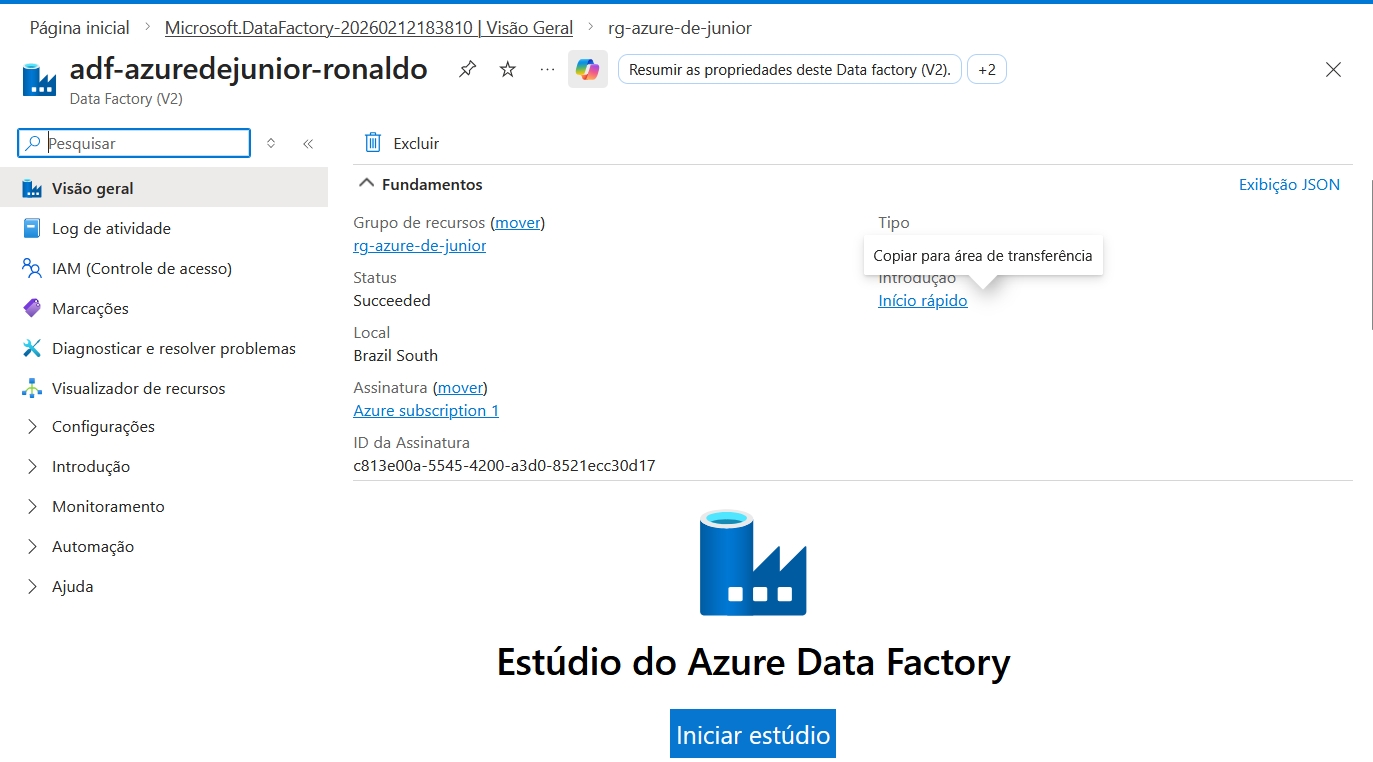
**Aba “Networking”:**

* **Public endpoint:** Habilitado (padrão)
* Para desenvolvimento, acesso público OK
* Produção: Managed Virtual Network (private endpoints)

**Deixe padrão e clique em “Next”**

### Passo 5: Review + Create

1. **Revise o resumo:**
   * Name: adf-azuredejunior-[seunome]
   * Region: Brazil South
   * Git: Configure later
2. **Clique em “Create”**
3. **Aguarde deployment** (1-2 minutos)
4. **Clique em “Go to resource”**

**

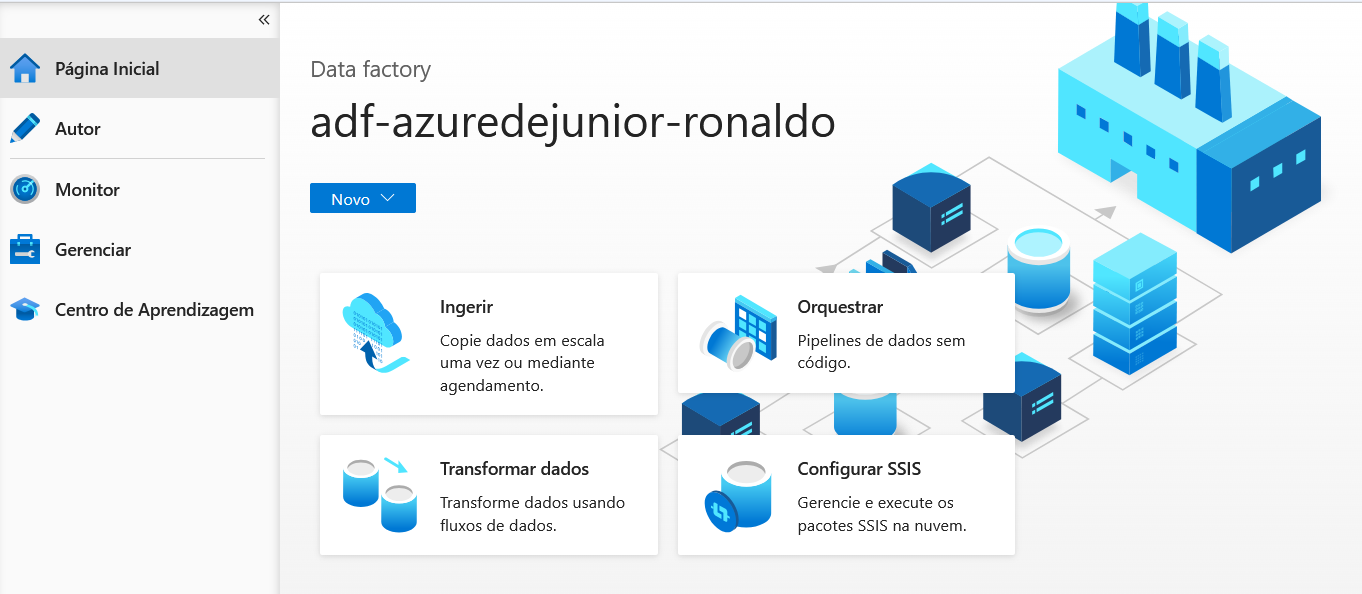
*Figura 4.3 - Overview do Azure Data Factory criado*

### Passo 6: Abrir Data Factory Studio

**Na página Overview:**

1. **Clique no botão grande:** **“Open Azure Data Factory Studio”**
   * Ou **“Launch Studio”**
2. **Nova aba abre:** ADF Studio (interface visual de desenvolvimento)

**ADF Studio** é onde você cria pipelines, datasets, linked services, etc.

**

*Figura 4.4 - Azure Data Factory Studio - Home*

**Principais seções do ADF Studio:**

* **Home:** Acesso rápido a tarefas comuns
* **Author:** Criar pipelines, datasets, data flows (onde passamos mais tempo!)
* **Monitor:** Acompanhar execuções, debugar falhas
* **Manage:** Linked services, integration runtimes, triggers, credenciais

## 4.4 Criar Linked Services (Conexões)

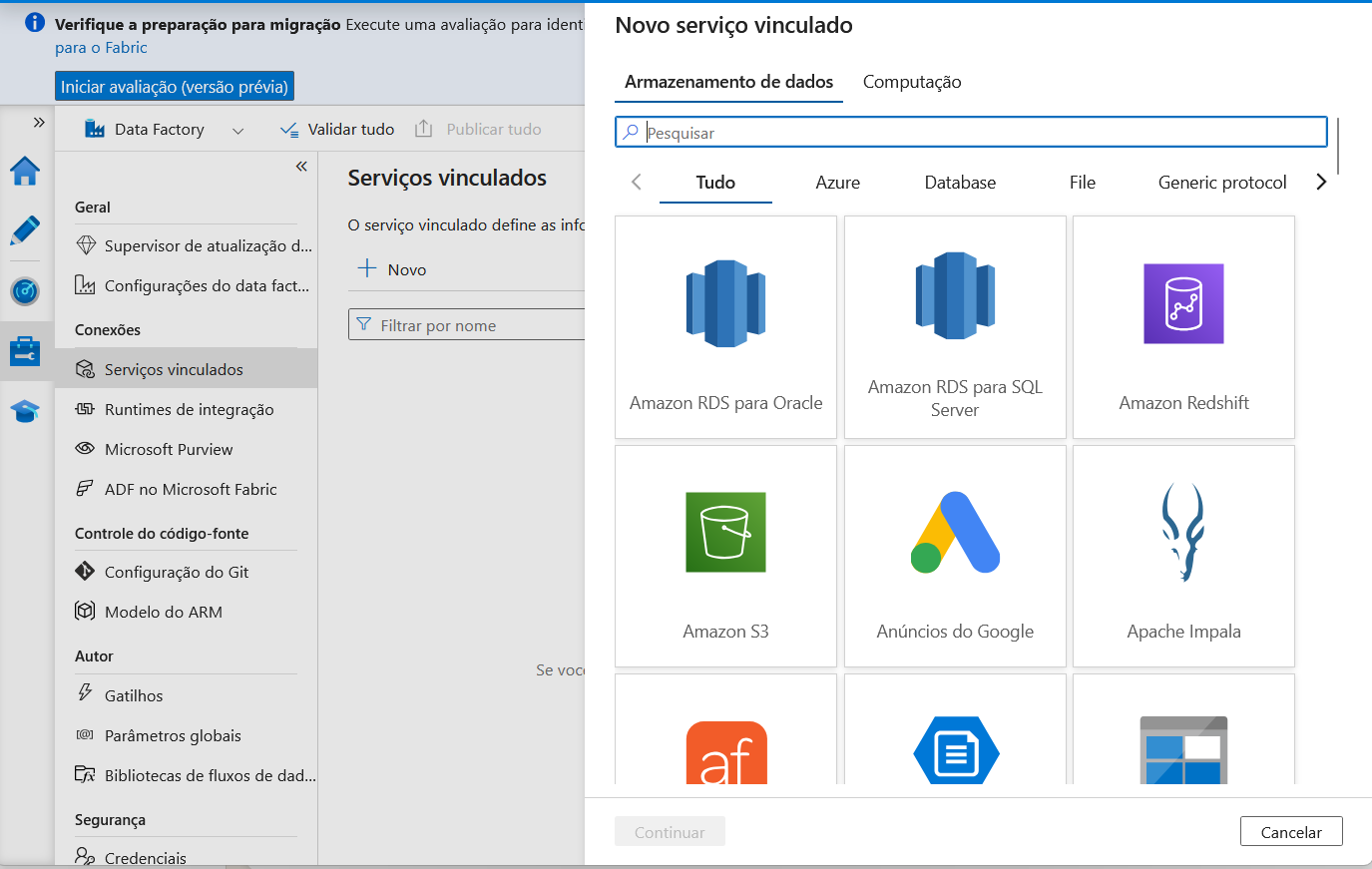
Antes de criar pipelines, precisamos conectar ADF às fontes/destinos de dados.

### Linked Service 1: Azure SQL Database

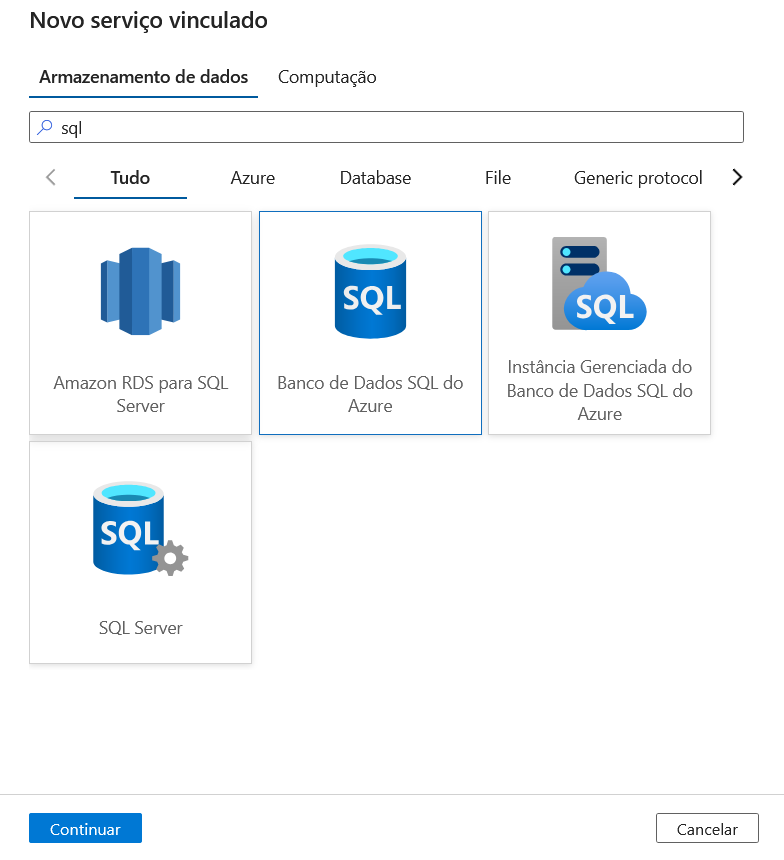
Vamos conectar ao SQL Database criado no Cap 3.

**No ADF Studio:**

1. **Clique em “Manage”** (ícone de engrenagem na barra lateral esquerda)
2. **No menu lateral, clique em “Linked services”**
3. **Clique em “+ New”**

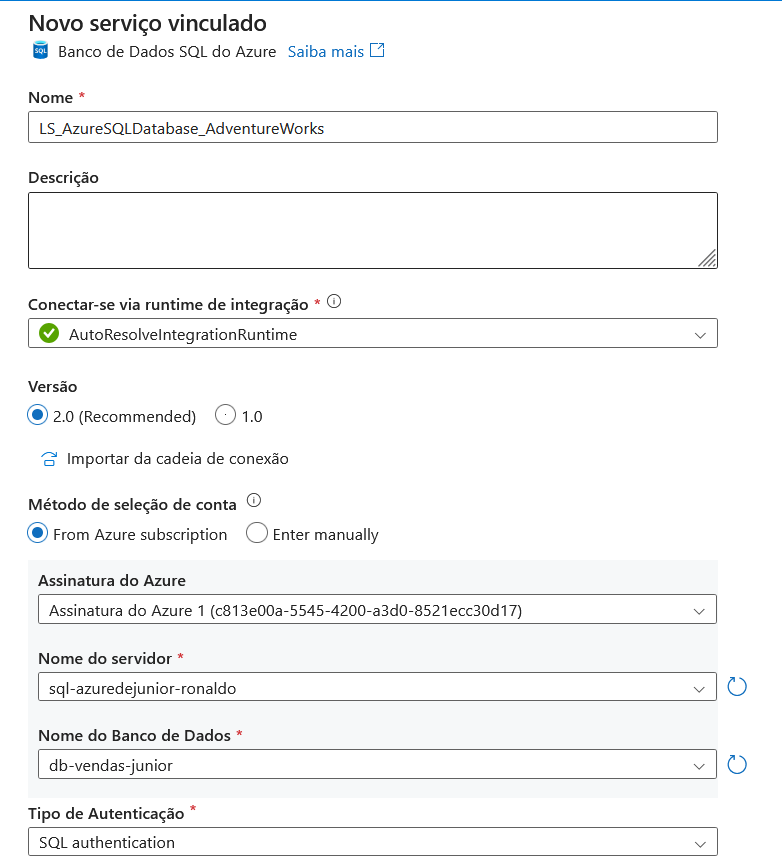
* Figura 4.5 - Página de Linked Services*

1. **No painel “New linked service”:**
   * **Barra de pesquisa:** Digite azure sql database
   * **Selecione:** “Azure SQL Database”
   * **Clique em “Continue”**

**

*Figura 4.6 - Seleção de tipo: Azure SQL Database*

1. **Configurar Linked Service SQL:**
   * **Name:** LS\_AzureSQLDatabase\_AdventureWorks
     + Convenção: LS\_ (Linked Service) + tipo + nome descritivo
   * **Description:** Conexão com SQL Database - AdventureWorksLT (opcional)
   * **Integration runtime:** AutoResolveIntegrationRuntime (padrão)
   * **Account selection method:** From Azure subscription
   * **Azure subscription:** Selecione sua subscription
   * **Server name:** Selecione seu SQL Server (sql-azuredejunior-[seunome])
   * **Database name:** db-vendas-junior
   * **Authentication type:** SQL authentication
   * **User name:** sqladmin
   * **Password:** Sua senha SQL (ex: Azur3@2026!)
   * **Azure Key Vault:** Deixe desmarcado (para simplificar)
     + Produção: Use Key Vault para senhas!

**

*Figura 4.7 - Configuração do Linked Service SQL Database*

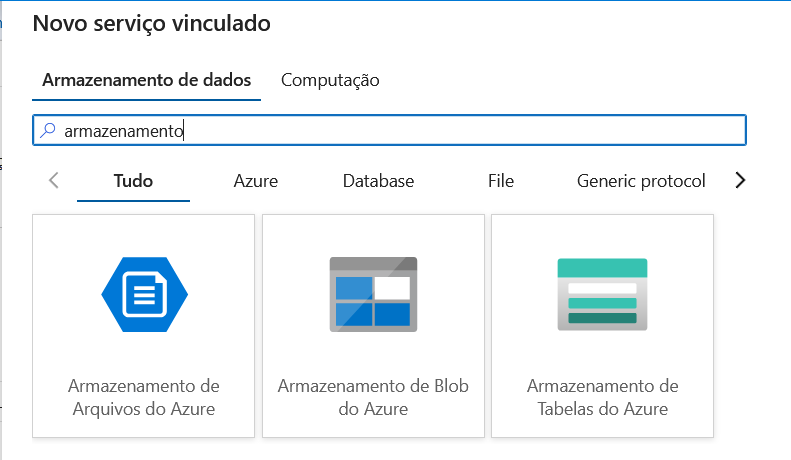
1. **Clique em “Test connection”** (parte inferior do formulário)
   * **Se sucesso:** “Connection successful”
   * **Se falhar:** Verifique firewall do SQL Server (adicione IP do ADF). Algo como “Client with IP address 'xxx.xxx.xxx.x' is not allowed to access the server”, vá até Network e adicione esse IP na lista de exceções. Talvez ao longo dos exercícios seja necessário repetir algumas vezes. Em ambiente de produção, rodando com as Vnets configuradas isso tende a não ocorrer. Evite abrir todo range de IP, mesmo em ambiente DEV.
2. **Clique em “Create”**

**Linked Service SQL criado!**

### Linked Service 2: Azure Data Lake Gen2

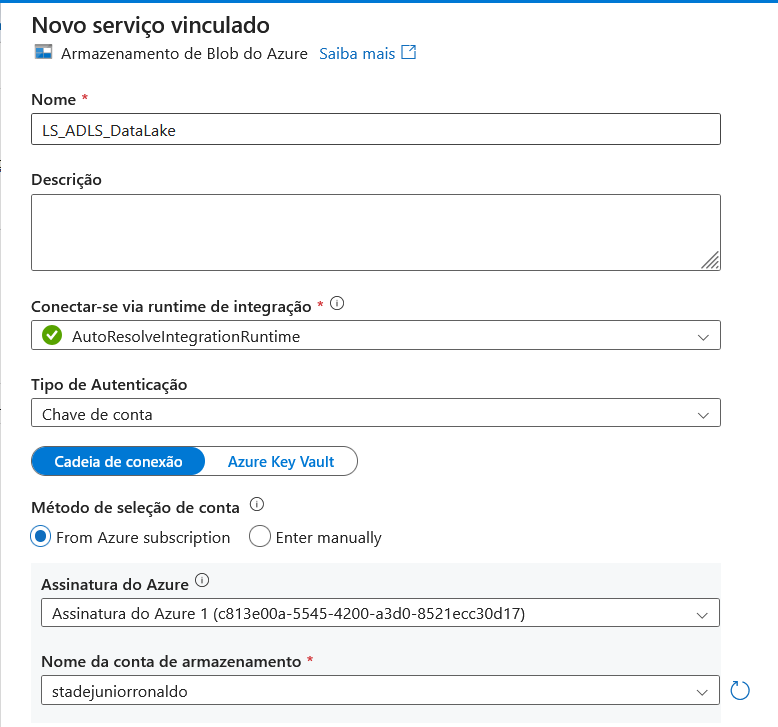
Agora conectar ao Data Lake criado no Cap 2.

1. **Ainda em “Manage” → “Linked services”**
2. **Clique em “+ New”**
3. **Pesquise:** azure data lake storage gen2
4. **Selecione:** “Azure Data Lake Storage Gen2”
5. **Clique em “Continue”**

**

*Figura 4.8 - Seleção de tipo: Azure Data Lake Gen2*

1. **Configurar Linked Service ADLS Gen2:**
   * **Name:** LS\_DataLake\_Bronze
     + Ou LS\_ADLS\_DataLake (nome genérico)
   * **Authentication method:** Account key (mais simples)
     + Alternativa: Managed Identity (mais seguro, produção)
   * **Account selection method:** From Azure subscription
   * **Azure subscription:** Selecione sua subscription
   * **Storage account name:** stadejunior<seunome>
   * **Authentication:** Account key (auto-preenchido após selecionar storage)

**

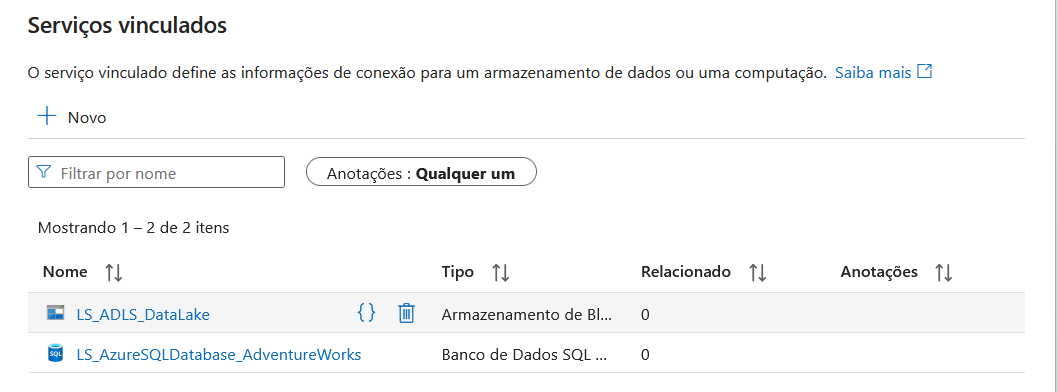
*Figura 4.9 - Configuração do Linked Service Data Lake Gen2*

1. **Clique em “Test connection”**
   * **Se sucesso:** “Connection successful”
2. **Clique em “Create”**

**Linked Service Data Lake criado!**

**Você agora tem 2 Linked Services:**

* LS\_AzureSQLDatabase\_AdventureWorks
* LS\_DataLake\_Bronze (ou LS\_ADLS\_DataLake)

**

*Figura 4.10 - Linked Services criados*

## 4.5 Criar Datasets

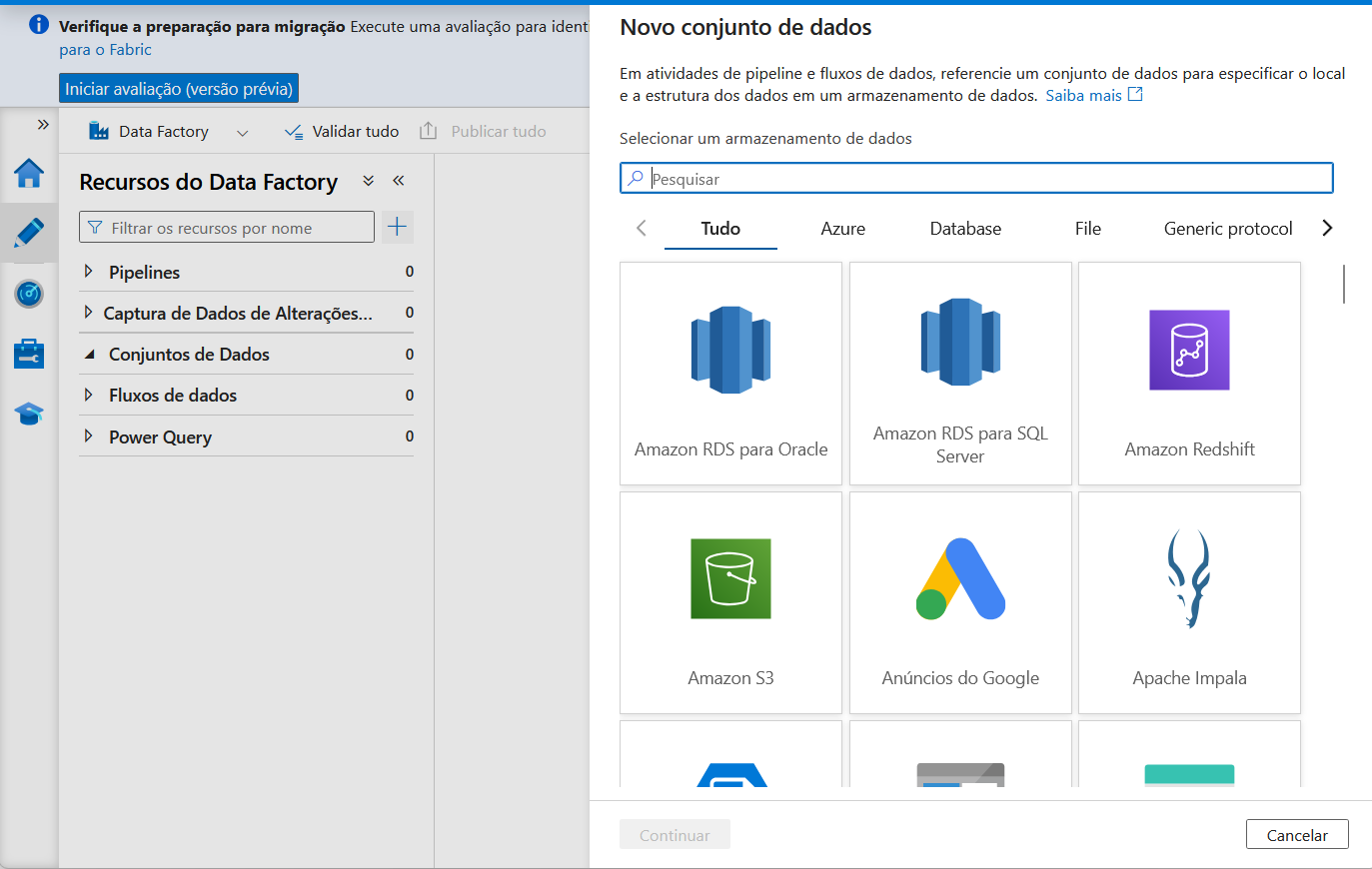
Datasets representam os dados específicos que vamos ler/escrever.

### Dataset 1: Tabela SQL (Source)

Vamos criar dataset apontando para tabela SalesLT.Customer no SQL Database.

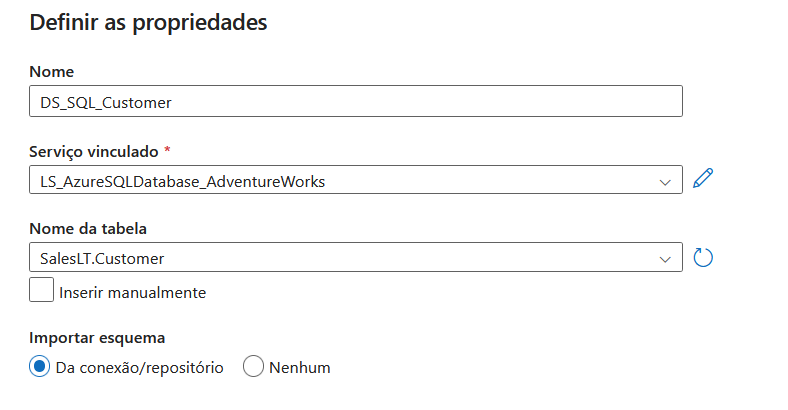
**No ADF Studio:**

1. **Clique em “Author”** (ícone de lápis na barra lateral esquerda)
2. **No painel Factory Resources (esquerda), clique no “+” ao lado de “Datasets”**
3. **Ou:** Clique no menu “...” ao lado de “Datasets” → “New dataset”

**

*Figura 4.11 - Criar novo Dataset*

1. **No painel “New dataset”:**
   * **Pesquise:** azure sql database
   * **Selecione:** “Azure SQL Database”
   * **Clique em “Continue”**
2. **Configurar Dataset SQL:**
   * **Name:** DS\_SQL\_Customer
     + Convenção: DS\_ + tipo + nome da tabela
   * **Linked service:** Selecione LS\_AzureSQLDatabase\_AdventureWorks
   * **Table:** Selecione SalesLT.Customer (dropdown)
     + Se não aparecer, clique em “Refresh” ou digite manualmente
   * **Import schema:** From connection/store (recomendado)
     + ADF lê schema automaticamente

**

*Figura 4.12 - Configuração do Dataset SQL (Customer)*

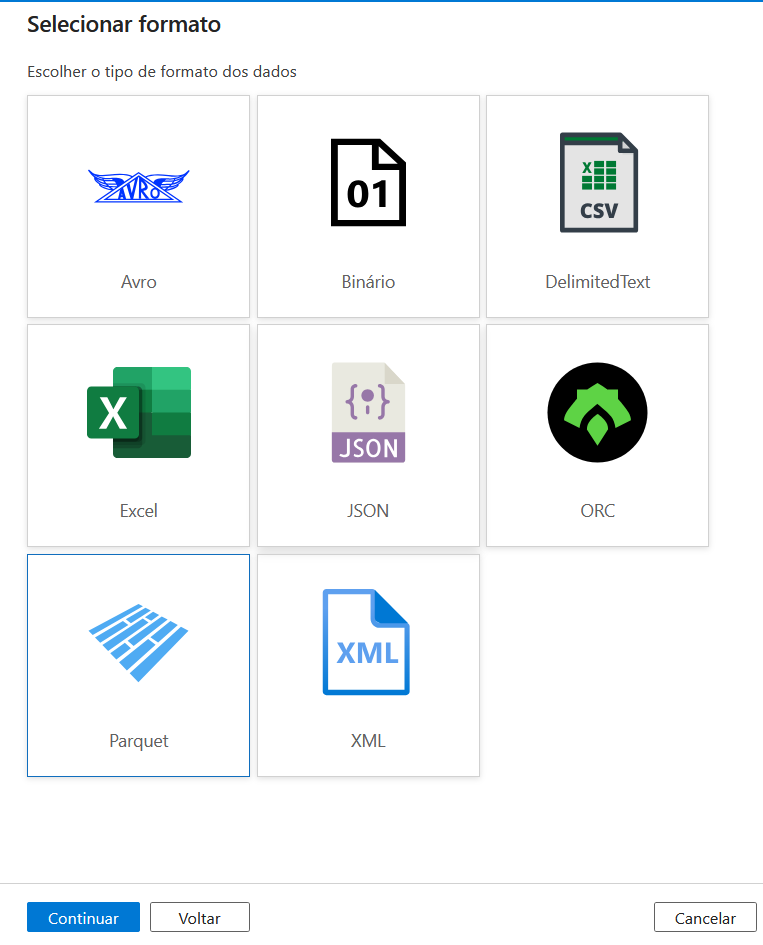
1. **Clique em “OK”**
2. **Dataset criado!** Você verá aba de configuração com:
   * **Schema:** Colunas da tabela (CustomerID, FirstName, LastName, etc.)
   * **Parameters:** Para datasets dinâmicos (veremos depois)

**Clique em “Publish all”** (topo da página) para salvar

### Dataset 2: Arquivo Parquet no Data Lake (Sink/Destino)

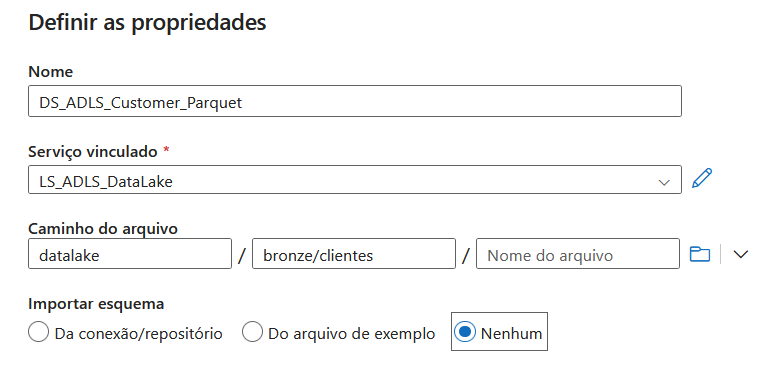
Vamos criar dataset para salvar dados no Data Lake em formato Parquet.

1. **No painel Author, clique novamente em “+” → “Dataset”**
2. **Selecione:** “Azure Data Lake Storage Gen2” + formato “Parquet”
3. **Clique em “Continue”**

**

*Figura 4.13 - Seleção de tipo: Parquet (Data Lake)*

1. **Configurar Dataset Parquet:**
   * **Name:** DS\_ADLS\_Customer\_Parquet
   * **Linked service:** LS\_DataLake\_Bronze (ou LS\_ADLS\_DataLake)
   * **File path:**
     + **Container/filesystem:** datalake
     + **Directory:** bronze/clientes
     + **File name:** customer.parquet (ou deixe vazio para nome dinâmico)
   * **Import schema:** None (destino, não precisa importar)

**

*Figura 4.14 - Configuração do Dataset Parquet (destino Data Lake)*

1. **Clique em “OK”**
2. **Clique em “Publish all”** para salvar

**Datasets criados:**

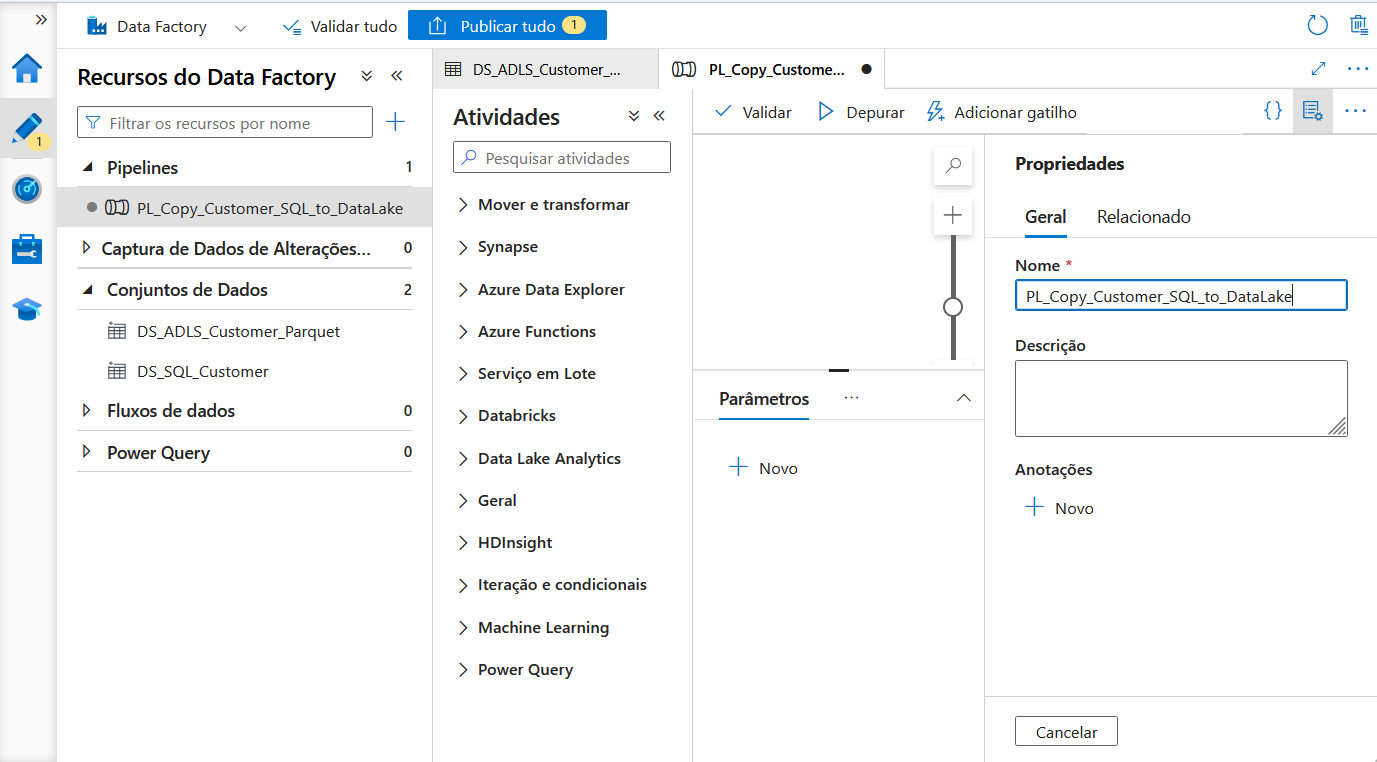
* DS\_SQL\_Customer (source: SQL)
* DS\_ADLS\_Customer\_Parquet (sink: Data Lake)

## 4.6 Criar Pipeline com Copy Activity

Agora a parte divertida: criar pipeline para copiar dados!

### Passo 1: Criar Pipeline

1. **No painel Author, clique em “+” → “Pipeline”**
2. **Nome do pipeline:** PL\_Copy\_Customer\_SQL\_to\_DataLake
   * Convenção: PL\_ + descrição clara

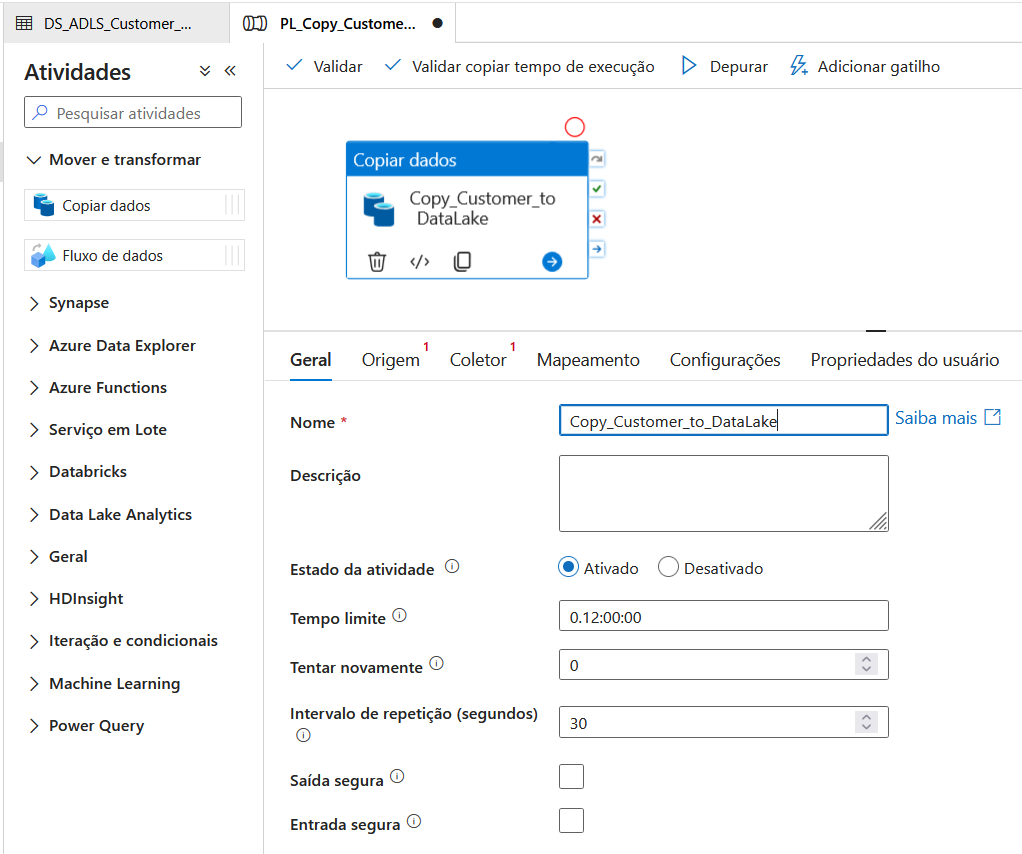
**

*Figura 4.15 - Novo pipeline criado (vazio)*

### Passo 2: Adicionar Copy Activity

**Na área de design do pipeline:**

1. **No painel “Activities” (esquerda), expanda “Move & transform”**
2. **Arraste “Copy data”** para o canvas central
3. **Aba “General” da Copy Activity:**
   * **Name:** Copy\_Customer\_to\_DataLake

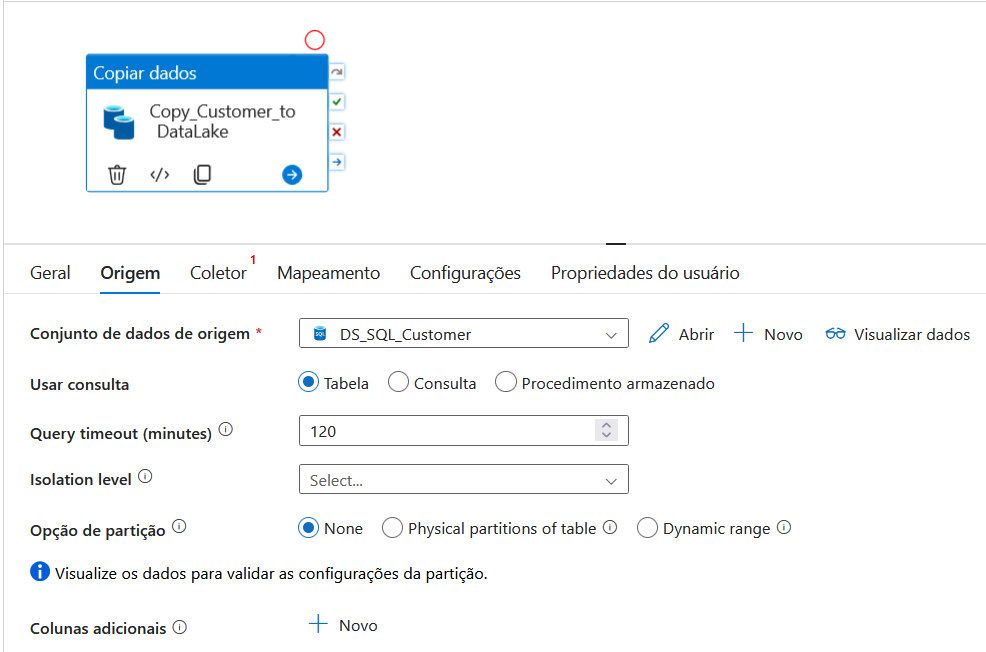
**

*Figura 4.16 - Copy Activity adicionada ao pipeline*

### Passo 3: Configurar Source (Origem)

**Clique na aba “Source”:**

1. **Source dataset:** Selecione DS\_SQL\_Customer
2. **Use query:** Table (padrão)
   * Alternativa: Query (escrever SQL customizado)
3. **Deixe demais configurações padrão**

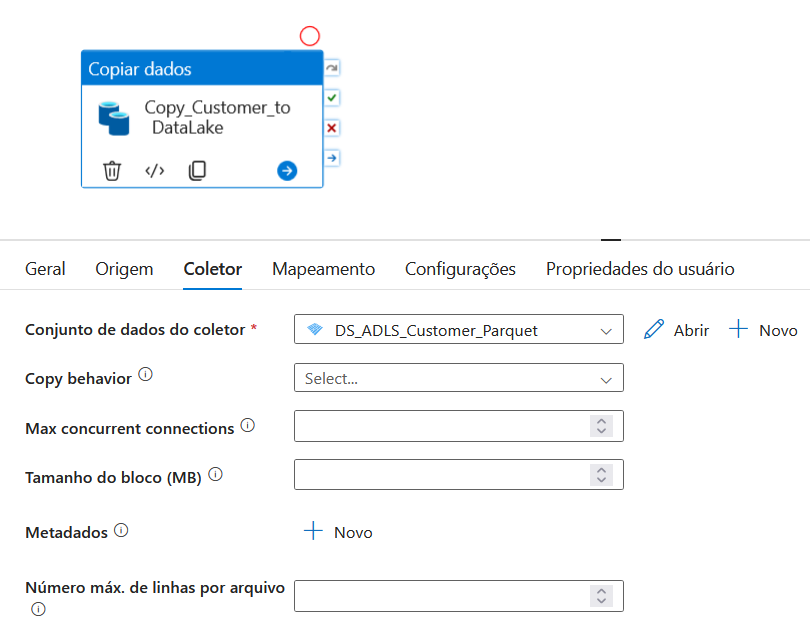
**

*Figura 4.17 - Configuração do Source (origem SQL)*

### Passo 4: Configurar Sink (Destino)

**Clique na aba “Sink”:**

1. **Sink dataset:** Selecione DS\_ADLS\_Customer\_Parquet
2. **Copy behavior:** Deixe em branco
   * Alternativas: Merge, Flaten, Preserve

**

*Figura 4.18 - Configuração do Sink (destino Data Lake Parquet)*

### Passo 5: Configurar Mapping (Opcional)

**Aba “Mapping”:**

* **Import schemas:** Clique para importar schema de source e sink
* **Auto map:** ADF mapeia colunas automaticamente (mesmo nome)
* **Manual mapping:** Ajuste se colunas tiverem nomes diferentes

**Para este exercício:** Deixe auto map (padrão OK)

### Passo 6: Configurar Settings

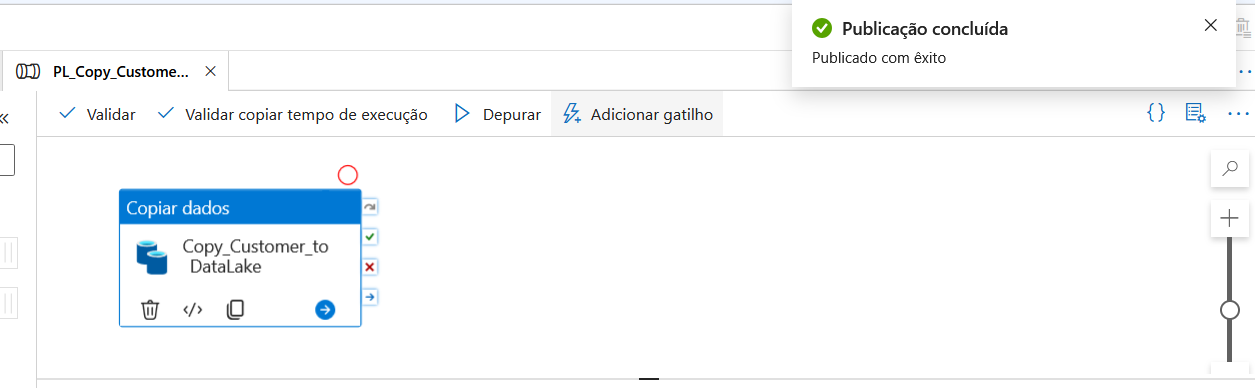
**Aba “Settings”:**

* **Data integration units (DIU):** Auto (padrão)
  + DIU = unidades de processamento paralelo (quanto maior, mais rápido, mais caro)
  + Auto = ADF escolhe baseado em volume de dados
* **Degree of copy parallelism:** Auto (padrão)
* **Enable staging:** Desabilitado
  + Staging: Usa Blob intermediário (útil para PolyBase em Synapse)

**Deixe padrão**

### Passo 7: Salvar e Validar

1. **Clique em “Validate”** (ícone de checklist no topo do pipeline)
2. **Verifica erros de configuração**
3. **Se “Pipeline validation: successful” → tudo certo!**
4. **Clique em “Publish all”** (salva mudanças)

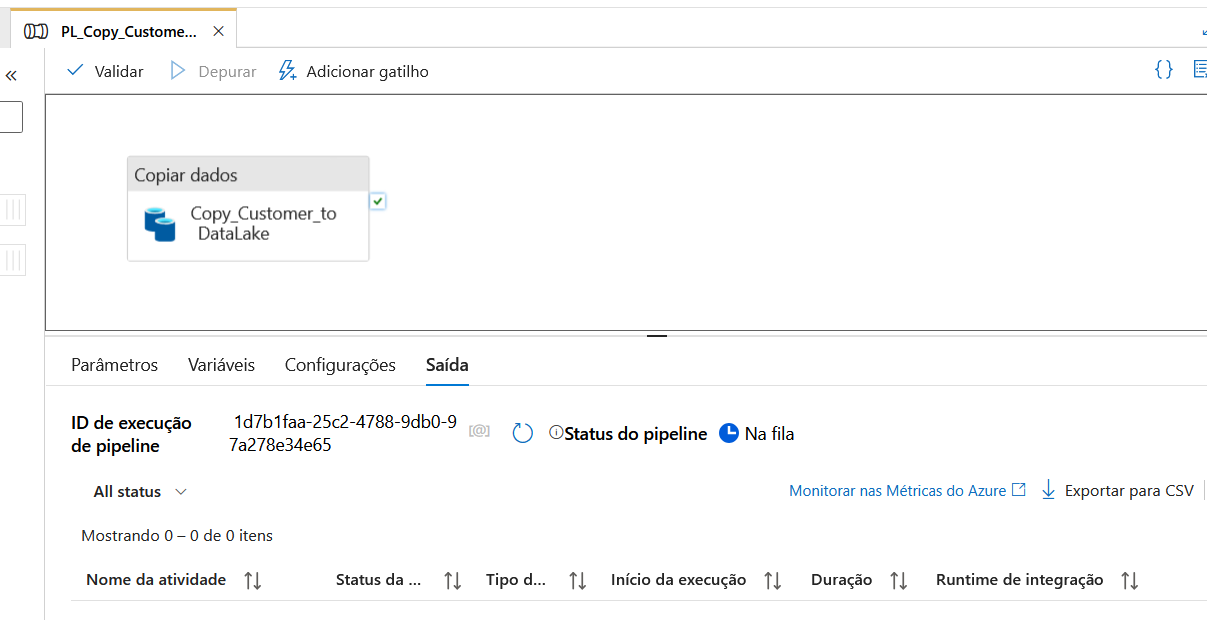
* Figura 4.19 - Pipeline completo e validado*

## 4.7 Debug e Executar Pipeline

Antes de agendar, vamos testar manualmente!

### Debug Manual

1. **No topo do pipeline, clique em “Debug”**
2. **Se o pipeline tiver parâmetros (não tem ainda), popup aparece pedindo valores → clique “OK”**
3. **Execução inicia!** Você verá:
   * Status: In Progress → Succeeded (ou Failed)
   * Painel inferior “Output” mostra progresso em tempo real

**

*Figura 4.20 - Debug do pipeline em execução*

1. **Aguarde conclusão** (10-30 segundos para tabela Customer)
2. **Status “Succeeded”!**
3. **Clique no ícone de óculos** () na linha da Copy Activity no Output para ver detalhes:
   * **Rows read:** Quantidade de linhas lidas da fonte
   * **Rows written:** Quantidade de linhas escritas no destino
   * **Data read:** Volume de dados (MB)
   * **Duration:** Tempo de execução

**

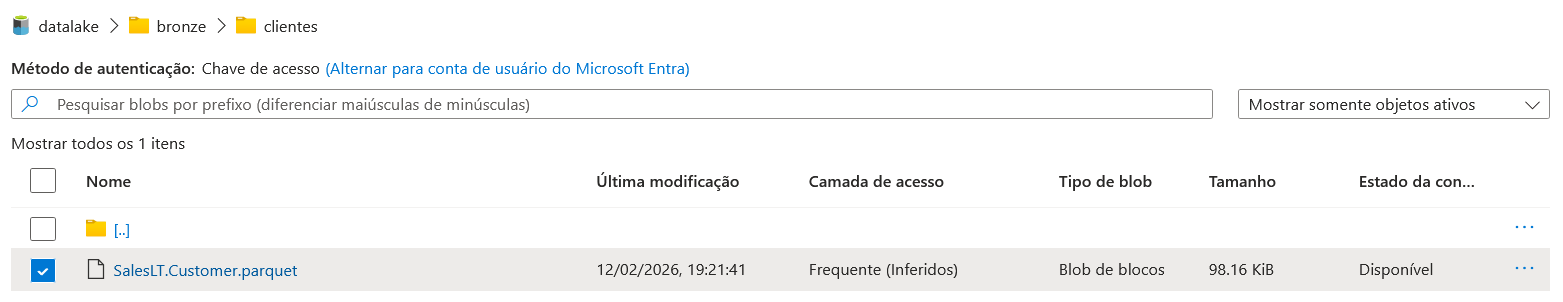
*Figura 4.21 - Detalhes da execução da Copy Activity*

### Verificar Dados no Data Lake

Vamos confirmar que dados foram copiados!

**Azure Portal ou Storage Explorer:**

1. **Navegue até:** datalake/bronze/clientes/
2. **Você verá:** SalesLT.Customer.parquet (arquivo criado!)
3. **Tamanho:** ~50-100 KB (comprimido)

* Figura 4.22 - Arquivo Parquet criado no Data Lake*

**Sucesso! Você executou seu primeiro pipeline ETL!**

**Fluxo completo:**

SQL Database (SalesLT.Customer)

↓ (Copy Activity)

Data Lake (bronze/clientes/customer.parquet)

## 4.8 Monitoramento de Pipelines

ADF oferece monitoramento robusto de execuções.

### Monitor Page

1. **No ADF Studio, clique em “Monitor”** (ícone de gráfico na barra lateral esquerda)
2. **Aba “Pipeline runs”** (debug/depurar)
3. **Você verá lista de execuções:**
   * **Pipeline name:** PL\_Copy\_Customer\_SQL\_to\_DataLake
   * **Run start:** Data/hora de início
   * **Duration:** Duração
   * **Status:** Succeeded, Failed, In Progress, Cancelled
   * **Triggered by:** Debug (manual), Schedule (trigger), etc.

* Figura 4.23 - Monitoramento de execuções de pipelines*

1. **Clique no nome do pipeline** para ver detalhes
2. **Você verá execução das Activities:**
   * Copy\_Customer\_to\_DataLake: Succeeded
   * Duração individual
   * Input/Output de cada activity
3. **Clique no ícone de óculos () da activity** para ver estatísticas detalhadas

### Filtros e Pesquisa

**No Monitor:**

* **Date range:** Últimas 24h, 7 dias, 30 dias, custom
* **Status filter:** Todos, Succeeded, Failed, In Progress
* **Search:** Buscar por nome de pipeline
* **Triggered by filter:** Debug, Schedule, Manual, etc.

### Alertas (Opcional)

**Para produção:**

1. **No Azure Portal, vá para Data Factory**
2. **“Alerts” → “+ New alert rule”**
3. **Configure:**
   * **Condition:** “Pipeline run failed”
   * **Actions:** Enviar email, chamar webhook, Logic App

**Exemplo:** “Se pipeline falhar, enviar email para engenheiro-de-dados@empresa.com”

## 4.9 Triggers: Agendar Pipelines

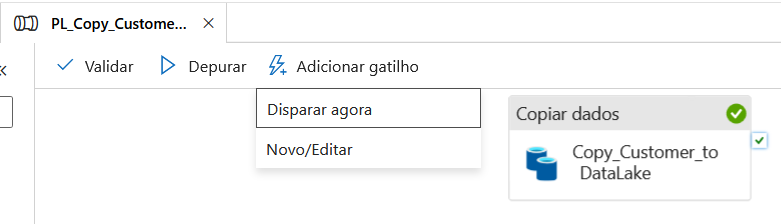
Pipelines podem ser executados automaticamente com Triggers.

### Criar Schedule Trigger

Vamos agendar nosso pipeline para executar todo dia às 6h.

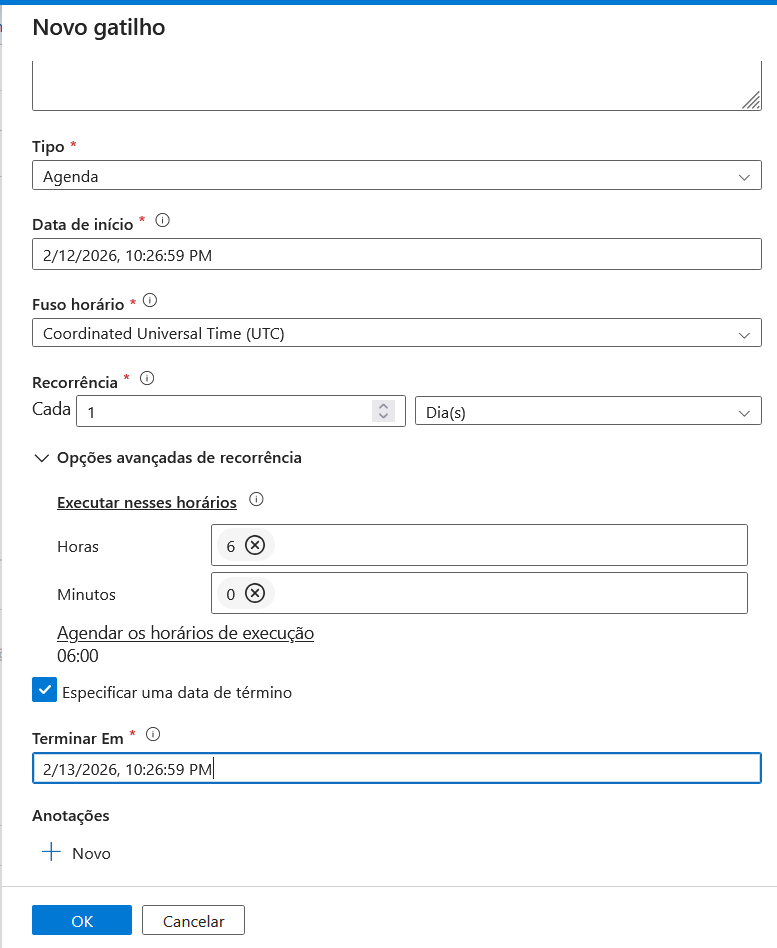
**No ADF Studio:**

1. **Vá para “Author”**
2. **Abra o pipeline:** PL\_Copy\_Customer\_SQL\_to\_DataLake
3. **Clique em “Add trigger”** (topo do pipeline) → **“New/Edit”**

**

*Figura 4.24 - Menu para adicionar Trigger*

1. **No popup “Add triggers”:**
   * **Choose trigger dropdown:** Clique em “+ New”
2. **Configurar Trigger:**
   * **Name:** TR\_Daily\_6AM\_Customer\_Ingest
   * **Type:** Schedule (recorrência)
   * **Start date:** Hoje (ou data futura)
   * **Time zone:** (UTC-03:00) Brasilia
   * **Recurrence:**
     + **Every:** 1 Day(s)
     + **At:** 06:00 (6h da manhã)
   * **End date:** No End (executa indefinidamente)
     + Ou defina data de término (sugestão do livro, defina uma data, para não correr riscos)

**

*Figura 4.25 - Configuração de Trigger agendado (diário às 6h)*

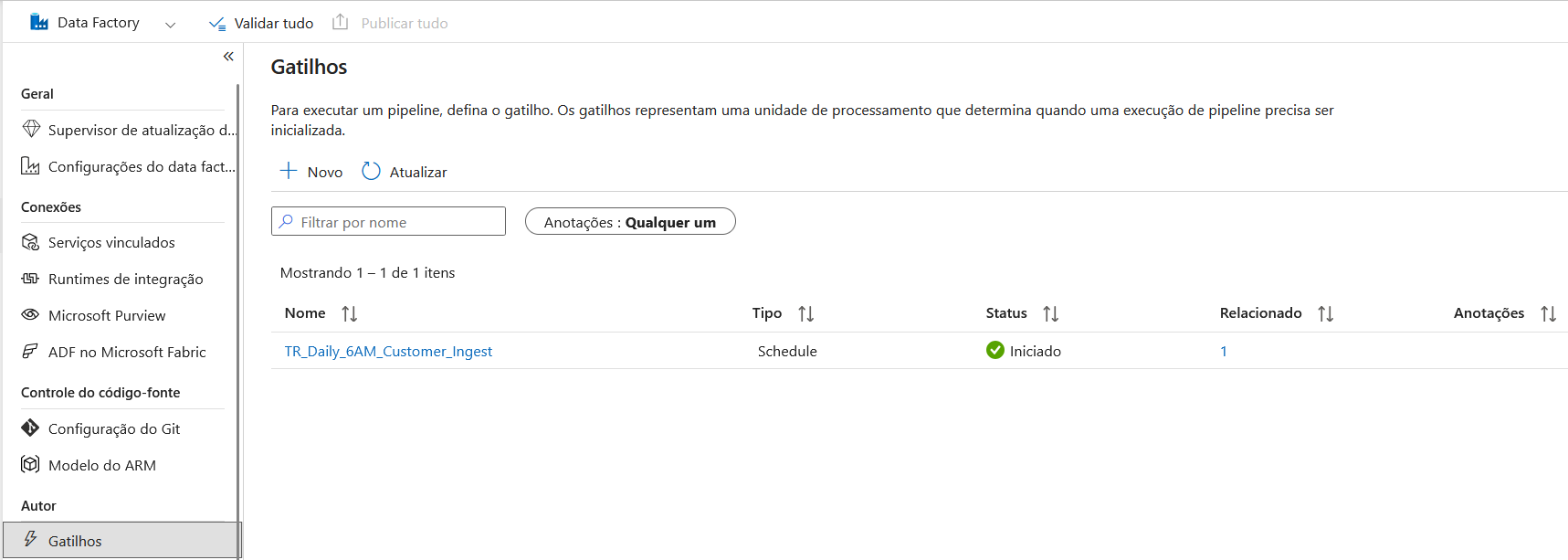
1. **Clique em “OK”**
2. **Popup “Trigger Run Parameters”** (se pipeline tiver parâmetros):
   * Defina valores padrão
   * Nosso pipeline não tem, então clique “OK” diretamente
3. **Clique em “Publish all”** para ativar o trigger

**Trigger criado e ativado!**

### Gerenciar Triggers

**No ADF Studio → “Manage” → “Triggers”:**

* Lista de todos triggers
* **Ativar/Desativar:** Toggle switch
* **Editar:** Clique no trigger
* **Deletar:** Menu “...”

* Figura 4.26 - Lista de Triggers criados*

### Outros Tipos de Trigger

Dominar quando usar cada tipo de trigger é primordial para o Engenheiro de dados. Estude pelo Learn da Microsoft cada um dos apresentados abaixo.

**Tumbling Window Trigger:**

* Executa em janelas de tempo fixas (ex: a cada 1 hora, processando dados da última hora)
* Útil para processamento incremental com garantias

**Event-based Trigger (Storage Event):**

* Executa quando arquivo chega em Blob/Data Lake
* Exemplo: “Quando .csv chegar em bronze/vendas/, executar pipeline de processamento”

**Manual Trigger:**

* “Trigger Now” (execução sob demanda)

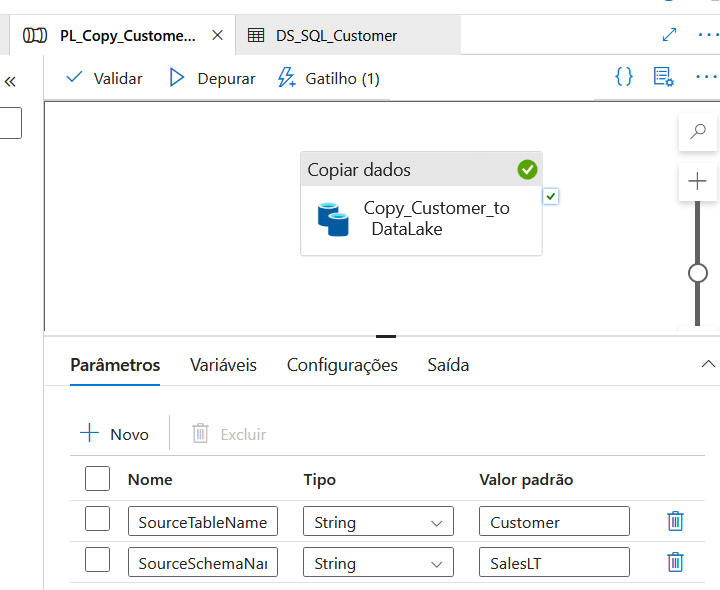
## 4.10 Parametrização de Pipelines

Pipelines parametrizados são **reutilizáveis** e **dinâmicos**.

**Exemplo:** Em vez de criar 10 pipelines para copiar 10 tabelas, crie 1 pipeline parametrizado e passe nome da tabela como parâmetro.

### Adicionar Parâmetros ao Pipeline

1. **Abra o pipeline:** PL\_Copy\_Customer\_SQL\_to\_DataLake
2. **Clique em área vazia do canvas** (fundo)
3. **No painel inferior, aba “Parameters”**
4. **Clique em “+ New”**
5. **Configure parâmetro:**
   * **Name:** SourceSchemaName, SourceTableName
   * **Type:** String
   * **Default value:** SalesLT, Customer

**

*Figura 4.27 - Parâmetro de pipeline criado*

### Usar Parâmetro em Dataset

**Modificar Dataset SQL para aceitar parâmetro:**

1. **Vá para “Author” → “Datasets” → `DS\_SQL\_Customer`**
2. **Aba “Parameters”**
3. **Clique em “+ New”**
   * **Name:** TableName
   * **Type:** String
4. **Aba “Connection”**
5. **No campo “Table”:**
   * Clique em Inserir Manualmente
   * Clique no ícone “Add dynamic content”
   * Na expressão, escreva:

@dataset().TableName

@dataset().SchemaName

* + Clique “OK”

**Agora o dataset aceita nome de tabela dinâmico!**

### Passar Parâmetro no Pipeline

1. **Volte ao pipeline:** PL\_Copy\_Customer\_SQL\_to\_DataLake
2. **Clique na Copy Activity**
3. **Aba “Source”**
4. **No campo “Source dataset” → Ícone de configuração (engrenagem)**
5. **Popup “Dataset properties” abre:**
   * **TableName:** Clique em “Add dynamic content”
   * Expressão:

@pipeline().parameters.SourceTableName

@pipeline().parameters.SourceSchemaName

**Agora pipeline passa parâmetro para dataset!**

### Testar com Debug

1. **Clique em “Debug”**
2. **Popup “Parameters” aparece:**
   * **SourceTableName:** Deixe padrão (SalesLT.Customer) ou mude para SalesLT.Product
3. **Clique “OK” → Pipeline executa com parâmetro**

**Reutilização:** Agora você pode chamar este pipeline para copiar qualquer tabela, apenas mudando parâmetro!

## 4.11 Loops e Condicionais

### ForEach Activity

Processar lista de itens (ex: copiar múltiplas tabelas).

**Exemplo:**

1. **Criar parâmetro de pipeline tipo Array:**
   * **Name:** TableList
   * **Default value:**

**JSON**

[“SalesLT.Customer”, “SalesLT.Product”, “SalesLT.SalesOrderHeader”]

1. **Adicionar “ForEach” activity ao pipeline**
2. **Configurar ForEach:**
   * **Items:** @pipeline().parameters.TableList
   * **Sequential:** False (paralelo)
3. **Dentro do ForEach, adicionar Copy Activity:**
   * Source table: @item() (item atual do loop)

**Resultado:** Copia 3 tabelas em paralelo!

### If Condition Activity

Lógica condicional (se/senão).

**Exemplo:** “Se arquivo existe, copie; senão, envie alerta”

1. **Adicionar “Get Metadata” activity** (verifica se arquivo existe)
2. **Adicionar “If Condition” activity**
3. **Configurar:**
   * **Expression:**

@activity('Get Metadata').output.exists

* + **True activities:** Copy Activity
  + **False activities:** Web Activity (chamar API de alerta)

## 4.12 Integração com Databricks

ADF pode orquestrar notebooks Databricks (transformações Spark). Não vamos entrar em detalhes nesse momento, mas como poderoso orquestrador que é, o ADF consegue orquestrar um notebook dentro de um workspace no Azure Databricks.

**Passos:**

1. **Criar Linked Service Databricks:**
   * Tipo: Azure Databricks
   * Workspace URL
   * Access token
2. **No pipeline, adicionar “Databricks Notebook” activity**
3. **Configurar:**
   * Linked service: LS\_Databricks
   * Notebook path: /Users/..../transform\_silver
   * Parameters: Passar parâmetros ao notebook

**Fluxo típico:**

ADF Pipeline:

1. Copy Activity → Ingest to bronze (Data Lake)

2. Databricks Notebook → Transform bronze → silver

3. Copy Activity → Load silver → SQL Database (DW)

## 4.13 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Explicar papel do ADF em arquiteturas de dados
* [ ] Diferenciar Linked Service, Dataset, Pipeline, Activity, Trigger
* [ ] Entender quando usar ADF vs Databricks vs Synapse

**Práticos:**

* [ ] Criar Azure Data Factory
* [ ] Criar Linked Services (SQL Database, Data Lake Gen2)
* [ ] Criar Datasets (SQL table, Parquet file)
* [ ] Criar Pipeline com Copy Activity
* [ ] Configurar Source e Sink
* [ ] Executar Debug de pipeline
* [ ] Monitorar execuções (Monitor page)
* [ ] Criar Schedule Trigger (agendamento)

**Avançado:**

* [ ] Parametrizar pipelines e datasets
* [ ] Usar ForEach para loops
* [ ] Usar If Condition para lógica condicional

## 4.14 Próximos Passos

No **Capítulo 5**, você vai:

Criar Data Flows (transformações visuais no ADF)

Transformar dados sem código (joins, filters, aggregates)

Implementar SCD Type 2 (Slowly Changing Dimensions)

Otimizar Data Flows (partitioning, caching)

**Preparação:**

* Mantenha Data Factory, SQL Database e Data Lake ativos
* Revise conceitos de transformações de dados (Cap 1)
* Tenha datasets de exemplo prontos

## 4.15 Recursos Complementares

**Documentação Oficial:**

* [Azure Data Factory Overview](https://learn.microsoft.com/azure/data-factory/introduction)
* [Copy Activity](https://learn.microsoft.com/azure/data-factory/copy-activity-overview)
* [Triggers](https://learn.microsoft.com/azure/data-factory/concepts-pipeline-execution-triggers)
* [Parameterization](https://learn.microsoft.com/azure/data-factory/parameterize-linked-services)

**Tutoriais:**

* [ADF Quickstart](https://learn.microsoft.com/azure/data-factory/quickstart-create-data-factory)
* [Copy data from SQL to Blob](https://learn.microsoft.com/azure/data-factory/tutorial-copy-data-portal)

**Pricing:**

* [ADF Pricing Calculator](https://azure.microsoft.com/pricing/details/data-factory/)

## 4.16 Exercícios Práticos

**Exercício 1: Copiar Múltiplas Tabelas**

Crie pipeline que copia 3 tabelas do AdventureWorksLT:

* SalesLT.Product
* SalesLT.ProductCategory
* SalesLT.SalesOrderHeader

Para: datalake/bronze/<nome-tabela>/

Use parâmetros e ForEach!

**Exercício 2: Ingestão Incremental**

Modifique pipeline para copiar apenas dados novos (baseado em data):

* Use “Query” no Source:

**SQL**

SELECT \* FROM SalesLT.Customer

WHERE ModifiedDate > '@{pipeline().parameters.LastRunDate}'

* Passe LastRunDate como parâmetro

**Exercício 3: Monitoramento e Alertas**

1. Configure alerta no Azure Portal para falhas de pipeline
2. Force uma falha (deletar Linked Service temporariamente)
3. Verifique se recebeu alerta

**Exercício 4: Event-based Trigger**

1. Crie trigger que executa pipeline quando arquivo .csv chegar em bronze/
2. Teste: Faça upload manual de CSV
3. Verifique se pipeline executou automaticamente

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Azure Data Factory:** Orquestrador ETL/ELT (mover e transformar dados)

**Componentes:** Linked Services (conexões), Datasets (dados), Pipelines (workflows), Activities (ações), Triggers (agendamento)

**Copy Activity:** Copiar dados entre 100+ fontes/destinos

**Hands-on completo:**

* Criar Data Factory
* Conectar SQL Database e Data Lake (Linked Services)
* Criar datasets (SQL table, Parquet file)
* Pipeline: SQL → Data Lake (formato Parquet)
* Debug e monitoramento
* Schedule Trigger (diário às 6h)
* Parametrização (pipelines reutilizáveis)

**Orquestração:** ForEach (loops), If Condition (lógica)

**Integração:** Databricks Notebook activity (transformações Spark)

**Próximo capítulo:** Data Flows - transformações visuais sem código!

**Parabéns! Você completou a Parte 1: Fundamentos!**

**Você agora sabe:**

* Estruturar Data Lake (Medallion)
* Modelar Data Warehouse (Star Schema)
* Orquestrar pipelines ETL (Azure Data Factory)

**Próxima parte:** Transformação de Dados (Caps 5-9) com Data Flows, Databricks e PySpark!

# Capítulo 5: Data Flows no Azure Data Factory

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender a diferença entre Copy Activity e Data Flow
* Criar e configurar Data Flows no Azure Data Factory
* Aplicar transformações sem código (visual Spark)
* Realizar operações: Select, Filter, Join, Aggregate, Derived Column
* Implementar Lookup e Conditional Split
* Otimizar Data Flows (partitioning, caching, cluster sizing)
* Debugar transformações com Data Preview
* Implementar SCD Type 1 (Slowly Changing Dimension)
* Monitorar performance e custos

**Tempo estimado:** 4-5 horas

**Custo Azure:** R$ 10-30 (execuções de Data Flow usam Spark clusters)

## 5.1 O que são Data Flows?

**Data Flows** são transformações visuais de dados no Azure Data Factory, executadas em clusters Apache Spark gerenciados pela Microsoft.

### Copy Activity vs Data Flow

|  |  |  |
| --- | --- | --- |
| **Característica** | **Copy Activity** | **Data Flow** |
| **Propósito** | Copiar dados (mover) | Transformar dados |
| **Código** | Sem código | Sem código (visual) |
| **Engine** | ADF nativo | Apache Spark |
| **Performance** | Otimizado para I/O | Otimizado para processamento |
| **Custo** | Baixo (DIUs) | Médio (Spark cluster) |
| **Quando usar** | Ingestão simples | Transformações complexas |
| **Exemplos** | SQL → Data Lake | Joins, agregações, limpeza |

### Quando usar Data Flow?

**Use Data Flow quando:**

* Precisar de transformações complexas (joins, agregações, pivots)
* Limpar/enriquecer dados antes de gravar
* Implementar SCD (Slowly Changing Dimensions)
* Processar grandes volumes (paralelização Spark)
* Evitar código (transformações visuais)

**Use Copy Activity quando:**

* Apenas copiar dados sem transformação
* Minimizar custos (Copy é mais barato)
* Dados já estão no formato correto

### Arquitetura de Data Flow

[Source Dataset]

↓

[Data Flow]

Select (colunas)

↓

Filter (linhas)

↓

Derived Column

↓

Aggregate

↓

Join

↓

[Sink Dataset]

**Sob o capô:**

* ADF traduz transformações visuais → código Spark (PySpark)
* Executa em cluster Spark efêmero (criado on-demand)
* Paralelização automática
* Otimizações: predicate pushdown, column pruning

## 5.2 Hands-On: Criar Primeiro Data Flow

Vamos criar um Data Flow para transformar dados de clientes do SQL Database.

### Cenário

**Objetivo:** Ler tabela SalesLT.Customer do SQL Database, aplicar transformações (filtrar, calcular idade, selecionar colunas), gravar em Data Lake (Parquet).

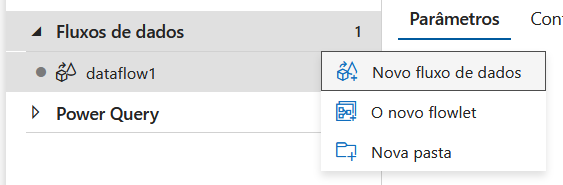
**Transformações:**

1. **Filter:** Apenas clientes do Brasil (CompanyName contém “Brazil”)
2. **Derived Column:** Calcular categoria de cliente (VIP, Standard)
3. **Select:** Selecionar apenas colunas relevantes
4. **Sink:** Gravar em silver/clientes/ como Parquet

### Passo 1: Criar Data Flow

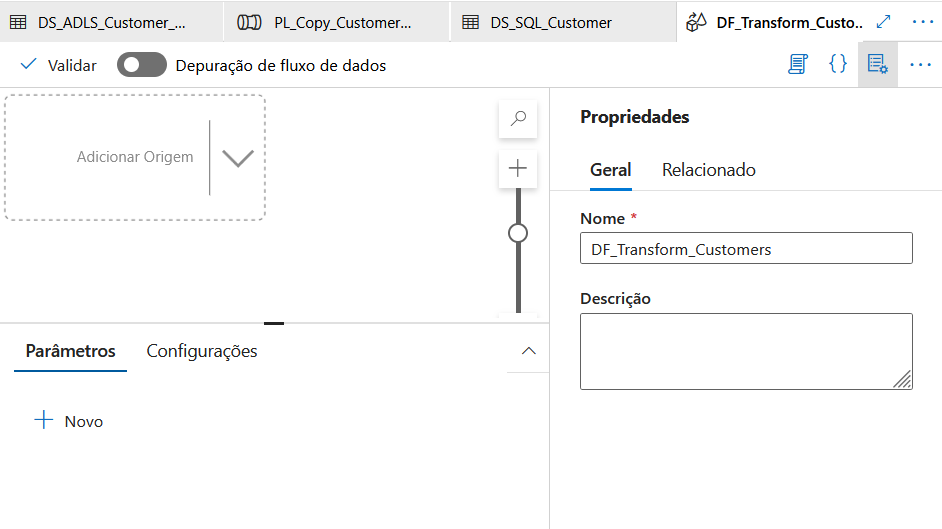
**No ADF Studio:**

1. **Clique em “Author”** (lápis)
2. **No painel Factory Resources, clique no “+” → “Data Flow”**
3. **Selecione “Mapping Data Flow”**



*Figura 5.1 - Criar novo Data Flow*

1. **Canvas de Data Flow abre:**
   * Área central em branco
   * Botão “Add Source” no centro
2. **Nome do Data Flow (topo):** DF\_Transform\_Customers

**

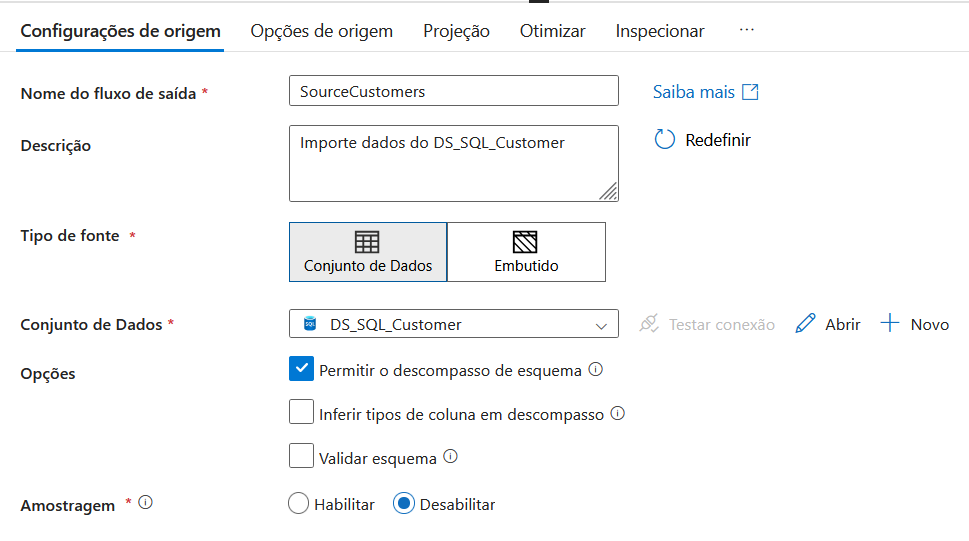
*Figura 5.2 - Canvas do Data Flow (vazio)*

### Passo 2: Configurar Source

1. **Clique em “Add Source”**
2. **Configurar Source:**

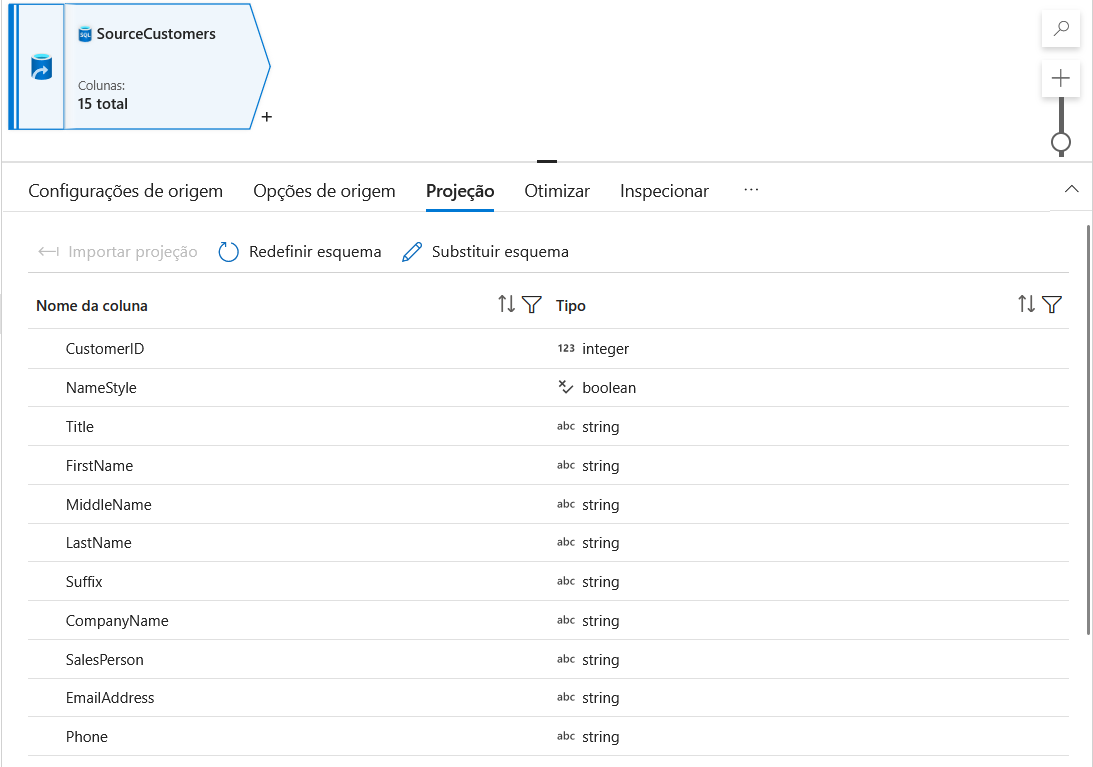
**Aba “Source settings”:**

* + **Output stream name:** SourceCustomers
    - Nome do “nó” no fluxo
  + **Source type:** Dataset
  + **Dataset:** Selecione DS\_SQL\_Customer (criado no Cap 4)
    - Se não existe, crie dataset apontando para SalesLT.Customer

**

*Figura 5.3 - Configuração do Source*

1. **Aba “Source options”:**
   * **Deixe padrão** (lê tabela inteira)
   * Alternativa: Query (SQL customizado)
2. **Aba “Projection”:**
   * Mostra schema (colunas e tipos)
   * **Clique em “Import schema”** se não aparecer automaticamente

**

*Figura 5.4 - Schema do Source (Projection)*

1. **Aba “Data preview”:**
   * **Clique em “Refresh”**
   * Preview dos dados reais (primeiras linhas)
   * **IMPORTANTE:** Data Flow precisa estar em modo Debug!

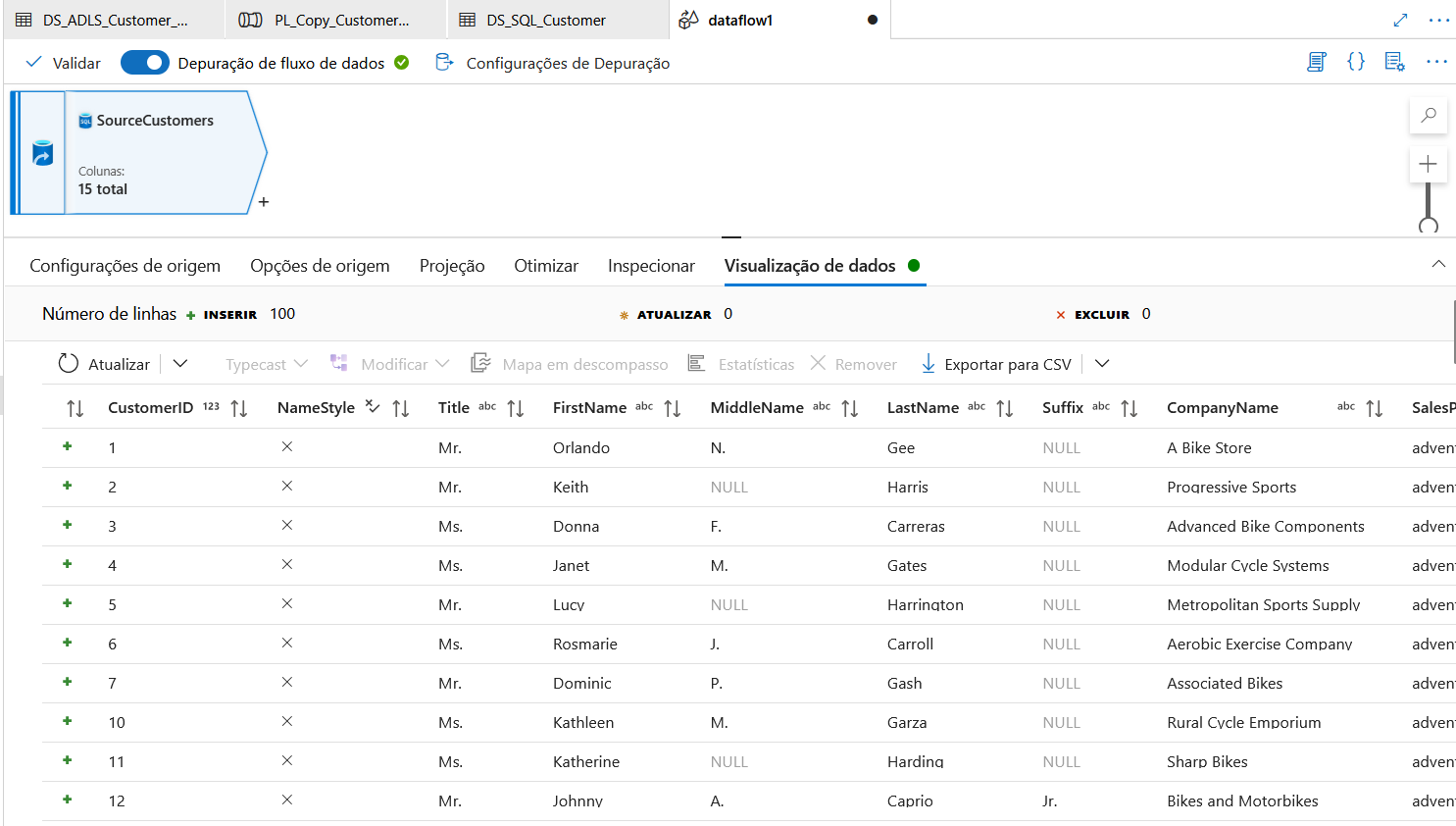
**Ativar Debug:**

* No topo do Data Flow, clique em **“Data flow debug: Off” → Ligar (On)**
* Aguarde ~2-3 minutos (provisiona cluster Spark).

Importante entender, aqui a pre-visualização vem através de um cluster spark, diferente do conjunto de dados que esta conectado com a tabela SQL diretamente como um wrapper odbc (para simplificar a história).

* Figura 5.5 - Ativar Data Flow Debug*

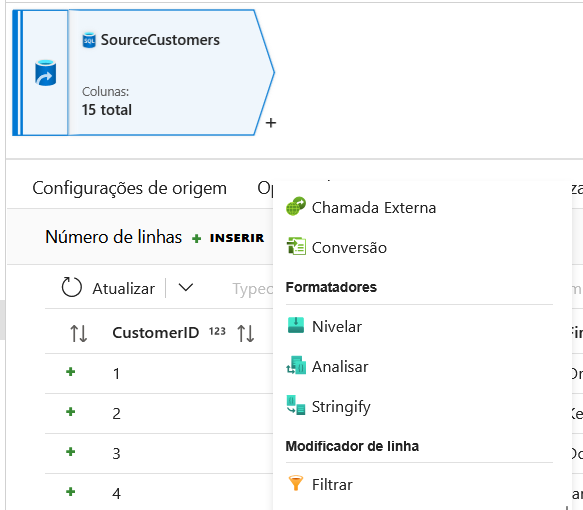
1. **Após Debug ativo, volte à aba “Data preview” e clique “Refresh”:**

* Figura 5.6 - Data Preview do Source*

### Passo 3: Adicionar Filter (Filtrar Linhas)

Vamos filtrar apenas clientes com CompanyName contendo “Bike” (exemplo).

1. **Clique no ícone “+” ao lado do nó SourceCustomers**
2. **No menu, selecione “Filter”** (categoria “Row modifier”)

**

*Figura 5.7 - Adicionar transformação Filter*

1. **Configurar Filter:**

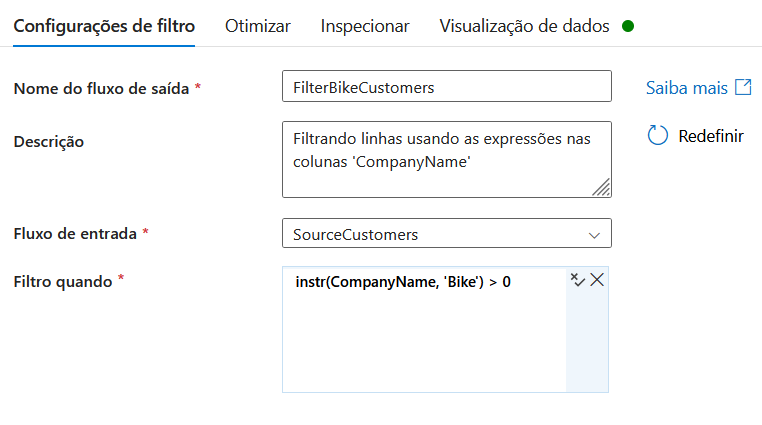
**Aba “Filter settings”:**

* + **Output stream name:** FilterBikeCustomers
  + **Filter on:** Expressão
  + **Expression:** Clique no campo e abra o Expression Builder

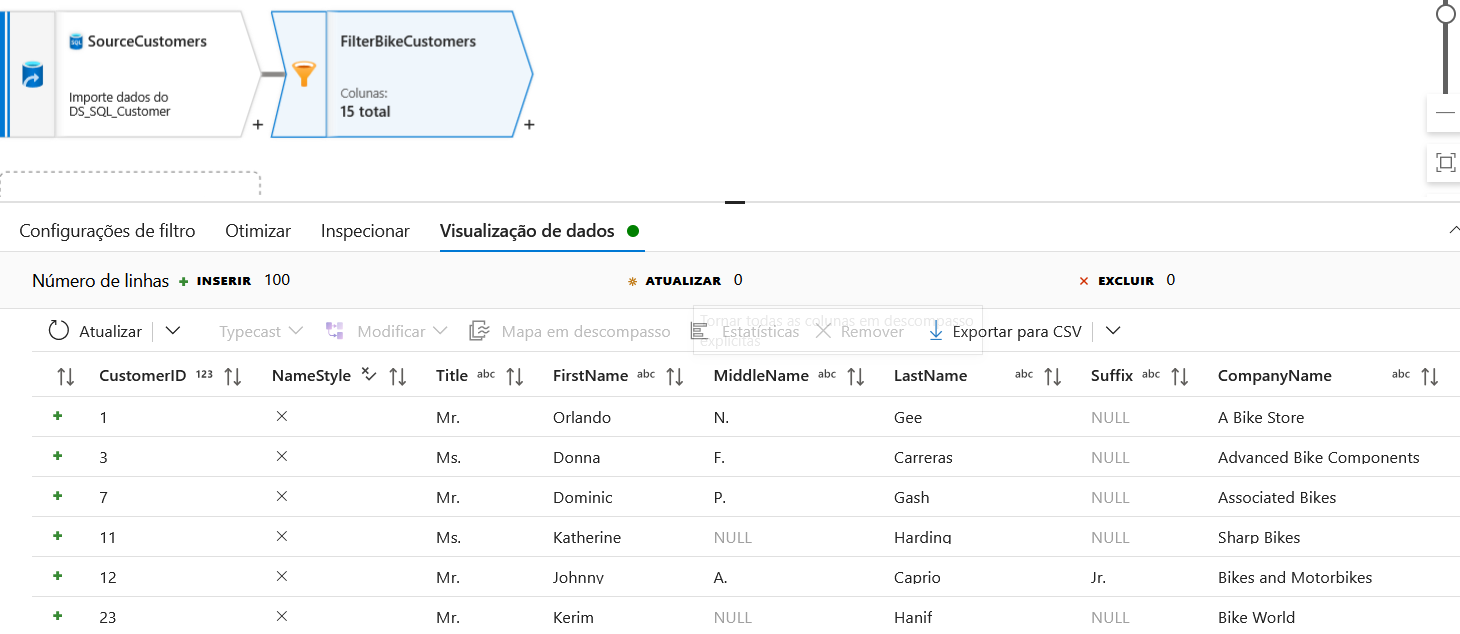
1. **No Expression Builder:**
   * Escreva:

instr(CompanyName, 'Bike') > 0

* + Significa: “CompanyName contém 'Bike'“. Ela retorna a posição do texto. Se for maior que zero, encontrou.
  + Clique “Save and finish”

*Figura 5.8 - Expressão de Filter*

1. **Aba “Data preview”:**
   * Clique “Refresh”
   * Verá apenas clientes filtrados (CompanyName contendo “Bike”)



*Figura 5.9 - Preview após Filter*

### Passo 4: Adicionar Derived Column (Coluna Calculada)

Vamos criar coluna CustomerCategory baseada em lógica:

* Se CompanyName contém “Bikes”: “VIP”
* Senão: “Standard”

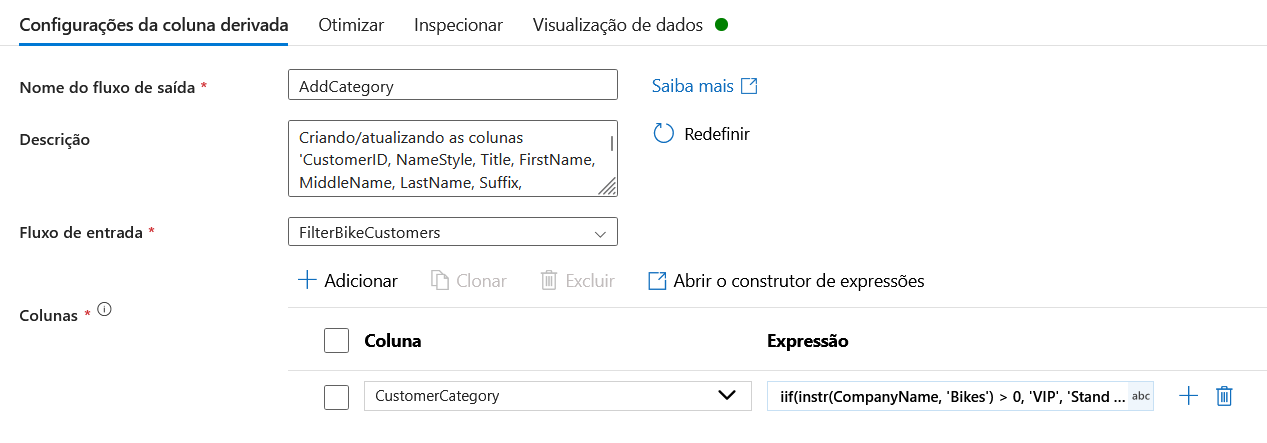
1. **Clique no “+” do nó FilterBikeCustomers**
2. **Selecione “Derived Column”** (categoria “Schema modifier”)
3. **Configurar Derived Column:**

**Aba “Derived column's settings”:**

* + **Output stream name:** AddCategory
  + **Columns:** Clique “+ Add”
  + **Column:** CustomerCategory (novo nome)
  + **Expression:** Abra Expression Builder

iif(instr(CompanyName, 'Bikes') > 0, 'VIP', 'Standard')

* + iif = if (ternário)

*Figura 5.10 - Derived Column: CustomerCategory*

1. **Aba “Data preview”:**
   * Refresh
   * Verá nova coluna CustomerCategory (VIP ou Standard)

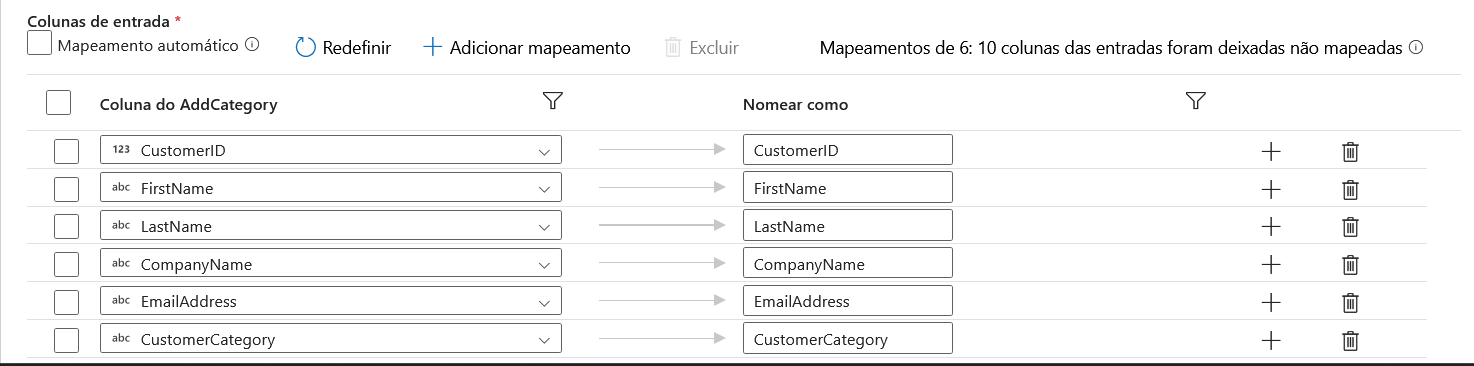
### Passo 5: Adicionar Select (Selecionar Colunas)

Vamos manter apenas colunas relevantes.

1. **Clique no “+” do nó AddCategory**
2. **Selecione “Select”** (categoria “Schema modifier”)
3. **Configurar Select:**

**Aba “Select settings”:**

* + **Output stream name:** SelectRelevantColumns
  + **Input columns:**
    - **Desmarque (apague)** colunas desnecessárias (ex: MiddleName, Suffix, PasswordHash, etc.)
    - **Mantenha:**
      1. CustomerID
      2. FirstName
      3. LastName
      4. CompanyName
      5. EmailAddress
      6. CustomerCategory (nova coluna)

* Figura 5.11 - Select: Selecionar colunas relevantes*

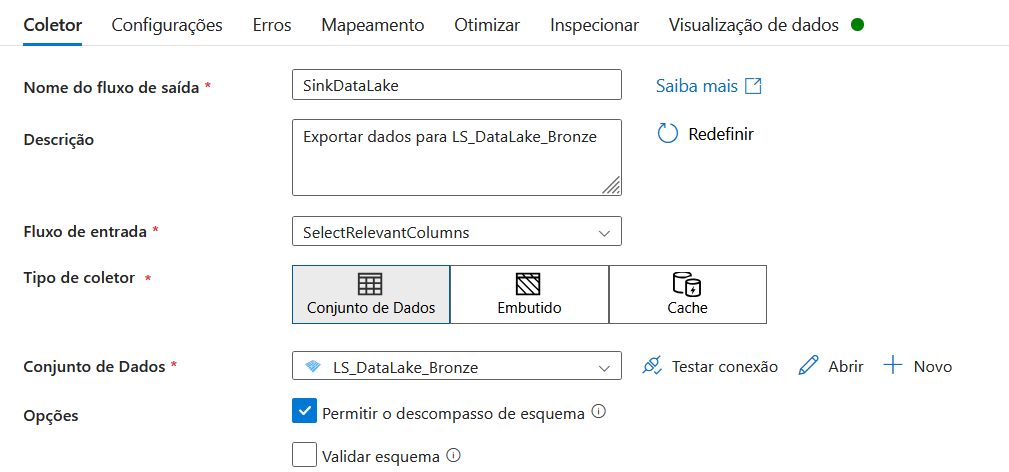
### Passo 6: Adicionar Sink (Destino)

Vamos gravar resultado no Data Lake.

1. **Clique no “+” do nó SelectRelevantColumns**
2. **Selecione “Sink”** (categoria “Destination”)
3. **Configurar Sink:**

**Aba “Sink”:**

* + **Output stream name:** SinkDataLake
  + **Sink type:** Dataset
  + **Dataset:** Crie novo ou selecione existente
    - **Novo dataset:**
      1. Tipo: Azure Data Lake Storage Gen2
      2. Formato: Parquet
      3. Linked service: LS\_DataLake\_Bronze
      4. File path:
         1. Container: datalake
         2. Directory: silver/clientes/
         3. File name: (deixe vazio ou customers\_transformed.parquet)

* Figura 5.12 - Sink: Gravar no Data Lake*

1. **Aba “Settings”:**
   * **Clear the folder:** Marque (sobrescrever arquivos antigos)
   * **File name option:** Output to single file (ou partition)
2. **Aba “Mapping”:**
   * Auto-map (ADF mapeia automaticamente)

### Passo 7: Validar e Salvar

1. **Clique em “Validate”** (topo do Data Flow)
2. **Verifica erros**
3. **Se “Validation successful” → tudo certo!**
4. **Clique em “Publish all”**

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap05\_13\_data\_flow\_complete.png

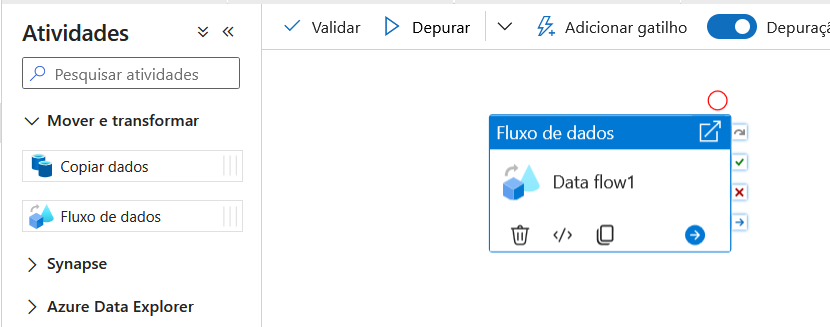
*Figura 5.13 - Data Flow completo*

## 5.3 Executar Data Flow em Pipeline

Data Flows não executam sozinhos — precisam ser chamados por um Pipeline.

### Criar Pipeline com Data Flow Activity

1. **Vá para “Author” → “Pipelines”**
2. **Crie novo pipeline:** PL\_Execute\_Transform\_Customers
3. **No painel Activities, expanda “Move & transform”**
4. **Arraste “Data Flow”** para o canvas

**

*Figura 5.14 - Adicionar Data Flow Activity ao pipeline*

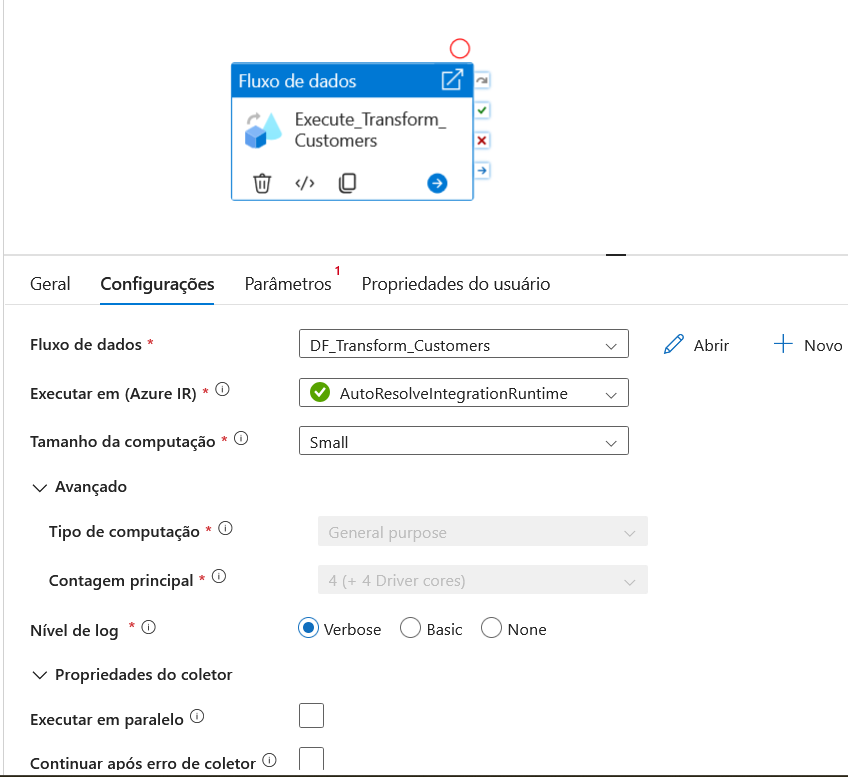
1. **Configurar Data Flow Activity:**

**Aba “General”:**

* + **Name:** Execute\_Transform\_Customers

**Aba “Settings”:**

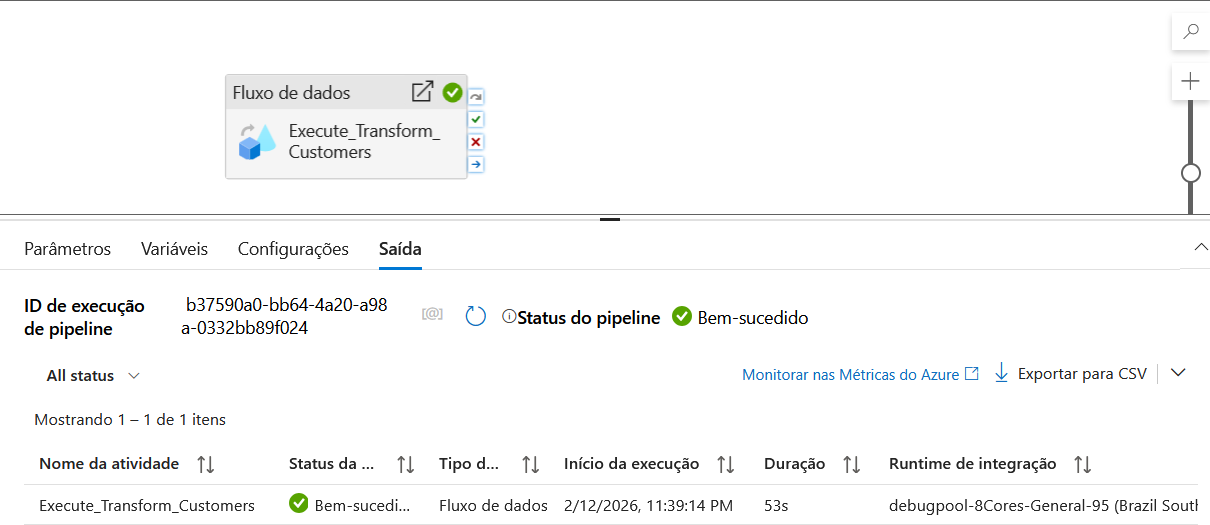
* + **Data flow:** Selecione DF\_Transform\_Customers
  + **Compute type:** General purpose (padrão)
  + **Core count:** 4 (+ 4 = Small cluster)
    - Small: 4 cores
    - Medium: 8 cores
    - Large: 16 cores
  + **Time to live (TTL):** 5 minutos
    - Cluster permanece ativo por 5 min após execução (evita cold start)

**

*Figura 5.15 - Configuração da Data Flow Activity*

### Executar Pipeline

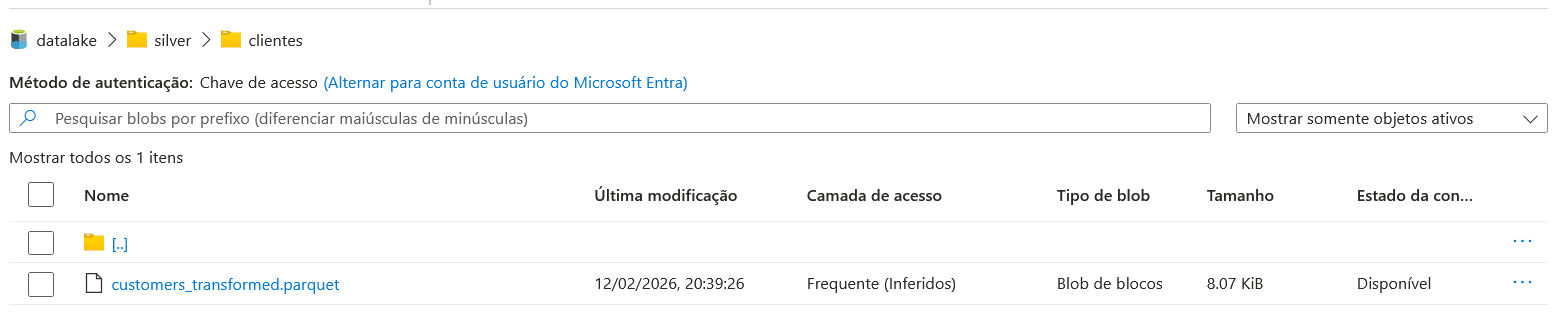
1. **Clique em “Debug”**
2. **Aguarde execução** (3-5 minutos na primeira vez, cold start do Spark)
3. **Status “Succeeded”!**

**

*Figura 5.16 - Execução do pipeline com Data Flow (sucesso)*

### Verificar Dados no Data Lake

1. **Navegue até:** datalake/silver/clientes/
2. **Você verá:** customers\_transformed.parquet (ou arquivos particionados)
3. **Tamanho:** Menor que original (apenas colunas selecionadas + filtradas)

* Figura 5.17 – Dados transformados gravados no Data Lake*

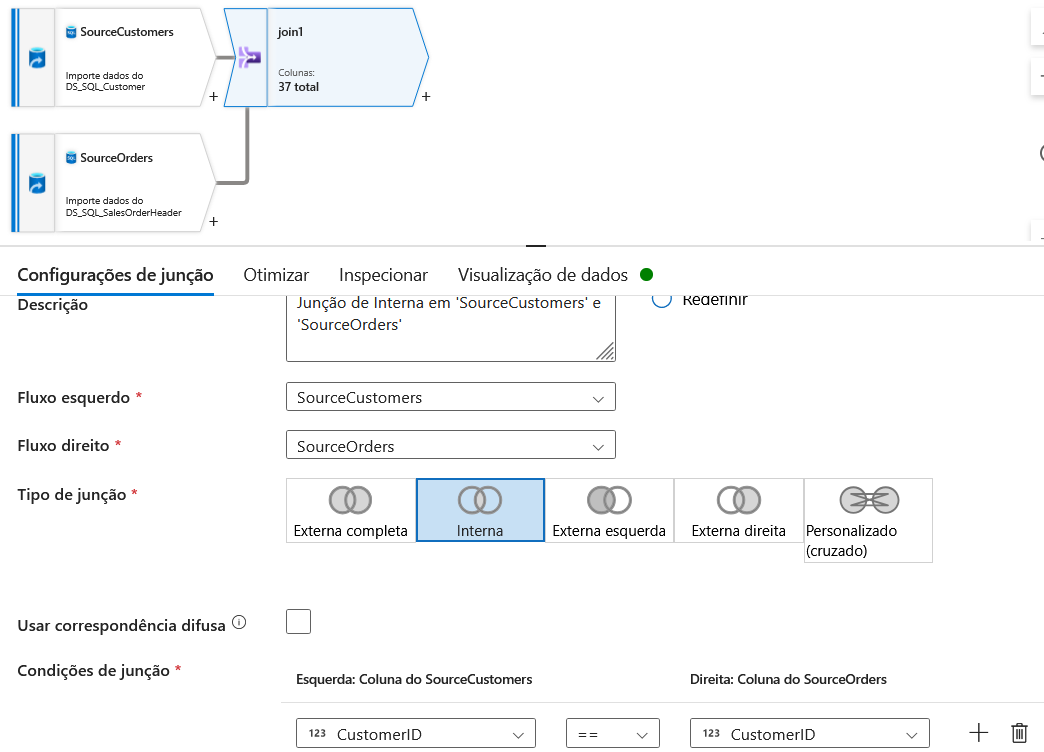
## 5.4 Transformações Avançadas

### Join (Combinar Datasets)

**Exemplo:** Juntar SalesLT.Customer com SalesLT.SalesOrderHeader (clientes + pedidos).

**Passos:**

1. **Crie um novo Dataflow:** DF\_Transfor\_Advanced
2. **Adicionar primeiro Source**: SourceCustomers (Customer)
3. **Adicionar segundo Source:** SourceOrders (SalesOrderHeader)
4. **Adicionar “Join” transformation:**
   * **Left stream:** SourceCustomers
   * **Right stream:** SourceOrders
   * **Join type:** Inner join
   * **Join conditions:**
     + Left: CustomerID
     + Right: CustomerID

**

*Figura 5.18 - Join: Combinar clientes e pedidos*

1. **Resultado:** Dataset combinado com colunas de ambas tabelas

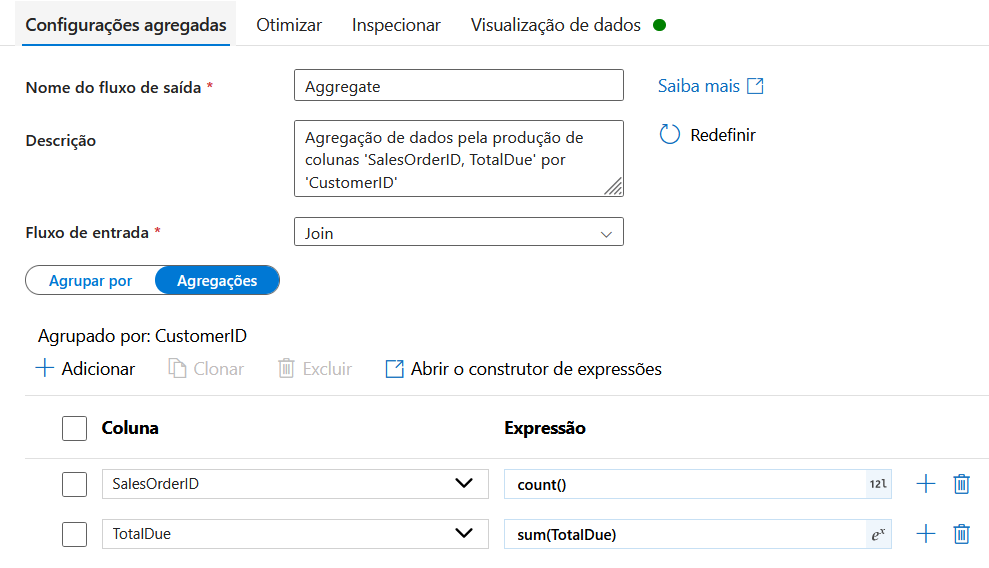
**Tipos de Join:**

* **Inner:** Apenas linhas que existem em ambos lados
* **Left outer:** Todas linhas da esquerda + matches da direita
* **Right outer:** Todas linhas da direita + matches da esquerda
* **Full outer:** Todas linhas de ambos lados
* **Cross:** Produto cartesiano (evitar!)

### Aggregate (Agrupar e Sumarizar)

**Exemplo:** Contar total de pedidos por cliente.

1. **Adicionar “Aggregate” transformation:**
2. **Configurar:**
   * **Group by:** CustomerID
   * **Aggregates:**
     + **Column:** SalesOrderID
     + **Expression:** count()
     + **Column:** TotalDue
     + **Expression:** sum(TotalDue)

*Figura 5.19 - Aggregate: Sumarizar pedidos por cliente*

**Funções de agregação:**

* count() - Contar linhas
* sum(coluna) - Somar
* avg(coluna) - Média
* min(coluna) - Mínimo
* max(coluna) - Máximo
* first(coluna) - Primeiro valor
* last(coluna) - Último valor

Após a agregação adicione um coletor. Você já consegue criar tudo o que precisa para isso. Recordar:

* Adicionar coletor SalesCustomerAgregate
* Criar um dataset para o diretório na camada silver (vendasclientesagregado) DS\_Lake\_SalesCustomerAgregate
* Criar um pipeline, adicionar uma atividade de Dataflow PL\_Execute\_Transform\_Advanced\_Agregate, configurar o dataflow criado e executar.

### Lookup (Enriquecer Dados)

Os próximos exemplos de operação, faça como exercício. Explore as opções.

**Exemplo:** Adicionar nome do cliente em tabela de pedidos (sem Join completo).

**Diferença Lookup vs Join:**

* **Join:** Combina todas as colunas de ambos datasets
* **Lookup:** Busca valores específicos de outra tabela (mais leve)

1. **Adicionar “Lookup” transformation:**
2. **Configurar:**
   * **Primary stream:** SourceOrders
   * **Lookup stream:** SourceCustomers
   * **Lookup conditions:**
     + Primary: CustomerID
     + Lookup: CustomerID
   * **Multiple rows:** First (primeira match)
   * **Columns to add:** FirstName, LastName

**Resultado:** Orders com nome do cliente anexado.

### Conditional Split (Dividir Fluxo)

**Exemplo:** Separar clientes VIP de Standard.

1. **Adicionar “Conditional Split”:**
2. **Configurar:**
   * **Stream 1:** VIPCustomers
     + Condition: CustomerCategory == 'VIP'
   * **Stream 2:** StandardCustomers
     + Condition: CustomerCategory == 'Standard'
   * **Default stream:** (se nenhuma condição atender)

*[INSERIR IMAGEM]*

**Nome do arquivo:** Junior\_Cap05\_20\_conditional\_split.png

*Figura 5.20 - Conditional Split: Separar VIP e Standard*

1. **Adicionar Sinks separados:**
   * VIPCustomers → silver/clientes\_vip/
   * StandardCustomers → silver/clientes\_standard/

### Window (Funções de Janela)

**Exemplo:** Ranquear clientes por total de pedidos.

1. **Adicionar “Window” transformation:**
2. **Configurar:**
   * **Over:** CustomerID (particionar por cliente? Não, queremos ranking global)
     + Deixe vazio (ranking global)
   * **Sort:** TotalOrders (descendente)
   * **Window columns:**
     + **Column:** Rank
     + **Expression:** rank()

**Resultado:** Coluna Rank com posição do cliente (1 = mais pedidos).

**Funções de Window:**

* rank() - Ranking (pode ter empates)
* denseRank() - Ranking denso (sem gaps em empates)
* rowNumber() - Número sequencial (sem empates)
* lag(coluna, 1) - Valor da linha anterior
* lead(coluna, 1) - Valor da próxima linha

## 5.5 Otimização de Data Flows

Data Flows executam em Spark, então otimizações Spark se aplicam. As otimizações abaixo você ira encontrar na atividade dos Fonte (Source) e Coletores (Sink). É nesse momento que a mágica acontece, estamos falando da técnicas de leitura e escrita.

### Partitioning (Paralelização)

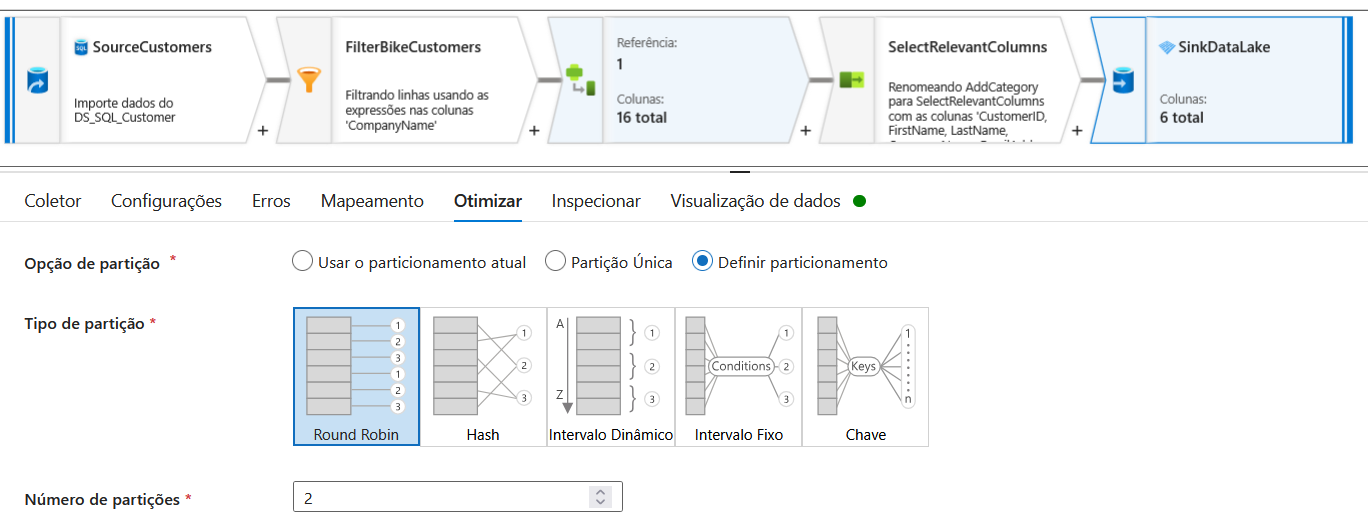
**Particionamento** controla como dados são distribuídos entre workers do Spark.

**Estratégias:**

1. **Default (Auto):**
   * ADF decide automaticamente
   * Geralmente bom para começar
2. **Round robin:**
   * Distribuição uniforme (balanceamento de carga)
   * Use para operações sem chave (ex: Filter, Select)
3. **Hash partitioning:**
   * Particiona por coluna (ex: CustomerID)
   * Use antes de Join ou Aggregate (co-location)
   * Evita shuffle desnecessário
4. **Range partitioning:**
   * Particiona por faixa de valores (ex: data)
   * Use para dados ordenados
5. **Fixed partitioning:**
   * Número fixo de partições
   * Use quando souber número ideal

**Configurar Partitioning:**

* Clique na transformação
* Aba “Optimize”
* Selecione estratégia

*Figura 5.21 - Configuração de Partitioning*

### Broadcast (Otimização de Join)

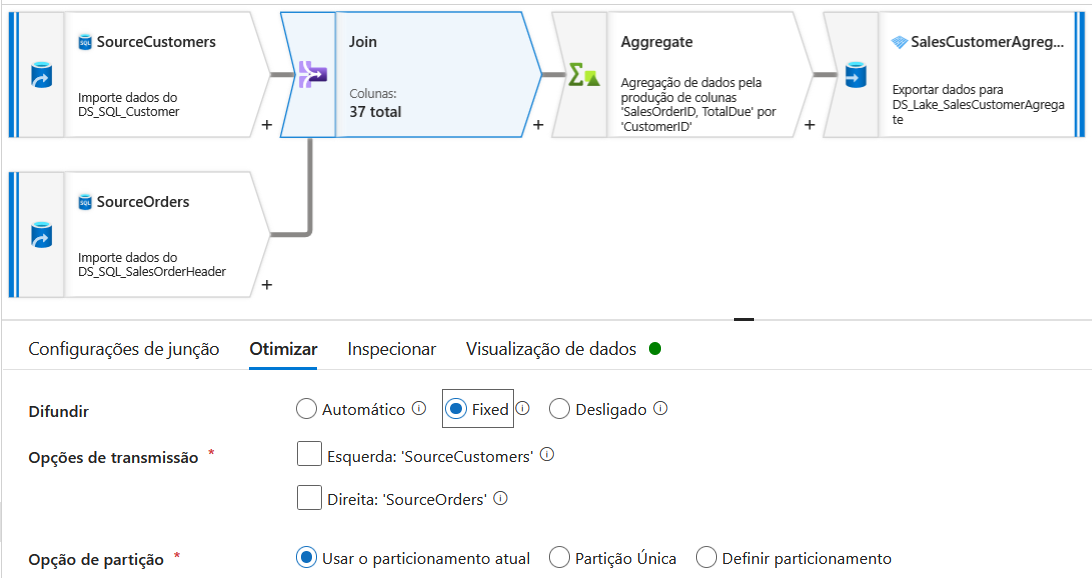
Quando uma tabela é pequena (ex: dimensões), **broadcast** a distribui para todos workers (evita shuffle).

**Configurar Broadcast:**

* No Join transformation
* Aba “Optimize”
* Selecione “Broadcast” no lado pequeno (ex: Right)

**Quando usar:**

* Tabela direita < 100 MB
* Join com dimensões (pequenas)



*Figura 5.22 - Configuração de Partitioning*

### Caching (Reutilizar Datasets)

Se um dataset é usado múltiplas vezes, **cache** evita recálculo.

**Configurar Cache:**

* Clique na transformação (ex: source)
* Aba “Optimize”
* Marque “Enable cache”

**Quando usar:**

* Dataset usado em múltiplos branches (Conditional Split)
* Joins múltiplos
* Debug (evita recarregar source)

### Cluster Sizing (Dimensionar Cluster)

Essa opção fica disponível, quando você adicionar uma atividade de fluxo de dados em um pipeline. Ao escolher um tamanho de computação “Custom”. Muito cuidado aqui. Quanto maior seu cluster, maior a fatura. E esse aumento é exponencial. 90% dos trabalhos vão performar perfeitamente com o tamanho Small.

**Compute type:**

* **General purpose:** Balanceado (padrão)
* **Compute optimized:** CPU intensivo (transformações complexas)
* **Memory optimized:** Grandes datasets em memória

**Core count:**

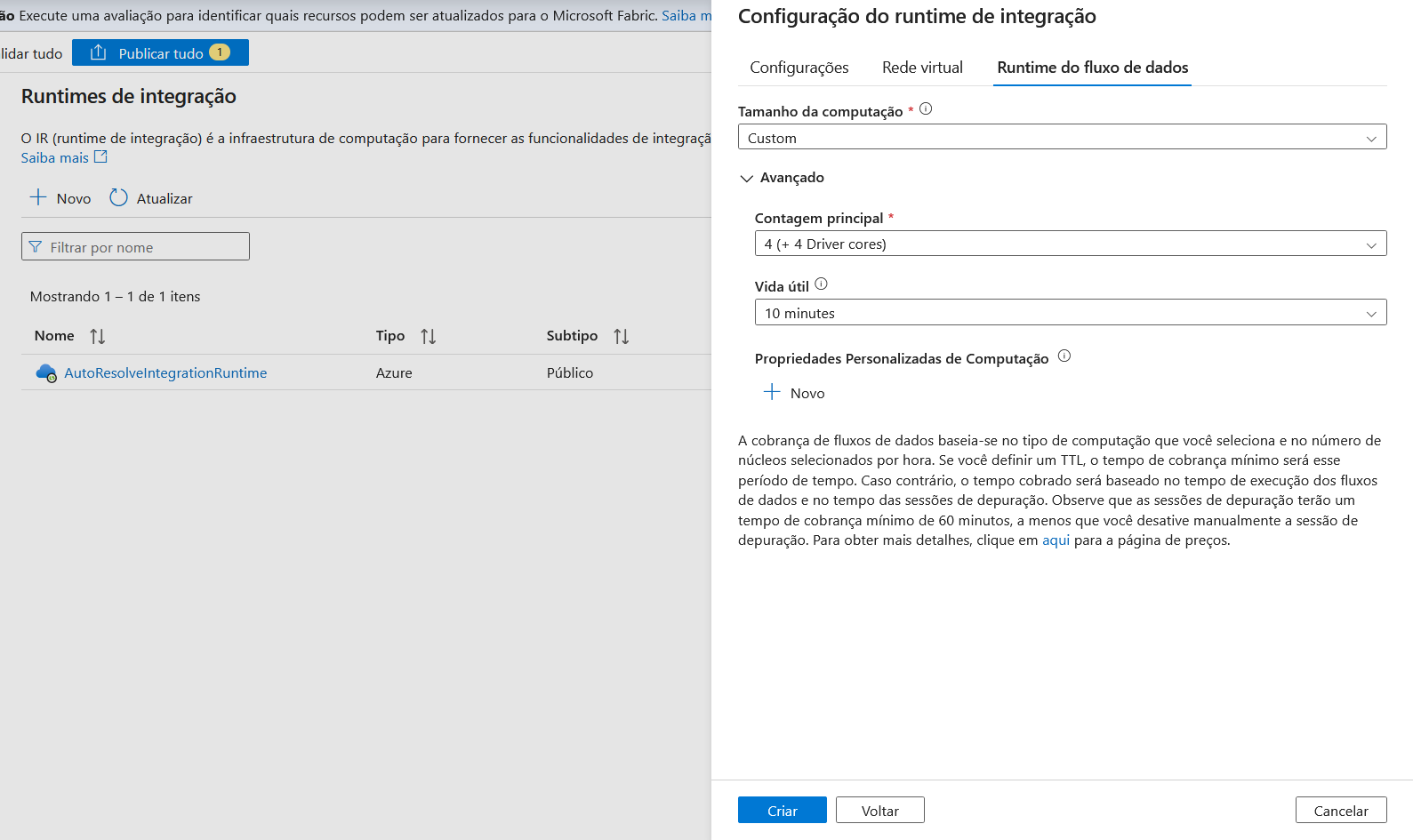
* Small: 4 cores (8 vCores total, incluindo driver)
* Medium: 8 cores (16 vCores)
* Large: 16 cores (32 vCores)
* Extra Large: 32 cores (64 vCores)

**Regra geral:**

* Start small (4 cores)
* Aumente se execução > 10 min ou falhar (Out of Memory)

### Time to Live (TTL)

**TTL:** Tempo que cluster Spark permanece ativo após execução. Essa configuração em especifico exige a criação de um novo runtime de Integração, no caso o autohospedado.



*Figura 5.23 - Configuração de Partitioning*

**Benefício:**

* Evita cold start (3-5 min de provisionamento)
* Próxima execução usa cluster já ativo

**Custo:**

* Cluster cobra por minuto ativo (mesmo sem executar)

**Recomendação:**

* Dev: TTL = 5 min (debug frequente)
* Prod: TTL = 0 ou 10 min (se pipelines consecutivos)

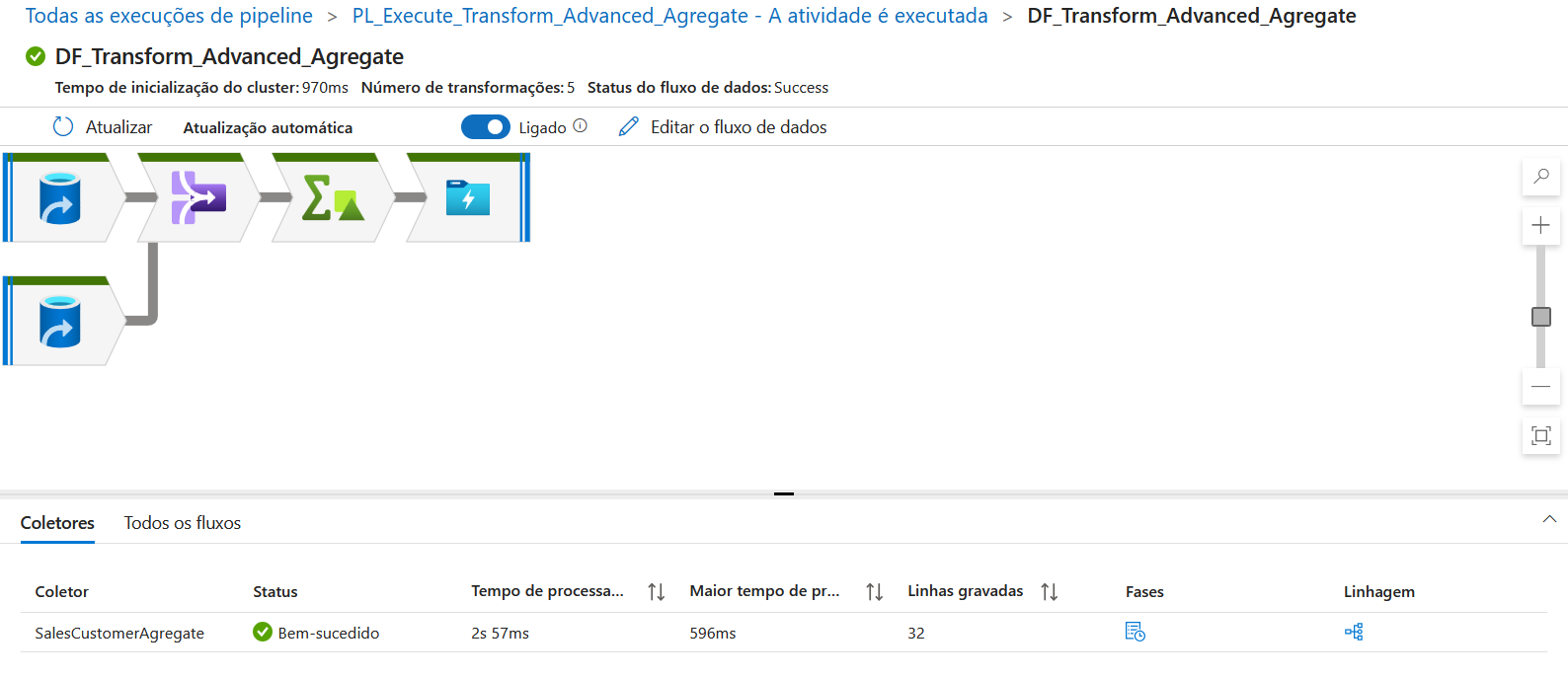
## 5.6 Monitoramento de Data Flows

### Pipeline Monitor

1. **ADF Studio → Monitor → Pipeline runs**
2. **Clique no pipeline com Data Flow**
3. **Clique no ícone de óculos () da Data Flow activity**

**Você verá:**

* **Duração total:** Tempo de execução (incluindo provisionamento)
* **Cluster startup time:** Tempo de cold start
* **Data flow execution time:** Tempo real de processamento
* **Rows processed:** Total de linhas processadas
* **Stage details:** Detalhes de cada transformação (Spark stages)

*Figura 5.22 - Monitoramento de Data Flow*

### Performance Tuning

**Métricas importantes:**

* **Cluster startup time > 50% duração total:** Considere aumentar TTL
* **Rows processed muito alto:** Adicione Filter mais cedo no fluxo
* **Join/Aggregate lentos:** Use Hash partitioning na coluna de join/group

**Ferramentas:**

* **Data Flow Debug:** Preview de dados em cada transformação
* **Pipeline Monitor:** Métricas de execução
* **Spark UI:** (avançado) acessível via link no monitor

## 5.7 Implementar SCD Type 1

**SCD (Slowly Changing Dimension):** Como lidar com mudanças em dimensões.

**Type 1:** Sobrescrever valor antigo (sem histórico).

**Exemplo:** Cliente muda de email.

### Fluxo SCD Type 1

1. **Source:** Carregar novos dados (ex: new\_customers.csv)
2. **Lookup:** Verificar se CustomerID já existe na tabela destino
3. **Alter Row:** Marcar linhas como INSERT ou UPDATE
   * Se existe: UPDATE
   * Se não existe: INSERT
4. **Sink:** Gravar com “Upsert” (insert + update)

### Hands-On SCD Type 1

**Passos:**

1. **Criar Data Flow:** DF\_SCD1\_Customers
2. **Source 1:** SourceNewCustomers (novos dados)
3. **Source 2:** SourceExistingCustomers (tabela destino atual)
4. **Lookup:**
   * Primary: SourceNewCustomers
   * Lookup: SourceExistingCustomers
   * Condition: CustomerID == CustomerID
5. **Derived Column:** IsExisting
   * Expression: !isNull(SourceExistingCustomers@CustomerID)
   * Se Lookup encontrou = true (existe), senão = false (novo)
6. **Alter Row:**
   * **Upsert if:** IsExisting == true()
   * **Insert if:** IsExisting == false()
7. **Sink:**
   * **Dataset:** Tabela SQL Database de clientes
   * **Aba “Settings”:**
     + **Update method:** Marque “Allow upsert”
     + **Key columns:** CustomerID

**Resultado:** Registros novos são inseridos, existentes são atualizados (sobrescritos).

## 5.8 Checklist de Habilidades

Ao final deste capítulo, você deve conseguir:

**Conceitos:**

* [ ] Diferenciar Copy Activity vs Data Flow
* [ ] Explicar quando usar cada um
* [ ] Entender arquitetura de Data Flow (Spark)

**Práticos:**

* [ ] Criar Data Flow no ADF
* [ ] Adicionar transformações: Filter, Select, Derived Column
* [ ] Realizar Join entre datasets
* [ ] Aplicar Aggregate (group by)
* [ ] Usar Lookup para enriquecimento
* [ ] Implementar Conditional Split
* [ ] Configurar Sink para Data Lake
* [ ] Executar Data Flow via Pipeline
* [ ] Debugar com Data Preview

**Avançado:**

* [ ] Otimizar partitioning (Hash, Broadcast)
* [ ] Implementar SCD Type 1 (Upsert)
* [ ] Monitorar performance e custos
* [ ] Ajustar cluster sizing (cores, TTL)

## 5.9 Próximos Passos

No **Capítulo 6**, você vai:

Conhecer Azure Databricks e notebooks

Escrever código PySpark para transformações

Integrar Databricks com ADF

Processar Big Data com Spark distribuído

**Preparação:**

* Mantenha Data Factory, Data Lake e SQL Database ativos
* Revise conceitos de Spark (opcional: tutorial básico)

## 5.10 Recursos Complementares

**Documentação Oficial:**

* [Data Flows Overview](https://learn.microsoft.com/azure/data-factory/concepts-data-flow-overview)
* [Transformations Reference](https://learn.microsoft.com/azure/data-factory/data-flow-transformation-overview)
* [Performance Optimization](https://learn.microsoft.com/azure/data-factory/concepts-data-flow-performance)

**Tutoriais:**

* [Create your first Data Flow](https://learn.microsoft.com/azure/data-factory/tutorial-data-flow)
* [Implement SCD Type 1](https://learn.microsoft.com/azure/data-factory/how-to-data-flow-scd)

**Pricing:**

* [Data Flow Pricing](https://azure.microsoft.com/pricing/details/data-factory/data-pipeline/)

## 5.11 Exercícios Práticos

**Exercício 1: Pipeline de Limpeza**

Crie Data Flow que:

1. Lê SalesLT.Product do SQL Database
2. Remove produtos descontinuados (DiscontinuedDate IS NOT NULL)
3. Calcula margem de lucro: (ListPrice - StandardCost) / ListPrice \* 100
4. Seleciona apenas: ProductID, Name, ListPrice, Margin
5. Grava em silver/produtos/

**Exercício 2: Join e Agregação**

Crie Data Flow que:

1. Join: SalesLT.SalesOrderHeader + SalesLT.SalesOrderDetail
2. Aggregate: Total de vendas por produto (group by ProductID)
3. Sort: Top 10 produtos por receita
4. Sink: gold/top\_produtos.parquet

**Exercício 3: SCD Type 1 Completo**

Implemente SCD Type 1 para tabela de clientes:

1. Source: CSV com novos/alterados clientes
2. Lookup: Verificar existentes no SQL Database
3. Alter Row: Upsert (atualizar) ou Insert (novo)
4. Sink: SQL Database (tabela DimCliente)

## Resumo do Capítulo

Neste capítulo você aprendeu:

**Data Flow:** Transformações visuais executadas em Spark (sem código)

**Copy vs Data Flow:**

* Copy: Mover dados (ingestão)
* Data Flow: Transformar dados (limpeza, agregação, joins)

**Transformações principais:**

* Filter (filtrar linhas)
* Select (selecionar colunas)
* Derived Column (colunas calculadas)
* Join (combinar datasets)
* Aggregate (group by, sumarizar)
* Lookup (enriquecer dados)
* Conditional Split (dividir fluxos)

**Otimizações:**

* Partitioning (Hash, Broadcast)
* Caching (reutilizar datasets)
* Cluster sizing (cores, compute type)
* TTL (evitar cold start)

**SCD Type 1:** Insert + Update (Upsert) para lidar com mudanças em dimensões

**Monitoramento:** Pipeline Monitor, Spark UI, métricas de performance

**Próximo capítulo:** Databricks e PySpark - transformações com código!

# Capítulo 6: Azure Databricks e PySpark - Fundamentos

## Objetivos do Capítulo

Ao final deste capítulo, você será capaz de:

* Compreender o que é Azure Databricks e quando usar
* Provisionar workspace e cluster Databricks
* Criar e executar notebooks interativos
* Escrever código PySpark para transformações
* Ler e gravar dados em Data Lake (Delta Lake, Parquet, CSV)
* Realizar transformações: filter, select, join, aggregate
* Usar Spark SQL (queries SQL em DataFrames)
* Integrar Databricks com Azure Data Factory
* Entender arquitetura Spark (Driver, Executors, Partitions)
* Monitorar jobs e otimizar performance básica

**Tempo estimado:** 5-6 horas

**Custo Azure:** R$ 50-150/mês (clusters on-demand, pagamento por minuto)

## 6.1 O que é Azure Databricks?

**Azure Databricks** é uma plataforma de analytics unificada baseada em Apache Spark, otimizada para Azure.

### Componentes Principais

**Apache Spark:**

* Engine de processamento distribuído (Big Data)
* In-memory computing (100x mais rápido que MapReduce)
* APIs: Python (PySpark), Scala, SQL, R, Java

**Databricks:**

* Camada gerenciada sobre Spark (da empresa criadora do Spark)
* Notebooks colaborativos (como Jupyter)
* Clusters gerenciados (auto-scaling, terminação automática)
* Otimizações proprietárias (Photon engine)
* Integração nativa com Azure (Data Lake, Key Vault, ADF, Synapse)

### Databricks vs Outras Ferramentas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Característica** | **Databricks** | **ADF Data Flow** | **Synapse Spark Pools** | **HDInsight** |
| **Engine** | Apache Spark | Spark (gerenciado ADF) | Apache Spark | Hadoop/Spark |
| **Interface** | Notebooks | Visual (low-code) | Notebooks | Scripts/SSH |
| **Colaboração** | Excelente | Não | Boa | Limitada |
| **ML/AI** | Sim (MLflow) | Não | Sim | Limitado |
| **Custo** | Médio-Alto | Médio | Médio | Baixo-Médio |
| **Quando usar** | Transformações complexas, ML, Big Data | ETL visual sem código | Data Warehousing + Spark | Hadoop legado |

### Quando usar Databricks?

**Use Databricks quando:**

* Processar Big Data (> 100 GB)
* Transformações complexas (múltiplos joins, agregações, window functions)
* Machine Learning (feature engineering, treinamento distribuído)
* Análise exploratória de dados (EDA)
* Equipe tem familiaridade com Python/SQL
* Precisa de colaboração em notebooks

**Evite Databricks se:**

* Dados pequenos (< 10 GB) → Use SQL Database ou ADF Data Flow
* ETL simples (copiar + filtros básicos) → Use ADF Copy Activity
* Budget limitado → Databricks cobra por DBU (Databricks Unit) + VM

## 6.2 Arquitetura do Apache Spark

Antes de provisionar Databricks, é essencial entender como Spark funciona.

### Conceitos Fundamentais

**1. Driver:**

* Processo principal que coordena o trabalho
* Executa código do notebook
* Divide jobs em tasks e distribui aos Executors

**2. Executors:**

* Processos workers que executam tasks
* Rodam em VMs separadas (nós do cluster)
* Armazenam dados em memória (cache)

**3. Cluster:**

* Conjunto de Driver + Executors
* Exemplo: 1 driver (DS13\_v2) + 2 executors (DS13\_v2) = cluster de 3 nós

**4. Partitions:**

* Dados são divididos em partições (chunks)
* Cada partition é processada por um executor em paralelo
* Default: 200 partitions (ajustável)

**5. DataFrames:**

* Estrutura de dados distribuída (como tabela SQL)
* Imutável (transformações criam novos DataFrames)
* Lazy evaluation (executa apenas quando necessário)

### Fluxo de Execução Spark

1. Driver recebe código do notebook

2. Driver analisa e cria DAG (Directed Acyclic Graph) de transformações

3. Driver divide DAG em Stages e Tasks

4. Tasks são enviados aos Executors

5. Executors processam partitions em paralelo

6. Resultados são agregados e retornados ao Driver

**Exemplo:**

**PYTHON**

# Código no notebook

df = spark.read.parquet(“path/to/data”) # Lazy (não executa ainda)

df = df.filter(df.age > 18) # Lazy

df = df.select(“name”, “age”) # Lazy

df.write.parquet(“path/to/output”) # Action (executa tudo!)

**Transformações (Lazy):**

* filter(), select(), join(), groupBy(), etc.
* Apenas definem o plano, não executam

**Actions (Executam):**

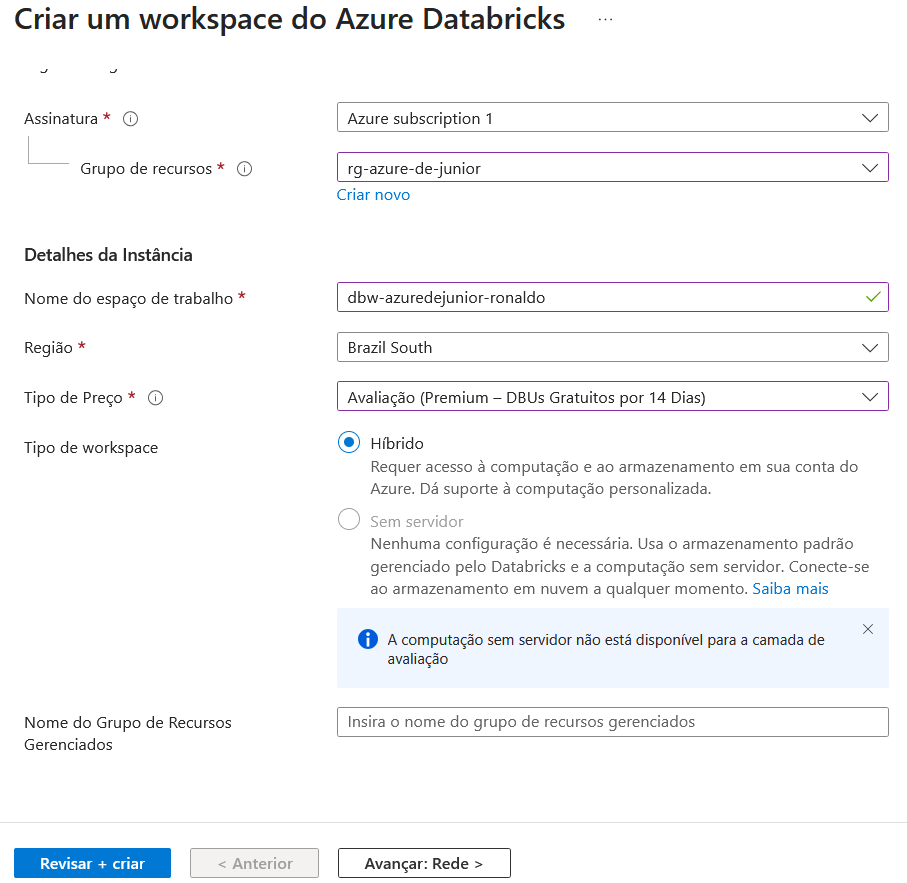
* write(), show(), count(), collect(), take()
* Disparam execução de todas as transformações acumuladas

## 6.3 Hands-On: Provisionar Azure Databricks

### Passo 1: Criar Workspace Databricks

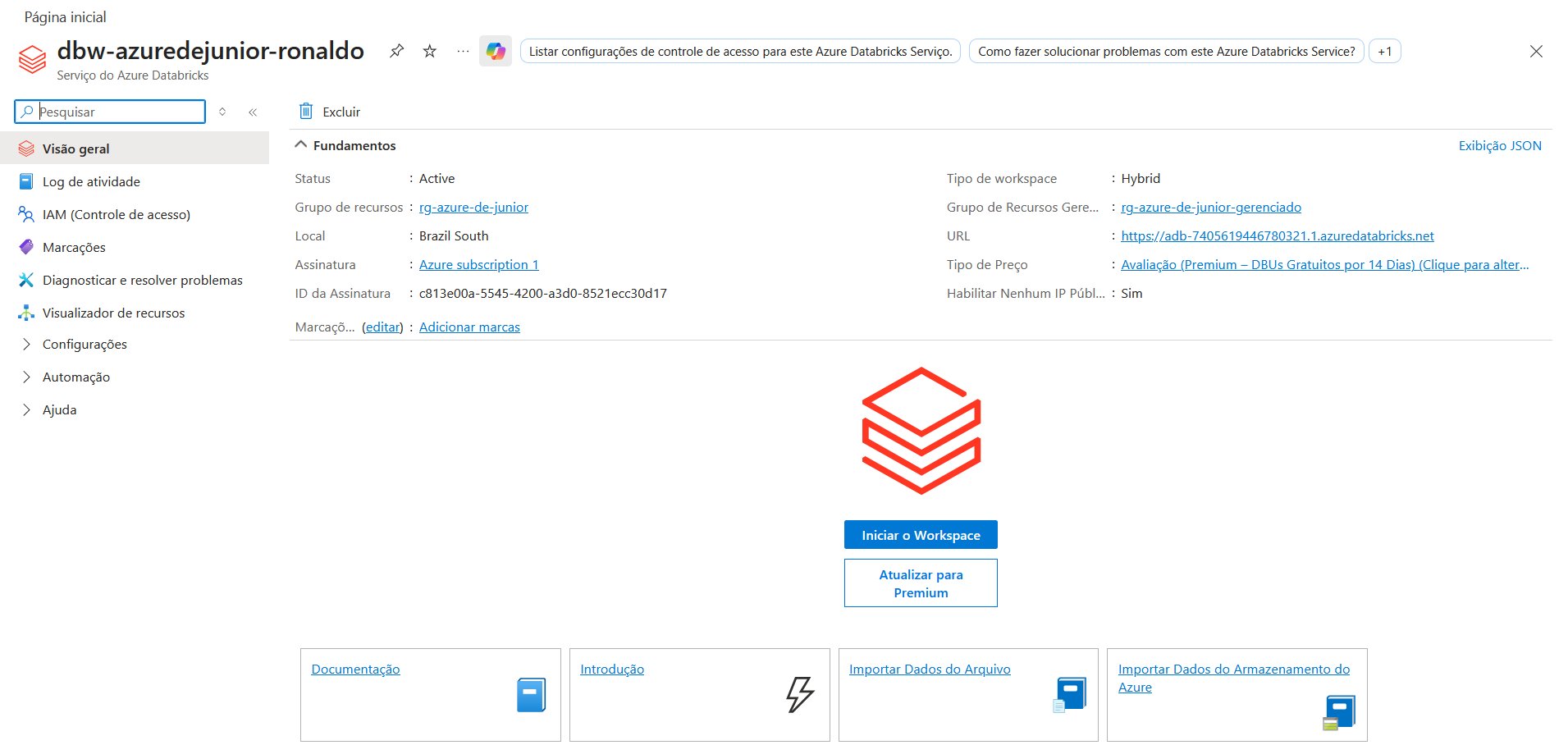
**No Azure Portal:**

1. **Barra de pesquisa:** databricks
2. **Clique em “Azure Databricks”**
3. **Clique em “+ Create”**
4. **Aba “Basics”:**
   * **Subscription:** Free Trial
   * **Resource group:** rg-azure-de-junior
   * **Workspace name:** dbw-azuredejunior-<seunome>
     + Exemplo: dbw-azuredejunior-ronaldo
     + Prefixo dbw- (Databricks Workspace)
   * **Region:** Brazil South
   * **Pricing tier:** Standard (recomendado para começar)
     + Standard: Recursos básicos
     + Premium: RBAC avançado, auditoria, ML extra
     + Trial: 14 dias grátis (depois vira Standard)

**

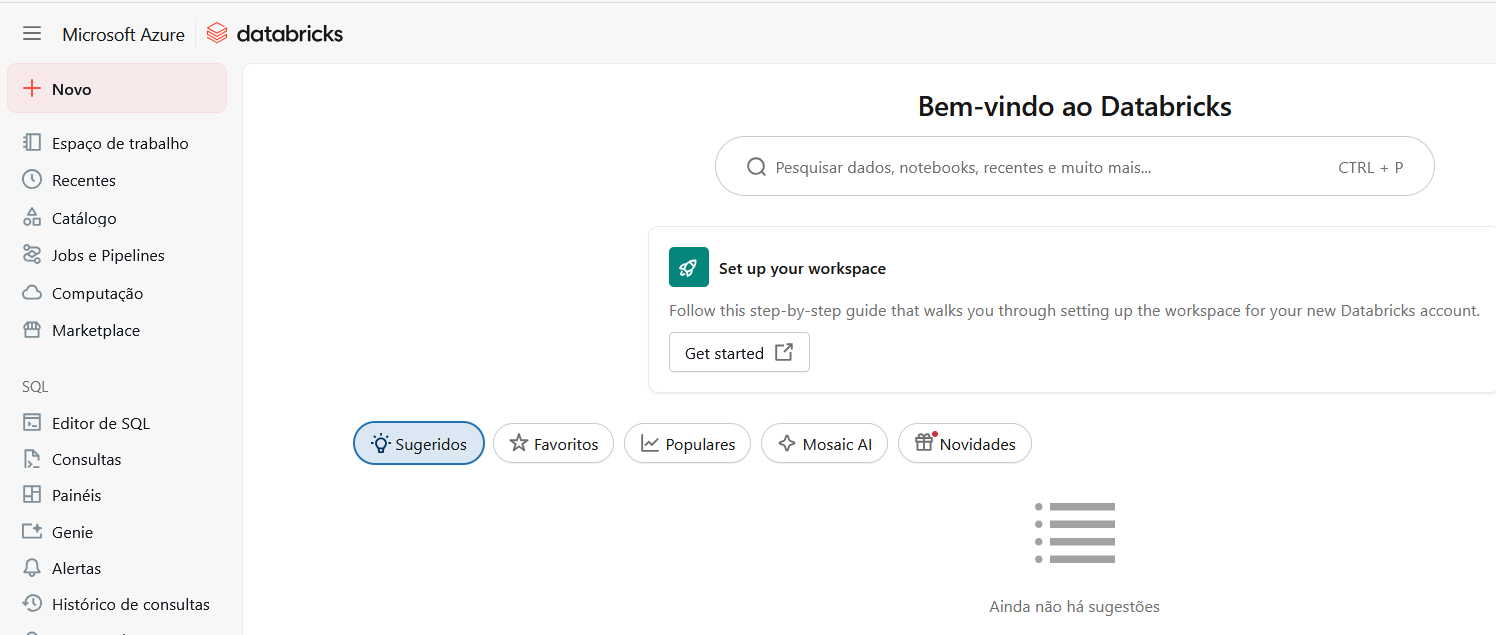
*Figura 6.2 - Configuração básica do Databricks Workspace*

1. **Aba “Networking”:**
   * **Deploy in Virtual Network:** Não (deixe desmarcado)
   * Para produção: Habilite (segurança extra)
2. **Aba “Advanced”:**
   * **Managed Resource Group:** Auto-gerado (OK)
   * Databricks cria resource group interno para recursos gerenciados
3. **Clique em “Review + create”**
4. **Clique em “Create”**
5. **Aguarde deployment** (7-10 minutos)

* Figura 6.3 - Databricks Workspace criado*

### Passo 2: Abrir Databricks Workspace

1. **Clique em “Launch Workspace”** (ou “Launch Portal”)
2. **Nova aba abre:** Databricks Workspace UI
3. **Login automático:** Azure AD SSO (Single Sign-On)

* Figura 6.4 - Databricks Workspace - Home*

**Principais seções:**

* **Workspace:** Notebooks, pastas, scripts
* **Repos:** Integração Git (GitHub, Azure DevOps)
* **Data:** Databases, tabelas, arquivos
* **Compute:** Clusters (criar/gerenciar)
* **Workflows:** Jobs agendados
* **Machine Learning:** Experimentos MLflow, Feature Store

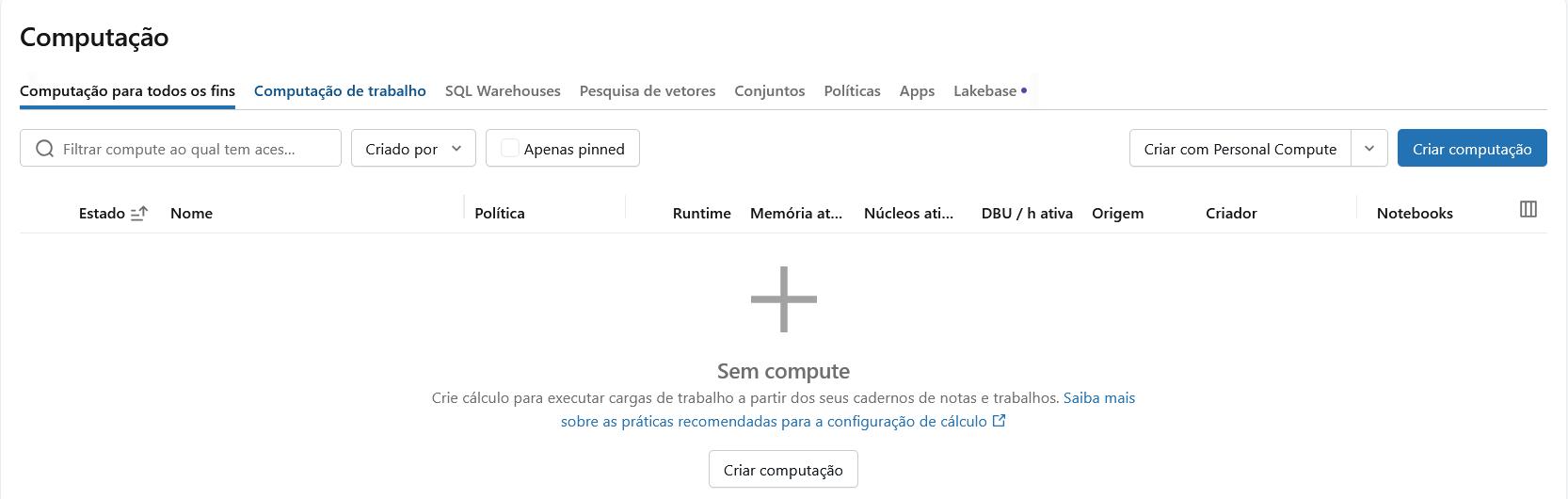
## 6.4 Criar Cluster Spark

Clusters são necessários para executar código.

### Passo 1: Configurar Cluster

**No Databricks Workspace:**

1. **Clique em “Compute”** (menu lateral esquerdo)
2. **Clique em “Create Personal Compute”**
   1. Vamos usar essa opção, por questões de custo para o estudo. No geral as opções são as mesmas e para estudo não teremos necessidade. O valor dela é de 1 DBU/h (Databricks Unity)

* Figura 6.5 - Página de Compute (Clusters)*

1. **Configurar cluster:**

**Cluster name:** cluster-dev-junior

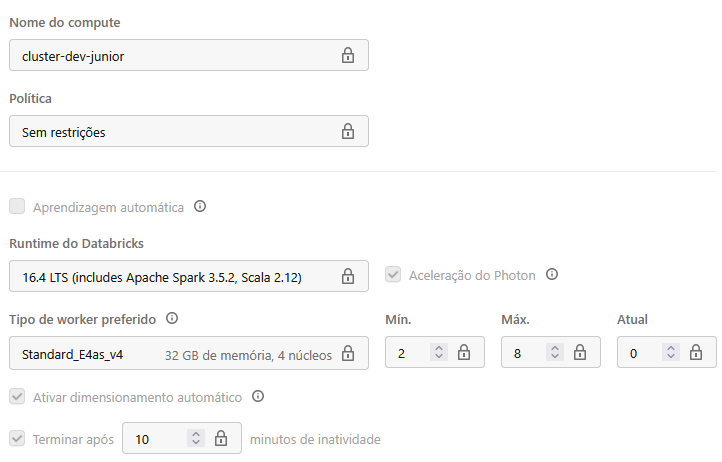
**Cluster Policy:** Sem restrições

**Databricks Runtime Version:** Selecione versão 16.4 LTS (Long Term Support)

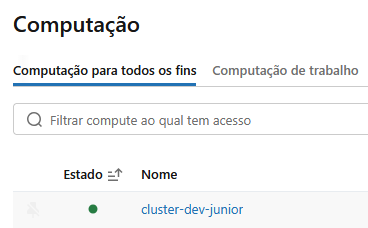
* + Exemplo: 12.2 LTS (Scala 2.12, Spark 3.5.2)
  + Inclui Apache Spark + otimizações Databricks

**Tipo de nó:** Standard\_E4as\_v4 32 GB de memória, 4 núcleos

* + **Terminate after:** 10 minutes of inactivity (seja conservador em DEV)
    - Cluster desliga automaticamente para economizar custos

* Figura 6.6 - Configuração do cluster*

1. **Clique em “Create”**
2. **Aguarde provisionamento** (3-5 minutos)
3. **Status “Running”** (ícone verde)

**

*Figura 6.7 - Cluster provisionado e rodando*

**Custo aproximado (Brazil South, 2026):**

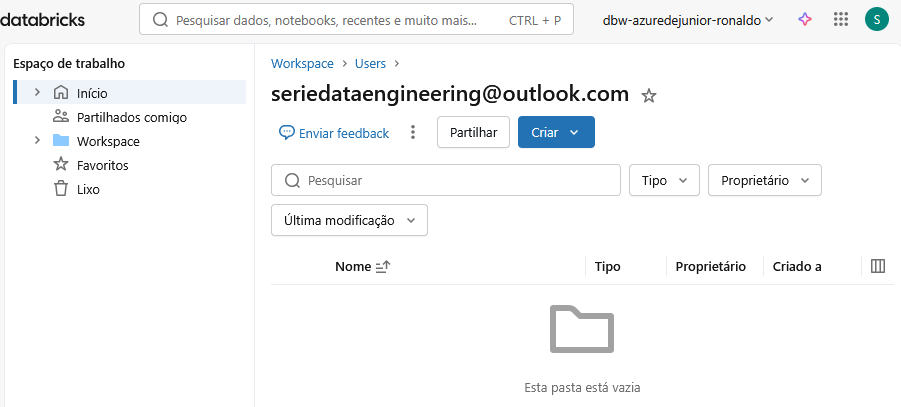
* **Standard\_E4as\_v4:** ~R$ 1,30/hora (VM) + ~R$ 1,00/hora (DBU\*) = ~R$ 2,30/hora
* **Cluster 1 driver + 2 workers:** ~R$ 6,90/hora
* **Dica:** Sempre termine cluster manualmente após uso para evitar custos desnecessários!

## 6.5 Criar Primeiro Notebook

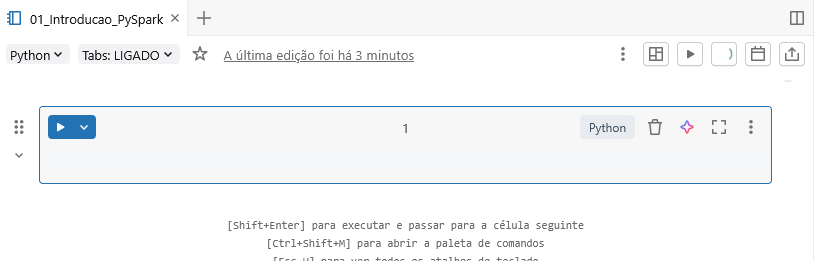
Notebooks são ambientes interativos para escrever código, visualizar dados e documentar análises.

### Passo 1: Criar Notebook

1. **Clique em “Workspace”** (menu lateral esquerdo)
2. **Navegue até pasta** (ex: Users/seu-email@dominio.com)
3. **Clique ao lado, no menu “...” da pasta → “Create” → “Notebook”**

* Figura 6.8 - Criar novo Notebook*

1. **Configurar notebook:**
   * **Name:** 01\_Introducao\_PySpark
   * **Default language:** Python
     + Alternativas: SQL, Scala, R
   * **Cluster:** Selecione cluster-dev-junior eventualmente pode aparecer a opção **Serverless,** clique em mais opções para procurar o cluster criado. Se ele estiver desligado, clique em “Anexar e Iniciar”

**

*Figura 6.9 - Notebook criado e conectado ao cluster*

### Passo 2: Primeira Execução

**No notebook, digite na primeira célula:**

**PYTHON**

# Verificar versão do Spark

import sys

import platform

print(f”Spark version: {spark.version}”)

print(f”Python version: {platform.python\_version()}”)

**Clique em “Run Cell” (Shift + Enter)**

**Saída esperada:**

Spark version: 3.5.2

Python version: 3.12.3

*Figura 6.10 - Primeira execução no notebook*

### Estrutura de Notebooks

**Células:**

* **Code cells:** Código Python, SQL, Scala
* **Markdown cells:** Documentação (inicia com %md)

**Magic commands:**

* %python - Executar Python (default)
* %sql - Executar SQL
* %scala - Executar Scala
* %sh - Executar shell commands
* %md - Markdown
* %fs - File system commands (DBFS)

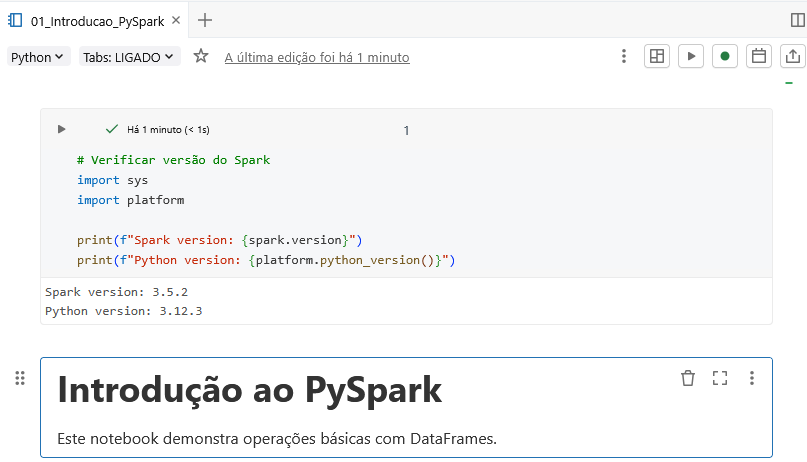
**Exemplo de célula Markdown:**

**MARKDOWN**

%md

# Introdução ao PySpark

Este notebook demonstra operações básicas com DataFrames.

*Figura 6.10 - Primeira execução no notebook*

## 6.6 PySpark Fundamentos

### Criar DataFrame Manualmente

**PYTHON**

# Criar DataFrame a partir de lista Python

data = [

(1, “Alice”, 25, “São Paulo”),

(2, “Bob”, 30, “Rio de Janeiro”),

(3, “Carol”, 28, “Brasília”),

(4, “David”, 35, “Salvador”)

]

schema = [“id”, “nome”, “idade”, “cidade”]

df = spark.createDataFrame(data, schema)

# Mostrar dados

df.show()

**Saída:**

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

| id| nome| idade| cidade|

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

| 1|Alice| 25| São Paulo|

| 2| Bob| 30|Rio de Janeiro|

| 3|Carol| 28| Brasília|

| 4|David| 35| Salvador|

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

### Ler Dados do Data Lake

**Configurar acesso ao Data Lake:**

**PYTHON**

# Configurar credenciais

storage\_account\_name = “stadejuniorronaldo”

storage\_account\_key = “SUA\_ACCESS\_KEY\_AQUI” # Obter do Azure Portal

spark.conf.set(

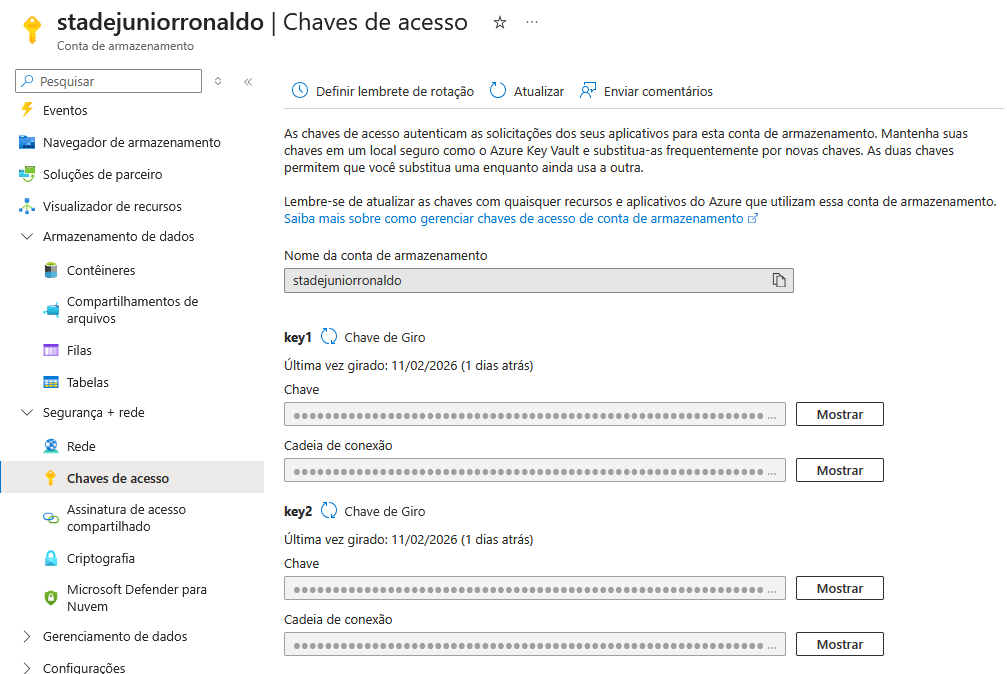
f”fs.azure.account.key.{storage\_account\_name}.dfs.core.windows.net”,

storage\_account\_key

)

**Segurança: Em produção, use Azure Key Vault para armazenar chaves!**

As chaves para acesso ficam disponíveis em “Segurança e Rede”, “Chaves de acesso”. Este livro não tem a intenção de detalhar esse nível de detalhe. Recomenda-se que o Junior (Pleno e Sênior também) tenham a certificação AZ900 e se possível, minimamente a DP900, e estou falando para começar a carreira na área.

****

**Ler arquivo Parquet:**

**PYTHON**

# Caminho no Data Lake

path = f”abfss://datalake@{storage\_account\_name}.dfs.core.windows.net/bronze/clientes/SalesLT.Customer.parquet”

# Ler Parquet

df\_customers = spark.read.parquet(path)

# Mostrar schema

df\_customers.printSchema()

# Mostrar primeiras 10 linhas

df\_customers.show(10)

**Saída:**

root

|-- CustomerID: integer (nullable = true)

|-- FirstName: string (nullable = true)

|-- LastName: string (nullable = true)

|-- CompanyName: string (nullable = true)

|-- EmailAddress: string (nullable = true)

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

|CustomerID|FirstName|LastName| CompanyName| EmailAddress|

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

| 1| Orlando| Gee|A Bike Store |orlando0@adventur...|

| 2| Keith| Harris|Progressive Sport...|keith0@adventure-...|

| 3| Donna| Carreras|Advanced Bike Com...|donna0@adventure-...|

...

### Transformações Básicas

**Select (selecionar colunas):**

**PYTHON**

df\_selected = df\_customers.select(“CustomerID”, “FirstName”, “LastName”)

df\_selected.show(5)

**Filter (filtrar linhas):**

**PYTHON**

# Filtrar clientes com ID > 10

df\_filtered = df\_customers.filter(df\_customers.CustomerID > 10)

# Alternativa (SQL-like)

df\_filtered = df\_customers.filter(“CustomerID > 10”)

# Alternativa (coluna como string)

from pyspark.sql import functions as F

df\_filtered = df\_customers.filter(F.col(“CustomerID”) > 10)

df\_filtered.show(5)

**WithColumn (adicionar/modificar coluna):**

**PYTHON**

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

# Criar coluna FullName

df\_with\_fullname = df\_customers.withColumn(

“FullName”,

concat(col(“FirstName”), lit(“ “), col(“LastName”))

)

df\_with\_fullname.select(“CustomerID”, “FullName”).show(5)

**Saída:**

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

|CustomerID| FullName|

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

| 1| Orlando Gee|

| 2| Keith Harris|

| 3| Donna Carreras|

...

**Aggregate (agrupar e sumarizar):**

**PYTHON**

from pyspark.sql.functions import count, avg, sum

# Contar clientes

total\_customers = df\_customers.count()

print(f”Total de clientes: {total\_customers}”)

# Média de idade (se tivesse coluna Age)

# df\_customers.agg(avg(“Age”)).show()

### Join (combinar DataFrames)

**PYTHON**

# Ler tabela de pedidos

path\_orders = f”abfss://datalake@{storage\_account\_name}.dfs.core.windows.net/bronze/orders/SalesLT.SalesOrderHeader.parquet”

df\_orders = spark.read.parquet(path\_orders)

# Join Inner

df\_joined = df\_customers.join(

df\_orders,

df\_customers.CustomerID == df\_orders.CustomerID,

“inner”

)

df\_joined.select(“FirstName”, “LastName”, “SalesOrderID

”, “OrderDate”).show(10)

**Tipos de Join:**

* “inner” - Inner join (padrão)
* “left” / “left\_outer” - Left outer join
* “right” / “right\_outer” - Right outer join
* “outer” / “full” / “full\_outer” - Full outer join
* “cross” - Cross join (produto cartesiano)
* “left\_semi” - Left semi join (apenas colunas da esquerda)
* “left\_anti” - Left anti join (linhas da esquerda sem match)

### GroupBy (agrupar)

**PYTHON**

# Total de pedidos por cliente

orders = df\_orders.alias("o")

customers = df\_customers.alias("c")

df\_joined = orders.join(

    customers,

    orders.CustomerID == customers.CustomerID

)

df\_grouped = df\_joined.groupBy(

    customers.CustomerID,

    customers.FirstName,

    customers.LastName

).agg(

    count(orders.SalesOrderID).alias("TotalOrders"),

    sum(orders.TotalDue).alias("TotalRevenue")

).orderBy("TotalRevenue", ascending=False)

df\_grouped.show(10)

**Saída:**

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

|CustomerID|FirstName| LastName|TotalOrders|TotalRevenue|

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

| 123| Alice| Smith| 15| 45000.50|

| 456| Bob| Johnson| 12| 38000.00|

...

## 6.7 Spark SQL

Além de DataFrames API, você pode escrever queries SQL diretamente!

### Registrar DataFrame como Tabela Temporária

**PYTHON**

# Registrar tabela temporária

df\_customers.createOrReplaceTempView(“customers”)

# Agora pode usar SQL!

result = spark.sql(“““

SELECT CustomerID, FirstName, LastName

FROM customers

WHERE CustomerID > 10

LIMIT 10

“““)

result.show()

### Query SQL Complexa

**PYTHON**

# Registrar ambas tabelas

df\_customers.createOrReplaceTempView(“customers”)

df\_orders.createOrReplaceTempView(“orders”)

# Query com JOIN e agregação

result = spark.sql(“““

SELECT

c.CustomerID,

c.FirstName,

c.LastName,

COUNT(o.SalesOrderID) AS TotalOrders,

SUM(o.TotalDue) AS TotalRevenue

FROM customers c

INNER JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

ORDER BY TotalRevenue DESC

LIMIT 10

“““)

result.show()

### Magic Command %sql

Você pode criar células SQL direto no notebook:

**SQL**

%sql

SELECT

FirstName,

LastName,

COUNT(\*) AS OrderCount

FROM customers c

INNER JOIN orders o ON c.CustomerID = o.CustomerID

GROUP BY FirstName, LastName

ORDER BY OrderCount DESC

LIMIT 5

**Vantagem:** Sintaxe SQL pura (sem quotes, mais legível)

**Desvantagem:** Não pode misturar Python e SQL na mesma célula

## 6.8 Gravar Dados no Data Lake

### Escrever Parquet

**PYTHON**

# Caminho de saída

output\_path = f”abfss://datalake@{storage\_account\_name}.dfs.core.windows.net/silver/clientes\_transformados/”

# Gravar como Parquet

df\_transformed = df\_customers.select(“CustomerID”, “FirstName”, “LastName”, “EmailAddress”)

df\_transformed.write \

.mode(“overwrite”) \

.parquet(output\_path)

print(f”Dados gravados em: {output\_path}”)

**Modes:**

* “overwrite” - Sobrescreve arquivos existentes
* “append” - Adiciona aos arquivos existentes
* “ignore” - Ignora se já existe
* “error” / “errorifexists” - Erro se já existe (padrão)

### Escrever Delta Lake

**Delta Lake** é formato open-source otimizado (ACID transactions, time travel).

**PYTHON**

# Gravar como Delta

output\_delta = f”abfss://datalake@{storage\_account\_name}.dfs.core.windows.net/silver/clientes\_delta/”

df\_transformed.write \

.format(“delta”) \

.mode(“overwrite”) \

.save(output\_delta)

print(“Dados gravados em formato Delta Lake!”)

**Vantagens Delta Lake:**

* ACID transactions (atomicidade)
* Time travel (versões históricas)
* Schema evolution (mudar schema sem quebrar)
* Performance otimizada (Z-Ordering, file compaction)

# Capítulo 7: Azure Synapse Analytics - Fundamentos

## Objetivos do Capítulo

**Ao final deste capítulo, você será capaz de:**

• Compreender o que é Azure Synapse Analytics e suas principais funcionalidades

• Criar e configurar um workspace do Azure Synapse Analytics

• Entender a diferença entre SQL Pools dedicados e serverless

• Configurar e utilizar Apache Spark Pools

• Criar seu primeiro notebook no Synapse

• Integrar Synapse com Azure Data Lake Storage

• Executar queries SQL e processar dados com Spark

• Gerenciar recursos e monitorar custos

**Tempo estimado:** 4-5 horas

**Custo Azure:** ~$10-20 USD (com créditos gratuitos cobertos)

## 7.1 O que é Azure Synapse Analytics?

### Visão Geral

Azure Synapse Analytics é uma plataforma de análise unificada que reúne Data Warehouse, Big Data e integração de dados em um único serviço. É a evolução do SQL Data Warehouse (SQL DW) com capacidades expandidas.

Pense no Synapse como um "canivete suíço" de dados: você tem Data Warehouse (SQL), processamento Big Data (Spark), pipelines de ETL, e tudo integrado com seu Data Lake.

### O Problema que o Synapse Resolve

**Antes do Synapse, você precisava de múltiplos serviços Azure:**

• Azure Data Factory para orquestração de pipelines

• Azure Synapse para processamento Spark

• Azure SQL Data Warehouse para análises SQL

• Múltiplas ferramentas de monitoramento

**Com Synapse, tudo está em um único workspace:**

**PYTHON**

┌─────────────────────────────────────────────────────────────┐  
│ AZURE SYNAPSE WORKSPACE │  
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │  
│ │ SQL Pools │ │ Spark Pools │ │ Pipelines │ │  
│ │ (dedicado/ │ │ (Big Data) │ │ (ETL/ELT) │ │  
│ │ serverless) │ │ │ │ │ │  
│ └──────────────┘ └──────────────┘ └──────────────┘ │  
│ │ │  
│ ┌─────▼─────┐ │  
│ │ Data Lake │ │  
│ │ Storage │ │  
│ └───────────┘ │  
└─────────────────────────────────────────────────────────────┘

### Componentes Principais

|  |  |  |
| --- | --- | --- |
| **Componente** | **Descrição** | **Caso de Uso** |
| **SQL Pool Dedicado** | Data Warehouse MPP tradicional | Cargas de trabalho previsíveis, BI enterprise |
| **SQL Pool Serverless** | Query sob demanda (pague por query) | Exploração de dados, consultas ad-hoc |
| **Spark Pool** | Apache Spark gerenciado | Big Data, ML, processamento distribuído |
| **Pipelines** | Orquestração (similar ao ADF) | ETL/ELT, automação |
| **Synapse Link** | Conexão real-time com Cosmos DB | HTAP, analytics operacional |
| **Data Explorer** | Time-series e logs (Kusto) | IoT, telemetria, logs |

## 7.2 Criando seu Primeiro Workspace Synapse

### Pré-requisitos

**Antes de começar, você precisa ter:**

• Conta Azure ativa (pode usar a gratuita)

• Um Resource Group criado

• Uma Storage Account com Data Lake Gen2 habilitado

### Passo 1: Acessar o Portal Azure

1. Acesse portal.azure.com

2. Na barra de pesquisa, digite "Synapse"

3. Clique em "Azure Synapse Analytics"

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_synapse\_search.png*

*Figura 7.1 - Pesquisando Azure Synapse no Portal*

### Passo 2: Criar Workspace

**Clique em "+ Create" e preencha:**

**Basics:**

• **Subscription:** Sua assinatura

• Resource Group: rg-dados-junior (ou o seu)

• Workspace name: synapse-dados-junior

• Region: Brazil South (ou sua região)

• **Data Lake Storage:** Selecione sua storage account

• **File system:** Crie um novo chamado "synapse"

|  |
| --- |
| CODE |
| Nomenclatura recomendada: - Workspace: synapse-{projeto}-{ambiente} - Exemplo: synapse-vendas-dev, synapse-analytics-prod |

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_create\_workspace\_basics.png*

*Figura 7.2 - Configurações básicas do Workspace*

**Security:**

• **SQL Administrator login:** sqladmin

• **SQL Administrator password:** (crie uma senha forte)

**Networking:**

• Para ambiente de desenvolvimento: "Enable public access"

• Para produção: Configure Private Endpoints

### Passo 3: Revisar e Criar

1. Revise todas as configurações

2. Clique em "Create"

3. Aguarde ~5 minutos para o deployment

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_workspace\_created.png*

*Figura 7.3 - Workspace criado com sucesso*

### Passo 4: Abrir o Synapse Studio

**Após a criação:**

1. Acesse o recurso Synapse

2. Clique em **"Open Synapse Studio"**

3. Você será redirecionado para web.azuresynapse.net

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_synapse\_studio.png*

*Figura 7.4 - Interface do Synapse Studio*

## 7.3 SQL Pool Serverless - Queries sob Demanda

### O que é SQL Pool Serverless?

SQL Pool Serverless permite executar queries SQL diretamente sobre arquivos no Data Lake sem provisionar infraestrutura. Você paga apenas pelos dados processados (~$5 por TB).

**Vantagens:**

• Zero configuração de infraestrutura

• Escala automaticamente

• Ideal para exploração de dados

• Suporta Parquet, CSV, JSON, Delta Lake

### Sua Primeira Query no Data Lake

**Vamos consultar arquivos CSV diretamente do Data Lake:**

|  |
| --- |
| SQL |
| -- Query arquivos CSV no Data Lake SELECT TOP 100 \* FROM OPENROWSET(  BULK 'https://suastorage.dfs.core.windows.net/bronze/vendas/\*.csv',  FORMAT = 'CSV',  HEADER\_ROW = TRUE ) AS vendas |

**Explicação:**

• `OPENROWSET`: Função que lê arquivos externos

• `BULK`: Caminho para os arquivos (pode usar wildcards \*)

• `FORMAT`: Tipo do arquivo

• `HEADER\_ROW`: Indica que a primeira linha é cabeçalho

### Consultando Parquet

**Parquet é mais eficiente que CSV:**

|  |
| --- |
| SQL |
| -- Query arquivos Parquet SELECT  produto\_id,  SUM(quantidade) as total\_quantidade,  SUM(valor\_total) as receita FROM OPENROWSET(  BULK 'https://suastorage.dfs.core.windows.net/silver/fato\_vendas/\*\*',  FORMAT = 'PARQUET' ) AS vendas GROUP BY produto\_id ORDER BY receita DESC |

### Consultando Delta Lake

**Para ler tabelas Delta Lake:**

|  |
| --- |
| SQL |
| -- Query Delta Lake SELECT \* FROM OPENROWSET(  BULK 'https://suastorage.dfs.core.windows.net/gold/dim\_cliente/',  FORMAT = 'DELTA' ) AS clientes WHERE estado = 'SP' |

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_serverless\_query.png*

*Figura 7.5 - Executando query no SQL Pool Serverless*

## 7.4 Criando Views e External Tables

### Views sobre o Data Lake

**Views facilitam o acesso aos dados sem repetir a sintaxe OPENROWSET:**

|  |
| --- |
| SQL |
| -- Criar database CREATE DATABASE vendas\_analytics; GO  USE vendas\_analytics; GO  -- Criar view sobre Parquet CREATE VIEW vw\_vendas\_diarias AS SELECT  CAST(data\_venda AS DATE) as data,  COUNT(\*) as qtd\_pedidos,  SUM(valor\_total) as receita\_total FROM OPENROWSET(  BULK 'https://suastorage.dfs.core.windows.net/gold/fato\_vendas/',  FORMAT = 'PARQUET' ) AS v GROUP BY CAST(data\_venda AS DATE); GO |

**Agora você pode usar simplesmente:**

|  |
| --- |
| SQL |
| SELECT \* FROM vw\_vendas\_diarias WHERE data >= '2026-01-01' ORDER BY data; |

### External Tables

**External Tables são ainda mais poderosas - parecem tabelas normais:**

|  |
| --- |
| SQL |
| -- Criar External Data Source CREATE EXTERNAL DATA SOURCE DataLakeSource WITH (  LOCATION = 'https://suastorage.dfs.core.windows.net/gold/' ); GO  -- Criar External File Format CREATE EXTERNAL FILE FORMAT ParquetFormat WITH (  FORMAT\_TYPE = PARQUET ); GO  -- Criar External Table CREATE EXTERNAL TABLE dim\_produto (  produto\_id INT,  nome VARCHAR(100),  categoria VARCHAR(50),  preco DECIMAL(10,2) ) WITH (  LOCATION = 'dim\_produto/',  DATA\_SOURCE = DataLakeSource,  FILE\_FORMAT = ParquetFormat ); GO |

**Agora você pode fazer JOIN normalmente:**

|  |
| --- |
| SQL |
| SELECT  p.nome,  p.categoria,  SUM(v.valor\_total) as receita FROM fato\_vendas v JOIN dim\_produto p ON v.produto\_id = p.produto\_id GROUP BY p.nome, p.categoria; |

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_external\_tables.png*

*Figura 7.6 - External Tables no Synapse*

## 7.5 SQL Pool Dedicado (Data Warehouse)

### Quando Usar SQL Pool Dedicado?

**Use SQL Pool Dedicado quando precisar de:**

• Performance previsível e consistente

• Cargas de trabalho de BI intensivas

• Dados modelados (Star Schema)

• SLAs de disponibilidade

### Arquitetura MPP

**SQL Pool Dedicado usa arquitetura Massively Parallel Processing (MPP):**

|  |
| --- |
| CODE |
| ┌─────────────────────────────────────────────────────────────┐ │ CONTROL NODE │ │ (recebe query, cria plano) │ └─────────────────────────────────────────────────────────────┘  │  ┌───────────────┼───────────────┐  ▼ ▼ ▼  ┌──────────────┐ ┌──────────────┐ ┌──────────────┐  │ Compute Node │ │ Compute Node │ │ Compute Node │  │ DW1 │ │ DW2 │ │ DW3 │  └──────────────┘ └──────────────┘ └──────────────┘  │ │ │  ┌──────────────┐ ┌──────────────┐ ┌──────────────┐  │ Distribution │ │ Distribution │ │ Distribution │  │ (Storage) │ │ (Storage) │ │ (Storage) │  └──────────────┘ └──────────────┘ └──────────────┘ |

• **Control Node:** Recebe queries, otimiza, distribui trabalho

• **Compute Nodes:** Executam a query em paralelo

• **Distributions:** 60 distribuições onde os dados são armazenados

### Criando um SQL Pool Dedicado

**No Synapse Studio:**

1. Vá em **Manage** > **SQL pools**

2. Clique em **+ New**

3. Configure:

- **Name:** dwpool01

- **Performance level:** DW100c (menor para testes)

|  |
| --- |
| CODE |
| DW Units (DWU) - Capacidade de processamento: - DW100c: Desenvolvimento (~$1.20/hora) - DW500c: Testes de carga - DW1000c+: Produção |

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_dedicated\_pool.png*

*Figura 7.7 - Criando SQL Pool Dedicado*

### Criando Tabelas no Pool Dedicado

|  |
| --- |
| SQL |
| -- Criar tabela de dimensão (replicada em todos os nodes) CREATE TABLE dim\_cliente (  cliente\_id INT NOT NULL,  nome VARCHAR(100),  email VARCHAR(100),  cidade VARCHAR(50),  estado CHAR(2) ) WITH (  DISTRIBUTION = REPLICATE,  CLUSTERED COLUMNSTORE INDEX );  -- Criar tabela fato (distribuída por hash) CREATE TABLE fato\_vendas (  venda\_id BIGINT NOT NULL,  cliente\_id INT,  produto\_id INT,  data\_venda DATE,  quantidade INT,  valor\_unitario DECIMAL(10,2),  valor\_total DECIMAL(12,2) ) WITH (  DISTRIBUTION = HASH(cliente\_id),  CLUSTERED COLUMNSTORE INDEX ); |

**Tipos de Distribution:**

• **REPLICATE:** Cópia em cada node (dimensões pequenas)

• **HASH:** Distribui por valor de coluna (fatos grandes)

• ROUND\_ROBIN: Distribui uniformemente (staging)

## 7.6 Apache Spark Pool

### O que é Spark Pool?

**Spark Pool é um cluster Apache Spark gerenciado dentro do Synapse. Ideal para:**

• Processamento de grandes volumes (Big Data)

• Transformações complexas

• Machine Learning

• Streaming

### Criando um Spark Pool

1. No Synapse Studio, vá em **Manage** > **Apache Spark pools**

2. Clique em **+ New**

3. Configure:

- **Name:** sparkpool01

- **Node size:** Small (4 vCores, 32GB)

- **Number of nodes:** 3-10 (auto-scale)

- **Auto-pause:** Enabled (após 15 min de inatividade)

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_spark\_pool.png*

*Figura 7.8 - Criando Spark Pool*

### Seu Primeiro Notebook Spark

1. Vá em **Develop** > **+** > **Notebook**

2. Conecte ao seu Spark Pool

3. Execute:

|  |
| --- |
| PYTHON |
| # Célula 1: Ler dados do Data Lake df = spark.read.parquet("abfss://gold@suastorage.dfs.core.windows.net/fato\_vendas/")  # Ver schema df.printSchema()  # Preview display(df.limit(10)) |

|  |
| --- |
| PYTHON |
| # Célula 2: Análise exploratória from pyspark.sql.functions import sum, count, avg  # Agregação por produto resumo = df.groupBy("produto\_id") \  .agg(  count("\*").alias("qtd\_vendas"),  sum("valor\_total").alias("receita\_total"),  avg("valor\_total").alias("ticket\_medio")  ) \  .orderBy("receita\_total", ascending=False)  display(resumo) |

|  |
| --- |
| PYTHON |
| # Célula 3: Salvar resultado resumo.write \  .mode("overwrite") \  .parquet("abfss://gold@suastorage.dfs.core.windows.net/resumo\_vendas\_produto/")  print("Dados salvos com sucesso!") |

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_spark\_notebook.png*

*Figura 7.9 - Notebook Spark no Synapse*

## 7.7 Pipelines de Integração

### Synapse Pipelines vs Azure Data Factory

Synapse Pipelines é praticamente idêntico ao Azure Data Factory, mas integrado ao workspace. Se você já conhece ADF, se sentirá em casa.

**Vantagens de usar Pipelines no Synapse:**

• Tudo no mesmo workspace

• Execução de notebooks Spark

• Execução de scripts SQL

• Triggers integrados

### Criando um Pipeline Simples

1. Vá em **Integrate** > **+** > **Pipeline**

2. Dê o nome: "pl\_processar\_vendas\_diarias"

3. Arraste atividades do painel esquerdo

**Exemplo de Pipeline:**

|  |
| --- |
| CODE |
| ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │ Copy Data │───▶│ Notebook │────▶│ SQL Script │ │ (Bronze→Raw) │ │ (Transform) │ │ (Refresh) │ └──────────────┘ └──────────────┘ └──────────────┘ |

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_pipeline.png*

*Figura 7.10 - Pipeline no Synapse Studio*

## 7.8 Monitoramento e Custos

### Monitor Hub

**O Synapse Studio tem um hub de monitoramento integrado:**

1. Clique em **Monitor** no menu lateral

2. Visualize:

- **Pipeline runs:** Execuções de pipelines

- **SQL requests:** Queries executadas

- **Spark applications:** Jobs Spark

### Gerenciando Custos

**Principais custos do Synapse:**

|  |  |  |
| --- | --- | --- |
| **Componente** | **Cobrança** | **Dica de Economia** |
| **SQL Pool Dedicado** | Por DWU/hora | Pause quando não usar |
| **SQL Pool Serverless** | Por TB processado | Use formatos colunares (Parquet) |
| **Spark Pool** | Por node/hora | Habilite auto-pause |
| **Storage** | Por GB/mês | Lifecycle policies |

**Script para pausar SQL Pool Dedicado:**

|  |
| --- |
| SQL |
| -- Pausar pool (via T-SQL) ALTER DATABASE dwpool01 SET (ALLOW\_PAUSING = ON); |

**Ou via Portal/PowerShell:**

|  |
| --- |
| POWERSHELL |
| # Pausar SQL Pool Suspend-AzSynapseSqlPool -WorkspaceName synapse-dados-junior -Name dwpool01  # Resumir SQL Pool Resume-AzSynapseSqlPool -WorkspaceName synapse-dados-junior -Name dwpool01 |

*[INSERIR IMAGEM]*

*Nome do arquivo: Junior\_Cap07\_monitor.png*

*Figura 7.11 - Monitor Hub do Synapse*

## 7.9 Hands-On: Projeto Prático

**Vamos criar um mini projeto usando tudo que aprendemos:**

### Cenário

**Você recebeu arquivos CSV de vendas e precisa:**

1. Criar External Tables sobre os dados

2. Transformar com Spark

3. Criar views para BI

### Passo 1: Upload dos Dados

**Faça upload do arquivo vendas\_2026.csv para:**

abfss://bronze@suastorage.dfs.core.windows.net/vendas/

### Passo 2: Query Exploratória (Serverless)

|  |
| --- |
| SQL |
| -- Explorar dados brutos SELECT TOP 100 \* FROM OPENROWSET(  BULK 'https://suastorage.dfs.core.windows.net/bronze/vendas/\*.csv',  FORMAT = 'CSV',  HEADER\_ROW = TRUE,  PARSER\_VERSION = '2.0' ) AS dados;  -- Verificar tipos e qualidade SELECT  COUNT(\*) as total\_linhas,  COUNT(DISTINCT produto\_id) as produtos\_unicos,  MIN(data\_venda) as primeira\_venda,  MAX(data\_venda) as ultima\_venda FROM OPENROWSET(  BULK 'https://suastorage.dfs.core.windows.net/bronze/vendas/\*.csv',  FORMAT = 'CSV',  HEADER\_ROW = TRUE ) AS dados; |

### Passo 3: Transformação com Spark

**Crie um notebook e execute:**

|  |
| --- |
| PYTHON |
| # Ler CSV df\_raw = spark.read \  .option("header", "true") \  .option("inferSchema", "true") \  .csv("abfss://bronze@suastorage.dfs.core.windows.net/vendas/")  # Limpeza e transformação from pyspark.sql.functions import \*  df\_clean = df\_raw \  .withColumn("data\_venda", to\_date("data\_venda", "yyyy-MM-dd")) \  .withColumn("valor\_total", col("quantidade") \* col("preco\_unitario")) \  .withColumn("ano", year("data\_venda")) \  .withColumn("mes", month("data\_venda")) \  .dropDuplicates(["venda\_id"]) \  .filter(col("valor\_total") > 0)  # Salvar em Parquet particionado df\_clean.write \  .mode("overwrite") \  .partitionBy("ano", "mes") \  .parquet("abfss://silver@suastorage.dfs.core.windows.net/vendas\_clean/")  print(f"Total de registros processados: {df\_clean.count()}") |

### Passo 4: Criar Views Analíticas

|  |
| --- |
| SQL |
| -- Criar database de analytics CREATE DATABASE IF NOT EXISTS analytics; GO  USE analytics; GO  -- View de vendas diárias CREATE OR ALTER VIEW vw\_vendas\_diarias AS SELECT  CAST(data\_venda AS DATE) as data,  COUNT(\*) as pedidos,  SUM(valor\_total) as receita,  AVG(valor\_total) as ticket\_medio FROM OPENROWSET(  BULK 'https://suastorage.dfs.core.windows.net/silver/vendas\_clean/',  FORMAT = 'PARQUET' ) AS v GROUP BY CAST(data\_venda AS DATE); GO  -- Testar SELECT \* FROM vw\_vendas\_diarias ORDER BY data DESC; |

## 7.10 Checklist de Habilidades

**Antes de prosseguir para o próximo capítulo, verifique se você consegue:**

• [ ] Criar um workspace Azure Synapse

• [ ] Navegar no Synapse Studio

• [ ] Executar queries com SQL Pool Serverless

• [ ] Criar External Tables sobre o Data Lake

• [ ] Entender diferença entre SQL Pool Dedicado e Serverless

• [ ] Criar e usar um Spark Pool

• [ ] Criar notebooks e executar código PySpark

• [ ] Criar pipelines básicos

• [ ] Monitorar execuções e custos

• [ ] Pausar/resumir recursos para economia

## 7.11 Resumo

**Neste capítulo você aprendeu:**

• **Azure Synapse Analytics** é uma plataforma unificada de analytics

• **SQL Pool Serverless** permite queries ad-hoc sem infraestrutura

• **SQL Pool Dedicado** oferece Data Warehouse MPP para cargas intensivas

• **Spark Pool** processa Big Data com Apache Spark gerenciado

• **Pipelines** orquestram ETL/ELT dentro do workspace

• **External Tables** e Views abstraem a complexidade do Data Lake

• **Monitoramento** e gestão de custos são essenciais

## Próximos Passos

No próximo capítulo, você aprenderá sobre **Delta Lake**, o formato de tabela que adiciona ACID transactions ao Data Lake, permitindo updates, deletes e time travel.

# Capítulo 8: Delta Lake - Fundamentos

## Objetivos do Capítulo

**Ao final deste capítulo, você será capaz de:**

• Compreender o que é Delta Lake e suas vantagens

• Criar tabelas Delta Lake no Synapse

• Realizar operações ACID (INSERT, UPDATE, DELETE, MERGE)

• Usar Time Travel para consultar versões históricas

• Implementar Slowly Changing Dimensions (SCD Type 2)

• Otimizar tabelas Delta (OPTIMIZE, Z-ORDERING, VACUUM)

• Trabalhar com schema evolution

• Monitorar histórico de transações

• Integrar Delta Lake com ADF e SQL

**Tempo estimado:** 4-5 horas

**Custo Azure:** Incluído no Synapse (sem custo adicional)

## 8.1 O que é Delta Lake?

**Delta Lake** é um formato de armazenamento open-source que traz **ACID transactions** para Data Lakes.

### Problema: Data Lakes Tradicionais

**Parquet/CSV/JSON:**

• Sem transações atômicas (falhas parciais corrompem dados)

• Sem controle de concorrência (múltiplos writes simultâneos causam conflitos)

• Sem versionamento (não há histórico de mudanças)

• Difícil atualizar/deletar dados (reescrever arquivos inteiros)

• Schema inconsistente (pode mudar sem controle)

### Solução: Delta Lake

**Delta Lake = Parquet + Transaction Log**

**Vantagens:**

• **ACID transactions** (Atomicity, Consistency, Isolation, Durability)

• **Time Travel** (query versões históricas)

• **Schema enforcement** (garante consistência)

• **Schema evolution** (evolui schema de forma segura)

• **UPSERT/MERGE** (atualizar dados facilmente)

• **DELETE** (remover dados sem reescrever tudo)

• **Otimizações** (Z-Ordering, compaction)

• **Streaming + Batch** (mesma tabela para ambos)

### Arquitetura Delta Lake

|  |
| --- |
| CODE |
| Delta Table   \_delta\_log/ (Transaction Log - JSON)  00000000000000000000.json  00000000000000000001.json  00000000000000000002.json   data/ (Parquet files)  part-00000.parquet  part-00001.parquet  part-00002.parquet |

**Transaction Log:**

• Cada transação gera novo arquivo JSON no `\_delta\_log/`

• Contém metadados: arquivos adicionados/removidos, schema, operação

• Garante ACID (leitura sempre consistente)

**Data files:**

• Arquivos Parquet (formato colunar comprimido)

• Delta Lake gerencia automaticamente (compaction, cleanup)

## 8.2 Criar Tabela Delta Lake

### Opção 1: DataFrame API

|  |
| --- |
| PYTHON |
| # Criar DataFrame data = [  (1, "Alice", 25, "São Paulo"),  (2, "Bob", 30, "Rio de Janeiro"),  (3, "Carol", 28, "Brasília") ]  schema = ["id", "nome", "idade", "cidade"] df = spark.createDataFrame(data, schema)  # Gravar como Delta Lake df.write \  .format("delta") \  .mode("overwrite") \  .save("abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta")  print("Tabela Delta criada!") |

### Opção 2: SQL

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark CREATE TABLE clientes\_delta USING DELTA LOCATION 'abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta' AS SELECT  1 AS id,  'Alice' AS nome,  25 AS idade,  'São Paulo' AS cidade UNION ALL SELECT 2, 'Bob', 30, 'Rio de Janeiro' UNION ALL SELECT 3, 'Carol', 28, 'Brasília' |

### Opção 3: CREATE TABLE (managed table)

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark CREATE TABLE IF NOT EXISTS clientes\_delta (  id INT,  nome STRING,  idade INT,  cidade STRING ) USING DELTA |

**Diferença:**

• **External table** (LOCATION): Dados em caminho específico (Data Lake)

• **Managed table** (sem LOCATION): Synapse gerencia localização (Data Lake Storage)

Recomendação: Use external tables (LOCATION) para integração com outros sistemas.

## 8.3 Operações ACID

### INSERT

**Adicionar novos dados:**

|  |
| --- |
| PYTHON |
| from pyspark.sql.functions import lit  # Novo registro new\_data = [  (4, "David", 35, "Salvador") ]  df\_new = spark.createDataFrame(new\_data, schema)  # Append (adicionar) df\_new.write \  .format("delta") \  .mode("append") \  .save("abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta") |

**SQL:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark INSERT INTO clientes\_delta VALUES (4, 'David', 35, 'Salvador') |

### UPDATE

**Atualizar registros existentes:**

|  |
| --- |
| PYTHON |
| from delta.tables import DeltaTable  # Carregar tabela Delta delta\_table = DeltaTable.forPath(spark, "abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta")  # Update delta\_table.update(  condition="id = 2",  set={"idade": "31"} )  print("Registro atualizado!") |

**SQL:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark UPDATE clientes\_delta SET idade = 31 WHERE id = 2 |

**Verificar:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark SELECT \* FROM clientes\_delta WHERE id = 2 |

### DELETE

**Remover registros:**

|  |
| --- |
| PYTHON |
| delta\_table.delete(condition="id = 3") |

**SQL:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark DELETE FROM clientes\_delta WHERE id = 3 |

### MERGE (UPSERT)

**MERGE** combina INSERT + UPDATE em uma operação.

**Cenário:** Recebemos novos dados. Se ID já existe, atualiza; senão, insere.

|  |
| --- |
| PYTHON |
| # Novos dados (mix de updates e inserts) updates\_data = [  (2, "Bob", 32, "Rio de Janeiro"), # ID 2 existe → UPDATE  (5, "Eva", 27, "Fortaleza") # ID 5 não existe → INSERT ]  df\_updates = spark.createDataFrame(updates\_data, schema)  # MERGE delta\_table.alias("target").merge(  df\_updates.alias("source"),  "target.id = source.id" # Condição de match ).whenMatchedUpdate(  set={  "nome": "source.nome",  "idade": "source.idade",  "cidade": "source.cidade"  } ).whenNotMatchedInsert(  values={  "id": "source.id",  "nome": "source.nome",  "idade": "source.idade",  "cidade": "source.cidade"  } ).execute()  print("MERGE concluído!") |

**SQL:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark MERGE INTO clientes\_delta AS target USING (  SELECT 2 AS id, 'Bob' AS nome, 32 AS idade, 'Rio de Janeiro' AS cidade  UNION ALL  SELECT 5, 'Eva', 27, 'Fortaleza' ) AS source ON target.id = source.id WHEN MATCHED THEN UPDATE SET \* WHEN NOT MATCHED THEN INSERT \* |

**MERGE é fundamental para SCD Type 1 e Type 2!**

## 8.4 Time Travel

Delta Lake mantém histórico de todas transações.

### Consultar Versão Específica

**Por número de versão:**

|  |
| --- |
| PYTHON |
| # Ler versão 0 (primeira) df\_v0 = spark.read \  .format("delta") \  .option("versionAsOf", 0) \  .load("abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta")  df\_v0.show() |

**SQL:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark SELECT \* FROM clientes\_delta VERSION AS OF 0 |

**Por timestamp:**

|  |
| --- |
| PYTHON |
| # Ler dados como estavam em data específica df\_history = spark.read \  .format("delta") \  .option("timestampAsOf", "2024-02-08 10:00:00") \  .load("abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta")  df\_history.show() |

**SQL:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark SELECT \* FROM clientes\_delta TIMESTAMP AS OF '2024-02-08 10:00:00' |

### Visualizar Histórico de Transações

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark DESCRIBE HISTORY clientes\_delta |

**Saída:**

|  |
| --- |
| CODE |
| version | timestamp | operation | operationParameters --------|---------------------|-----------|-------------------- 3 | 2024-02-08 12:30:00 | MERGE | {"predicate": "..."} 2 | 2024-02-08 12:00:00 | DELETE | {"predicate": "id = 3"} 1 | 2024-02-08 11:30:00 | UPDATE | {"predicate": "id = 2"} 0 | 2024-02-08 11:00:00 | WRITE | {"mode": "Overwrite"} |

### Restaurar Versão Anterior

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark RESTORE TABLE clientes\_delta TO VERSION AS OF 1 |

**Use case:** Rollback após erro.

## 8.5 Slowly Changing Dimensions (SCD Type 2)

SCD Type 2: Manter histórico completo de mudanças (não sobrescrever).

**Exemplo:** Cliente muda de cidade. Queremos manter registro da cidade antiga.

### Schema SCD Type 2

|  |
| --- |
| PYTHON |
| # Schema expandido com controle de versão scd\_schema = [  "id", # ID do cliente  "nome",  "idade",  "cidade",  "valid\_from", # Data início de validade  "valid\_to", # Data fim de validade (NULL = atual)  "is\_current" # Booleano (registro atual?) ] |

### Implementação SCD Type 2 com MERGE

|  |
| --- |
| PYTHON |
| from pyspark.sql.functions import col, lit, current\_timestamp  # Dados novos (cliente 1 mudou de cidade) new\_data = [  (1, "Alice", 25, "Belo Horizonte") # Cidade mudou ]  df\_new = spark.createDataFrame(new\_data, ["id", "nome", "idade", "cidade"])  # Adicionar campos SCD df\_new = df\_new \  .withColumn("valid\_from", current\_timestamp()) \  .withColumn("valid\_to", lit(None).cast("timestamp")) \  .withColumn("is\_current", lit(True))  # MERGE SCD Type 2 delta\_table.alias("target").merge(  df\_new.alias("source"),  "target.id = source.id AND target.is\_current = true" ).whenMatchedUpdate(  condition="target.cidade <> source.cidade", # Mudou cidade  set={  "valid\_to": "current\_timestamp()",  "is\_current": "false"  } ).whenNotMatchedInsert(  values={  "id": "source.id",  "nome": "source.nome",  "idade": "source.idade",  "cidade": "source.cidade",  "valid\_from": "source.valid\_from",  "valid\_to": "source.valid\_to",  "is\_current": "source.is\_current"  } ).execute()  # Inserir novo registro (versão atual) df\_new.write \  .format("delta") \  .mode("append") \  .save("abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta") |

**Resultado:**

|  |
| --- |
| CODE |
| id | nome | cidade | valid\_from | valid\_to | is\_current ---|-------|-----------------|---------------------|---------------------|------------ 1 | Alice | São Paulo | 2024-02-01 10:00:00 | 2024-02-08 12:00:00 | false 1 | Alice | Belo Horizonte | 2024-02-08 12:00:00 | NULL | true |

**Histórico completo mantido!**

## 8.6 Schema Evolution

Delta Lake permite evolução de schema segura.

### Adicionar Coluna

|  |
| --- |
| PYTHON |
| # Adicionar coluna "email" df\_with\_email = df.withColumn("email", lit("exemplo@email.com"))  # Gravar com opção mergeSchema df\_with\_email.write \  .format("delta") \  .mode("append") \  .option("mergeSchema", "true") \  .save("abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta") |

**SQL:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark ALTER TABLE clientes\_delta ADD COLUMNS (email STRING) |

### Schema Enforcement

**Se tentar gravar dados com schema incompatível, Delta Lake rejeita:**

|  |
| --- |
| PYTHON |
| # Tentar gravar com coluna errada bad\_data = [(1, "Teste", 99)] # Falta coluna "cidade" df\_bad = spark.createDataFrame(bad\_data, ["id", "nome", "idade"])  # Erro: "A schema mismatch detected..." df\_bad.write.format("delta").mode("append").save("abfss://datalake@storage.dfs.core.windows.net/silver/clientes\_delta") |

**Proteção contra corrupção de dados!**

## 8.7 Otimizações Delta Lake

### OPTIMIZE (File Compaction)

Delta Lake pode gerar muitos arquivos pequenos (cada INSERT/UPDATE). Isso degrada performance.

OPTIMIZE combina arquivos pequenos em arquivos maiores (ideal: 1 GB cada).

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark OPTIMIZE clientes\_delta |

**Quando executar:**

• Após muitos INSERTs/UPDATEs

• Performance de leitura degradando

• Recomendado: Diariamente (em janelas de manutenção)

### Z-ORDERING

**Z-Ordering** reorganiza dados para otimizar queries com filtros específicos.

**Exemplo: Se sempre filtra por `cidade`, Z-Order por essa coluna:**

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark OPTIMIZE clientes\_delta ZORDER BY (cidade) |

Resultado: Dados da mesma cidade ficam fisicamente próximos (menos I/O).

**Quando usar:**

• Filtros frequentes em colunas específicas

• Joins em colunas específicas

• Não use em colunas com alta cardinalidade (ex: ID único)

### VACUUM (Limpeza de Arquivos)

Delta Lake mantém arquivos antigos para Time Travel. **VACUUM** remove arquivos não mais necessários.

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark VACUUM clientes\_delta RETAIN 168 HOURS -- Manter últimos 7 dias |

**Default:** 7 dias (168 horas)

**Atenção:** Após VACUUM, Time Travel além do período retido não funciona!

**Recomendação produção:**

• VACUUM semanalmente

• RETAIN 7-30 dias (depende de compliance)

## 8.8 Monitoramento e Debugging

### Estatísticas da Tabela

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark DESCRIBE DETAIL clientes\_delta |

**Saída:**

|  |
| --- |
| CODE |
| format | location | numFiles | sizeInBytes | ... -------|---------------------------|----------|-------------|---- delta | abfss://datalake@storage.dfs.core.windows.net/silver/... | 5 | 1024000 | ... |

### Histórico de Operações

|  |
| --- |
| SQL |
| -- SQL no Synapse Spark DESCRIBE HISTORY clientes\_delta LIMIT 10 |

### Metadados de Arquivos

|  |
| --- |
| PYTHON |
| # Ver arquivos da versão atual delta\_table.detail().select("numFiles", "sizeInBytes").show() |

## 8.9 Integração Delta Lake

### Com Azure Data Factory

**ADF pode ler/gravar Delta Lake via Synapse:**

1. **Copy Activity:**

- Source: Delta Lake (via Synapse Linked Service)

- Sink: SQL Database / Synapse / Outro destino

2. **Notebook Activity:**

- Executar transformações Delta no Synapse

- ADF orquestra

### Com Power BI

**Power BI pode conectar em Delta Lake:**

1. **Via Synapse SQL Endpoint:**

- Synapse expõe tabelas Delta como SQL

- Power BI conecta via ODBC/JDBC

2. Direct Lake (Preview):

- Power BI lê Delta Lake diretamente (sem Synapse)

- Requer Microsoft Fabric

### Com Synapse Analytics

**Synapse pode ler Delta Lake:**

|  |
| --- |
| SQL |
| -- Synapse SQL Serverless SELECT \* FROM OPENROWSET(  BULK 'https://stadatalake.dfs.core.windows.net/datalake/silver/clientes\_delta/',  FORMAT = 'DELTA' ) AS clientes |

## 8.10 Checklist de Habilidades

**Ao final deste capítulo, você deve conseguir:**

**Conceitos:**

• [ ] Explicar vantagens Delta Lake vs Parquet

• [ ] Entender ACID transactions

• [ ] Diferenciar SCD Type 1 vs Type 2

**Práticos:**

• [ ] Criar tabelas Delta Lake (DataFrame API e SQL)

• [ ] Realizar operações: INSERT, UPDATE, DELETE

• [ ] Implementar MERGE (UPSERT)

• [ ] Usar Time Travel (versões históricas)

• [ ] Implementar SCD Type 2 com Delta

• [ ] Evoluir schema (adicionar colunas)

• [ ] Otimizar tabelas (OPTIMIZE, Z-ORDERING)

• [ ] Limpar arquivos antigos (VACUUM)

• [ ] Monitorar histórico de transações

## 8.11 Próximos Passos

**No Capítulo 8, você vai:**

Usar Delta Lake com Structured Streaming (dados em tempo real)

Implementar Change Data Capture (CDC)

Orquestrar pipelines Delta com ADF

Integrar Delta Lake com Synapse Analytics

**Preparação:**

• Pratique operações Delta Lake no Synapse

• Revise conceitos de streaming (opcional)

## 8.12 Recursos Complementares

**Documentação Oficial:**

• [Delta Lake Documentation](https://docs.delta.io/latest/index.html)

• [Synapse Delta Lake Guide](https://learn.microsoft.com/azure/synapse/delta/)

• [Best Practices](https://docs.delta.io/latest/best-practices.html)

**Tutoriais:**

• [Delta Lake Quickstart](https://docs.delta.io/latest/quick-start.html)

• [SCD Type 2 Tutorial](https://learn.microsoft.com/azure/synapse/delta/scd-type-2)

## 8.13 Exercícios Práticos

**Exercício 1: CRUD Completo**

**Crie tabela Delta produtos\_delta:**

1. INSERT: Adicionar 5 produtos

2. UPDATE: Atualizar preço de 2 produtos

3. DELETE: Remover 1 produto descontinuado

4. SELECT: Consultar tabela final

**Exercício 2: Time Travel**

1. Criar tabela Delta com dados iniciais

2. Fazer 3 operações (INSERT, UPDATE, DELETE)

3. Usar DESCRIBE HISTORY para ver versões

4. Consultar versões 0, 1, 2

5. Restaurar para versão 1

**Exercício 3: SCD Type 2**

**Implementar SCD Type 2 para tabela clientes:**

1. Criar schema com valid\_from, valid\_to, is\_current

2. Inserir dados iniciais

3. Simular mudança de endereço de 2 clientes

4. Implementar MERGE para SCD Type 2

5. Verificar histórico mantido

## Resumo do Capítulo

**Neste capítulo você aprendeu:**

**Delta Lake:** Formato ACID para Data Lakes (Parquet + Transaction Log)

**Vantagens:**

• ACID transactions

• Time Travel (query histórico)

• Schema enforcement/evolution

• MERGE (UPSERT)

**Operações ACID:** INSERT, UPDATE, DELETE, MERGE

Time Travel: Consultar versões históricas, restaurar, auditar

**SCD Type 2:** Manter histórico completo de mudanças

**Otimizações:**

• OPTIMIZE (compaction)

• Z-ORDERING (reorganizar dados)

• VACUUM (limpeza)

**Integração:** ADF, Power BI, Synapse

**Próximo capítulo:** Streaming com Delta Lake!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Capítulo 9: Streaming e Orquestração com Delta Lake

## 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 Synapse (custo do Spark Pool)

## 9.1 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.

|  |
| --- |
| CODE |
| 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

## 9.2 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)

## 9.3 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 = "abfss://datalake@storage.dfs.core.windows.net/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):**

• Synapse-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 = "abfss://datalake@storage.dfs.core.windows.net/silver/eventos\_compra/" checkpoint\_path = "abfss://datalake@storage.dfs.core.windows.net/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:**

|  |
| --- |
| CODE |
| +---+----------+-------+-------------------+-------------------+ | 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| ... |

## 9.4 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 = "abfss://datalake@storage.dfs.core.windows.net/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", "abfss://datalake@storage.dfs.core.windows.net/checkpoints/clientes\_cdc/") \  .start("abfss://datalake@storage.dfs.core.windows.net/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) |

## 9.5 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() |

## 9.6 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:**

|  |
| --- |
| CODE |
| +-------------+-----+ | 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", "abfss://datalake@storage.dfs.core.windows.net/checkpoints/windowed/") \  .start("abfss://datalake@storage.dfs.core.windows.net/gold/eventos\_windowed/") |

**Output:**

|  |
| --- |
| CODE |
| +------------------------------------------+-------+-----+ |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| ... |

## 9.7 Checkpointing e Tolerância a Falhas

**Checkpointing** é essencial para produção.

### Como Funciona

|  |
| --- |
| CODE |
| 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", "abfss://datalake@storage.dfs.core.windows.net/checkpoints/my\_query/") \  .start("abfss://datalake@storage.dfs.core.windows.net/output/")  # Se falhar e reiniciar, Spark: # 1. Lê checkpoint # 2. Retoma do offset exato # 3. Continua processamento |

## 9.8 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[abfss://datalake@storage.dfs.core.windows.net/input]",  "startOffset": 100,  "endOffset": 200,  "numInputRows": 1000,  "inputRowsPerSecond": 100.0,  "processedRowsPerSecond": 200.0  }  ],  "sink": {  "description": "DeltaSink[abfss://datalake@storage.dfs.core.windows.net/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}") |

## 9.9 Integração com Azure Data Factory

ADF pode orquestrar streaming jobs Synapse.

### Opção 1: Notebook com Trigger Once

**No Synapse 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 Synapse:**

• Job permanente (não notebook)

• Cluster dedicado com auto-restart

**No ADF:**

• Não necessário (streaming roda independente)

• ADF só usado para orquestrar ETL batch

## 9.10 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("abfss://datalake@storage.dfs.core.windows.net/unified\_table/")  # Batch READ (em outro notebook) df\_batch = spark.read.format("delta").load("abfss://datalake@storage.dfs.core.windows.net/unified\_table/") df\_batch.show()  # Streaming READ (CDC) df\_stream\_read = spark.readStream.format("delta").load("abfss://datalake@storage.dfs.core.windows.net/unified\_table/") |

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

## 9.11 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) |

## 9.12 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 Synapse:**

- 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("abfss://datalake@storage.dfs.core.windows.net/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, "abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/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", "abfss://datalake@storage.dfs.core.windows.net/checkpoints/cdc/") \  .start() |

## 9.13 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

## 9.14 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 Synapse

## 9.15 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/synapse/ingestion/auto-loader/)

**Tutoriais:**

• [Streaming with Delta Lake](https://learn.microsoft.com/azure/synapse/delta/delta-streaming)

• [CDC Tutorial](https://docs.delta.io/latest/delta-change-data-feed.html)

## 9.16 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!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Volume 1: JUNIOR

# Capítulo 10: 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

## 10.1 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("abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/landing/vendas\_2024-02-09.parquet")  # Tabela destino delta\_table = DeltaTable.forPath(spark, "abfss://datalake@storage.dfs.core.windows.net/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!") |

## 10.2 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("abfss://datalake@storage.dfs.core.windows.net/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() |

## 10.3 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("abfss://datalake@storage.dfs.core.windows.net/bronze/clientes/") df\_quality = data\_quality\_checks(df, "clientes")  df\_quality.show(truncate=False) |

**Output:**

|  |
| --- |
| CODE |
| +----------+------------------+------+---------------+ |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("abfss://datalake@storage.dfs.core.windows.net/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... |

## 10.4 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("abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/silver/clientes/")  # Gravar inválidos em DLQ if df\_invalid.count() > 0:  df\_invalid.withColumn("dlq\_timestamp", current\_timestamp()) \  .write.format("delta").mode("append").save("abfss://datalake@storage.dfs.core.windows.net/dlq/clientes/")   print(f" {df\_invalid.count()} registros enviados para DLQ") |

## 10.5 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("abfss://datalake@storage.dfs.core.windows.net/control/runs/")  # Verificar se run\_id já existe antes de processar (evitar duplicação) existing\_runs = spark.read.format("delta").load("abfss://datalake@storage.dfs.core.windows.net/control/runs/") \  .filter(f"run\_id = '{run\_id}'").count()  if existing\_runs > 0:  print("Run ID já processado, pulando...")  mssparkutils.notebook.exit("SKIPPED") |

## 10.6 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("abfss://datalake@storage.dfs.core.windows.net/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("abfss://datalake@storage.dfs.core.windows.net/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.`abfss://datalake@storage.dfs.core.windows.net/silver/clientes/`")  delta\_history.select(  "version",  "timestamp",  "operation",  "operationParameters.mode",  "operationMetrics.numOutputRows" ).show(20, truncate=False) |

## 10.7 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("abfss://datalake@storage.dfs.core.windows.net/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 Synapse, retornar status:**

|  |
| --- |
| PYTHON |
| # Se erro, retornar status FAILED if failed\_checks > 0:  mssparkutils.notebook.exit("FAILED") else:  mssparkutils.notebook.exit("SUCCESS") |

**No ADF, verificar output:**

|  |
| --- |
| JSON |
| // Pipeline ADF {  "activities": [  {  "type": "SynapseNotebook",  "outputs": [{  "name": "notebookOutput"  }]  },  {  "type": "IfCondition",  "dependsOn": ["SynapseNotebook"],  "condition": "@equals(activity('SynapseNotebook').output.runOutput, 'FAILED')",  "ifTrueActivities": [  {  "type": "WebActivity",  "url": "https://hooks.slack.com/...", // Webhook Slack/Teams  "body": {  "text": "Pipeline vendas\_etl FALHOU!"  }  }  ]  }  ] } |

## 10.8 Best Practices de ETL

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

|  |
| --- |
| CODE |
| 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("abfss://datalake@storage.dfs.core.windows.net/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.`abfss://datalake@storage.dfs.core.windows.net/silver/vendas/` ZORDER BY (customer\_id)")  # VACUUM mensalmente spark.sql("VACUUM delta.`abfss://datalake@storage.dfs.core.windows.net/silver/vendas/` RETAIN 168 HOURS") |

## 10.9 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

## 10.10 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

## 10.11 Recursos Complementares

**Documentação:**

• [Delta Lake Best Practices](https://docs.delta.io/latest/best-practices.html)

• [Synapse ETL Patterns](https://learn.microsoft.com/azure/synapse/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)

• Synapse 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!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Volume 1: JUNIOR

# Capítulo 11: 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, Synapse)

• 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)

## 11.1 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 |

## 11.2 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**

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap11\_01\_powerbi\_desktop\_home.png`

**Descrição:** Tela inicial do Power BI Desktop mostrando opções: Get data, Recent sources, Open other reports.

Legenda: \*Figura 11.1 - Power BI Desktop (tela inicial)\*

## 11.3 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\_Cap11\_02\_get\_data\_azure\_sql.png`

**Descrição:** Janela "Get Data" mostrando lista de fontes, "Azure SQL Database" selecionado.

**Legenda:** \*Figura 11.2 - 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\_Cap11\_03\_sql\_connection\_config.png`

**Descrição:** Formulário de conexão SQL Database mostrando Server, Database, modo Import selecionado.

**Legenda:** \*Figura 11.3 - 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\_Cap11\_04\_navigator\_select\_tables.png`

Descrição: Navigator mostrando lista de tabelas, 4 tabelas selecionadas (checkboxes), preview dos dados à direita.

**Legenda:** \*Figura 11.4 - Seleção de tabelas no Navigator\*

## 11.4 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\_Cap11\_05\_data\_model\_relationships.png`

Descrição: View de modelo mostrando 4 tabelas conectadas por linhas (relacionamentos), formato Star Schema.

Legenda: \*Figura 11.5 - 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)

## 11.5 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\_Cap11\_06\_power\_query\_editor.png`

**Descrição:** Power Query Editor mostrando lista de queries à esquerda, preview de dados ao centro, Applied Steps à direita.

**Legenda:** \*Figura 11.6 - 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)

## 11.6 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\_Cap11\_07\_card\_total\_vendas.png`

**Descrição:** Card mostrando "Total de Vendas: R$ 5,234,567.89" formatado.

**Legenda:** \*Figura 11.7 - 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\_Cap11\_08\_bar\_chart\_category.png`

Descrição: Gráfico de barras horizontais mostrando vendas por categoria (Bikes, Accessories, Clothing).

**Legenda:** \*Figura 11.8 - 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\_Cap11\_09\_line\_chart\_time.png`

**Descrição:** Gráfico de linha mostrando evolução de vendas por mês/ano.

Legenda: \*Figura 11.9 - 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\_Cap11\_10\_table\_customers.png`

**Descrição:** Tabela mostrando top clientes por valor de vendas.

**Legenda:** \*Figura 11.10 - 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\_Cap11\_11\_slicer\_date.png`

**Descrição:** Slicer de data mostrando range slider para selecionar período.

Legenda: \*Figura 11.11 - Slicer de data (filtro interativo)\*

## 11.7 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]) |

3. **Formatar:** Home → Format → Currency → R$

4. **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**

## 11.8 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" |

5. **Save**

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap11\_12\_rls\_role\_creation.png`

**Descrição:** Interface de criação de Role mostrando nome da role, tabela selecionada, DAX filter aplicado.

Legenda: \*Figura 11.12 - 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**

## 11.9 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\_Cap11\_13\_publish\_success.png`

**Descrição:** Janela de sucesso mostrando "Publishing to Power BI... Success!" com link para abrir relatório.

**Legenda:** \*Figura 11.13 - 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\_Cap11\_14\_powerbi\_service\_report.png`

**Descrição:** Power BI Service mostrando relatório publicado, dashboard interativo.

**Legenda:** \*Figura 11.14 - Relatório no Power BI Service\*

## 11.10 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\_Cap11\_15\_schedule\_refresh.png`

Descrição: Configuração de refresh agendado mostrando frequência (Daily), horário (08:00), timezone.

**Legenda:** \*Figura 11.15 - Agendamento de refresh automático\*

### Refresh Manual

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

## 11.11 Conectar em Data Lake (Delta Lake)

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

### Passo 1: Habilitar SQL Endpoint no Synapse

1. **Synapse Workspace → SQL → SQL Warehouses**

2. Create SQL Warehouse (ou usar existente)

3. **Copiar Server hostname e HTTP path**

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap11\_16\_synapse\_sql\_endpoint.png`

**Descrição:** Synapse SQL Warehouse mostrando Server hostname e HTTP path copiados.

**Legenda:** \*Figura 11.16 - Synapse SQL Endpoint\*

### Passo 2: Conectar no Power BI

1. **Power BI Desktop → Get data → More → Azure Synapse**

2. **Server hostname:** `adb-123456789.azuresynapse.net`

3. **HTTP path:** `/sql/1.0/warehouses/abc123`

4. **Data Connectivity mode:** Import

5. Autenticação: Personal Access Token (gerar no Synapse)

6. **Selecionar tabelas Delta**

7. **Load**

## 11.12 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.

## 11.13 Integração com Pipelines de Dados

### Fluxo Completo

|  |
| --- |
| CODE |
| 1. Azure Data Factory:  - Ingest dados (bronze)  - Transform (silver/gold)  2. Synapse:  - Processar Big Data  - Gravar Delta Lake (gold layer)  3. Power BI:  - Conectar em Synapse SQL Endpoint  - Ler tabelas gold (agregadas)  - Dashboards para negócio  4. Refresh automático:  - ADF roda diariamente → atualiza gold  - Power BI refresh lê novos dados |

## 11.14 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 Synapse (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

## 11.15 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/)

## 11.16 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

• Synapse (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!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Volume 1: JUNIOR

# Capítulo 12: 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)

## 12.1 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** | **Synapse** | **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 Synapse)

**3. Pipelines:**

• ETL/ELT (mesma engine do ADF)

**4. Synapse Studio:**

• Interface web unificada

## 12.2 Provisionar Synapse Workspace

### Passo 1: Criar Workspace

**Azure Portal:**

1. **Barra de pesquisa:** `synapse`

2. **Azure Synapse Analytics → + Create**

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap12\_01\_synapse\_create.png`

Descrição: Página de Azure Synapse no portal com botão "+ Create" destacado.

**Legenda:** \*Figura 12.1 - Criar Synapse Workspace\*

3. **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)

4. **Aba "Security":**

- **SQL admin login:** sqladmin

- **Password:** <senha\_forte>

5. **Review + create → Create**

6. Aguardar deployment (5-7 minutos)

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap12\_02\_synapse\_workspace\_created.png`

Descrição: Overview do Synapse Workspace criado, botão "Open Synapse Studio" destacado.

**Legenda:** \*Figura 12.2 - 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\_Cap12\_03\_synapse\_studio\_home.png`

**Descrição:** Synapse Studio mostrando menu lateral: Data, Develop, Integrate, Monitor, Manage.

Legenda: \*Figura 12.3 - Synapse Studio (home)\*

## 12.3 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 |

4. **Run**

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap12\_04\_serverless\_query\_parquet.png`

**Descrição:** SQL script no Synapse Studio mostrando query OPENROWSET em Parquet, resultados abaixo.

**Legenda:** \*Figura 12.4 - 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!**

## 12.4 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\_Cap12\_05\_dedicated\_pool\_created.png`

Descrição: Lista de SQL Pools mostrando sqldw\_junior com status "Online", Performance level DW100c.

**Legenda:** \*Figura 12.5 - 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

## 12.5 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\_Cap12\_06\_synapse\_pipeline\_copy.png`

Descrição: Pipeline do Synapse mostrando Copy data activity, Source (Data Lake), Sink (Dedicated Pool).

Legenda: \*Figura 12.6 - 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).

## 12.6 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 ); |

## 12.7 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; |

## 12.8 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\_Cap12\_07\_powerbi\_synapse\_connection.png`

Descrição: Power BI conectado em Synapse Dedicated Pool, Navigator mostrando Star Schema tables.

**Legenda:** \*Figura 12.7 - Power BI conectado em Synapse\*

## 12.9 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

## 12.10 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

## 12.11 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)

## 12.12 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!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Volume 1: JUNIOR

# Capítulo 13: 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)

## 13.1 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

## 13.2 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, Synapse)

**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**

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap13\_01\_azure\_ad\_user.png`

**Descrição:** Página de criação de usuário no Azure AD mostrando campos: User principal name, Name, Password.

**Legenda:** \*Figura 13.1 - Criar usuário no Azure AD\*

## 13.3 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\_Cap13\_02\_rbac\_assignment.png`

Descrição: Interface de atribuição de role mostrando Role selecionado (Storage Blob Data Reader), usuário selecionado.

**Legenda:** \*Figura 13.2 - 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\_Cap13\_03\_adf\_managed\_identity.png`

**Descrição:** Página Identity do ADF mostrando System assigned identity habilitada, Object ID visível.

**Legenda:** \*Figura 13.3 - 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!**

## 13.4 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\_Cap13\_04\_key\_vault\_created.png`

**Descrição:** Overview do Key Vault criado mostrando nome, região, pricing tier.

**Legenda:** \*Figura 13.4 - 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\_Cap13\_05\_key\_vault\_secret.png`

Descrição: Formulário de criação de secret mostrando Name, Value (mascarado).

**Legenda:** \*Figura 13.5 - 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\_Cap13\_06\_adf\_key\_vault\_linked\_service.png`

**Descrição:** Linked Service SQL mostrando password configurado via Azure Key Vault.

**Legenda:** \*Figura 13.6 - 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

## 13.5 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\_Cap13\_07\_private\_endpoint.png`

**Descrição:** Private endpoint criado mostrando conexão entre VNet e Storage Account.

Legenda: \*Figura 13.7 - 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!**

## 13.6 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\_Cap13\_08\_data\_lake\_acl.png`

**Descrição:** Interface de gerenciamento de ACL mostrando usuário adicionado com permissões r-x.

Legenda: \*Figura 13.8 - Data Lake ACL (permissões granulares)\*

**Maria agora pode:**

• Ler arquivos em bronze/

• Não pode modificar/deletar

## 13.7 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!**

## 13.8 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\_Cap13\_09\_data\_masking.png`

**Descrição:** Configuração de Dynamic Data Masking mostrando coluna EmailAddress com função Email masking.

Legenda: \*Figura 13.9 - 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 |

## 13.9 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\_Cap13\_10\_diagnostic\_settings.png`

Descrição: Configuração de Diagnostic settings mostrando logs selecionados, destino (Log Analytics).

Legenda: \*Figura 13.10 - 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\_Cap13\_11\_log\_analytics\_query.png`

Descrição: Log Analytics mostrando query KQL e resultados (logs de Storage).

Legenda: \*Figura 13.11 - 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!**

## 13.10 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\_Cap13\_12\_purview\_account.png`

**Descrição:** Overview do Microsoft Purview account criado.

Legenda: \*Figura 13.12 - 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/Synapse).

**Exemplo de lineage:**

|  |
| --- |
| CODE |
| 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\_Cap13\_13\_purview\_lineage.png`

Descrição: Purview mostrando data lineage visual (fluxo de dados entre fontes).

Legenda: \*Figura 13.13 - Data Lineage (rastreabilidade)\*

## 13.11 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!**

## 13.12 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)

## 13.13 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)

## 13.14 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/Synapse

**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!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Volume 1: JUNIOR

# Capítulo 14: 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)

## 14.1 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

## 14.2 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

## 14.3 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\_Cap14\_01\_adf\_diagnostic\_settings.png`

**Descrição:** Configuração de Diagnostic Settings do ADF mostrando logs e métricas selecionados.

**Legenda:** \*Figura 14.1 - 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\_Cap14\_02\_adf\_metrics\_chart.png`

**Descrição:** Gráfico de métricas mostrando PipelineFailedRuns ao longo dos últimos 7 dias.

Legenda: \*Figura 14.2 - 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\_Cap14\_03\_adf\_failed\_pipelines\_query.png`

Descrição: Log Analytics mostrando query KQL e resultados (pipelines falhados).

**Legenda:** \*Figura 14.3 - 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 |

## 14.4 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\_Cap14\_04\_alert\_rule\_adf.png`

Descrição: Configuração de Alert Rule mostrando condição (PipelineFailedRuns > 0), action group (email).

Legenda: \*Figura 14.4 - 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!**

## 14.5 Monitorar Synapse

### Cluster Metrics

**Synapse Workspace → Compute → Cluster → Metrics:**

**Métricas disponíveis:**

• CPU Utilization

• Memory Usage

• Network I/O

• Disk I/O

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap14\_05\_synapse\_spark\_metrics.png`

**Descrição:** Métricas do cluster Synapse mostrando gráficos de CPU, memória, I/O.

**Legenda:** \*Figura 14.5 - Synapse 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\_Cap14\_06\_spark\_ui\_jobs.png`

Descrição: Spark UI mostrando lista de jobs, duração, stages, tasks.

Legenda: \*Figura 14.6 - Spark UI (monitoramento de jobs)\*

### Synapse Logs (Log Analytics)

**Synapse 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

## 14.6 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\_Cap14\_07\_sql\_database\_metrics.png`

**Descrição:** Gráfico de métricas SQL Database mostrando DTU%, CPU%, IO%.

**Legenda:** \*Figura 14.7 - 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\_Cap14\_08\_query\_performance\_insight.png`

**Descrição:** Query Performance Insight mostrando top queries por CPU, gráficos de duração.

**Legenda:** \*Figura 14.8 - 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!**

## 14.7 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\_Cap14\_09\_storage\_metrics.png`

**Descrição:** Métricas de Storage Account mostrando Transactions, Ingress, Egress.

**Legenda:** \*Figura 14.9 - 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 |

## 14.8 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\_Cap14\_10\_azure\_dashboard.png`

**Descrição:** Dashboard customizado mostrando 4 tiles com métricas de ADF, SQL, Storage.

**Legenda:** \*Figura 14.10 - Dashboard customizado\*

### Compartilhar Dashboard

**Dashboard → Share:**

• **Publish to Dashboard:** Yes

• **Access control:** Adicionar usuários/grupos

**Agora equipe toda vê dashboard!**

## 14.9 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:**

|  |
| --- |
| CODE |
| User Request (Power BI)  ↓ [request\_id: abc123] ADF Pipeline  ↓ [request\_id: abc123] Synapse Notebook  ↓ [request\_id: abc123] SQL Database |

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

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

## 14.10 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, Synapse, SQL, Storage)

• **By resource:** Detalhamento por recurso específico

• **Forecast:** Projeção de gastos futuros

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap14\_11\_cost\_analysis.png`

Descrição: Cost Analysis mostrando gráfico de custos por serviço (últimos 30 dias).

**Legenda:** \*Figura 14.11 - 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!**

## 14.11 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 Synapse Connection",  "type": "SynapseNotebook",  "linkedServiceName": "LS\_Synapse",  "typeProperties": {  "notebookPath": "/health\_check",  "baseParameters": {  "test": "connectivity"  }  }  }  ] } |

**Agendar:** Daily 7:00 AM

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

## 14.12 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 Synapse): 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 |

## 14.13 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 Synapse (Cluster Metrics, Spark UI)

• [ ] Usar Query Performance Insight (SQL)

• [ ] Criar dashboard customizado

• [ ] Analisar custos (Cost Management)

• [ ] Criar health check pipeline

## 14.14 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)

## 14.15 Exercícios Práticos

**Exercício 1: Monitoramento Completo**

1. Habilitar Diagnostic Settings para:

- ADF

- SQL Database

- Storage Account

- Synapse

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 Synapse, 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

• Synapse: 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!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Volume 1: JUNIOR

# Capítulo 15: 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 Synapse

• Criar ambientes (Dev, Staging, Prod)

• Implementar testes automatizados

• Rollback de deployments

**Tempo estimado:** 5-6 horas

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

## 15.1 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

## 15.2 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

|  |
| --- |
| CODE |
| 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

## 15.3 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\_Cap15\_01\_azure\_devops\_repo.png`

Descrição: Azure DevOps mostrando repositório criado, estrutura de pastas inicial.

**Legenda:** \*Figura 15.1 - 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\_Cap15\_02\_adf\_git\_config.png`

Descrição: Configuração Git no ADF mostrando campos preenchidos (Azure DevOps, repository, branches).

**Legenda:** \*Figura 15.2 - ADF Git Configuration\*

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

### Estrutura do Repositório ADF

|  |
| --- |
| CODE |
| 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!**

## 15.4 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\_Cap15\_03\_pull\_request.png`

Descrição: Pull Request no Azure DevOps mostrando diff de arquivos JSON (pipeline criado).

Legenda: \*Figura 15.3 - 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`!**

## 15.5 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:**

|  |
| --- |
| CODE |
| 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!**

## 15.6 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' |

5. **Save and run**

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

**![INSERIR\_IMAGEM]**

**Nome do arquivo:** `Junior\_Cap15\_04\_ci\_pipeline.png`

Descrição: Azure Pipelines mostrando build CI executando, stages (Validate), status verde.

Legenda: \*Figura 15.4 - 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\_Cap15\_05\_cd\_pipeline.png`

Descrição: Release Pipeline mostrando 2 stages (Dev, Prod), aprovação manual entre eles.

Legenda: \*Figura 15.5 - CD Pipeline (deploy automático)\*

**Fluxo completo:**

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

## 15.7 Synapse + Git

### Repos (Synapse Git Integration)

**Synapse Workspace → Repos → Add Repo:**

1. **Git provider:** GitHub / Azure DevOps

2. Git repository URL: https://dev.azure.com/azuredejunior/data-platform/\_git/synapse-notebooks

3. **Clone**

**Estrutura:**

|  |
| --- |
| CODE |
| synapse-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\_Cap15\_06\_synapse\_repos.png`

Descrição: Synapse Repos mostrando estrutura de pastas, notebooks versionados.

Legenda: \*Figura 15.6 - Synapse Repos (Git integration)\*

### Workflow Synapse

**1. Criar branch:**

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

**2. Editar notebooks:**

• Mudanças salvas localmente (Synapse)

**3. Commit:**

• Repos → Git → Commit & Push

4. Pull Request (Azure DevOps):

• Mesmos passos que ADF

**5. Merge → Deploy:**

• CD Pipeline executa notebooks via Synapse CLI

## 15.8 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!**

## 15.9 Testes Automatizados

### Unit Tests (Synapse)

**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 (Synapse)

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 |

## 15.10 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**

## 15.11 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 Synapse notebooks (Repos)

• [ ] Parametrizar pipelines (Dev vs Prod)

• [ ] Criar testes automatizados

• [ ] Fazer rollback de deploy

## 15.12 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/)

• [Synapse Repos](https://learn.microsoft.com/azure/synapse/repos/)

**Best Practices:**

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

## 15.13 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: Synapse Versioning**

1. Conectar Synapse Repos (Git)

2. Criar notebook transformação

3. Criar branch, editar, commit

4. Criar PR, merge

5. Deploy via CD Pipeline (Synapse 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

**Synapse:**

• Repos (Git integration)

• Versioning de notebooks

**Testes:**

• Unit tests (Synapse)

• Integration tests (ADF)

**Rollback:** Estratégias para reverter deploys

**Próximo capítulo:** Projeto Final Integrado!

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

# Volume 1: JUNIOR

# Capítulo 16: 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)

## 16.1 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 (Synapse)

• Criar Data Warehouse (Synapse)

• Dashboards executivos (Power BI)

• Atualização automática (diária)

### Arquitetura Alvo

|  |
| --- |
| CODE |
| 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    Synapse (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)    Synapse (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 |

## 16.2 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`

• Synapse 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 |

## 16.3 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 |

## 16.4 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:**

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

**Trigger:**

**Schedule Trigger:**

• Name: `TR\_Daily\_Bronze`

• Recurrence: Daily, 01:00 AM (UTC-3)

## 16.5 Fase 4: Pipeline Silver (Transformação)

### Notebook Synapse: `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 = mssparkutils.notebook.get("execution\_date") bronze\_path = f"abfss://datalake@storage.dfs.core.windows.net/bronze" silver\_path = f"abfss://datalake@storage.dfs.core.windows.net/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()}")  mssparkutils.notebook.exit("SUCCESS") |

**Pipeline ADF: `PL\_Transform\_Silver`**

**Activity:**

• **Synapse Notebook:**

◦ Notebook path: `/bronze\_to\_silver`

◦ Base parameters: `{"execution\_date": "@formatDateTime(utcnow(), 'yyyy-MM-dd')"}`

**Dependency:**

• Runs after `PL\_Ingest\_Bronze` succeeds

## 16.6 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; |

## 16.7 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

## 16.8 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`

• `synapse-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.**

## 16.9 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)

• Synapse: Cluster utilization

• Synapse: Query performance

• Storage: Transactions

• Cost: Daily spend

## 16.10 Fase 9: CI/CD

### Git Repository Structure

|  |
| --- |
| CODE |
| techstore-data-platform/  infrastructure/  arm-templates/  storage.json  adf.json  synapse.json  synapse.json  adf/  pipeline/  dataset/  linkedService/  synapse/  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 synapse/tests/  displayName: 'Run Unit Tests' |

### CD Pipeline (Releases)

**Stages:**

**1. Deploy to Dev:**

• Deploy ARM templates (dev parameters)

• Deploy ADF pipelines

• Deploy Synapse notebooks

• Run smoke tests

2. Deploy to Prod (manual approval):

• Deploy ARM templates (prod parameters)

• Deploy ADF pipelines

• Deploy Synapse notebooks

• Health check pipeline

## 16.11 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)

## 16.12 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 Synapse (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 Synapse | 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

|  |
| --- |
| CODE |
| 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, Synapse, Synapse Analytics, Power BI, Python, Spark, SQL   Código completo no GitHub: [link]  #DataEngineering #Azure #Synapse #PowerBI #BigData |

## 16.13 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, Synapse)

• [ ] 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

## 16.14 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 (Synapse, 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 Synapse

• Delta Lake

• Azure Synapse Analytics

• Power BI

• Segurança Azure

• Monitoramento

• CI/CD

**Próximo passo:** Construir projetos reais, aplicar em empresas, continuar aprendendo (Volume 2 - Pleno)!

**Tempo gasto neste capítulo:** \_\_\_\_\_ horas

**Data de conclusão:** \_\_\_/\_\_\_/\_\_\_\_\_\_

**Projeto publicado:** [ ] GitHub [ ] LinkedIn

**Autor:** Ronaldo Santana Ramires

**Versão:** 1.0

**Data:** 09/02/2026

## Agradecimentos

Obrigado por completar esta jornada! Continue praticando, construindo projetos e compartilhando conhecimento com a comunidade.

**Sucesso na sua carreira de Data Engineering!**