

O'REILLY®

Fundamentos De dados Engenharia

Planejar e Construir

Sistemas de Dados Robustos



joe reis
Matt Housle

Fundamentos da Engenharia de Dados

A engenharia de dados cresceu rapidamente na última década, deixando muitos engenheiros de software, cientistas de dados e analistas em busca de uma visão abrangente dessa prática. Com este livro prático, você aprenderá como planejar e construir sistemas para atender às necessidades de sua organização e clientes, avaliando as melhores tecnologias disponíveis por meio da estrutura do ciclo de vida da engenharia de dados.

Os autores Joe Reis e Matt Housley conduzem você pelo ciclo de vida da engenharia de dados e mostram como unir uma variedade de tecnologias de nuvem para atender às necessidades dos consumidores de dados downstream. Você entenderá como aplicar os conceitos de geração, ingestão, orquestração, transformação, armazenamento e governança de dados que são essenciais em qualquer ambiente de dados, independentemente da tecnologia subjacente.

Este livro irá ajudá-lo a:

- Obtenha uma visão geral concisa de todo o cenário da engenharia de dados
- Avalie os problemas de engenharia de dados usando uma estrutura de ponta a ponta das melhores práticas
- Elimine o hype de marketing ao escolher tecnologias, arquitetura e processos de dados
- Use o ciclo de vida da engenharia de dados para projetar e construir uma arquitetura robusta
- Incorpore governança e segurança de dados em todo o ciclo de vida da engenharia de dados

joe reisé um “cientista de dados em recuperação” e engenheiro e arquiteto de dados.

Matt Housleyé consultor de engenharia de dados e especialista em nuvem.

DADOS

US \$ 69,99 PODE US\$ 87,99

ISBN: 978-1-098-10830-4



9 781098 108304

“O mundo dos dados tem sido evoluindo por um tempo agora. Primeiro foram os designers. Em seguida, administradores de banco de dados. Em seguida, CIOs. então dados arquitetos. Este livro sinaliza o próximo passo na evolução e maturidade da indústria. É uma leitura obrigatória para quem leva sua profissão e carreira honestamente.”

— Bill Inmon
criador do data warehouse

“Fundamentos de dados Engenhariaé uma ótima introdução ao negócio de mover, processar e manipular dados. Eu o recomendo fortemente para qualquer pessoa que queira se atualizar em engenharia de dados ou analytics, ou para profissionais existentes que desejam preencher quaisquer lacunas em seu entendimento”.

— Jordan Tigani
fundador e CEO, MotherDuck,
e engenheiro fundador e
co-criador do BigQuery

Twitter: @oreillymedia
[linkedin.com/company/oreilly-media](https://www.linkedin.com/company/oreilly-media)
[youtube.com/oreillymedia](https://www.youtube.com/oreillymedia)

Louvor para *Fundamentos da Engenharia de Dados*

O mundo dos dados vem evoluindo há algum tempo. Primeiro foram os designers. Em seguida, administradores de banco de dados. Em seguida, CIOs. Em seguida, arquitetos de dados. Este livro sinaliza o próximo passo na evolução e maturidade da indústria. É uma leitura obrigatória para quem leva a profissão e carreira honestamente.

— Bill Inmon, criador do *data warehouse*

Fundamentos da Engenharia de Dados é uma ótima introdução ao negócio de mudanças, processamento e tratamento de dados. Ele explica a taxonomia dos conceitos de dados, sem focar muito em ferramentas ou fornecedores individuais, de modo que as técnicas e ideias devem durar mais que qualquer tendência ou produto individual. Eu o recomendo fortemente para qualquer pessoa que queira se atualizar em engenharia ou análise de dados ou para profissionais existentes que desejam preencher quaisquer lacunas em sua compreensão.

— Jordan Tigani, fundador e CEO da MotherDuck e fundador engenheiro e cocriador do BigQuery

Se você deseja liderar em seu setor, deve desenvolver os recursos necessários para fornecer experiências excepcionais a clientes e funcionários. Este não é apenas um problema de tecnologia. É uma oportunidade de pessoas. E isso vai transformar o seu negócio. Os engenheiros de dados estão no centro dessa transformação. Mas hoje a disciplina é mal compreendida. Este livro vai desmistificar a engenharia de dados e tornar-se seu melhor guia para ter sucesso com os dados.

— Bruno Aziza, chefe de análise de dados, Google Cloud

Que livro! Joe e Matt estão dando a você a resposta para a pergunta: "O que devo entender para fazer engenharia de dados?" Se você está começando como engenheiro de dados ou fortalecendo suas habilidades, você não está procurando mais um manual de tecnologia. Você estão buscando aprender mais sobre os princípios subjacentes e os conceitos centrais da função, suas responsabilidades, seu ambiente técnico e organizacional, sua missão - isso é exatamente o que Joe e Matt oferecem neste livro.

— Andy Petrella, fundador da Kensu

Este é o livro que faltava na engenharia de dados. Um relato maravilhosamente completo do que é preciso para ser um bom engenheiro de dados prático, incluindo considerações ponderadas da vida real.

Eu recomendaria que toda a educação futura de profissionais de dados incluisse o trabalho de Joe e Matt.

— Sarah Krasnik, líder de engenharia de dados

É incrível perceber a amplitude de conhecimento que um engenheiro de dados deve ter. Mas não deixe que isso te assuste. Este livro fornece uma grande visão geral fundamental de várias arquiteturas, abordagens, metodologias e padrões que qualquer pessoa que trabalhe com dados precisa conhecer. Mas o que é ainda mais valioso é que este livro está cheio de pepitas de ouro de sabedoria, conselhos de melhores práticas e coisas a considerar ao tomar decisões relacionadas a engenharia de dados. É uma leitura obrigatória para engenheiros de dados experientes e novos.

— Veronika Durgin, líder de dados e análises

Tive a honra e a humildade de receber o pedido de Joe e Matt para ajudar na revisão técnica de sua obra-prima de conhecimento de dados, *Fundamentos da Engenharia de Dados*. Sua capacidade de dividir os principais componentes que são críticos para qualquer pessoa que queira mudar para uma empresa de dados papel de engenharia é incomparável. Seu estilo de escrita facilita a absorção das informações e eles não deixam pedra sobre pedra. Foi um prazer absoluto trabalhar com alguns dos melhores líderes de pensamento no espaço de dados. Mal posso esperar para ver o que eles farão a seguir.

— Chris Tabb, cofundador da LEIT DATA

Fundamentos da Engenharia de Dados é o primeiro livro a ter uma visão aprofundada e holística nos requisitos do engenheiro de dados de hoje. Como você verá, o livro mergulha nas áreas críticas da engenharia de dados, incluindo conjuntos de habilidades, ferramentas e arquiteturas usadas para gerencie, move e organize dados nos complexos ambientes técnicos atuais.

Mais importante, Joe e Matt transmitem seu mestre em entender a engenharia de dados e reserve um tempo para mergulhar ainda mais nas áreas mais sutis da engenharia de dados e torná-las relacionáveis ao leitor. Seja você um gerente, engenheiro de dados experiente ou alguém querendo entrar no espaço, este livro fornece uma visão prática sobre a atualidade panorama da engenharia de dados.

— Jon King, principal arquiteto de dados

Duas coisas permanecerão relevantes para os engenheiros de dados em 2042: SQL e este livro. Joe e Matt cortaram o hype em torno das ferramentas para extrair as dimensões que mudam lentamente de nossa disciplina. Esteja você iniciando sua jornada com dados ou adicionando listras ao seu preto cinto, *Fundamentos da Engenharia de Dados* estabelece as bases para a maestria.

— Kevin Hu, CEO da Metaplane

Em um campo que está mudando rapidamente, com novas soluções de tecnologia surgindo constantemente, Joe e Matt fornecem orientação clara e atemporal, concentrando-se nos principais conceitos e conhecimentos fundamentais necessários para se destacar como engenheiro de dados. Este livro está repleto de informações que o capacitarão a fazer as perguntas certas, entendendo os trade-offs e tome as melhores decisões ao projetar sua arquitetura de dados e implementação de soluções em todo o ciclo de vida da engenharia de dados. Se você é apenas pensando em se tornar um engenheiro de dados ou já está no campo há anos, Garanto que você aprenderá algo com este livro!

— Julie Price, gerente sênior de produtos, SingleStore

Fundamentos da Engenharia de Dados não é apenas um manual de instruções - ele ensina como pense como um engenheiro de dados. Parte aula de história, parte teoria e parte conhecimento adquirido das décadas de experiência de Joe e Matt, o livro definitivamente ganhou seu lugar na estante de todos os profissionais de dados.

— Scott Breitenother, fundador e CEO, Brooklyn Data Co.

Não há outro livro que aborde de forma tão abrangente o que significa ser um engenheiro de dados. Joe e Matt se aprofundam nas responsabilidades, impactos, escolhas arquitetônicas e muito mais. Apesar de tratar de temas tão complexos, o livro é de fácil leitura e digerir. Uma combinação muito poderosa.

— Danny Leybzon, arquiteto MLOps

Eu gostaria que este livro existisse anos atrás, quando comecei a trabalhar com engenheiros de dados. O ampla cobertura do campo torna claros os papéis envolvidos e cria empatia com os muitas funções necessárias para construir uma disciplina de dados competente.

— Tod Hansmann, vice-presidente de engenharia

Uma leitura obrigatória e um clássico instantâneo para qualquer pessoa no campo da engenharia de dados. Este livro preenche uma lacuna na base de conhecimento atual, discutindo tópicos fundamentais não encontrados em outros livros. Você obterá compreensão dos conceitos fundamentais e insights sobre a história contexto sobre engenharia de dados que preparará um para o sucesso.

— Matthew Sharp, engenheiro de dados e ML

A engenharia de dados é a base de toda análise, modelo de aprendizado de máquina e dados produto, por isso é fundamental que seja bem feito. Existem inúmeros manuais, livros e referências para cada uma das tecnologias usadas pelos engenheiros de dados, mas muito poucos (se houver) recursos que fornecem uma visão holística do que significa trabalhar como engenheiro de dados. Este livro atende a uma necessidade crítica do setor e o faz bem, estabelecendo as bases para que engenheiros de dados novos e em atividade sejam bem-sucedidos e eficazes em suas funções. este é o livro que recomendarei a todos que desejam trabalhar com dados em qualquer nível.

— Tobias Macey, apresentador do podcast de engenharia de dados

Fundamentos da Engenharia de Dados

Planeje e construa sistemas de dados robustos

Joe Reis e Matt Housley

Pequim • Boston • Farnham • Sebastopol • Tóquio

O'REILLY®

Fundamentos da Engenharia de Dados

por Joe Reis e Matt Housley

Copyright © 2022 Joseph Reis e Matthew Housley. Todos os direitos reservados.

Impresso nos Estados Unidos da América.

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Os livros da O'Reilly podem ser adquiridos para uso educacional, comercial ou promocional de vendas. Edições online também estão disponíveis para a maioria dos títulos (<http://oreilly.com>). Para mais informações, entre em contato com nosso departamento de vendas corporativo/institucional: 800-998-9938 ou corporate@oreilly.com.

Editor de Aquisições:Jéssica Haberman

Editor de Desenvolvimento:Michele Cronin

Editor de Produção:Gregory Hyman

Editor de cópia:Sharon Wilkey

Revisor:Amnet Systems, LLC

Indexador:Judith McConville

Designer de interiores:David Futato

Designer de capa:Karen Montgomery

Ilustrador:Kate Dullea

julho de 2022:

Primeira edição

Histórico de revisão para a primeira edição

2022-06-22: Primeiro lançamento

Ver <http://oreilly.com/catalog/errata.csp?isbn=9781098108304>para detalhes do lançamento.

O logotipo da O'Reilly é uma marca registrada da O'Reilly Media, Inc.*Fundamentos da Engenharia de Dados*, a imagem da capa e a identidade visual relacionada são marcas registradas da O'Reilly Media, Inc.

As opiniões expressas neste trabalho são das autores e não representam as opiniões do editor. Embora a editora e os autores tenham feito esforços de boa fé para garantir que as informações e instruções contidas neste trabalho sejam precisas, a editora e os autores se isentam de toda responsabilidade por erros ou omissões, incluindo, sem limitação, responsabilidade por danos resultantes do uso ou confiança neste trabalho. O uso das informações e instruções contidas neste trabalho é por sua conta e risco. Se qualquer amostra de código ou outra tecnologia que este trabalho contém ou descreve estiver sujeita a licenças de código aberto ou direitos de propriedade intelectual de terceiros, é sua responsabilidade garantir que seu uso esteja em conformidade com tais licenças e/ou direitos.

978-1-098-10830-4

[LSI]

Índice

Prefácio.....	xiii
----------------------	-------------

Parte I. Fundação e blocos de construção

1. Engenharia de dados descrita.....	3
O que é Engenharia de Dados?	3
Definição de engenharia de dados A	4
evolução do ciclo de vida da engenharia de	5
dados do engenheiro de dados	6
Habilidades e atividades de engenharia de	11
dados e ciência de dados	13
Maturidade de dados e o engenheiro de dados O histórico	13
e as habilidades de um engenheiro de dados	17
Responsabilidades de negócios	18
Responsabilidades Técnicas	19
O continuum de funções de engenharia de dados, de A a B	21
Engenheiros de dados dentro de uma organização	22
Engenheiros de dados internos versus externos	23
Engenheiros de dados e outras funções técnicas	24
Conclusão dos engenheiros de dados e da liderança	28
empresarial	31
Recursos adicionais	32
2.0 ciclo de vida da engenharia de dados.....	33
O que é o ciclo de vida da engenharia de dados?	33
O ciclo de vida dos dados versus a geração do ciclo de vida da	34
engenharia de dados: sistemas de origem	35

Armazenar	38
Ingestão	39
Transformação	43
Servindo dados	44
Principais subcorrentes em todo o ciclo de vida da engenharia de dados	48
Segurança	49
Gestão de dados	50
DataOps	59
Arquitetura de dados	64
Orquestração	64
Engenharia de software	66
Conclusão	68
Recursos adicionais	69
3. Projetando uma boa arquitetura de dados.	71
O que é arquitetura de dados?	71
Arquitetura Corporativa Definida	72
Arquitetura de Dados Definida “Boa”	75
Arquitetura de Dados Princípios da Boa	76
Arquitetura de Dados	77
Princípio 1: Escolha sabiamente os componentes comuns	78
Princípio 2: Planeje para o fracasso	79
Princípio 3: Arquitetar para escalabilidade	80
Princípio 4: Arquitetura é liderança Princípio 5:	80
Sempre estar arquitetando Princípio 6: Construir sistemas fracamente acoplados Princípio 7:	81
Tomar decisões reversíveis Princípio 8: Priorizar a segurança	83
Princípio 9: Adote os principais conceitos de arquitetura de FinOps	84
Domínios e Serviços	87
Sistemas distribuídos, escalabilidade e design para falhas Acoplamento rígido versus flexível: camadas, monólitos e microserviços Acesso do usuário: único versus multilocatário	88
Arquitetura Orientada a Eventos	90
Brownfield versus Greenfield Exemplos e	94
Tipos de Arquitetura de Dados	95
Arrozal de dados	96
Data Lake	101
Convergência, data lakes de próxima geração e a pilha de dados moderna da plataforma de dados	102
Arquitetura lambda	103

Arquitetura Kappa	105
O modelo de fluxo de dados e a arquitetura unificada de lote e streaming para IoT	105
Malha de dados	106
Outros exemplos de arquitetura de dados	109
Quem está envolvido no projeto de uma arquitetura de dados?	111
Conclusão	111
Recursos adicionais	111
4. Escolhendo tecnologias ao longo do ciclo de vida da engenharia de dados.....	115
Tamanho da equipe e capacidades	116
Velocidade para o mercado	117
Interoperabilidade	117
Otimização de custos e valor comercial	118
Custo Total de Propriedade	118
Custo Total de Oportunidade de Propriedade	119
FinOps	120
Hoje versus futuro: tecnologias imutáveis versus transitórias	120
Nosso conselho	122
Localização	123
No local	123
Nuvem	124
nuvem híbrida	127
Multicloud	128
Descentralizado: Blockchain e o Edge Nosso	129
Conselho	129
Argumentos de repatriação na nuvem	130
Construir versus comprar	132
Software livre	133
Jardins murados proprietários	137
Nosso conselho	138
Monólito Versus Modular	139
Monólito	139
Modularidade	140
O Padrão Monolítico Distribuído	142
Nosso Conselho	142
Sem servidor versus servidores	143
sem servidor	143
Containers	144
Como avaliar o servidor versus sem servidor	145
Nosso conselho	146
Otimização, desempenho e as guerras de benchmark	147

Big Data... para a década de 1990	148
Comparações de custos absurdas	148
Advertência de otimização assimétrica	148
Emptor	149
Subcorrentes e seus impactos na escolha de tecnologias	149
Gestão de dados	149
DataOps	149
Arquitetura de dados	150
Exemplo de Orquestração:	150
Engenharia de Software Airflow	151
Conclusão	151
Recursos adicionais	151

Parte II.O ciclo de vida da engenharia de dados em profundidade

5.Geração de Dados em Sistemas de Origem.	155
Fontes de dados: como os dados são criados?	156
Sistemas de Origem: Ideias Principais	156
Arquivos e APIs de dados não estruturados	156
Bancos de dados de aplicativos (Sistemas OLTP)	157
Sistema de processamento analítico on-line	159
Captura de dados alterados	159
Histórico	160
Registros do banco de dados	161
CRUD	162
Somente Inserção	162
Mensagens e fluxos	163
Tipos de Tempo	164
Detalhes práticos do sistema de origem	165
bancos de dados	166
APIs	174
Compartilhamento de dados	176
Fontes de dados de terceiros	177
Filas de mensagens e plataformas de transmissão de eventos com quem você trabalhará	177
Subcorrentes e seu impacto nos sistemas de origem	181
Segurança	183
Gestão de dados	184
DataOps	184
Arquitetura de dados	185

Orquestração	186
Engenharia de software	187
Conclusão	187
Recursos adicionais	188
6. Armazenar	189
Ingredientes brutos de armazenamento de dados	191
Unidade de disco magnético	191
Disco de Estado Sólido	193
Memória de acesso aleatório	194
Rede e CPU	195
Serialização	195
Compressão	196
Cache	197
Sistemas de armazenamento de dados	197
Máquina única versus armazenamento	198
distribuído Eventual versus consistência forte	198
Armazenamento de arquivo	199
Armazenamento em bloco	202
Armazenamento de objetos	205
Cache e sistemas de armazenamento baseados em	211
memória O Hadoop Distributed File System	211
Streaming Storage	212
Índices, particionamento e clustering Engenharia	213
de dados Abstrações de armazenamento	215
O Armazém de Dados	215
o lago de dados	216
A Casa do Lago de Dados	216
Plataformas de dados	217
Arquitetura de armazenamento de fluxo para lote	217
Grandes ideias e tendências em armazenamento	218
Catálogo de Dados	218
Compartilhamento de dados	219
Esquema	219
Separação entre computação e armazenamento Ciclo de vida do	220
armazenamento de dados e retenção de dados Armazenamento de	223
locatário único versus armazenamento de vários locatários Com	226
quem você trabalhará	227
Subcorrentes	228
Segurança	228
Gestão de dados	228
DataOps	229

Arquitetura de dados	230
Orquestração	230
Engenharia de software	230
Conclusão	230
Recursos adicionais	231
7. Ingestão.	233
O que é ingestão de dados?	234
Principais considerações de engenharia para a fase de ingestão	235
Frequência de dados limitada versus	236
ilimitada	237
Serialização e desserialização de ingestão	238
síncrona versus assíncrona	239
Taxa de transferência e escalabilidade	239
Confiabilidade e durabilidade Carga	240
útil	241
Padrões Push versus Pull versus Poll	244
Considerações sobre ingestão de lote	244
Captura Instantânea ou Extração Diferencial de	246
Exportação e Ingestão Baseada em Arquivo ETL	246
Versus ELT	246
Migração de dados de inserções, atualizações e	247
tamanho de lote	247
Considerações sobre a ingestão de mensagens e streams	248
Evolução do Esquema	248
Dados de Chegada Atrasada	248
Repetição de pedido e entrega	248
múltipla	249
Tempo de Viver	249
Tamanho da mensagem	249
Tratamento de erros e filas de mensagens mortas	249
Pull and Push do consumidor	250
Localização	250
Formas de ingerir dados	250
Captura de dados de alteração de conexão	251
direta com o banco de dados	252
APIs	254
Filas de mensagens e plataformas de streaming de eventos	255
Conectores de dados gerenciados	256
Movendo Dados com Object Storage	257
EDI	257
Bancos de dados e exportação de arquivos	257

Problemas práticos com formatos de arquivo comuns	258
Shell	258
SSH	259
SFTP e SCP	259
Webhooks	259
Interface web	260
Raspagem da web	260
Dispositivos de transferência para compartilhamento de dados de migração de dados	261
Com quem você trabalhará	262
Partes interessadas a montante	262
Subcorrentes das partes interessadas a jusante	263
Segurança	264
Gestão de dados	264
DataOps	266
Orquestração	268
Engenharia de software	268
Conclusão	268
Recursos adicionais	269
8.Consultas, Modelagem e Transformação.	271
Consultas	272
O que é uma consulta? A Vida de uma Consulta O	273
Otimizador de Consultas	274
Melhorando o desempenho da consulta	275
Consultas em dados de streaming	281
Modelagem de dados	287
O que é um modelo de dados?	288
Normalização de modelos de dados conceituais, lógicos e físicos	289
Técnicas para modelagem de dados analíticos em lote	290
Modelagem de dados de streaming	294
Transformações	307
Transformações em Lote	309
Visões materializadas, federação e virtualização de consulta	310
Transformações e processamento de streaming	323
Com quem você trabalhará	326
Partes interessadas a montante	329
Subcorrentes das partes interessadas a jusante	330

Segurança	330
Gestão de dados	331
DataOps	332
Arquitetura de dados	333
Orquestração	333
Engenharia de software	333
Conclusão	334
Recursos adicionais	335
9. Servindo dados para análise, aprendizado de máquina e ETL reverso.	337
Considerações Gerais para Servir Dados	338
Confiar	338
Qual é o caso de uso e quem é o usuário?	339
Produtos de dados	340
Autoatendimento ou não?	341
Definições de dados e malha de	342
dados lógicos	343
Análise	344
Analista de negócios	344
Análise Operacional	346
Análise incorporada	348
Aprendizado de máquina	349
O que um engenheiro de dados deve saber sobre	350
maneiras de ML de fornecer dados para análise e ML	351
troca de arquivos	351
banços de dados	352
Sistemas de streaming	354
Consultar Federação	354
Compartilhamento de dados	355
Camadas Semânticas e Métricas	355
Servindo Dados em Notebooks	356
ETL Reverso	358
Com quem você trabalhará	360
Undercurrents	360
Segurança	361
Gestão de dados	362
DataOps	362
Arquitetura de dados	363
Orquestração	363
Engenharia de software	364
Conclusão	365
Recursos adicionais	365

Parte III.Segurança, privacidade e o futuro da engenharia de dados

10.Segurança e privacidade.....	369
Pessoas	370
O Poder do Pensamento Negativo	370
Seja Sempre Paranóico	370
Processos	371
Teatro de segurança versus hábito de segurança	371
Segurança ativa	371
O Princípio do Menor Privilégio	372
Responsabilidade Compartilhada na Nuvem	372
Sempre Faça Backup de Seus Dados	372
Um exemplo de tecnologia de política	373
de segurança	374
Corrija e atualize a criptografia	374
de sistemas	375
Registro, monitoramento e alerta de	375
acesso à rede	376
Conclusão de segurança para engenharia de dados	377
de baixo nível	378
Recursos adicionais	378
11.O Futuro da Engenharia de Dados.....	379
O ciclo de vida da engenharia de dados não vai acabar	380
O declínio da complexidade e o surgimento de ferramentas de dados fáceis de usar O	380
sistema operacional de dados em escala de nuvem e a interoperabilidade aprimorada	381
Engenharia de dados "corporativos"	383
Títulos e responsabilidades irão se transformar...	384
Indo além da pilha de dados moderna, em direção à pilha de dados ao vivo	385
A pilha de dados ao vivo	385
Pipelines de streaming e bancos de dados analíticos em tempo	386
real A fusão de dados com aplicativos	387
O feedback apertado entre aplicativos e dados de	388
matéria escura de ML e o surgimento de... Planilhas?!	388
Conclusão	389
A.Detalhes técnicos de serialização e compactação.....	391
B.Rede em Nuvem.....	399
Índice.....	403

Prefácio

Como surgiu este livro? A origem está profundamente enraizada em nossa jornada da ciência de dados para a engenharia de dados. Muitas vezes nos referimos de brincadeira a nós mesmos como *recuperando cientistas de dados*. Nós dois tivemos a experiência de sermos designados para projetos de ciência de dados, depois lutamos para executar esses projetos devido à falta de fundamentos adequados. Nossa jornada na engenharia de dados começou quando assumimos tarefas de engenharia de dados para construir fundações e infraestrutura.

Com o surgimento da ciência de dados, as empresas investiram generosamente em talentos de ciência de dados, na esperança de colher grandes recompensas. Muitas vezes, os cientistas de dados lutavam com problemas básicos que sua formação e treinamento não abordavam - coleta de dados, limpeza de dados, acesso a dados, transformação de dados e infraestrutura de dados. Esses são problemas que a engenharia de dados visa resolver.

O que este livro não é

Antes de abordarmos o assunto deste livro e o que você obterá dele, vamos abordar rapidamente o que este livro *não é*. Este livro não é sobre engenharia de dados usando uma ferramenta, tecnologia ou plataforma específica. Embora muitos livros excelentes abordem as tecnologias de engenharia de dados a partir dessa perspectiva, esses livros têm vida útil curta. Em vez disso, nos concentramos nos conceitos fundamentais por trás da engenharia de dados.

Sobre o que é este livro

Este livro visa preencher uma lacuna no conteúdo e materiais atuais de engenharia de dados. Embora não haja escassez de recursos técnicos que abordam ferramentas e tecnologias específicas de engenharia de dados, as pessoas lutam para entender como montar esses componentes em um todo coerente que se aplica ao mundo real. Este livro conecta os pontos do ciclo de vida de dados de ponta a ponta. Ele mostra como unir várias tecnologias para atender às necessidades dos consumidores de dados downstream, como analistas, cientistas de dados e engenheiros de aprendizado de máquina. Este livro funciona como um complemento

aos livros da O'Reilly que cobrem os detalhes de tecnologias, plataformas e linguagens de programação específicas.

A grande ideia deste livro é *ciclo de vida da engenharia de dados*: geração de dados, armazenamento, ingestão, transformação e serviço. Desde o surgimento dos dados, vimos a ascensão e queda de inúmeras tecnologias específicas e produtos de fornecedores, mas os estágios do ciclo de vida da engenharia de dados permaneceram essencialmente inalterados. Com essa estrutura, o leitor obterá um bom entendimento da aplicação de tecnologias a problemas de negócios do mundo real.

Nosso objetivo aqui é mapear princípios que perpassam dois eixos. Primeiro, queremos destilar a engenharia de dados em princípios que podem abranger *qualquer tecnologia relevante*. Em segundo lugar, desejamos apresentar princípios que resistirão ao teste de *tempo*. Esperamos que essas ideias reflitam as lições aprendidas durante a revolução da tecnologia de dados dos últimos vinte anos e que nossa estrutura mental permaneça útil por uma década ou mais no futuro.

Uma coisa a observar: nós assumidamente adotamos uma abordagem que prioriza a nuvem. Vemos a nuvem como um desenvolvimento fundamentalmente transformador que perdurará por décadas; a maioria dos sistemas de dados e cargas de trabalho no local acabará migrando para a hospedagem na nuvem.

Assumimos que a infraestrutura e os sistemas são *efêmeros e escaláveis*, e que os engenheiros de dados se inclinarão para a implantação de serviços gerenciados na nuvem. Dito isso, a maioria dos conceitos deste livro serão traduzidos para ambientes fora da nuvem.

Quem deveria ler esse livro

Nosso público-alvo principal para este livro consiste em profissionais técnicos, engenheiros de software de nível médio a sênior, cientistas de dados ou analistas interessados em ingressar na engenharia de dados; ou engenheiros de dados trabalhando com tecnologias específicas, mas querendo desenvolver uma perspectiva mais abrangente. Nosso público-alvo secundário consiste em partes interessadas em dados que trabalham ao lado de profissionais técnicos, por exemplo, um líder de equipe de dados com experiência técnica que supervisiona uma equipe de engenheiros de dados ou um diretor de armazenamento de dados que deseja migrar da tecnologia local para um solução baseada em nuvem.

Idealmente, você é curioso e quer aprender - por que outro motivo você estaria lendo este livro? Você se mantém atualizado com as tecnologias e tendências de dados lendo livros e artigos sobre data warehousing/data lakes, sistemas batch e streaming, orquestração, modelagem, gerenciamento, análise, desenvolvimentos em tecnologias de nuvem etc. leia uma imagem completa da engenharia de dados em tecnologias e paradigmas.

Pré-requisitos

Assumimos bastante familiaridade com os tipos de sistemas de dados encontrados em um ambiente corporativo. Além disso, presumimos que os leitores tenham alguma familiaridade com SQL e Python (ou alguma outra linguagem de programação) e experiência com serviços em nuvem.

Vários recursos estão disponíveis para aspirantes a engenheiros de dados praticarem Python e SQL. Recursos on-line gratuitos são abundantes (postagens de blog, sites de tutoriais, vídeos do YouTube) e muitos novos livros sobre Python são publicados todos os anos.

A nuvem oferece oportunidades sem precedentes para obter experiência prática com ferramentas de dados. Sugerimos que aspirantes a engenheiros de dados configurem contas com serviços em nuvem, como AWS, Azure, Google Cloud Platform, Snowflake, Databricks, etc. Observe que muitas dessas plataformas têm nível gratuito/optionais, mas os leitores devem ficar de olho nos custos e trabalhar com pequenas quantidades de dados e clusters de nó único enquanto estudam.

Desenvolver familiaridade com sistemas de dados corporativos fora de um ambiente corporativo continua sendo difícil, e isso cria certas barreiras para aspirantes a engenheiros de dados que ainda não conseguiram seu primeiro emprego em dados. Este livro pode ajudar. Sugerimos que os novatos em dados leiam as ideias de alto nível e, em seguida, consultem os materiais na seção Recursos adicionais no final de cada capítulo. Em uma segunda leitura, observe quaisquer termos e tecnologias desconhecidos. Você pode utilizar o Google, Wikipedia, postagens de blog, vídeos do YouTube e sites de fornecedores para se familiarizar com novos termos e preencher lacunas em seu entendimento.

O que você aprenderá e como isso melhorará suas habilidades

Este livro visa ajudá-lo a construir uma base sólida para resolver problemas de engenharia de dados do mundo real.

Ao final deste livro você entenderá:

- Como a engenharia de dados afeta sua função atual (cientista de dados, engenheiro de software ou líder de equipe de dados)
- Como superar o hype de marketing e escolher as tecnologias, arquitetura de dados e processos certos
- Como usar o ciclo de vida da engenharia de dados para projetar e construir uma arquitetura robusta
- Melhores práticas para cada estágio do ciclo de vida dos dados

E você será capaz de:

- Incorpore princípios de engenharia de dados em sua função atual (cientista de dados, analista, engenheiro de software, líder de equipe de dados, etc.)
- Unir uma variedade de tecnologias de nuvem para atender às necessidades dos consumidores de dados downstream
- Avaliar problemas de engenharia de dados com uma estrutura de ponta a ponta das melhores práticas
- Incorporar governança e segurança de dados em todo o ciclo de vida da engenharia de dados

Navegando neste livro

Este livro é composto por quatro partes:

- Parte I, "Fundamentos e blocos de construção"
- Parte II, "O Ciclo de Vida da Engenharia de Dados em Profundidade"
- Parte III, "Segurança, Privacidade e o Futuro da Engenharia de Dados"
- ApêndicesAeB: abrangendo serialização e compactação e rede em nuvem, respectivamente

Em Parte I, começamos definindo a engenharia de dados em Capítulo 1, mapeando o ciclo de vida da engenharia de dados em Capítulo 2. Em Capítulo 3, nós discutimos *boa arquitetura*. Em Capítulo 4, apresentamos uma estrutura para escolher a tecnologia certa - embora frequentemente vejamos tecnologia e arquitetura confundidas, na verdade são tópicos muito diferentes.

parte IIConstrói Capítulo 2 cobrir o ciclo de vida da engenharia de dados em profundidade; cada estágio do ciclo de vida — geração, armazenamento, ingestão, transformação e serviço de dados — é abordado em seu próprio capítulo. parte IIé indiscutivelmente o coração do livro, e os outros capítulos existem para apoiar as ideias centrais abordadas aqui.

Parte IIIcobre tópicos adicionais. Em Capítulo 10, nós discutimos *segurança e privacidade*. Embora a segurança sempre tenha sido uma parte importante da profissão de engenharia de dados, ela só se tornou mais crítica com o surgimento de hackers com fins lucrativos e ataques cibernéticos patrocinados pelo estado. E o que podemos dizer da privacidade? A era do niilismo da privacidade corporativa acabou — nenhuma empresa quer ver seu nome aparecer na manchete de um artigo sobre práticas desleixadas de privacidade. O manuseio imprudente de dados pessoais também pode ter ramificações legais significativas com o advento do GDPR, CCPA e outros regulamentos. Em resumo, segurança e privacidade devem ser as principais prioridades em qualquer trabalho de engenharia de dados.

No decorrer do trabalho em engenharia de dados, fazendo pesquisas para este livro e entrevistando vários especialistas, pensamos bastante sobre para onde o campo está indo no curto e no longo prazo. [Capítulo 11](#) descreve nossas ideias altamente especulativas sobre o futuro da engenharia de dados. Por sua natureza, o futuro é uma coisa escorregadia. O tempo dirá se algumas de nossas ideias estão corretas. Gostaríamos muito de ouvir nossos leitores sobre como suas visões do futuro concordam ou diferem das nossas.

Nos apêndices, abordamos alguns tópicos técnicos extremamente relevantes para a prática cotidiana da engenharia de dados, mas que não se encaixam no corpo principal do texto. Especificamente, os engenheiros precisam entender a serialização e a compactação (consulte [Apêndice A](#)) tanto para trabalhar diretamente com arquivos de dados quanto para avaliar considerações de desempenho em sistemas de dados e rede em nuvem (consulte [Apêndice B](#)) é um tópico crítico à medida que a engenharia de dados muda para a nuvem.

Convenções utilizadas neste livro

As seguintes convenções tipográficas são usadas neste livro:

italílico

Indica novos termos, URLs, endereços de e-mail, nomes de arquivo e extensões de arquivo

Largura constante

Usado para listas de programas, bem como dentro de parágrafos para se referir a elementos do programa, como nomes de variáveis ou funções, bancos de dados, tipos de dados, variáveis de ambiente, declarações e palavras-chave



Este elemento significa uma dica ou sugestão.



Este elemento significa uma nota geral.



Este elemento indica um aviso ou cuidado.

Como entrar em contato conosco

Envie comentários e perguntas sobre este livro para o editor:

O'Reilly Media, Inc.
1005 Gravestein Highway North
Sebastopol, CA 95472
800-998-9938 (nos Estados Unidos ou Canadá)
707-829-0515 (internacional ou local)
707-829-0104 (fax)

Temos uma página web para este livro, onde listamos errata, exemplos e qualquer informação adicional. Você pode acessar esta página em <https://oreil.ly/fundamentals-of-data>.

E-mail bookquestions@oreilly.com para comentar ou fazer perguntas técnicas sobre este livro.

Para notícias e informações sobre nossos livros e cursos, visite <https://oreilly.com>.

Encontre-nos no LinkedIn: <https://linkedin.com/company/oreilly-media> Siga-nos no

Twitter: <https://twitter.com/oreillymedia> Assista-nos no YouTube: <https://www.youtube.com/oreillymedia>

Agradecimentos

Quando começamos a escrever este livro, fomos advertidos por muitas pessoas de que enfrentaríamos uma tarefa difícil. Um livro como este tem muitas partes móveis e, devido à sua visão abrangente do campo da engenharia de dados, exigiu uma tonelada de pesquisa, entrevistas, discussões e reflexão profunda. Não afirmaremos ter capturado todas as nuances da engenharia de dados, mas esperamos que os resultados ressoem com você. Inúmeras pessoas contribuíram para nossos esforços e somos gratos pelo apoio que recebemos de muitos especialistas.

Primeiro, graças à nossa incrível equipe de revisores técnicos. Eles se arrastaram por muitas leituras e deram feedback inestimável (e muitas vezes impiedosamente contundente). Este livro seria uma fração de si mesmo sem seus esforços. Em nenhuma ordem específica, agradecemos infinitamente a Bill Inmon, Andy Petrella, Matt Sharp, Tod Hanseman, Chris Tabb, Danny Lebzyon, Martin Kleppman, Scott Lorimor, Nick Schrock, Lisa Steckman, Veronika Durgin e Alex Woolford.

Em segundo lugar, tivemos uma oportunidade única de conversar com os principais especialistas na área de dados em nossos shows ao vivo, podcasts, encontros e inúmeras chamadas privadas. Suas ideias ajudaram a moldar nosso livro. Há muitas pessoas para nomear individualmente, mas gostaríamos de agradecer a Jordan Tigani, Zhamak Dehghani, Ananth Packkildurai,

Shruti Bhat, Eric Tschetter, Benn Stancil, Kevin Hu, Michael Rogove, Ryan Wright, Adi Polak, Shinji Kim, Andreas Kretz, Egor Gryaznov, Chad Sanderson, Julie Price, Matt Turck, Monica Rogati, Mars Lan, Pardhu Gunnam, Brian Suk , Barr Moses, Lior Gavish, Bruno Aziza, Gian Merlino, DeVaris Brown, Todd Beauchene, Tudor Girba, Scott Taylor, Ori Rafael, Lee Edwards, Bryan Offutt, Ollie Hughes, Gilbert Eijkelen- boom, Chris Bergh, Fabiana Clemente, Andreas Kretz , Ori Reshef, Nick Singh, Mark Balkenende, Kenten Danas, Brian Olsen, Lior Gavish, Rhaghu Murthy, Greg Coquillo, David Aponte, Demetrios Brinkmann, Sarah Catanzaro, Michel Tricot, Levi Davis, Ted Walker, Carlos Kemeny, Josh Benamram, Chanin Nantasenamat, George Firican, Jordan Goldmeir, Minhaaj Rehmam, Luigi Patruno, Vin Vashista, Danny Ma, Jesse Anderson, Alessya Visnjic, Vishal Singh, Dave Langer, Roy Hasson, Todd Odess, Che Sharma, Scott Breitenother, Ben Taylor, Thom Ives, John Thompson, Brent Dykes, Josh Tobin, Mark Kosiba, Tyler Pugliese, Douwe Maan, Martin Traverso, Curtis Kowalski, Bob Davis, Koo Ping Shung, Ed Chenard, Matt Sciorma, Tyler Folkman, Jeff Baird, Tejas Manohar, Paul Singman, Kevin Stumpf, Willem Pineaar e Michael Del Balso de Tecton, Emma Dahl, Harpreet Sahota, Ken Jee, Scott Taylor, Kate Strachnyi, Kristen Kehrer, Taylor Miller , Abe Gong, Ben Castleton, Ben Rogojan, David Mertz, Emmanuel Raj, Andrew Jones, Avery Smith, Brock Cooper, Jeff Larson, Jon King, Holden Ackerman, Miriah Peterson, Felipe Hoffa, David Gonzalez, Richard Wellman, Susan Walsh, Ravit Jain, Lauren Balik, Mikiko Bazeley, Mark Freeman, Mike Wimmer, Alexey Shchedrin, Mary Clair Thompson, Julie Burroughs, Jason Pedley, Freddy Drennan, Jason Pedley, Kelly e Matt Phillipps, Brian Campbell, Faris Chebib, Dylan Gregerson, Ken Myers, Jake Carter, Seth Paul, Ethan Aaron e muitos outros.

Se você não for mencionado especificamente, não leve para o lado pessoal. Você sabe quem você é. Deixe-nos saber e nós vamos levá-lo na próxima edição.

Também gostaríamos de agradecer à equipe da Ternary Data (Colleen McAuley, Maike Wells, Patrick Dahl, Aaron Hunsaker e outros), aos nossos alunos e às inúmeras pessoas ao redor do mundo que nos apoiam. É um ótimo lembrete de que o mundo é um lugar muito pequeno.

Trabalhar com a equipe da O'Reilly foi incrível! Agradecimentos especiais a Jess Haberman por ter confiado em nós durante o processo de proposta do livro, a nossas incríveis e extremamente pacientes editoras de desenvolvimento, Nicole Taché e Michele Cronin, pela inestimável edição, feedback e apoio. Obrigado também à excelente equipe de produção da O'Reilly (Greg e equipe).

Joe gostaria de agradecer a sua família — Cassie, Milo e Ethan — por deixá-lo escrever um livro. Eles tiveram que aguentar muito, e Joe promete nunca mais escrever um livro. ;)

Matt gostaria de agradecer a seus amigos e familiares por sua paciência e apoio duradouros. Ele ainda está esperançoso de que Sêneca se dignará a dar uma avaliação cinco estrelas depois de muito trabalho e perder o tempo com a família nas férias.

Fundação e blocos de construção

Engenharia de dados descrita

Se você trabalha com dados ou software, deve ter notado a engenharia de dados emergindo das sombras e agora dividindo o palco com a ciência de dados. A engenharia de dados é um dos campos mais quentes em dados e tecnologia, e por um bom motivo. Ele constrói a base para ciência de dados e análise na produção. Este capítulo explora o que é engenharia de dados, como o campo nasceu e sua evolução, as habilidades dos engenheiros de dados e com quem eles trabalham.

O que é Engenharia de Dados?

Apesar da popularidade atual da engenharia de dados, há muita confusão sobre o que significa engenharia de dados e o que os engenheiros de dados fazem. A engenharia de dados existe de alguma forma desde que as empresas começaram a fazer coisas com dados – como análise preditiva, análise descritiva e relatórios – e ganhou foco junto com o surgimento da ciência de dados na década de 2010. Para o propósito deste livro, é fundamental definir o que *engenharia de dados* e *engenheiro de dados* significar.

Primeiro, vamos ver como a engenharia de dados é descrita e desenvolver alguma terminologia que podemos usar ao longo deste livro. Definições infinitas de *engenharia de dados* existir. No início de 2022, uma pesquisa de correspondência exata do Google para “o que é engenharia de dados?” retorna mais de 91.000 resultados únicos. Antes de darmos nossa definição, aqui estão alguns exemplos de como alguns especialistas da área definem a engenharia de dados:

A engenharia de dados é um conjunto de operações voltadas para a criação de interfaces e mecanismos para o fluxo e acesso de informações. São necessários especialistas dedicados — engenheiros de dados — para manter os dados de forma que permaneçam disponíveis e utilizáveis por outros. Em suma, os engenheiros de dados configuram e operam a infraestrutura de dados da organização, preparando-a para análise posterior por analistas de dados e cientistas.

— De “Data Engineering and Its Main Concepts” de AlexSoft¹

O primeiro tipo de engenharia de dados é focado em SQL. O trabalho e o armazenamento primário dos dados são em bancos de dados relacionais. Todo o processamento de dados é feito com SQL ou uma linguagem baseada em SQL. Às vezes, esse processamento de dados é feito com uma ferramenta ETL.² O segundo tipo de engenharia de dados é focado em Big Data. O trabalho e o armazenamento primário dos dados estão nas tecnologias de Big Data, como Hadoop, Cassandra e HBase. Todo o processamento de dados é feito em estruturas de Big Data como MapReduce, Spark e Flink. Enquanto o SQL é usado, o processamento primário é feito com linguagens de programação como Java, Scala e Python.

— Jesse Anderson³

Em relação às funções existentes anteriormente, o campo de engenharia de dados pode ser pensado como um superconjunto de inteligência de negócios e armazenamento de dados que traz mais elementos da engenharia de software. Esta disciplina também integra especialização em torno da operação dos chamados sistemas distribuídos de “big data”, juntamente com conceitos em torno do ecossistema Hadoop estendido, processamento de fluxo e computação em escala.

— Maxime Beauchemin⁴

A engenharia de dados trata da movimentação, manipulação e gerenciamento de dados.

—Lewis Gavins

Uau! É totalmente comprehensível se você estiver confuso sobre engenharia de dados. Isso é apenas um punhado de definições, e elas contêm uma enorme variedade de opiniões sobre o significado de *engenharia de dados*.

Engenharia de dados definida

Quando descompactamos as linhas comuns de como várias pessoas definem a engenharia de dados, surge um padrão óbvio: um engenheiro de dados obtém dados, armazena-os e os prepara para consumo por cientistas de dados, analistas e outros. Nós definimos *engenharia de dados* e *engenheiro de dados* do seguinte modo:

Engenharia de dados é o desenvolvimento, implementação e manutenção de sistemas e processos que recebem dados brutos e produzem informações consistentes e de alta qualidade que dão suporte a casos de uso downstream, como análise e aprendizado de máquina. A engenharia de dados é a interseção de segurança, gerenciamento de dados, DataOps, arquitetura de dados, orquestração e engenharia de software. A *engenheiro de dados* gerencia o ciclo de vida da engenharia de dados, começando com a obtenção de dados dos sistemas de origem e terminando com o fornecimento de dados para casos de uso, como análise ou aprendizado de máquina.

¹ “Data Engineering and Its Main Concepts”, AlexSoft, última atualização em 26 de agosto de 2021, <https://oreil.ly/e94py>.

² ETL significa extraír, transformar, carregar, um padrão comum que abordamos no livro.

³ Jesse Anderson, “Os dois tipos de engenharia de dados”, 27 de junho de 2018, <https://oreil.ly/dxDt6>. ⁴ Maxime

Beauchemin, “The Rise of the Data Engineer”, 20 de janeiro de 2017, <https://oreil.ly/kNDmd>. ⁵ Lewis Gavin, *O que é Engenharia de Dados?* (Sebastopol, CA: O'Reilly, 2020), <https://oreil.ly/ELxLi>.

O ciclo de vida da engenharia de dados

É muito fácil se fixar na tecnologia e perder o quadro geral de forma míope. Este livro gira em torno de uma grande ideia chamada *ciclo de vida da engenharia de dados* (Figura 1-1), que acreditamos fornecer aos engenheiros de dados o contexto holístico para visualizar sua função.

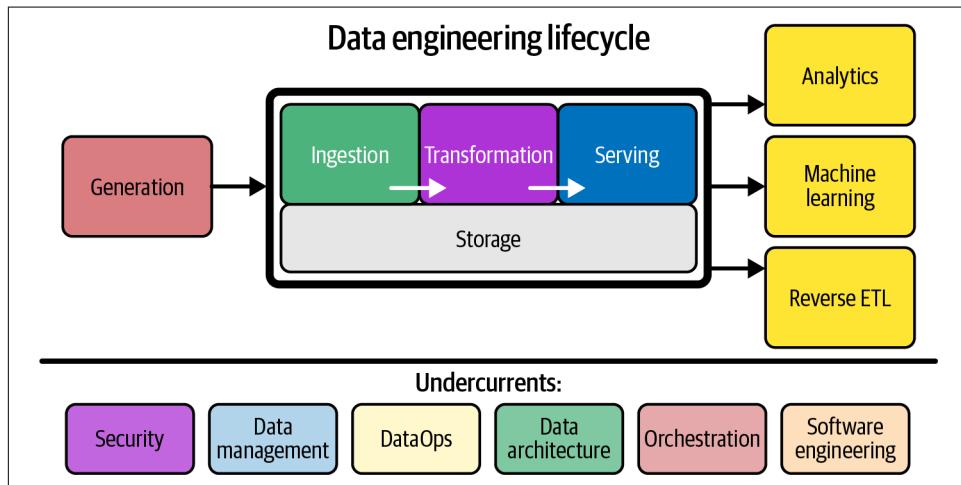


Figura 1-1. O ciclo de vida da engenharia de dados

O ciclo de vida da engenharia de dados muda a conversa da tecnologia para os próprios dados e os objetivos finais que devem servir. As etapas do ciclo de vida da engenharia de dados são as seguintes:

- Geração
- Armazenar
- Ingestão
- Transformação
- Servir

O ciclo de vida da engenharia de dados também tem uma noção *desubcorrentes*—ideias críticas ao longo de todo o ciclo de vida. Isso inclui segurança, gerenciamento de dados, DataOps, arquitetura de dados, orquestração e engenharia de software. Cobrimos o ciclo de vida da engenharia de dados e suas subcorrentes mais extensivamente em [Capítulo 2](#). Ainda assim, nós o apresentamos aqui porque é essencial para nossa definição de engenharia de dados e a discussão que se segue neste capítulo.

Agora que você tem uma definição funcional de engenharia de dados e uma introdução ao seu ciclo de vida, vamos dar um passo para trás e examinar um pouco da história.

Evolução do Engenheiro de Dados

A história não se repete, mas rima.

— Um ditado famoso frequentemente atribuído a Mark Twain

Compreender a engenharia de dados hoje e amanhã requer um contexto de como o campo evoluiu. Esta seção não é uma aula de história, mas olhar para o passado é inestimável para entender onde estamos hoje e para onde as coisas estão indo. Um tema comum reaparece constantemente: o que é velho é novo de novo.

Os primeiros dias: 1980 a 2000, do armazenamento de dados à web

O nascimento do engenheiro de dados indiscutivelmente tem suas raízes no armazenamento de dados, que datam da década de 1970, com o *armazém de dados de negócios* tomando forma na década de 1980 e Bill Inmon cunhando oficialmente o termo *armazém de dados* em 1989. Depois que os engenheiros da IBM desenvolveram o banco de dados relacional e a linguagem de consulta estruturada (SQL), a Oracle popularizou a tecnologia. À medida que os sistemas de dados nascentes cresciam, as empresas precisavam de ferramentas dedicadas e pipelines de dados para geração de relatórios e inteligência de negócios (BI). Para ajudar as pessoas a modelar corretamente sua lógica de negócios no data warehouse, Ralph Kimball e Inmon desenvolveram suas respectivas técnicas e abordagens de modelagem de dados homônimas, que ainda são amplamente utilizadas hoje.

O data warehousing inaugurou a primeira era da análise escalável, com novos bancos de dados de processamento paralelo massivo (MPP) que usam vários processadores para processar grandes quantidades de dados que chegam ao mercado e suportam volumes de dados sem precedentes. Funções como engenheiro de BI, desenvolvedor de ETL e engenheiro de data warehouse atenderam às várias necessidades do data warehouse. Data warehouse e engenharia de BI foram precursores da engenharia de dados de hoje e ainda desempenham um papel central na disciplina.

A internet tornou-se popular em meados da década de 1990, criando toda uma nova geração de empresas pioneiras na web, como AOL, Yahoo e Amazon. O boom das pontocom gerou uma tonelada de atividades em aplicativos da Web e nos sistemas de back-end para apoiá-los - servidores, bancos de dados e armazenamento. Grande parte da infraestrutura era cara, monolítica e fortemente licenciada. Os fornecedores que vendem esses sistemas de back-end provavelmente não previram a escala absoluta dos dados que os aplicativos da web produziriam.

O início dos anos 2000: o nascimento da engenharia de dados contemporânea

Avanço rápido para o início dos anos 2000, quando o boom das pontocom do final dos anos 90 estourou, deixando para trás um pequeno grupo de sobreviventes. Algumas dessas empresas, como Yahoo, Google e Amazon, se transformaram em poderosas empresas de tecnologia. Inicialmente, essas empresas continuaram a contar com os tradicionais bancos de dados relacionais monolíticos e data warehouses da década de 1990, levando esses sistemas ao limite. Como esses sistemas

Abordagens desatualizadas e atualizadas eram necessárias para lidar com o crescimento dos dados. A nova geração dos sistemas deve ser econômica, escalável, disponível e confiável.

Coincidindo com a explosão de dados, o hardware comum – como servidores, RAM, discos unidade flash – também se tornou barato e onipresente. Várias inovações permitiram computação e armazenamento distribuídos em clusters de computação massivos em grande escala. Essas inovações começaram a descentralizar e separar os serviços tradicionalmente monolíticos. A era do “big data” havia começado.

O *Oxford English Dictionary* define **grandes dados** como “conjuntos de dados extremamente grandes que podem ser analisados computacionalmente para revelar padrões, tendências e associações, especialmente relacionadas ao comportamento e interações humanas”. Outra descrição famosa e sucinta de big data são os três *Vs* de dados: velocidade, variedade e volume.

Em 2003, o Google publicou um artigo sobre o Google File System e, logo depois, em 2004, um artigo sobre MapReduce, um paradigma de processamento de dados ultraescalável. Na verdade, big data tem antecedentes anteriores em data warehouses MPP e gerenciamento de dados para projetos de física experimental, mas as publicações do Google constituíram um “big bang” para tecnologias de dados e as raízes culturais da engenharia de dados como a conhecemos hoje. Você aprenderá mais sobre sistemas MPP e MapReduce nos capítulos 3 e 8, respectivamente.

Os documentos do Google inspiraram os engenheiros do Yahoo a desenvolver e posteriormente o Apache Hadoop de código aberto em 2006.⁶ É difícil exagerar o impacto do Hadoop. Engenheiros de software interessados em problemas de dados em grande escala foram atraídos pelas possibilidades desse novo ecossistema de tecnologia de código aberto. À medida que empresas de todos os tamanhos e tipos viram seus dados crescerem em muitos terabytes e até petabytes, nasceu a era do engenheiro de big data.

Ao mesmo tempo, a Amazon teve que acompanhar suas próprias necessidades crescentes de dados e criou ambientes de computação elásticos (Amazon Elastic Compute Cloud, ou EC2), sistemas de armazenamento infinitamente escaláveis (Amazon Simple Storage Service, ou S3), bancos de dados NoSQL altamente escaláveis (Amazon DynamoDB) e muitos outros blocos de construção de dados principais.⁷ A Amazon optou por oferecer esses serviços para consumo interno e externo por meio de *Amazon Web Services* (AWS), tornando-se a primeira nuvem pública popular. A AWS criou um mercado de recursos de pagamento conforme o uso ultraflexível, virtualizando e revendendo vastos conjuntos de hardware de commodities. Em vez de comprar hardware para um data center, os desenvolvedores podem simplesmente alugar computação e armazenamento da AWS.

6 Cade Metz, “How Yahoo Spawned Hadoop, the Future of Big Data,” *Com fio*, 18 de outubro de 2011, <https://oreil.ly/iaD9G>.

7 Ron Miller, “Como surgiu a AWS,” *TechCrunch*, 2 de julho de 2016, <https://oreil.ly/Vjehv>.

Como a AWS se tornou um mecanismo de crescimento altamente lucrativo para a Amazon, outras nuvens públicas logo se seguiriam, como Google Cloud, Microsoft Azure e DigitalOcean. A nuvem pública é indiscutivelmente uma das inovações mais significativas do século 21 e gerou uma revolução na forma como os aplicativos de software e dados são desenvolvidos e implantados.

As primeiras ferramentas de big data e a nuvem pública lançaram as bases para o ecossistema de dados atual. O cenário de dados moderno – e a engenharia de dados como a conhecemos agora – não existiria sem essas inovações.

Décadas de 2000 e 2010: engenharia de big data

As ferramentas de big data de código aberto no ecossistema Hadoop amadureceram rapidamente e se espalharam do Vale do Silício para empresas com experiência em tecnologia em todo o mundo. Pela primeira vez, qualquer empresa teve acesso às mesmas ferramentas de dados de ponta usadas pelas principais empresas de tecnologia. Outra revolução ocorreu com a transição da computação em lote para o streaming de eventos, inaugurando uma nova era de grandes dados em “tempo real”. Você aprenderá sobre streaming de eventos e lotes ao longo deste livro.

Os engenheiros podiam escolher as melhores e mais recentes — Hadoop, Apache Pig, Apache Hive, Dremel, Apache HBase, Apache Storm, Apache Cassandra, Apache Spark, Presto e várias outras novas tecnologias que entraram em cena. As ferramentas de dados tradicionais orientadas para empresas e baseadas em GUI de repente pareciam obsoletas, e a engenharia que priorizava o código estava em voga com a ascensão do MapReduce. Nós (os autores) estávamos por aí durante esse tempo, e parecia que velhos dogmas morreram repentinamente no altar do big data.

A explosão de ferramentas de dados no final dos anos 2000 e 2010 inaugurou o engenheiro de big data. Para usar efetivamente essas ferramentas e técnicas - ou seja, o ecossistema Hadoop, incluindo Hadoop, YARN, Hadoop Distributed File System (HDFS) e MapReduce - os engenheiros de big data precisavam ser proficientes em desenvolvimento de software e hacking de infraestrutura de baixo nível, mas com uma ênfase deslocada. Os engenheiros de big data geralmente mantinham grandes clusters de hardware de commodities para fornecer dados em escala. Embora possam ocasionalmente enviar solicitações pull ao código principal do Hadoop, eles mudaram seu foco do desenvolvimento da tecnologia principal para a entrega de dados.

Big data rapidamente se tornou vítima de seu próprio sucesso. Como uma palavra da moda, *grandes dados* ganhou popularidade durante o início dos anos 2000 até meados dos anos 2010. O big data capturou a imaginação das empresas que tentavam entender os volumes cada vez maiores de dados e a enxurrada interminável de marketing descarado de empresas que vendiam ferramentas e serviços de big data. Devido ao imenso hype, era comum ver empresas usando ferramentas de big data para pequenos problemas de dados, às vezes montando um cluster Hadoop para processar apenas alguns gigabytes. Parecia que todos queriam participar da ação de big data. Dan Ariely *tuitou*, “Big data é como sexo na adolescência: todo mundo fala sobre isso, ninguém

realmente sabe como fazer, todo mundo pensa que todo mundo está fazendo, então todo mundo afirma que está fazendo."

Figura 1-2 mostra um instantâneo do Google Trends para o termo de pesquisa "big data" para ter uma ideia da ascensão e queda do big data.

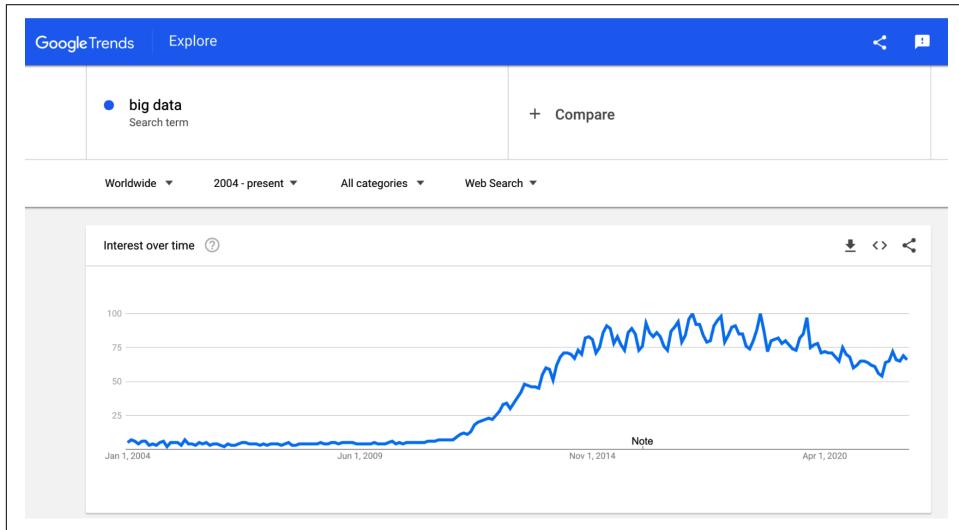


Figura 1-2. Google Trends para "big data" (março de 2022)

Apesar da popularidade do termo, big data perdeu força. O que aconteceu? Uma palavra: simplificação. Apesar do poder e da sofisticação das ferramentas de big data de código aberto, gerenciá-las dava muito trabalho e exigia atenção constante. Muitas vezes, as empresas empregavam equipes inteiras de engenheiros de big data, custando milhões de dólares por ano, para cuidar dessas plataformas. Os engenheiros de big data geralmente gastam muito tempo mantendo ferramentas complicadas e, sem dúvida, não tanto tempo entregando os insights e o valor do negócio.

Desenvolvedores de código aberto, nuvens e terceiros começaram a procurar maneiras de abstrair, simplificar e disponibilizar big data sem a alta sobrecarga administrativa e o custo de gerenciar seus clusters e instalar, configurar e atualizar seu código de código aberto. O termo *grandes dados* é essencialmente uma relíquia para descrever um determinado momento e abordagem para lidar com grandes quantidades de dados.

Hoje, os dados estão se movendo mais rápido do que nunca e crescendo cada vez mais, mas o processamento de big data tornou-se tão acessível que não merece mais um termo separado; toda empresa visa resolver seus problemas de dados, independentemente do tamanho real dos dados. Engenheiros de big data agora são simplesmente *engenheiros de dados*.

Década de 2020: Engenharia para o ciclo de vida dos dados

No momento da redação deste artigo, a função de engenharia de dados está evoluindo rapidamente. Esperamos que essa evolução continue em ritmo acelerado no futuro previsível. Enquanto os engenheiros de dados historicamente tendiam aos detalhes de baixo nível de estruturas monolíticas como Hadoop, Spark ou Informatica, a tendência está se movendo em direção a ferramentas descentralizadas, modularizadas, gerenciadas e altamente abstratas.

De fato, as ferramentas de dados proliferaram em um ritmo surpreendente (veja [Figura 1-3](#)). As tendências populares no início de 2020 incluem *opilha de dados moderna*, representando uma coleção de produtos de código aberto e de terceiros prontos para uso, reunidos para facilitar a vida dos analistas. Ao mesmo tempo, as fontes e formatos de dados estão crescendo em variedade e tamanho. A engenharia de dados é cada vez mais uma disciplina de interoperação e conecta várias tecnologias, como peças de LEGO, para atender aos objetivos comerciais finais.

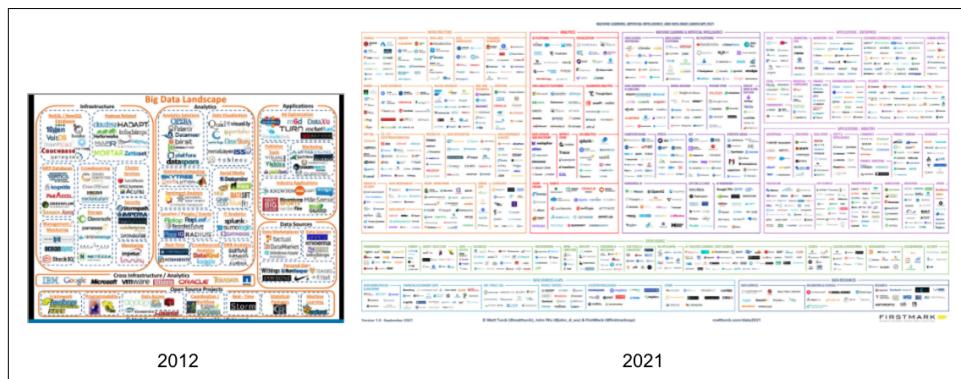


Figura 1-3. de Matt Turck Cenário de dados em 2012 versus 2021

O engenheiro de dados que discutimos neste livro pode ser descrito mais precisamente como um *engenheiro de ciclo de vida de dados*. Com maior abstração e simplificação, um engenheiro de ciclo de vida de dados não é mais sobrecarregado pelos detalhes sangrentos das estruturas de big data de ontem. Embora os engenheiros de dados mantenham habilidades em programação de dados de baixo nível e as usem conforme necessário, eles encontram cada vez mais seu papel focado em coisas mais altas na cadeia de valor: segurança, gerenciamento de dados, DataOps, arquitetura de dados, orquestração e gerenciamento geral do ciclo de vida dos dados.⁸

À medida que as ferramentas e os fluxos de trabalho são simplificados, observamos uma mudança perceptível nas atitudes dos engenheiros de dados. Em vez de focar em quem tem os “maiores dados”, os projetos e serviços de código aberto estão cada vez mais preocupados em gerenciar e governar os dados, facilitando seu uso e descoberta e melhorando sua qualidade. Os engenheiros de dados são

⁸DataOpsé uma abreviação de *operações de dados*. Cobrimos este tópico em [Capítulo 2](#). Para mais informações, leia o [Manifesto DataOps](#).

agora familiarizado com siglas como CCPA e RGPD,⁹ enquanto projetam pipelines, eles se preocupam com privacidade, anonimização, coleta de lixo de dados e conformidade com os regulamentos.

O que é velho é novo de novo. Embora coisas “empresariais” como gerenciamento de dados (incluindo qualidade e governança de dados) fossem comuns para grandes empresas na era pré-big data, elas não eram amplamente adotadas em empresas menores. Agora que muitos dos problemas desafiadores dos sistemas de dados de ontem foram resolvidos, cuidadosamente produzidos e empacotados, tecnólogos e empreendedores mudaram o foco de volta para o material “empresarial”, mas com ênfase na descentralização e agilidade, que contrasta com o comando empresarial tradicional abordagem -e-controle.

Vemos o presente como uma era de ouro do gerenciamento do ciclo de vida dos dados. Os engenheiros de dados que gerenciam o ciclo de vida da engenharia de dados têm ferramentas e técnicas melhores do que nunca. Discutimos o ciclo de vida da engenharia de dados e suas subcorrentes com mais detalhes no próximo capítulo.

Engenharia de Dados e Ciência de Dados

Onde a engenharia de dados se encaixa na ciência de dados? Há algum debate, com alguns argumentando que a engenharia de dados é uma subdisciplina da ciência de dados. Acreditamos que a engenharia de dados é separada da ciência de dados e análise. Eles se complementam, mas são bem diferentes. A engenharia de dados fica acima da ciência de dados (Figura 1-4), o que significa que os engenheiros de dados fornecem as entradas usadas pelos cientistas de dados (downstream da engenharia de dados), que convertem essas entradas em algo útil.

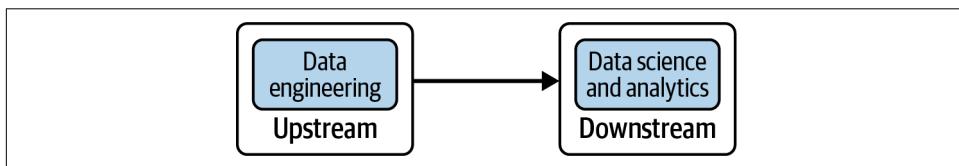


Figura 1-4. A engenharia de dados está a montante da ciência de dados

Considere a hierarquia de necessidades da ciência de dados (Figura 1-5). Em 2017, Monica Rogati publicou essa hierarquia em um artigo que mostrou onde a IA e o aprendizado de máquina (ML) ficavam próximos a áreas mais “mundanas”, como movimentação/armazenamento de dados, coleta e infraestrutura.

⁹ Essas siglas significam Lei de Privacidade do Consumidor da Califórnia e Regulamento Geral de Proteção de Dados, respectivamente.

THE DATA SCIENCE HIERARCHY OF NEEDS

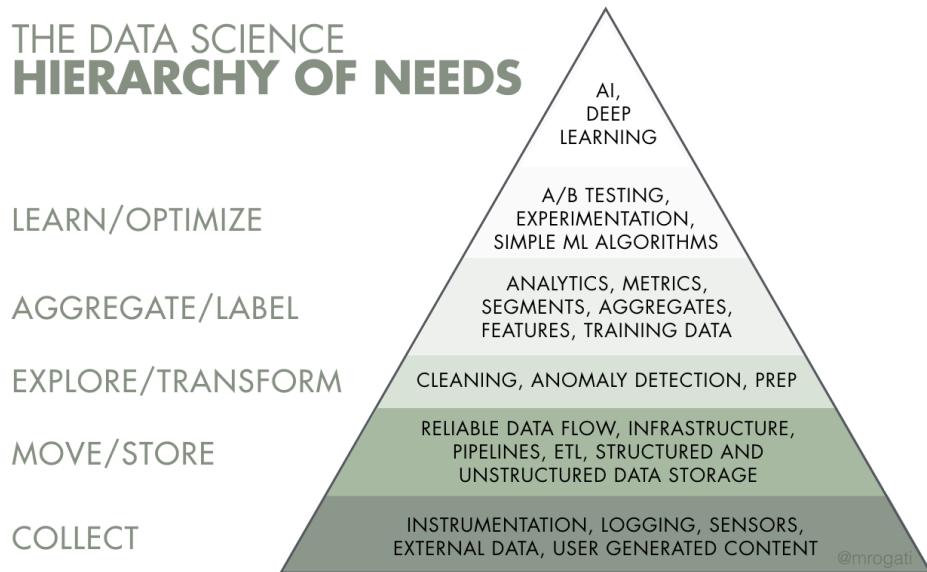


Figura 1-5. A hierarquia de necessidades da ciência de dados

Embora muitos cientistas de dados estejam ansiosos para criar e ajustar modelos de ML, a realidade é que cerca de 70% a 80% de seu tempo é gasto labutando nas três partes inferiores da hierarquia - coleta de dados, limpeza de dados, processamento de dados - e apenas uma pequena fatia de seu tempo em análise e ML. Rogati argumenta que as empresas precisam construir uma base sólida de dados (os três níveis inferiores da hierarquia) antes de abordar áreas como IA e ML.

Os cientistas de dados normalmente não são treinados para projetar sistemas de dados de nível de produção e acabam fazendo esse trabalho ao acaso porque não têm o suporte e os recursos de um engenheiro de dados. Em um mundo ideal, os cientistas de dados deveriam passar mais de 90% do tempo focados nas camadas superiores da pirâmide: análise, experimentação e ML. Quando os engenheiros de dados se concentram nessas partes inferiores da hierarquia, eles constroem uma base sólida para o sucesso dos cientistas de dados.

Com a ciência de dados impulsionando análises avançadas e ML, a engenharia de dados atravessa a divisão entre obter dados e obter valor dos dados (consulte Figura 1-6). Acreditamos que a engenharia de dados é de igual importância e visibilidade para a ciência de dados, com os engenheiros de dados desempenhando um papel vital para tornar a ciência de dados bem-sucedida na produção.

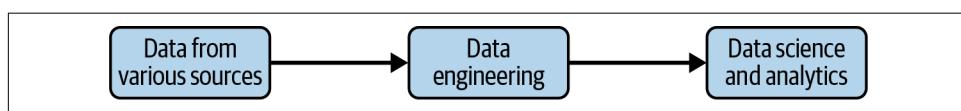


Figura 1-6. Um engenheiro de dados obtém dados e fornece valor a partir dos dados

Habilidades e atividades de engenharia de dados

O conjunto de habilidades de um engenheiro de dados abrange as “subcorrentes” da engenharia de dados: segurança, gerenciamento de dados, DataOps, arquitetura de dados e engenharia de software. Esse conjunto de habilidades requer uma compreensão de como avaliar as ferramentas de dados e como elas se encaixam no ciclo de vida da engenharia de dados. Também é fundamental saber como os dados são produzidos nos sistemas de origem e como os analistas e cientistas de dados consumirão e criará valor após o processamento e a curadoria dos dados. Por fim, um engenheiro de dados manipula muitas partes móveis complexas e deve otimizar constantemente os eixos de custo, agilidade, escalabilidade, simplicidade, reutilização e interoperabilidade (Figura 1-7). Abordaremos esses tópicos com mais detalhes nos próximos capítulos.

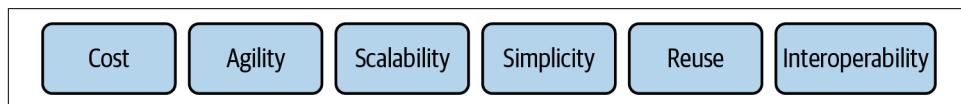


Figura 1-7. O equilíbrio da engenharia de dados

Como discutimos, no passado recente, esperava-se que um engenheiro de dados soubesse e entendesse como usar um pequeno punhado de tecnologias poderosas e monolíticas (Hadoop, Spark, Teradata, Hive e muitas outras) para criar uma solução de dados. A utilização dessas tecnologias geralmente requer um entendimento sofisticado de engenharia de software, redes, computação distribuída, armazenamento ou outros detalhes de baixo nível. Seu trabalho seria dedicado à administração e manutenção do cluster, gerenciamento de despesas gerais e criação de trabalhos de pipeline e transformação, entre outras tarefas.

Atualmente, o cenário de ferramentas de dados é consideravelmente menos complicado de gerenciar e implantar. As ferramentas de dados modernas abstraem e simplificam consideravelmente os fluxos de trabalho. Como resultado, os engenheiros de dados agora estão focados em equilibrar os serviços mais simples e econômicos, os melhores da categoria, que agregam valor aos negócios. Espera-se também que o engenheiro de dados crie arquiteturas de dados ágeis que evoluem à medida que surgem novas tendências.

Quais são algumas coisas que um engenheiro de dados faz/não faz? Um engenheiro de dados normalmente não cria diretamente modelos de ML, cria relatórios ou painéis, realiza análises de dados, cria indicadores-chave de desempenho (KPIs) ou desenvolve aplicativos de software. Um engenheiro de dados deve ter uma boa compreensão dessas áreas para atender melhor às partes interessadas.

Maturidade de dados e o engenheiro de dados

O nível de complexidade da engenharia de dados dentro de uma empresa depende muito da maturidade dos dados da empresa. Isso afeta significativamente as responsabilidades do trabalho diário de um engenheiro de dados e a progressão na carreira. O que é maturidade de dados, exatamente?

Maturidade dos dados é a progressão para maior utilização de dados, capacidades e integração em toda a organização, mas a maturidade dos dados não depende simplesmente do

idade ou receita de uma empresa. Uma startup em estágio inicial pode ter maior maturidade de dados do que uma empresa de 100 anos com receita anual na casa dos bilhões. O que importa é a forma como os dados são aproveitados como uma vantagem competitiva.

Os modelos de maturidade de dados têm muitas versões, como **Data Management Maturity (DMM)** e outros, e é difícil escolher um que seja simples e útil para engenharia de dados. Assim, criaremos nosso próprio modelo simplificado de maturidade de dados. Nosso modelo de maturidade de dados (**Figura 1-8**) tem três estágios: começando com dados, escalando com dados e liderando com dados. Vejamos cada um desses estágios e o que um engenheiro de dados normalmente faz em cada estágio.

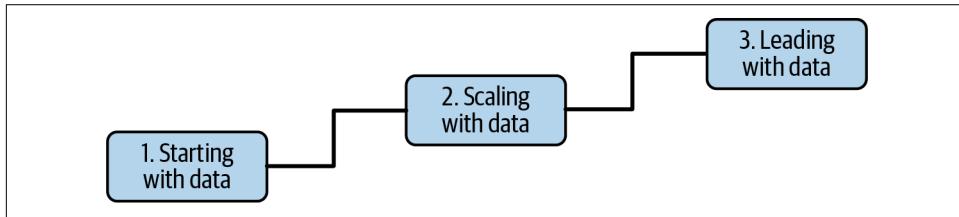


Figura 1-8. Nossa modelo simplificado de maturidade de dados para uma empresa

Estágio 1: Começando com os dados

Uma empresa que está começando com dados está, por definição, nos estágios iniciais de sua maturidade de dados. A empresa pode ter objetivos confusos e vagamente definidos ou nenhum objetivo. A arquitetura de dados e a infraestrutura estão nos estágios iniciais de planejamento e desenvolvimento. A adoção e a utilização provavelmente são baixas ou inexistentes. A equipe de dados é pequena, geralmente com um número de funcionários de um dígito. Nesse estágio, um engenheiro de dados geralmente é um generalista e normalmente desempenha várias outras funções, como cientista de dados ou engenheiro de software. O objetivo de um engenheiro de dados é mover-se rapidamente, obter tração e agregar valor.

Os aspectos práticos de obter valor a partir dos dados geralmente são mal compreendidos, mas o desejo existe. Relatórios ou análises carecem de estrutura formal e a maioria das solicitações de dados são ad hoc. Embora seja tentador mergulhar de cabeça no ML neste estágio, não o recomendamos. Vimos inúmeras equipes de dados travarem e falharem quando tentam pular para o ML sem construir uma base sólida de dados.

Isso não quer dizer que você não possa obter vitórias do ML neste estágio - é raro, mas possível. Sem uma base de dados sólida, você provavelmente não terá os dados para treinar modelos de ML confiáveis nem os meios para implantar esses modelos na produção de maneira escalonável e repetível. Nós meio que brincando nos chamamos **“recuperando cientistas de dados”**, principalmente por experiência pessoal de envolvimento em projetos prematuros de ciência de dados sem maturidade de dados adequada ou suporte de engenharia de dados.

Um engenheiro de dados deve se concentrar no seguinte em organizações que estão começando com dados:

- Obtenha a adesão dos principais interessados, incluindo a gerência executiva. Idealmente, o engenheiro de dados deve ter um patrocinador para iniciativas críticas para projetar e construir uma arquitetura de dados para dar suporte aos objetivos da empresa.
- Definir a arquitetura de dados correta (geralmente individual, já que um arquiteto de dados provavelmente não está disponível). Isso significa determinar as metas de negócios e a vantagem competitiva que você pretende alcançar com sua iniciativa de dados. Trabalhe em direção a uma arquitetura de dados que suporte esses objetivos. Ver [Capítulo 3](#) pelos nossos conselhos sobre uma "boa" arquitetura de dados.
- Identifique e audite os dados que darão suporte às principais iniciativas e operarão dentro da arquitetura de dados que você projetou.
- Construir uma base sólida de dados para futuros analistas e cientistas de dados gerarem relatórios e modelos que fornecem valor competitivo. Nesse ínterim, você também pode ter que gerar esses relatórios e modelos até que essa equipe seja contratada.

Esta é uma fase delicada com muitas armadilhas. Seguem algumas dicas para esta etapa:

- A força de vontade organizacional pode diminuir se muitos sucessos visíveis não ocorrerem com os dados. Obter ganhos rápidos estabelecerá a importância dos dados dentro da organização. Apenas tenha em mente que vitórias rápidas provavelmente criará dívida técnica. Tenha um plano para reduzir essa dívida, caso contrário, adicionará atrito para a entrega futura.
- Saia e converse com as pessoas e evite trabalhar em silos. Muitas vezes vemos a equipe de dados trabalhando em uma bolha, não se comunicando com pessoas fora de seus departamentos e obtendo perspectivas e feedback das partes interessadas nos negócios. O perigo é você passar muito tempo trabalhando em coisas de pouca utilidade para as pessoas.
- Evite levantamento de peso indiferenciado. Não se prenda a complexidades técnicas desnecessárias. Use soluções prontas para uso sempre que possível.
- Construir soluções personalizadas e codificar apenas quando isso criar uma vantagem competitiva.

Estágio 2: escalonamento com dados

Neste ponto, uma empresa se afastou das solicitações de dados ad hoc e tem práticas formais de dados. Agora, o desafio é criar arquiteturas de dados escaláveis e planejar um futuro em que a empresa seja genuinamente orientada por dados. As funções de engenharia de dados passam de generalistas para especialistas, com pessoas se concentrando em aspectos específicos do ciclo de vida da engenharia de dados.

Em organizações que estão no estágio 2 de maturidade de dados, os objetivos de um engenheiro de dados são:

- Estabelecer práticas formais de dados
- Crie arquiteturas de dados escaláveis e robustas

- Adotar práticas de DevOps e DataOps
- Construa sistemas que suportem ML
- Continue a evitar o trabalho pesado indiferenciado e personalize apenas quando resultar uma vantagem competitiva

Voltaremos a cada um desses objetivos mais adiante neste livro. Os

problemas a serem observados incluem o seguinte:

- À medida que nos tornamos mais sofisticados com os dados, há uma tentação de adotar tecnologias de ponta baseadas em provas sociais de empresas do Vale do Silício. Isso raramente é um bom uso de seu tempo e energia. Quaisquer decisões de tecnologia devem ser orientadas pelo valor que elas entregará aos seus clientes.
- O principal gargalo para dimensionamento não são os nós de cluster, armazenamento ou tecnologia, mas a equipe de engenharia de dados. Concentre-se em soluções simples de implantar e gerenciar para expandir o rendimento de sua equipe.
- Você ficará tentado a se enquadrar como um tecnólogo, um gênio dos dados que pode fornecer produtos mágicos. Mude seu foco para a liderança pragmática e comece a transição para o próximo estágio de maturidade; comunicar-se com outras equipes sobre a utilidade prática dos dados. Ensine a organização a consumir e alavancar dados.

Estágio 3: Liderando com dados

Nesta fase, a empresa é orientada por dados. Os pipelines e sistemas automatizados criados por engenheiros de dados permitem que as pessoas dentro da empresa façam análises de autoatendimento e ML. A introdução de novas fontes de dados é perfeita e o valor tangível é derivado. Os engenheiros de dados implementam controles e práticas adequados para garantir que os dados estejam sempre disponíveis para as pessoas e sistemas. As funções de engenharia de dados continuam a se especializar mais profundamente do que no estágio 2.

Em organizações no estágio 3 de maturidade de dados, um engenheiro de dados continuará desenvolvendo os estágios anteriores, além de fazer o seguinte:

- Criar automação para a introdução e uso contínuo de novos dados
- Foco na criação de ferramentas e sistemas personalizados que aproveitam os dados como uma vantagem competitiva
- Foco nos aspectos “empresariais” dos dados, como gerenciamento de dados (incluindo governança e qualidade de dados) e DataOps
- Implantar ferramentas que expõem e disseminam dados em toda a organização, incluindo catálogos de dados, ferramentas de linhagem de dados e sistemas de gerenciamento de metadados

- Colaborar de forma eficiente com engenheiros de software, engenheiros de ML, analistas e outros
- Crie uma comunidade e um ambiente onde as pessoas possam colaborar e falar abertamente, independentemente de sua função ou posição

Os problemas a serem observados incluem o seguinte:

- Nesta fase, a complacência é um perigo significativo. Quando as organizações atingem o estágio 3, elas devem se concentrar constantemente na manutenção e na melhoria ou correm o risco de cair para um estágio inferior.
- As distrações tecnológicas são um perigo mais significativo aqui do que nos outros estágios. Há uma tentação de perseguir projetos de hobby caros que não agregam valor ao negócio. Utilize a tecnologia personalizada apenas onde ela oferece uma vantagem competitiva.

O histórico e as habilidades de um engenheiro de dados

A engenharia de dados é um campo em rápido crescimento e muitas questões permanecem sobre como se tornar um engenheiro de dados. Como a engenharia de dados é uma disciplina relativamente nova, há pouco treinamento formal disponível para entrar no campo. As universidades não têm um caminho de engenharia de dados padrão. Embora um punhado de campos de treinamento de engenharia de dados e tutoriais on-line cubram tópicos aleatórios, ainda não existe um currículo comum para o assunto.

As pessoas que ingressam na engenharia de dados chegam com experiências variadas em educação, carreira e conjunto de habilidades. Todos que entram no campo devem esperar investir uma quantidade significativa de tempo em auto-estudo. A leitura deste livro é um bom ponto de partida; um dos principais objetivos deste livro é fornecer uma base para o conhecimento e as habilidades que consideramos necessárias para ter sucesso como engenheiro de dados.

Se você está direcionando sua carreira para a engenharia de dados, descobrimos que a transição é mais fácil ao passar de um campo adjacente, como engenharia de software, desenvolvimento de ETL, administração de banco de dados, ciência de dados ou análise de dados. Essas disciplinas tendem a ser “conscientes dos dados” e fornecem um bom contexto para funções de dados em uma organização. Eles também equipam as pessoas com as habilidades técnicas e o contexto relevantes para resolver problemas de engenharia de dados.

Apesar da falta de um caminho formalizado, existe um corpo de conhecimento necessário que acreditamos que um engenheiro de dados deve saber para ser bem-sucedido. Por definição, um engenheiro de dados deve entender os dados e a tecnologia. Com relação aos dados, isso implica conhecer várias práticas recomendadas em relação ao gerenciamento de dados. No lado da tecnologia, um engenheiro de dados deve estar ciente das várias opções de ferramentas, sua interação e suas compensações. Isso requer um bom entendimento de engenharia de software, DataOps e arquitetura de dados.

Olhando para fora, um engenheiro de dados também deve entender os requisitos dos consumidores de dados (analistas e cientistas de dados) e as implicações mais amplas dos dados em toda a organização. A engenharia de dados é uma prática holística; os melhores engenheiros de dados visualizam suas responsabilidades através de lentes comerciais e técnicas.

Responsabilidades Comerciais

As responsabilidades macro listadas nesta seção não são exclusivas dos engenheiros de dados, mas são cruciais para qualquer pessoa que trabalhe em um campo de dados ou tecnologia. Como uma simples pesquisa no Google renderá toneladas de recursos para aprender sobre essas áreas, simplesmente as listaremos para abreviar:

Saiba como se comunicar com pessoas não técnicas e técnicas.

A comunicação é fundamental e você precisa ser capaz de estabelecer relacionamento e confiança com as pessoas em toda a organização. Sugerimos prestar muita atenção às hierarquias organizacionais, quem se reporta a quem, como as pessoas interagem e quais silos existem. Essas observações serão inestimáveis para o seu sucesso.

Entenda como definir o escopo e reunir requisitos de negócios e produtos.

Você precisa saber o que construir e garantir que as partes interessadas concordem com sua avaliação. Além disso, desenvolva uma noção de como as decisões de dados e tecnologia afetam os negócios.

Entenda os fundamentos culturais de Agile, DevOps e DataOps.

Muitos tecnólogos acreditam erroneamente que essas práticas são resolvidas por meio da tecnologia. Achamos que isso é perigosamente errado. Agile, DevOps e DataOps são fundamentalmente culturais, exigindo adesão de toda a organização.

Controlar custos.

Você será bem-sucedido quando conseguir manter os custos baixos e, ao mesmo tempo, fornecer um valor extraordinário. Saiba como otimizar o tempo de retorno, o custo total de propriedade e o custo de oportunidade. Aprenda a monitorar custos para evitar surpresas.

Aprenda continuamente.

O campo de dados parece estar mudando na velocidade da luz. As pessoas bem-sucedidas são ótimas em aprender coisas novas enquanto aprimoram seus conhecimentos fundamentais. Eles também são bons em filtrar, determinando quais novos desenvolvimentos são mais relevantes para seu trabalho, quais ainda são imaturos e quais são apenas modismos. Fique a par do campo e aprenda a aprender.

Um engenheiro de dados bem-sucedido sempre reduz o zoom para entender o quadro geral e como obter valor descomunal para os negócios. A comunicação é vital, tanto para pessoas técnicas quanto para pessoas não técnicas. Muitas vezes vemos equipes de dados bem-sucedidas com base em sua comunicação com outras partes interessadas; sucesso ou fracasso raramente é uma questão de tecnologia. Saber como navegar em uma organização, escopo e reunir requisitos,

controlar os custos e aprender continuamente irá diferenciá-lo dos engenheiros de dados que dependem exclusivamente de suas habilidades técnicas para realizar sua carreira.

Responsabilidades Técnicas

Você deve entender como criar arquiteturas que otimizam o desempenho e o custo em alto nível, usando componentes pré-empacotados ou desenvolvidos internamente. Em última análise, arquiteturas e tecnologias constituintes são blocos de construção para atender ao ciclo de vida da engenharia de dados.

Lembre-se dos estágios do ciclo de vida da engenharia de dados:

- Geração
- Armazenar
- Ingestão
- Transformação
- Servir

As subcorrentes do ciclo de vida da engenharia de dados são as seguintes:

- Segurança
- Gestão de dados
- DataOps
- Arquitetura de dados
- Orquestração
- Engenharia de software

Ampliando um pouco, discutimos alguns dos dados táticos e habilidades tecnológicas que você precisará como engenheiro de dados nesta seção; discutimos isso com mais detalhes nos capítulos subsequentes.

As pessoas costumam perguntar: um engenheiro de dados deve saber codificar? Resposta curta: sim. Um engenheiro de dados deve ter habilidades de engenharia de software de nível de produção. Observamos que a natureza dos projetos de desenvolvimento de software realizados por engenheiros de dados mudou fundamentalmente nos últimos anos. Os serviços totalmente gerenciados agora substituem grande parte do esforço de programação de baixo nível anteriormente esperado dos engenheiros, que agora usam código aberto gerenciado e ofertas simples de software como serviço (SaaS) plug-and-play. Por exemplo, os engenheiros de dados agora se concentram em abstrações de alto nível ou em escrever pipelines como código em uma estrutura de orquestração.

Mesmo em um mundo mais abstrato, as melhores práticas de engenharia de software fornecem uma vantagem competitiva, e os engenheiros de dados que podem mergulhar nos detalhes arquitetônicos profundos de uma base de código oferecem às suas empresas uma vantagem quando surgem necessidades técnicas específicas. Resumindo, um engenheiro de dados que não pode escrever código de nível de produção será severamente prejudicado e

não vemos isso mudando tão cedo. Os engenheiros de dados permanecem engenheiros de software, além de suas muitas outras funções.

Quais idiomas um engenheiro de dados deve conhecer? Dividimos as linguagens de programação de engenharia de dados em categorias primárias e secundárias. No momento da redação deste artigo, as principais linguagens de engenharia de dados são SQL, Python, uma linguagem Java Virtual Machine (JVM) (geralmente Java ou Scala) e bash:

SQL

A interface mais comum para bancos de dados e data lakes. Depois de ser brevemente marginalizado pela necessidade de escrever código MapReduce personalizado para processamento de big data, o SQL (em várias formas) ressurgiu como a língua franca dos dados.

Pitão

A linguagem ponte entre engenharia de dados e ciência de dados. Um número crescente de ferramentas de engenharia de dados é escrito em Python ou possui APIs Python. É conhecido como “o segundo melhor idioma em tudo”. Python é a base de ferramentas de dados populares, como pandas, NumPy, Airflow, sci-kit learning, TensorFlow, PyTorch e PySpark. Python é a cola entre os componentes subjacentes e é frequentemente uma linguagem de API de primeira classe para interface com um framework.

Linguagens JVM como Java e Scala

Prevalente para projetos de código aberto Apache, como Spark, Hive e Druid. A JVM geralmente tem mais desempenho do que o Python e pode fornecer acesso a recursos de nível inferior do que uma API do Python (por exemplo, esse é o caso do Apache Spark e do Beam). Compreender Java ou Scala será benéfico se você estiver usando uma estrutura de dados de software livre popular.

bash

A interface de linha de comando para sistemas operacionais Linux. Conhecer os comandos bash e se sentir confortável usando CLIs melhorará significativamente sua produtividade e fluxo de trabalho quando você precisar criar scripts ou executar operações do sistema operacional. Ainda hoje, engenheiros de dados frequentemente usam ferramentas de linha de comando como awk ou sed para processar arquivos em um pipeline de dados ou chamar comandos bash de estruturas de orquestração. Se você estiver usando o Windows, sinta-se à vontade para substituir o PowerShell pelo bash.

A eficácia irracional do SQL

O advento do MapReduce e a era do big data relegaram o SQL ao status de ultrapassado. Desde então, vários desenvolvimentos melhoraram drasticamente a utilidade do SQL no ciclo de vida da engenharia de dados. Spark SQL, Google BigQuery, Snowflake, Hive e muitas outras ferramentas de dados podem processar grandes quantidades de dados usando semântica SQL declarativa e teórica de conjuntos. O SQL também é compatível com muitas estruturas de streaming, como Apache Flink, Beam e Kafka. Acreditamos que engenheiros de dados competentes devem ser altamente proficientes em SQL.

Estamos dizendo que o SQL é uma linguagem completa e definitiva? De jeito nenhum. O SQL é uma ferramenta poderosa que pode resolver rapidamente problemas complexos de análise e transformação de dados. Dado que o tempo é a principal restrição para o rendimento da equipe de engenharia de dados, os engenheiros devem adotar ferramentas que combinem simplicidade e alta produtividade. Os engenheiros de dados também devem desenvolver experiência na composição de SQL com outras operações, seja em estruturas como Spark e Flink ou usando orquestração para combinar várias ferramentas. Os engenheiros de dados também devem aprender a semântica SQL moderna para lidar com a análise de JavaScript Object Notation (JSON) e dados aninhados e considerar o uso de uma estrutura de gerenciamento SQL como **dbt** (ferramenta de construção de dados).

Um engenheiro de dados proficiente também reconhece quando o SQL não é a ferramenta certa para o trabalho e pode escolher e codificar uma alternativa adequada. Um especialista em SQL provavelmente poderia escrever uma consulta para derivar e tokenizar texto bruto em um pipeline de processamento de linguagem natural (NLP), mas também reconheceria que codificar no Spark nativo é uma alternativa muito superior a esse exercício masoquista.

Os engenheiros de dados também podem precisar desenvolver proficiência em linguagens de programação secundárias, incluindo R, JavaScript, Go, Rust, C/C++, C# e Julia. O desenvolvimento nessas linguagens geralmente é necessário quando popular em toda a empresa ou usado com ferramentas de dados específicas do domínio. Por exemplo, o JavaScript provou ser popular como linguagem para funções definidas pelo usuário em data warehouses na nuvem. Ao mesmo tempo, C# e PowerShell são essenciais em empresas que utilizam o Azure e o ecossistema da Microsoft.

Mantendo o ritmo em um campo em movimento rápido

Quando uma nova tecnologia passa por você, se você não faz parte do rolo compressor, faz parte da estrada.

—Stewart Brand

Como você mantém suas habilidades afiadas em um campo em rápida mudança como a engenharia de dados? Você deve se concentrar nas ferramentas mais recentes ou mergulhar fundo nos fundamentos? Fica o nosso conselho: foque nos fundamentos para entender o que não vai mudar; preste atenção aos desenvolvimentos em andamento para saber para onde o campo está indo. Novos paradigmas e práticas são introduzidos o tempo todo, e cabe a você se manter atualizado. Esforce-se para entender como as novas tecnologias serão úteis no ciclo de vida.

A continuidade das funções de engenharia de dados, de A a B

Embora as descrições de trabalho pintem um engenheiro de dados como um “único” que deve possuir todas as habilidades imagináveis em dados, nem todos os engenheiros de dados fazem o mesmo tipo de trabalho ou têm o mesmo conjunto de habilidades. A maturidade dos dados é um guia útil para entender os tipos de desafios de dados que uma empresa enfrentará à medida que aumenta sua capacidade de dados. É benéfico olhar

em algumas distinções críticas nos tipos de trabalho que os engenheiros de dados fazem. Embora essas distinções sejam simplistas, elas esclarecem o que os cientistas e engenheiros de dados fazem e evitam agrupar qualquer função no balde do unicórnio.

Na ciência de dados, existe a noção de cientistas de dados tipo A e tipo B.¹⁰ *Cientistas de dados tipo A*—onde *Apoia análise*—concentre-se na compreensão e na obtenção de informações a partir dos dados. *Cientistas de dados tipo B*—onde *Bapoiaprédio*—compartilham experiências semelhantes aos cientistas de dados do tipo A e possuem fortes habilidades de programação. O cientista de dados tipo B constrói sistemas que fazem a ciência de dados funcionar na produção. Tomando emprestado esse continuum de cientista de dados, criaremos uma distinção semelhante para dois tipos de engenheiros de dados:

Engenheiros de dados tipo A

Apoia abstração. Nesse caso, o engenheiro de dados evita o trabalho pesado indiferenciado, mantendo a arquitetura de dados o mais abstrata e direta possível e não reinventando a roda. Os engenheiros de dados do tipo A gerenciam o ciclo de vida da engenharia de dados principalmente usando produtos, serviços gerenciados e ferramentas totalmente disponíveis no mercado. Os engenheiros de dados tipo A trabalham em empresas de todos os setores e em todos os níveis de maturidade de dados.

Engenheiros de dados tipo B

Bapoiar construir. Os engenheiros de dados do Tipo B constroem ferramentas e sistemas de dados que escalam e alavancam a principal competência e vantagem competitiva de uma empresa. Na faixa de maturidade de dados, um engenheiro de dados tipo B é mais comumente encontrado em empresas nos estágios 2 e 3 (dimensionando e liderando com dados), ou quando um caso inicial de uso de dados é tão único e crítico que ferramentas de dados personalizadas são necessárias para começar.

Engenheiros de dados tipo A e tipo B podem trabalhar na mesma empresa e podem até ser a mesma pessoa! Mais comumente, um engenheiro de dados tipo A é contratado primeiro para definir a base, com conjuntos de habilidades de engenheiro de dados tipo B aprendidos ou contratados conforme a necessidade surge dentro de uma empresa.

Engenheiros de dados dentro de uma organização

Os engenheiros de dados não trabalham no vácuo. Dependendo do que estiverem trabalhando, eles interagirão com pessoas técnicas e não técnicas e enfrentarão diferentes direções (internas e externas). Vamos explorar o que os engenheiros de dados fazem dentro de uma organização e com quem eles interagem.

10 Robert Chang, "Fazendo ciência de dados no Twitter," *Médio*, 20 de junho de 2015, <https://oreil.ly/xqjAx>.

Engenheiros de dados internos versus externos

Um engenheiro de dados atende a vários usuários finais e enfrenta muitas direções internas e externas (Figura 1-9). Como nem todas as cargas de trabalho e responsabilidades da engenharia de dados são iguais, é essencial entender a quem o engenheiro de dados atende. Dependendo dos casos de uso final, as principais responsabilidades de um engenheiro de dados são externas, internas ou uma mistura das duas.

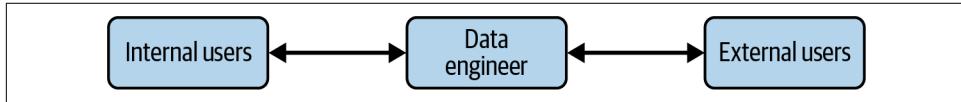


Figura 1-9. As direções que um engenheiro de dados enfrenta

Um *voltado para o exterior* O engenheiro de dados normalmente se alinha com os usuários de aplicativos voltados para o exterior, como aplicativos de mídia social, dispositivos de Internet das Coisas (IoT) e plataformas de comércio eletrônico. Este engenheiro de dados arquiteta, constrói e gerencia os sistemas que coletam, armazenam e processam dados transacionais e de eventos desses aplicativos. Os sistemas criados por esses engenheiros de dados têm um loop de feedback do aplicativo para o pipeline de dados e, em seguida, de volta ao aplicativo (Figura 1-10).

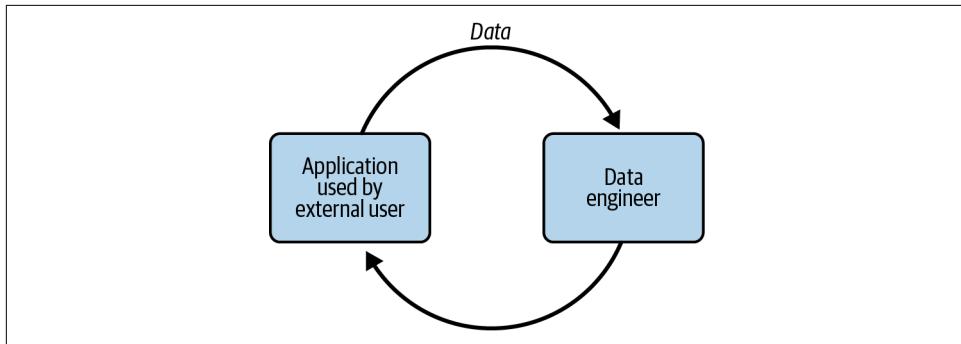


Figura 1-10. Sistemas de engenharia de dados voltados para o exterior

A engenharia de dados externa vem com um conjunto único de problemas. Os mecanismos de consulta externos geralmente lidam com cargas de simultaneidade muito maiores do que os sistemas internos. Os engenheiros também precisam considerar colocar limites rígidos nas consultas que os usuários podem executar para limitar o impacto na infraestrutura de qualquer usuário individual. Além disso, a segurança é um problema muito mais complexo e sensível para consultas externas, especialmente se os dados consultados forem multilocatários (dados de muitos clientes e alojados em uma única tabela).

Um *engenheiro de dados interno* normalmente se concentra em atividades cruciais para as necessidades do negócio e das partes interessadas internas (Figura 1-11). Exemplos incluem a criação e

manutenção de pipelines de dados e data warehouses para painéis de BI, relatórios, processos de negócios, ciência de dados e modelos de ML.

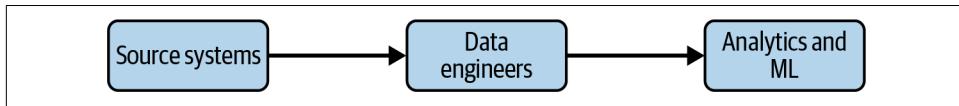


Figura 1-11. Engenheiro de dados interno

As responsabilidades externas e internas geralmente são combinadas. Na prática, os dados voltados para o interior geralmente são um pré-requisito para os dados voltados para o exterior. O engenheiro de dados tem dois conjuntos de usuários com requisitos muito diferentes para simultaneidade de consulta, segurança e muito mais.

Engenheiros de dados e outras funções técnicas

Na prática, o ciclo de vida da engenharia de dados abrange muitos domínios de responsabilidade. Os engenheiros de dados estão no nexo de várias funções, diretamente ou por meio de gerentes, interagindo com muitas unidades organizacionais.

Vejamos quem um engenheiro de dados pode impactar. Nesta seção, discutiremos as funções técnicas relacionadas à engenharia de dados (Figura 1-12).

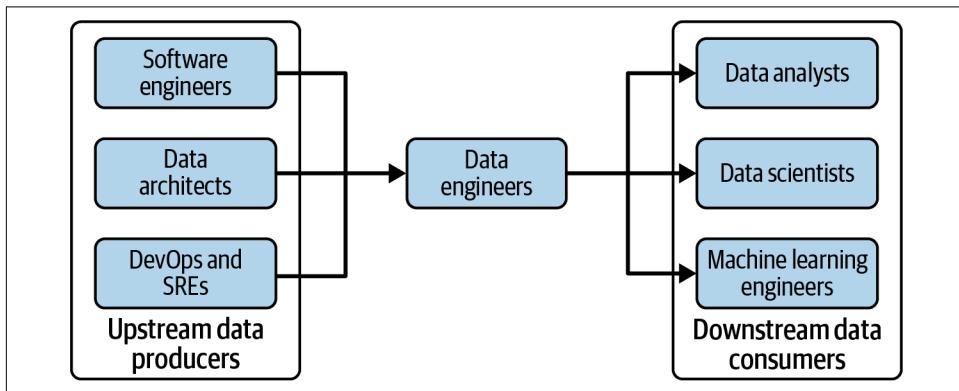


Figura 1-12. Principais partes interessadas técnicas da engenharia de dados

O engenheiro de dados é um hub entre *reprodutores de dados*, como engenheiros de software, arquitetos de dados e DevOps ou engenheiros de confiabilidade do site (SREs) e *consumidores de dados*, como analistas de dados, cientistas de dados e engenheiros de ML. Além disso, os engenheiros de dados irão interagir com aqueles em funções operacionais, como engenheiros de DevOps.

Dado o ritmo em que novas funções de dados estão em voga (analítica e engenheiros de ML vêm à mente), esta não é uma lista exaustiva.

partes interessadas a montante

Para ter sucesso como engenheiro de dados, você precisa entender a arquitetura de dados que está usando ou projetando e os sistemas de origem que produzem os dados necessários. Em seguida, discutimos algumas partes interessadas upstream conhecidas: arquitetos de dados, engenheiros de software e engenheiros de DevOps.

Arquitetos de dados. Os arquitetos de dados funcionam em um nível de abstração um passo distante dos engenheiros de dados. Os arquitetos de dados projetam o projeto para gerenciamento de dados organizacionais, mapeando processos e arquitetura e sistemas de dados gerais.¹¹ Eles também servem como uma ponte entre os lados técnicos e não técnicos de uma organização. Os arquitetos de dados bem-sucedidos geralmente têm “cicatrizes de batalha” de uma extensa experiência em engenharia, permitindo que eles orientem e auxiliem os engenheiros enquanto comunicam com sucesso os desafios de engenharia para as partes interessadas não técnicas do negócio.

Os arquitetos de dados implementam políticas para gerenciar dados entre silos e unidades de negócios, orientam estratégias globais, como gerenciamento e governança de dados, e orientam iniciativas significativas. Os arquitetos de dados geralmente desempenham um papel central nas migrações de nuvem e no design de nuvem totalmente novo.

O advento da nuvem mudou a fronteira entre arquitetura de dados e engenharia de dados. As arquiteturas de dados em nuvem são muito mais fluidas do que os sistemas locais, portanto, as decisões de arquitetura que tradicionalmente envolviam estudo extensivo, longos prazos de entrega, contratos de compra e instalação de hardware agora são frequentemente feitas durante o processo de implementação, apenas uma etapa em uma estratégia maior. No entanto, os arquitetos de dados continuarão sendo visionários influentes nas empresas, trabalhando lado a lado com os engenheiros de dados para determinar o panorama geral das práticas de arquitetura e estratégias de dados.

Dependendo da maturidade e tamanho dos dados da empresa, um engenheiro de dados pode se sobrepor ou assumir as responsabilidades de um arquiteto de dados. Portanto, um engenheiro de dados deve ter um bom entendimento das melhores práticas e abordagens de arquitetura.

Observe que colocamos os arquitetos de dados *nas partes interessadas a montante*. Os arquitetos de dados geralmente ajudam a projetar camadas de dados de aplicativos que são sistemas de origem para engenheiros de dados. Os arquitetos também podem interagir com engenheiros de dados em vários outros estágios do ciclo de vida da engenharia de dados. Cobrimos a “boa” arquitetura de dados em [Capítulo 3](#).

Engenheiros de software. Os engenheiros de software constroem o software e os sistemas que administram um negócio; são os grandes responsáveis pela geração de dados internos que os engenheiros de dados irão consumir e processar. Os sistemas construídos por engenheiros de software geralmente geram logs e dados de eventos de aplicativos, que são ativos significativos em si mesmos.

11 Paramita (Guha) Ghosh, “Data Architect vs. Data Engineer,” Dataversity, 12 de novembro de 2021, <https://oreil.ly/TlyZY>.

certo. Esses dados internos contrastam com *dados externos* extraídos de plataformas SaaS ou empresas parceiras. Em organizações técnicas bem administradas, engenheiros de software e engenheiros de dados coordenam desde o início de um novo projeto para projetar dados de aplicativos para consumo por aplicativos analíticos e de ML.

Um engenheiro de dados deve trabalhar em conjunto com engenheiros de software para entender os aplicativos que geram dados, o volume, a frequência e o formato dos dados gerados e qualquer outra coisa que afete o ciclo de vida da engenharia de dados, como segurança de dados e conformidade regulatória. Por exemplo, isso pode significar definir expectativas iniciais sobre o que os engenheiros de software de dados precisam para realizar seus trabalhos. Os engenheiros de dados devem trabalhar em estreita colaboração com os engenheiros de software.

Engenheiros de DevOps e engenheiros de confiabilidade do site. DevOps e SREs geralmente produzem dados através do monitoramento operacional. Nós os classificamos como upstream de engenheiros de dados, mas também podem ser downstream, consumindo dados por meio de painéis ou interagindo com engenheiros de dados diretamente na coordenação de operações de sistemas de dados.

partes interessadas a jusante

A engenharia de dados existe para atender consumidores de dados downstream e casos de uso. Esta seção discute como os engenheiros de dados interagem com várias funções downstream. Também apresentaremos alguns modelos de serviço, incluindo equipes centralizadas de engenharia de dados e equipes multifuncionais.

Cientistas de dados. Os cientistas de dados criam modelos prospectivos para fazer previsões e recomendações. Esses modelos são então avaliados em dados ao vivo para fornecer valor de várias maneiras. Por exemplo, a pontuação do modelo pode determinar ações automatizadas em resposta a condições em tempo real, recomendar produtos a clientes com base no histórico de navegação em sua sessão atual ou fazer previsões econômicas ao vivo usadas pelos comerciantes.

De acordo com o folclore comum da indústria, os cientistas de dados gastam de 70% a 80% de seu tempo coletando, limpando e preparando dados.¹² Em nossa experiência, esses números geralmente refletem práticas imaturas de ciência de dados e engenharia de dados. Em particular, muitas estruturas populares de ciência de dados podem se tornar gargalos se não forem dimensionadas adequadamente. Os cientistas de dados que trabalham exclusivamente em uma única estação de trabalho se forçam a reduzir a amostra dos dados, tornando a preparação dos dados significativamente mais complicada e potencialmente comprometendo a qualidade dos modelos que eles usam.

12 Existe uma variedade de referências para esta noção. Embora esse clichê seja amplamente conhecido, um debate saudável surgiu em torno de sua validade em diferentes contextos práticos. Para obter mais detalhes, consulte Leigh Dodds, "Do Data Scientists Spend 80% of Their Time Cleaning Data? Acontece que não? Blog Lost Boy, 31 de janeiro de 2020, <https://oreil.ly/szFww>; e Alex Woodie, "A preparação de dados ainda domina o tempo dos cientistas de dados, resultados de pesquisas," *datanami*, 6 de julho de 2020, <https://oreil.ly/jDVWF>.

produzir. Além disso, códigos e ambientes desenvolvidos localmente costumam ser difíceis de implantar na produção, e a falta de automação dificulta significativamente os fluxos de trabalho da ciência de dados. **Se os engenheiros de dados fazem seu trabalho e colaboram com sucesso, os cientistas de dados não devem gastar seu tempo coletando, limpando e preparando dados após o trabalho exploratório inicial.** Os engenheiros de dados devem automatizar esse trabalho o máximo possível.

A necessidade de ciência de dados pronta para produção é um fator significativo por trás do surgimento da profissão de engenharia de dados. Os engenheiros de dados devem ajudar os cientistas de dados a habilitar um caminho para a produção. Na verdade, nós (os autores) passamos da ciência de dados para a engenharia de dados depois de reconhecer essa necessidade fundamental. Os engenheiros de dados trabalham para fornecer a automação e a escala de dados que tornam a ciência de dados mais eficiente.

Analistas de dados. Os analistas de dados (ou analistas de negócios) procuram entender o desempenho e as tendências dos negócios. Enquanto os cientistas de dados estão voltados para o futuro, um analista de dados normalmente se concentra no passado ou no presente. Os analistas de dados geralmente executam consultas SQL em um data warehouse ou data lake. Eles também podem utilizar planilhas para computação e análise e várias ferramentas de BI, como Microsoft Power BI, Looker ou Tableau. Os analistas de dados são especialistas no domínio dos dados com os quais trabalham com frequência e se familiarizam intimamente com as definições, características e problemas de qualidade dos dados. Os clientes downstream típicos de um analista de dados são usuários de negócios, gerenciamento e executivos.

Os engenheiros de dados trabalham com analistas de dados para criar pipelines para novas fontes de dados exigidas pelos negócios. A experiência no assunto dos analistas de dados é inestimável para melhorar a qualidade dos dados, e eles frequentemente colaboram com os engenheiros de dados nessa função.

Engenheiros de aprendizado de máquina e pesquisadores de IA. Engenheiros de aprendizado de máquina (engineers) se sobrepõem a engenheiros de dados e cientistas de dados. Os engenheiros de ML desenvolvem técnicas avançadas de ML, treinam modelos e projetam e mantêm a infraestrutura executando processos de ML em um ambiente de produção em escala. Os engenheiros de ML geralmente têm conhecimento prático avançado de ML e técnicas e estruturas de aprendizado profundo, como PyTorch ou TensorFlow.

Os engenheiros de ML também entendem o hardware, os serviços e os sistemas necessários para executar essas estruturas, tanto para treinamento de modelo quanto para implantação de modelo em escala de produção. É comum que os fluxos de ML sejam executados em um ambiente de nuvem, onde os engenheiros de ML podem ativar e dimensionar recursos de infraestrutura sob demanda ou contar com serviços gerenciados.

Como mencionamos, os limites entre engenharia de ML, engenharia de dados e ciência de dados são confusos. Os engenheiros de dados podem ter algumas responsabilidades operacionais sobre os sistemas de ML, e os cientistas de dados podem trabalhar em estreita colaboração com a engenharia de ML na criação de processos avançados de ML.

O mundo da engenharia de ML é uma bola de neve e acompanha muitos dos mesmos desenvolvimentos que ocorrem na engenharia de dados. Enquanto há vários anos a atenção do ML estava focada em como construir modelos, a engenharia de ML agora enfatiza cada vez mais a incorporação das melhores práticas de operações de aprendizado de máquina (MLOps) e outras práticas maduras anteriormente adotadas em engenharia de software e DevOps.

Pesquisadores de IA trabalham em novas e avançadas técnicas de ML. Os pesquisadores de IA podem trabalhar em grandes empresas de tecnologia, startups especializadas em propriedade intelectual (OpenAI, DeepMind) ou instituições acadêmicas. Alguns profissionais se dedicam à pesquisa em meio período em conjunto com as responsabilidades de engenharia de ML dentro de uma empresa. Quem trabalha em laboratórios especializados de ML costuma se dedicar 100% à pesquisa. Os problemas de pesquisa podem ter como alvo aplicações práticas imediatas ou demonstrações mais abstratas de IA. DALL-E, Gato AI, AlphaGo e GPT-3/GPT-4 são ótimos exemplos de projetos de pesquisa de ML. Dado o ritmo dos avanços no ML, esses exemplos provavelmente serão estranhos daqui a alguns anos. Fornecemos algumas referências em “[Recursos adicionais](#)” na página 32.

Pesquisadores de IA em organizações bem financiadas são altamente especializados e operam com equipes de suporte de engenheiros para facilitar seu trabalho. Os engenheiros de ML na academia geralmente têm menos recursos, mas contam com equipes de alunos de pós-graduação, pós-doutorandos e funcionários da universidade para fornecer suporte de engenharia. Os engenheiros de ML que se dedicam parcialmente à pesquisa geralmente contam com as mesmas equipes de suporte para pesquisa e produção.

Engenheiros de dados e liderança empresarial

Discutimos as funções técnicas com as quais um engenheiro de dados interage. Mas os engenheiros de dados também operam de forma mais ampla como conectores organizacionais, muitas vezes em uma capacidade não técnica. As empresas passaram a depender cada vez mais dos dados como parte central de muitos produtos ou de um produto em si. Os engenheiros de dados agora participam do planejamento estratégico e lideram as principais iniciativas que se estendem além dos limites da TI. Os engenheiros de dados geralmente oferecem suporte aos arquitetos de dados, atuando como a cola entre os negócios e a ciência/análise de dados.

Dados no C-suite

Os executivos de nível C estão cada vez mais envolvidos em dados e análises, pois são reconhecidos como ativos significativos para as empresas modernas. Por exemplo, os CEOs agora se preocupam com iniciativas que antes eram domínio exclusivo da TI, como migrações para a nuvem ou implantação de uma nova plataforma de dados do cliente.

Diretor executivo. Os diretores executivos (CEOs) em empresas não tecnológicas geralmente não se preocupam com o âmago da questão de estruturas de dados e software. Em vez disso, eles definem uma visão em colaboração com funções técnicas de C-suite e liderança de dados da empresa. Os engenheiros de dados fornecem uma janela para o que é possível com

dados. Os engenheiros de dados e seus gerentes mantêm um mapa de quais dados estão disponíveis para a organização - tanto internamente quanto de terceiros - em que período de tempo. Eles também têm a tarefa de estudar as alterações arquitetônicas de dados primários em colaboração com outras funções de engenharia. Por exemplo, os engenheiros de dados geralmente estão fortemente envolvidos em migrações de nuvem, migrações para novos sistemas de dados ou implantação de tecnologias de streaming.

Diretor de informações. Um chief information officer (CIO) é o executivo sênior C-suite responsável pela tecnologia da informação dentro de uma organização; é um papel voltado para o interior. Um CIO deve possuir um conhecimento profundo de tecnologia da informação e processos de negócios — ambos sozinhos são insuficientes. Os CIOs dirigem a organização de tecnologia da informação, estabelecendo políticas contínuas ao mesmo tempo em que definem e executam iniciativas significativas sob a direção do CEO.

Os CIOs geralmente colaboram com a liderança da engenharia de dados em organizações com uma cultura de dados bem desenvolvida. Se uma organização não for muito alta em sua maturidade de dados, um CIO normalmente ajudará a moldar sua cultura de dados. Os CIOs trabalharão com engenheiros e arquitetos para mapear as principais iniciativas e tomar decisões estratégicas sobre a adoção dos principais elementos de arquitetura, como sistemas de planejamento de recursos empresariais (ERP) e gerenciamento de relacionamento com o cliente (CRM), migrações para a nuvem, sistemas de dados e sistemas internos. voltado para TI.

Diretor de tecnologia. Um diretor de tecnologia (CTO) é semelhante a um CIO, mas voltado para fora. Um CTO possui a estratégia tecnológica e as arquiteturas principais para aplicativos voltados para o exterior, como dispositivos móveis, aplicativos da Web e IoT - todas as fontes de dados essenciais para engenheiros de dados. O CTO provavelmente é um tecnólogo qualificado e tem um bom senso dos fundamentos da engenharia de software e da arquitetura do sistema. Em algumas organizações sem um CIO, o CTO ou às vezes o diretor de operações (COO) desempenha o papel de CIO. Os engenheiros de dados geralmente se reportam direta ou indiretamente por meio de um CTO.

Diretor de dados. O chief data officer (CDO) foi criado em 2002 na Capital One para reconhecer a crescente importância dos dados como um ativo comercial. O CDO é responsável pelos ativos de dados e pela estratégia de uma empresa. Os CDOs concentram-se na utilidade comercial dos dados, mas devem ter uma sólida base técnica. Os CDOs supervisionam produtos de dados, estratégias, iniciativas e funções essenciais, como gerenciamento de dados mestre e privacidade. Ocionalmente, CDOs gerenciam análise de negócios e engenharia de dados.

Diretor de análise. O chief analytics officer (CAO) é uma variante da função de CDO. Onde existem ambas as funções, o CDO se concentra na tecnologia e na organização necessárias para fornecer dados. O CAO é responsável pela análise, estratégia e tomada de decisões para o negócio. Um CAO pode supervisionar a ciência de dados e ML, embora isso dependa em grande parte se a empresa tem uma função de CDO ou CTO.

Diretor de algoritmos. Um diretor de algoritmos (CAO-2) é uma inovação recente no C-suite, uma função altamente técnica focada especificamente em ciência de dados e ML. Os CAO-2s normalmente têm experiência como colaboradores individuais e líderes de equipe em ciência de dados ou projetos de ML. Freqüentemente, eles têm experiência em pesquisa de ML e um grau avançado relacionado.

Espera-se que os CAO-2 estejam familiarizados com a pesquisa atual de ML e tenham profundo conhecimento técnico das iniciativas de ML de sua empresa. Além de criar iniciativas de negócios, eles fornecem liderança técnica, definem agendas de pesquisa e desenvolvimento e formam equipes de pesquisa.

Engenheiros de dados e gerentes de projeto

Os engenheiros de dados geralmente trabalham em iniciativas significativas, que podem durar muitos anos. Enquanto escrevemos este livro, muitos engenheiros de dados estão trabalhando em migrações para a nuvem, migrando pipelines e warehouses para a próxima geração de ferramentas de dados. Outros engenheiros de dados estão iniciando projetos novos, montando novas arquiteturas de dados a partir do zero, selecionando entre um número surpreendente de opções de arquitetura e ferramentas de ponta.

Essas grandes iniciativas geralmente se beneficiam de *gerenciamento de projetos* (em contraste com o gerenciamento de produtos, discutido a seguir). Enquanto os engenheiros de dados atuam em uma infraestrutura e capacidade de entrega de serviços, os gerentes de projeto direcionam o tráfego e atuam como guardiões. A maioria dos gerentes de projeto opera de acordo com alguma variação do Agile e do Scrum, com Waterfall ainda aparecendo ocasionalmente. Os negócios nunca dormem, e as partes interessadas nos negócios geralmente têm um acúmulo significativo de coisas que desejam resolver e novas iniciativas que desejam lançar. Os gerentes de projeto devem filtrar uma longa lista de solicitações e priorizar entregas críticas para manter os projetos em andamento e atender melhor a empresa.

Os engenheiros de dados interagem com os gerentes de projeto, muitas vezes planejando sprints para projetos e realizando reuniões relacionadas ao sprint. O feedback ocorre nos dois sentidos, com os engenheiros de dados informando os gerentes de projeto e outras partes interessadas sobre o progresso e os bloqueadores, e os gerentes de projeto equilibrando a cadênciia das equipes de tecnologia com as necessidades em constante mudança dos negócios.

Engenheiros de dados e gerentes de produto

Os gerentes de produto supervisionam o desenvolvimento de produtos, geralmente possuindo linhas de produtos. No contexto dos engenheiros de dados, esses produtos são chamados *produtos de dados*. Os produtos de dados são criados desde o início ou são melhorias incrementais em produtos existentes. Os engenheiros de dados interagem com mais frequência com *gerentes de produto* como o mundo corporativo adotou um foco centrado em dados. Assim como os gerentes de projeto, os gerentes de produto equilibram a atividade das equipes de tecnologia com as necessidades do cliente e do negócio.

Engenheiros de dados e outras funções de gerenciamento

Os engenheiros de dados interagem com vários gerentes além dos gerentes de projeto e produto. No entanto, essas interações geralmente seguem os modelos de serviços ou multifuncionais. Os engenheiros de dados atendem a uma variedade de solicitações recebidas como uma equipe centralizada ou trabalham como um recurso atribuído a um determinado gerente, projeto ou produto.

Para obter mais informações sobre equipes de dados e como estruturá-las, recomendamos o artigo de John Thompson *Construindo equipes de análise*(Packt) e Jesse Anderson *Equipes de dados* (Apresse). Ambos os livros fornecem estruturas e perspectivas sólidas sobre as funções dos executivos com dados, quem contratar e como construir a equipe de dados mais eficaz para sua empresa.



As empresas não contratam engenheiros simplesmente para hackear códigos isoladamente. Para serem dignos de seu título, os engenheiros devem desenvolver uma compreensão profunda dos problemas que devem resolver, das ferramentas tecnológicas à sua disposição e das pessoas com quem trabalham e atendem.

Conclusão

Este capítulo forneceu uma breve visão geral do cenário da engenharia de dados, incluindo o seguinte:

- Definir a engenharia de dados e descrever o que os engenheiros de dados fazem
- Descrever os tipos de maturidade de dados em uma empresa
- Engenheiros de dados tipo A e tipo B
- Com quem os engenheiros de dados trabalham

Esperamos que este primeiro capítulo tenha aguçado seu apetite, seja você um profissional de desenvolvimento de software, cientista de dados, engenheiro de ML, parte interessada nos negócios, empresário ou capitalista de risco. Claro, ainda há muito a ser elucidado nos capítulos subsequentes. [Capítulo 2](#)cobre o ciclo de vida da engenharia de dados, seguido pela arquitetura em[Capítulo 3](#). Os capítulos a seguir abordam o âmago da questão das decisões tecnológicas para cada parte do ciclo de vida. Todo o campo de dados está em fluxo e, tanto quanto possível, cada capítulo se concentra no/*imutáveis*—perspectivas que serão válidas por muitos anos em meio a mudanças implacáveis.

Recursos adicionais

- “A hierarquia de necessidades da IA” por Mônica Rogati
- A página web de pesquisa AlphaGo
- “Big Data estará morto em cinco anos” por Lewis Gavin
- *Construindo equipes de análise* por John K. Thompson (Packt)
- Capítulo 1 de *O que é Engenharia de Dados?* por Lewis Gavin (O'Reilly)
- “Dados como um produto x Dados como um serviço” por Justin Gage
- “Engenharia de dados: uma definição rápida e simples” por James Furbush (O'Reilly)
- *Equipes de dados* por Jesse Anderson (Apress)
- “Fazendo ciência de dados no Twitter” por Robert Chang
- “A Queda do Engenheiro de Dados” por Maxime Beauchemin
- “O Futuro da Engenharia de Dados é a Convergência das Disciplinas” por Liam Hausmann
- “Como os CEOs podem liderar uma cultura orientada por dados” por Thomas H. Davenport e Nitin Mittal
- “Como a criação de uma cultura orientada por dados pode gerar sucesso” por Frederik Bussler
- O site do Corpo de Conhecimento de Gerenciamento de Informações
- Página da Wikipédia “Gestão de Informações sobre o Conhecimento”
- Página da Wikipédia “Gerenciamento de Informações”
- “Sobre a Complexidade em Big Data” por Jesse Anderson (O'Reilly)
- “O novo gerador de linguagem GPT-3 da OpenAI é surpreendentemente bom — e completamente estúpido” por Will Douglas Heaven
- “A Ascensão do Engenheiro de Dados” por Maxime Beauchemin
- “Uma Breve História do Big Data” por Mark van Rijmenam
- “Habilidades do Arquiteto de Dados” por Bob Lambert
- “Os três níveis de análise de dados: uma estrutura para avaliar a maturidade da organização de dados” por Emilie Schario
- “O que é um arquiteto de dados? Visionário do Data Framework de TI” por Thor Olavsrud
- “Qual profissão é mais complexa para se tornar, um engenheiro de dados ou um cientista de dados?” tópico no Quora
- “Por que os CEOs devem liderar iniciativas de Big Data” por John Weathington

O ciclo de vida da engenharia de dados

O principal objetivo deste livro é encorajá-lo a ir além da visualização da engenharia de dados como uma coleção específica de tecnologias de dados. O cenário de dados está passando por uma explosão de novas tecnologias e práticas de dados, com níveis cada vez maiores de abstração e facilidade de uso. Devido ao aumento da abstração técnica, os engenheiros de dados se tornarão cada vez mais *engenheiros de ciclo de vida de dados*, pensando e operando em termos de *princípios de gerenciamento do ciclo de vida dos dados*.

Neste capítulo, você aprenderá sobre *o ciclo de vida da engenharia de dados*, que é o tema central deste livro. O ciclo de vida da engenharia de dados é nossa estrutura que descreve a engenharia de dados “do berço ao túmulo”. Você também aprenderá sobre as subcorrentes do ciclo de vida da engenharia de dados, que são os principais fundamentos que dão suporte a todos os esforços de engenharia de dados.

O que é o ciclo de vida da engenharia de dados?

O ciclo de vida da engenharia de dados compreende estágios que transformam ingredientes de dados brutos em um produto final útil, pronto para consumo por analistas, cientistas de dados, engenheiros de ML e outros. Este capítulo apresenta os principais estágios do ciclo de vida da engenharia de dados, concentrando-se nos principais conceitos de cada estágio e guardando os detalhes para capítulos posteriores.

Dividimos o ciclo de vida da engenharia de dados em cinco estágios ([Figura 2-1, principal](#)):

- Geração
- Armazenar
- Ingestão
- Transformação
- Dados de serviço

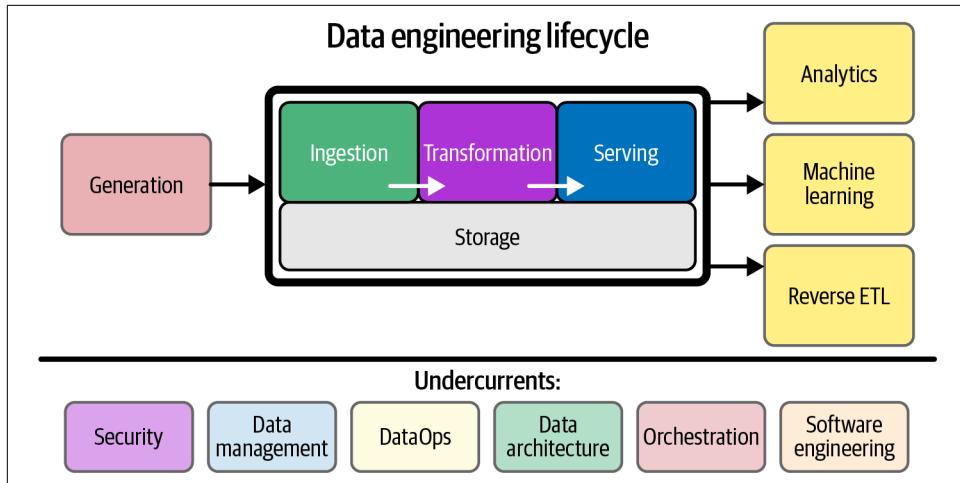


Figura 2-1. Componentes e subcorrentes do ciclo de vida da engenharia de dados

Começamos o ciclo de vida da engenharia de dados obtendo dados dos sistemas de origem e armazenando-os. Em seguida, transformamos os dados e prosseguimos para nosso objetivo central, fornecendo dados para analistas, cientistas de dados, engenheiros de ML e outros. Na realidade, o armazenamento ocorre durante todo o ciclo de vida à medida que os dados fluem do começo ao fim – portanto, o diagrama mostra o “estágio” de armazenamento como uma base que sustenta outros estágios.

Em geral, os estágios intermediários – armazenamento, ingestão, transformação – podem ficar um pouco confusos. E tudo bem. Embora dividamos as partes distintas do ciclo de vida da engenharia de dados, nem sempre é um fluxo contínuo e organizado. Vários estágios do ciclo de vida podem se repetir, ocorrer fora de ordem, se sobrepor ou se entrelaçar de maneiras interessantes e inesperadas.

Atuando como um alicerce são *correntes ocultas* (Figura 2-1, parte inferior) que atravessam vários estágios do ciclo de vida da engenharia de dados: segurança, gerenciamento de dados, DataOps, arquitetura de dados, orquestração e engenharia de software. Nenhuma parte do ciclo de vida da engenharia de dados pode funcionar adequadamente sem essas subcorrentes.

O ciclo de vida dos dados versus o ciclo de vida da engenharia de dados

Você pode estar se perguntando sobre a diferença entre o ciclo de vida geral dos dados e o ciclo de vida da engenharia de dados. Há uma distinção sutil entre os dois. O ciclo de vida da engenharia de dados é um subconjunto de todo o ciclo de vida dos dados (Figura 2-2). Enquanto o ciclo de vida completo dos dados abrange dados em toda a sua vida útil, o ciclo de vida da engenharia de dados se concentra nos estágios que um engenheiro de dados controla.

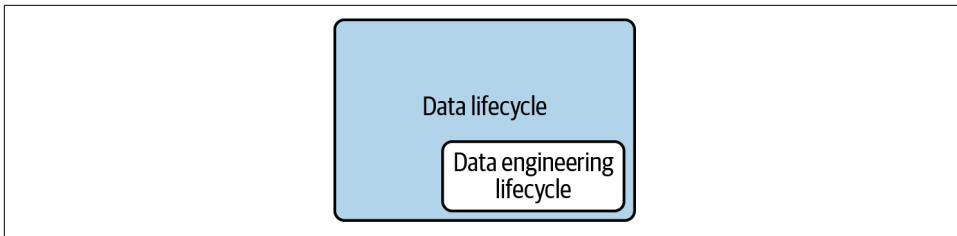


Figura 2-2. O ciclo de vida da engenharia de dados é um subconjunto do ciclo de vida completo dos dados

Geração: Sistemas de Origem

As *sistemas de origem* é a origem dos dados usados no ciclo de vida da engenharia de dados. Por exemplo, um sistema de origem pode ser um dispositivo IoT, uma fila de mensagens de aplicativo ou um banco de dados transacional. Um engenheiro de dados consome dados de um sistema de origem, mas normalmente não possui ou controla o próprio sistema de origem. O engenheiro de dados precisa ter uma compreensão prática de como os sistemas de origem funcionam, como eles geram dados, a frequência e a velocidade dos dados e a variedade de dados que eles geram.

Os engenheiros também precisam manter uma linha de comunicação aberta com os proprietários do sistema de origem sobre alterações que possam interromper pipelines e análises. O código do aplicativo pode alterar a estrutura dos dados em um campo ou a equipe do aplicativo pode até optar por migrar o back-end para uma tecnologia de banco de dados totalmente nova.

Um grande desafio na engenharia de dados é a variedade estonteante de sistemas de fonte de dados com os quais os engenheiros devem trabalhar e entender. Como ilustração, vejamos dois sistemas de origem comuns, um muito tradicional (um banco de dados de aplicativo) e outro um exemplo mais recente (enxames de IoT).

Figura 2-3 ilustra um sistema de origem tradicional com vários servidores de aplicativos suportados por um banco de dados. Esse padrão de sistema de origem tornou-se popular na década de 1980 com o sucesso explosivo dos sistemas de gerenciamento de banco de dados relacional (RDBMSs). O padrão aplicativo + banco de dados permanece popular hoje em dia com várias evoluções modernas de práticas de desenvolvimento de software. Por exemplo, os aplicativos geralmente consistem em muitos pares pequenos de serviço/banco de dados com microsserviços, em vez de um único monólito.

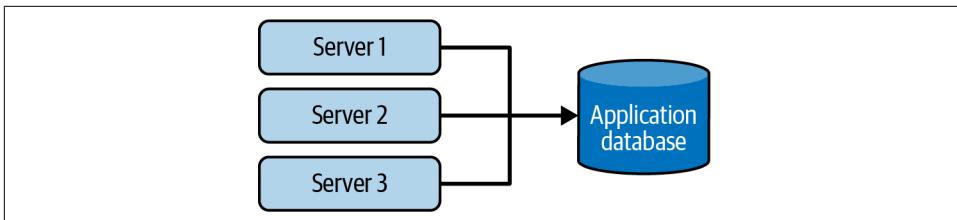


Figura 2-3. Exemplo de sistema de origem: um banco de dados de aplicativo

Vejamos outro exemplo de um sistema de origem. Figura 2-4 ilustra um enxame IoT: uma frota de dispositivos (círculos) envia mensagens de dados (retângulos) para um sistema de coleta central. Esse sistema de origem IoT é cada vez mais comum como dispositivos IoT, como sensores, dispositivos inteligentes e muito mais aumento na natureza.

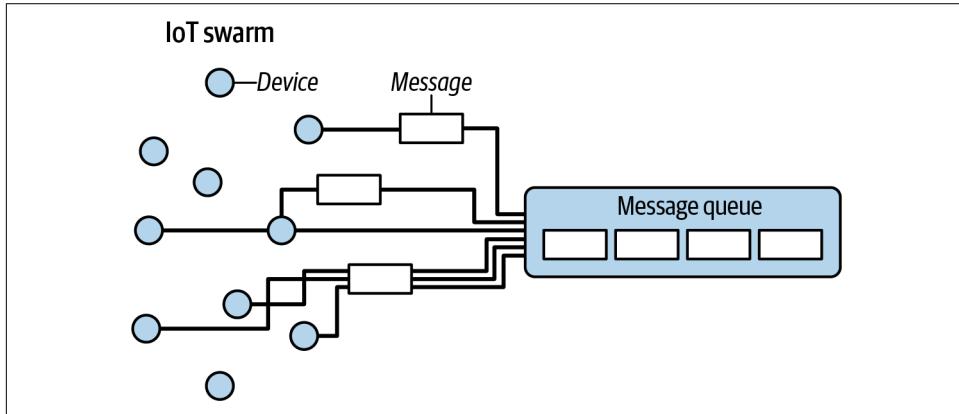


Figura 2-4. Exemplo de sistema de origem: um enxame IoT e fila de mensagens

Avaliação de sistemas de origem: principais considerações de engenharia

Há muitas coisas a serem consideradas ao avaliar os sistemas de origem, incluindo como o sistema lida com a ingestão, o estado e a geração de dados. A seguir, um conjunto inicial de perguntas de avaliação de sistemas de origem que os engenheiros de dados devem considerar:

- Quais são as características essenciais da fonte de dados? É um aplicativo? Um enxame de dispositivos IoT?
- Como os dados são mantidos no sistema de origem? Os dados são mantidos a longo prazo ou são temporários e rapidamente excluídos?
- Em que taxa os dados são gerados? Quantos eventos por segundo? Quantos gigabytes por hora?
- Que nível de consistência os engenheiros de dados podem esperar dos dados de saída? Se você estiver executando verificações de qualidade de dados nos dados de saída, com que frequência ocorrem inconsistências de dados - nulos onde não são esperados, formatação ruim etc.?
- Com que frequência ocorrem erros?
- Os dados conterão duplicatas?
- Alguns valores de dados chegarão atrasados, possivelmente muito mais tarde do que outras mensagens produzidas simultaneamente?
- Qual é o esquema dos dados ingeridos? Os engenheiros de dados precisarão unir várias tabelas ou mesmo vários sistemas para obter uma imagem completa dos dados?

- Se o esquema mudar (digamos, uma nova coluna for adicionada), como isso será tratado e comunicado às partes interessadas a jusante?
- Com que frequência os dados devem ser extraídos do sistema de origem?
- Para sistemas stateful (por exemplo, um banco de dados que rastreia as informações da conta do cliente), os dados são fornecidos como instantâneos periódicos ou eventos de atualização da captura de dados de alteração (CDC)? Qual é a lógica de como as alterações são executadas e como elas são rastreadas no banco de dados de origem?
- Quem/o que é o provedor de dados que transmitirá os dados para consumo downstream?
- A leitura de uma fonte de dados afetará seu desempenho?
- O sistema de origem possui dependências de dados upstream? Quais são as características desses sistemas upstream?
- Existem verificações de qualidade de dados para verificar se há dados atrasados ou ausentes?

As fontes produzem dados consumidos por sistemas downstream, incluindo planilhas geradas por humanos, sensores de IoT e aplicativos da Web e móveis. Cada fonte tem seu volume e cadências únicas de geração de dados. Um engenheiro de dados deve saber como a fonte gera dados, incluindo peculiaridades ou nuances relevantes. Os engenheiros de dados também precisam entender os limites dos sistemas de origem com os quais interagem. Por exemplo, as consultas analíticas em um banco de dados de aplicativo de origem causarão problemas de contenção de recursos e desempenho?

Uma das nuances mais desafiadoras dos dados de origem é o esquema. O *esquema* define a organização hierárquica dos dados. Logicamente, podemos pensar nos dados no nível de todo um sistema de origem, detalhando tabelas individuais até a estrutura dos respectivos campos. O esquema de dados enviados dos sistemas de origem é tratado de várias maneiras. Duas opções populares são esquema sem esquema e esquema fixo.

sem esquema não significa ausência de esquema. Em vez disso, significa que o aplicativo define o esquema à medida que os dados são gravados, seja em uma fila de mensagens, em um arquivo simples, em um blob ou em um banco de dados de documentos, como o MongoDB. Um modelo mais tradicional construído em armazenamento de banco de dados relacional usa um *esquema fixo* aplicado no banco de dados, ao qual as gravações do aplicativo devem estar em conformidade.

Qualquer um desses modelos apresenta desafios para engenheiros de dados. Os esquemas mudam com o tempo; na verdade, a evolução do esquema é incentivada na abordagem ágil para o desenvolvimento de software. Uma parte fundamental do trabalho do engenheiro de dados é obter a entrada de dados brutos no esquema do sistema de origem e transformá-los em uma saída valiosa para análises. Esse trabalho se torna mais desafiador à medida que o esquema de origem evolui.

Mergulhamos nos sistemas de origem com mais detalhes em [capítulo 5](#); também abordamos esquemas e modelagem de dados nos [capítulos 6 e 8](#), respectivamente.

Armazenar

Você precisa de um lugar para armazenar dados. A escolha de uma solução de armazenamento é a chave para o sucesso no restante do ciclo de vida dos dados e também é um dos estágios mais complicados do ciclo de vida dos dados por vários motivos. Primeiro, as arquiteturas de dados na nuvem geralmente aproveitam *diversas soluções* de armazenamento. Em segundo lugar, poucas soluções de armazenamento de dados funcionam apenas como armazenamento, com muitas consultas de transformação complexas de suporte; até mesmo as soluções de armazenamento de objetos podem oferecer suporte a recursos de consulta poderosos - por exemplo, [Seleção Amazon S3](#). Em terceiro lugar, embora o armazenamento seja um estágio do ciclo de vida da engenharia de dados, ele frequentemente envolve outros estágios, como ingestão, transformação e serviço.

O armazenamento é executado em todo o ciclo de vida da engenharia de dados, geralmente ocorrendo em vários locais em um pipeline de dados, com sistemas de armazenamento cruzando com sistemas de origem, ingestão, transformação e serviço. De muitas maneiras, a maneira como os dados são armazenados afeta como eles são usados em todos os estágios do ciclo de vida da engenharia de dados. Por exemplo, data warehouses em nuvem podem armazenar dados, processar dados em pipelines e distribuí-los aos analistas. Estruturas de streaming como Apache Kafka e Pulsar podem funcionar simultaneamente como sistemas de ingestão, armazenamento e consulta de mensagens, sendo o armazenamento de objetos uma camada padrão para transmissão de dados.

Avaliação de sistemas de armazenamento: principais considerações de engenharia

Aqui estão algumas perguntas importantes de engenharia a serem feitas ao escolher um sistema de armazenamento para um data warehouse, data lakehouse, banco de dados ou armazenamento de objetos:

- Esta solução de armazenamento é compatível com as velocidades de gravação e leitura exigidas pela arquitetura?
- O armazenamento criará um gargalo para os processos posteriores?
- Você entende como funciona essa tecnologia de armazenamento? Você está utilizando o sistema de armazenamento de forma otimizada ou cometendo erros não naturais? Por exemplo, você está aplicando uma alta taxa de atualizações de acesso aleatório em um sistema de armazenamento de objetos? (Este é um antipadrão com sobrecarga de desempenho significativa.)
- Este sistema de armazenamento lidará com a escala futura prevista? Você deve considerar todos os limites de capacidade do sistema de armazenamento: armazenamento total disponível, taxa de operação de leitura, volume de gravação, etc.
- Os usuários e processos downstream poderão recuperar dados no contrato de nível de serviço (SLA) necessário?
- Você está capturando metadados sobre a evolução do esquema, fluxos de dados, linhagem de dados e assim por diante? Os metadados têm um impacto significativo na utilidade dos dados. Os metadados representam um investimento no futuro, melhorando drasticamente a capacidade de descoberta e o conhecimento institucional para agilizar futuros projetos e mudanças de arquitetura.

- Esta é uma solução de armazenamento puro (armazenamento de objetos) ou oferece suporte a padrões de consulta complexos (ou seja, um data warehouse na nuvem)?
- O esquema do sistema de armazenamento é independente de esquema (armazenamento de objetos)? Esquema flexível (Cassandra)? Esquema imposto (um data warehouse na nuvem)?
- Como você está rastreando dados mestre, qualidade de dados de registros de ouro e linhagem de dados para governança de dados? (Temos mais a dizer sobre isso em "[Gerenciamento de dados](#)" na página 50.)
- Como você está lidando com a conformidade regulamentar e a soberania de dados? Por exemplo, você pode armazenar seus dados em determinadas localizações geográficas, mas não em outras?

Noções básicas sobre a frequência de acesso a dados

Nem todos os dados são acessados da mesma maneira. Os padrões de recuperação variam muito com base nos dados armazenados e consultados. Isso traz a noção das "temperaturas" dos dados. A frequência de acesso aos dados determinará a temperatura dos seus dados.

Os dados acessados com mais frequência são chamados *dados quentes*. Os dados quentes geralmente são recuperados várias vezes por dia, talvez até várias vezes por segundo, por exemplo, em sistemas que atendem a solicitações de usuários. Esses dados devem ser armazenados para recuperação rápida, onde "rápido" é relativo ao caso de uso. *dados mornos* pode ser acessado de vez em quando - digamos, toda semana ou mês.

dados frios raramente é consultado e é apropriado para armazenamento em um sistema de arquivo. Os dados frios geralmente são retidos para fins de conformidade ou em caso de falha catastrófica em outro sistema. Nos "velhos tempos", os dados frios seriam armazenados em fitas e enviados para instalações remotas de arquivamento. Em ambientes de nuvem, os fornecedores oferecem níveis de armazenamento especializados com custos mensais de armazenamento muito baixos, mas preços altos para recuperação de dados.

Selecionando um sistema de armazenamento

Que tipo de solução de armazenamento você deve usar? Isso depende de seus casos de uso, volumes de dados, frequência de ingestão, formato e tamanho dos dados sendo ingeridos essencialmente, as principais considerações listadas nas perguntas com marcadores anteriores. Não há recomendação de armazenamento universal de tamanho único. Toda tecnologia de armazenamento tem suas compensações. Existem inúmeras variedades de tecnologias de armazenamento e é fácil ficar sobrecarregado ao decidir a melhor opção para sua arquitetura de dados.

[Capítulo 6](#) aborda as melhores práticas e abordagens de armazenamento com mais detalhes, bem como o cruzamento entre o armazenamento e outros estágios do ciclo de vida.

Ingestão

Depois de entender a fonte de dados, as características do sistema de origem que você está usando e como os dados são armazenados, você precisa coletar os dados. A próxima etapa do ciclo de vida da engenharia de dados é a ingestão de dados dos sistemas de origem.

Em nossa experiência, os sistemas de origem e a ingestão representam os gargalos mais significativos do ciclo de vida da engenharia de dados. Os sistemas de origem normalmente estão fora de seu controle direto e podem deixar de responder aleatoriamente ou fornecer dados de baixa qualidade. Ou, seu serviço de ingestão de dados pode parar de funcionar misteriosamente por vários motivos. Como resultado, o fluxo de dados para ou fornece dados insuficientes para armazenamento, processamento e entrega.

Sistemas de origem e ingestão não confiáveis têm um efeito cascata em todo o ciclo de vida da engenharia de dados. Mas você está em boa forma, supondo que tenha respondido às grandes questões sobre sistemas de origem.

Principais considerações de engenharia para a fase de ingestão

Ao se preparar para arquitetar ou construir um sistema, aqui estão algumas perguntas principais sobre o estágio de ingestão:

- Quais são os casos de uso dos dados que estou ingerindo? Posso reutilizar esses dados em vez de criar várias versões do mesmo conjunto de dados?
- Os sistemas estão gerando e ingerindo esses dados de forma confiável e os dados estão disponíveis quando preciso deles?
- Qual é o destino dos dados após a ingestão?
- Com que frequência preciso acessar os dados?
- Em que volume os dados normalmente chegarão?
- Em que formato estão os dados? Meus sistemas downstream de armazenamento e transformação podem lidar com esse formato?
- Os dados de origem estão em boas condições para uso imediato a jusante? Em caso afirmativo, por quanto tempo e o que pode causar sua inutilização?
- Se os dados forem de uma fonte de streaming, eles precisam ser transformados antes de chegarem ao seu destino? Uma transformação em andamento seria apropriada, onde os dados são transformados dentro do próprio fluxo?

Estas são apenas uma amostra dos fatores que você precisará considerar com a ingestão e abordamos essas questões e muito mais em [Capítulo 7](#). Antes de sairmos, vamos voltar brevemente nossa atenção para dois conceitos principais de ingestão de dados: lote versus streaming e push versus pull.

Lote versus streaming

Praticamente todos os dados com os quais lidamos são inherentemente *transmissão*. Os dados são quase sempre produzidos e atualizados continuamente em sua fonte. *Ingestão em lote* é simplesmente uma maneira especializada e conveniente de processar esse fluxo em grandes partes — por exemplo, lidar com dados de um dia inteiro em um único lote.

A ingestão de streaming nos permite fornecer dados para sistemas downstream – sejam outros aplicativos, bancos de dados ou sistemas analíticos – de forma contínua e em tempo real. Aqui, *tempo real* (ou *quase em tempo real*) significa que os dados estão disponíveis para um sistema downstream logo após serem produzidos (por exemplo, menos de um segundo depois). A latência necessária para se qualificar como tempo real varia de acordo com o domínio e os requisitos.

Os dados em lote são ingeridos em um intervalo de tempo predeterminado ou quando os dados atingem um limite de tamanho predefinido. A ingestão de lote é uma porta de mão única: uma vez que os dados são divididos em lotes, a latência para consumidores downstream é inherentemente restrita. Devido às limitações dos sistemas legados, o lote foi por muito tempo a forma padrão de ingerir dados. O processamento em lote continua sendo uma forma extremamente popular de ingerir dados para consumo downstream, principalmente em análises e ML.

No entanto, a separação de armazenamento e computação em muitos sistemas e a onipresença de streaming de eventos e plataformas de processamento tornam o processamento contínuo de fluxos de dados muito mais acessível e cada vez mais popular. A escolha depende muito do caso de uso e das expectativas de pontualidade dos dados.

Principais considerações para ingestão de lote versus ingestão de fluxo

Você deve fazer o streaming primeiro? Apesar da atratividade de uma abordagem de streaming primeiro, há muitas compensações para entender e pensar. A seguir estão algumas perguntas a serem feitas ao determinar se a ingestão de streaming é uma escolha apropriada em relação à ingestão de lote:

- Se eu ingerir os dados em tempo real, os sistemas de armazenamento downstream podem lidar com a taxa de fluxo de dados?
- Preciso de ingestão de dados em milissegundos em tempo real? Ou uma abordagem de microlote funcionaria, acumulando e ingerindo dados, digamos, a cada minuto?
- Quais são meus casos de uso para ingestão de streaming? Quais são os benefícios específicos que obtenho ao implementar o streaming? Se eu obtiver dados em tempo real, quais ações posso executar nesses dados que seriam uma melhoria no lote?
- Minha abordagem de streaming primeiro custará mais em termos de tempo, dinheiro, manutenção, tempo de inatividade e custo de oportunidade do que simplesmente fazer lotes?
- Meu pipeline e sistema de streaming são confiáveis e redundantes se a infraestrutura falhar?
- Quais ferramentas são mais apropriadas para o caso de uso? Devo usar um serviço gerenciado (Amazon Kinesis, Google Cloud Pub/Sub, Google Cloud Dataflow) ou criar minhas próprias instâncias de Kafka, Flink, Spark, Pulsar etc.? Se eu fizer o último, quem o administrará? Quais são os custos e compensações?
- Se estou implantando um modelo de ML, quais benefícios tenho com as previsões on-line e possivelmente com o treinamento contínuo?

- Estou obtendo dados de uma instância de produção ativa? Em caso afirmativo, qual é o impacto do meu processo de ingestão neste sistema de origem?

Como você pode ver, transmitir primeiro pode parecer uma boa ideia, mas nem sempre é direto; custos extras e complexidades ocorrem inherentemente. Muitas grandes estruturas de ingestão lidam com os estilos de ingestão de lote e microlote. Acreditamos que o lote é uma abordagem excelente para muitos casos de uso comuns, como treinamento de modelo e relatórios semanais. Adote o verdadeiro streaming em tempo real somente depois de identificar um caso de uso de negócios que justifique as compensações contra o uso do lote.

Empurrar contra puxar

No *empurrar* modelo de ingestão de dados, um sistema de origem grava dados em um destino, seja um banco de dados, armazenamento de objeto ou sistema de arquivos. No *puxar* modelo, os dados são recuperados do sistema de origem. A linha entre os paradigmas push e pull pode ser bastante tênue; os dados geralmente são empurrados e puxados enquanto percorrem os vários estágios de um pipeline de dados.

Considere, por exemplo, o processo de extração, transformação, carregamento (ETL), comumente usado em fluxos de trabalho de ingestão orientados a lotes. ETLs *extraír (E)* esclarece que estamos lidando com um modelo de ingestão pull. No ETL tradicional, o sistema de ingestão consulta um instantâneo da tabela de origem atual em um cronograma fixo. Você aprenderá mais sobre ETL e extração, carregamento, transformação (ELT) ao longo deste livro.

Em outro exemplo, considere o CDC contínuo, que é alcançado de algumas maneiras. Um método comum aciona uma mensagem toda vez que uma linha é alterada no banco de dados de origem. Esta mensagem é *empurrada* para uma fila, onde o sistema de ingestão o coleta. Outro método CDC comum usa logs binários, que registram cada confirmação no banco de dados. O banco de dados *empurra* os seus logs. O sistema de ingestão lê os logs, mas não interage diretamente com o banco de dados. Isso adiciona pouca ou nenhuma carga adicional ao banco de dados de origem. Algumas versões do CDC em lote usam o *puxar* padrão. Por exemplo, no CDC baseado em timestamp, um sistema de ingestão consulta o banco de dados de origem e extrai as linhas que foram alteradas desde a atualização anterior.

Com a ingestão de streaming, os dados ignoram um banco de dados de back-end e são enviados diretamente para um endpoint, geralmente com dados armazenados em buffer por uma plataforma de streaming de eventos. Esse padrão é útil com frotas de sensores IoT que emitem dados do sensor. Em vez de depender de um banco de dados para manter o estado atual, simplesmente pensamos em cada leitura registrada como um evento. Esse padrão também está crescendo em popularidade em aplicativos de software, pois simplifica o processamento em tempo real, permite que os desenvolvedores de aplicativos personalizem suas mensagens para análises downstream e simplifica muito a vida dos engenheiros de dados.

Discutimos as melhores práticas e técnicas de ingestão em profundidade em [Capítulo 7](#). Em seguida, vamos passar para o estágio de transformação do ciclo de vida da engenharia de dados.

Transformação

Depois de ingerir e armazenar dados, você precisa fazer algo com eles. A próxima etapa do ciclo de vida da engenharia de dados é *transformação*, o que significa que os dados precisam ser alterados de sua forma original para algo útil para casos de uso downstream. Sem as transformações adequadas, os dados ficarão inertes e não estarão em uma forma útil para relatórios, análises ou ML. Normalmente, o estágio de transformação é onde os dados começam a criar valor para o consumo do usuário downstream.

Imediatamente após a ingestão, as transformações básicas mapeiam os dados em tipos corretos (alterando os dados de string ingeridos em tipos numéricos e de data, por exemplo), colocando os registros em formatos padrão e removendo os incorretos. Os estágios posteriores da transformação podem transformar o esquema de dados e aplicar a normalização. Downstream, podemos aplicar agregação em larga escala para relatórios ou caracterizar dados para processos de ML.

Principais considerações para a fase de transformação

Ao considerar as transformações de dados dentro do ciclo de vida da engenharia de dados, é útil considerar o seguinte:

- Qual é o custo e o retorno sobre o investimento (ROI) da transformação? Qual é o valor comercial associado?
- A transformação é tão simples e auto-isolada quanto possível?
- Quais regras de negócios são suportadas pelas transformações?

Você pode transformar dados em lote ou durante o streaming em trânsito. Como mencionado em “[Ingestão](#)” na [página 39](#), praticamente todos os dados começam a vida como um fluxo contínuo; batch é apenas uma forma especializada de processar um fluxo de dados. As transformações em lote são extremamente populares, mas dada a crescente popularidade das soluções de processamento de fluxo e o aumento geral na quantidade de dados de streaming, esperamos que a popularidade das transformações de streaming continue crescendo, talvez substituindo totalmente o processamento em lote em certos domínios em breve.

Logicamente, tratamos a transformação como uma área independente do ciclo de vida da engenharia de dados, mas as realidades do ciclo de vida podem ser muito mais complicadas na prática. A transformação é muitas vezes emaranhada em outras fases do ciclo de vida. Normalmente, os dados são transformados em sistemas de origem ou em trânsito durante a ingestão. Por exemplo, um sistema de origem pode adicionar um carimbo de data/hora de evento a um registro antes de encaminhá-lo para um processo de ingestão. Ou um registro em um pipeline de streaming pode ser “enriquecido” com campos e cálculos adicionais antes de ser enviado para um data warehouse. As transformações são onipresentes em várias partes do ciclo de vida. Preparação de dados, organização de dados e limpeza — essas tarefas transformadoras agregam valor para os consumidores finais de dados.

A lógica de negócios é um dos principais impulsionadores da transformação de dados, geralmente na modelagem de dados. Os dados traduzem a lógica de negócios em elementos reutilizáveis (por exemplo, uma venda significa “alguém comprou

12 molduras minhas por US\$ 30 cada, ou US\$ 360 no total"). Neste caso, alguém comprou 12 porta-retratos por US\$ 30 cada. A modelagem de dados é crítica para obter uma imagem clara e atual dos processos de negócios. Uma visão simples das transações brutas de varejo pode não ser útil sem adicionar a lógica das regras contábeis para que o CFO tenha uma visão clara da saúde financeira. Garanta uma abordagem padrão para implementar a lógica de negócios em suas transformações.

A caracterização de dados para ML é outro processo de transformação de dados. A caracterização pretende extraír e aprimorar recursos de dados úteis para treinar modelos de ML. A caracterização pode ser uma arte obscura, combinando conhecimento de domínio (para identificar quais recursos podem ser importantes para a previsão) com ampla experiência em ciência de dados. Para este livro, o ponto principal é que, uma vez que os cientistas de dados determinam como caracterizar os dados, os processos de caracterização podem ser automatizados por engenheiros de dados no estágio de transformação de um pipeline de dados.

A transformação é um assunto profundo e não podemos fazer justiça a ela nesta breve introdução. [Capítulo 8](#) investiga consultas, modelagem de dados e várias práticas e nuances de transformação.

Servindo dados

Você atingiu o último estágio do ciclo de vida da engenharia de dados. Agora que os dados foram ingeridos, armazenados e transformados em estruturas coerentes e úteis, é hora de obter valor de seus dados. "Obter valor" dos dados significa coisas diferentes para usuários diferentes.

dados tem *valor* quando é usado para fins práticos. Os dados que não são consumidos ou consultados são simplesmente inertes. Projetos de vaidade de dados são um grande risco para as empresas. Muitas empresas perseguiram projetos vaidosos na era do big data, reunindo enormes conjuntos de dados em data lakes que nunca foram consumidos de maneira útil. A era da nuvem está desencadeando uma nova onda de projetos vaidosos construídos nos mais recentes data warehouses, sistemas de armazenamento de objetos e tecnologias de streaming. Os projetos de dados devem ser intencionais em todo o ciclo de vida. Qual é o propósito comercial final dos dados tão cuidadosamente coletados, limpos e armazenados?

O serviço de dados é talvez a parte mais empolgante do ciclo de vida da engenharia de dados. É aqui que a mágica acontece. É aqui que os engenheiros de ML podem aplicar as técnicas mais avançadas. Vejamos alguns dos usos populares de dados: análise, ML e ETL reverso.

Análise

A análise é o núcleo da maioria dos empreendimentos de dados. Depois que seus dados são armazenados e transformados, você está pronto para gerar relatórios ou painéis e fazer análises ad hoc dos dados. Considerando que a maior parte da análise costumava abranger BI, agora inclui outras facetas

como análise operacional e análise incorporada ([Figura 2-5](#)). Vamos abordar brevemente essas variações de análise.

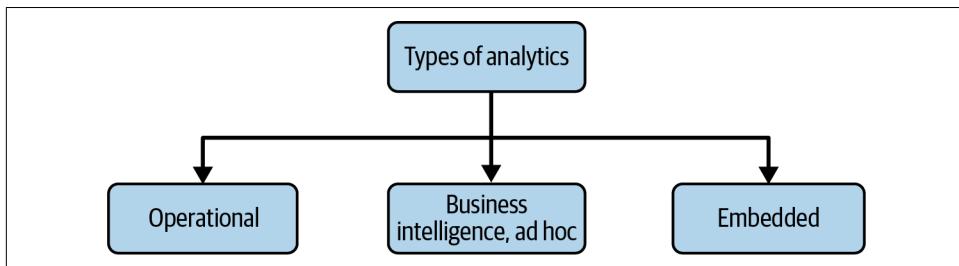


Figura 2-5. Tipos de análise

Inteligência de negócios. Os fiscais de BI coletavam dados para descrever o estado passado e atual de uma empresa. BI requer o uso de lógica de negócios para processar dados brutos. Observe que o fornecimento de dados para análise é outra área em que os estágios do ciclo de vida da engenharia de dados podem se emaranhar. Como mencionamos anteriormente, a lógica de negócios geralmente é aplicada aos dados no estágio de transformação do ciclo de vida da engenharia de dados, mas uma abordagem de lógica na leitura tornou-se cada vez mais popular. Os dados são armazenados de forma limpa, mas razoavelmente bruta, com lógica de negócios de pós-processamento mínima. Um sistema de BI mantém um repositório de lógica e definições de negócios. Essa lógica de negócios é usada para consultar o data warehouse para que os relatórios e painéis se alinhem com as definições de negócios.

À medida que uma empresa aumenta sua maturidade de dados, ela passará da análise de dados ad hoc para a análise de autoatendimento, permitindo acesso democratizado aos dados para usuários corporativos sem a necessidade de intervenção da TI. A capacidade de fazer análises de autoatendimento pressupõe que os dados sejam bons o suficiente para que as pessoas em toda a organização possam simplesmente acessá-los, cortá-los e cortá-los como quiserem e obter insights imediatos. Embora a análise de autoatendimento seja simples na teoria, é difícil realizá-la na prática. A principal razão é que a baixa qualidade dos dados, os silos organizacionais e a falta de habilidades adequadas de dados geralmente impedem o uso generalizado de análises.

Análise operacional. A análise operacional concentra-se nos detalhes refinados das operações, promovendo ações nas quais um usuário dos relatórios pode agir imediatamente. A análise operacional pode ser uma exibição ao vivo do inventário ou painéis em tempo real da integridade do site ou do aplicativo. Nesse caso, os dados são consumidos em tempo real, diretamente de um sistema de origem ou de um pipeline de dados de streaming. Os tipos de insights na análise operacional diferem do BI tradicional, pois a análise operacional é focada no presente e não necessariamente diz respeito às tendências históricas.

Análise incorporada. Você pode se perguntar por que dividimos a análise incorporada (análise voltada para o cliente) separadamente do BI. Na prática, as análises fornecidas aos clientes em uma plataforma SaaS vêm com um conjunto separado de requisitos e

complicações. O BI interno enfrenta um público limitado e geralmente apresenta um número limitado de visualizações unificadas. Os controles de acesso são críticos, mas não particularmente complicados. O acesso é gerenciado usando várias funções e níveis de acesso.

Com a análise incorporada, a taxa de solicitação de relatórios e a carga correspondente nos sistemas de análise aumentam drasticamente; o controle de acesso é significativamente mais complicado e crítico. As empresas podem fornecer análises e dados separados para milhares ou mais clientes. Cada cliente deve ver seus dados e apenas seus dados. Um erro interno de acesso a dados em uma empresa provavelmente levaria a uma revisão processual. Um vazamento de dados entre clientes seria considerado uma violação maciça de confiança, levando à atenção da mídia e a uma perda significativa de clientes. Minimize seu raio de explosão relacionado a vazamentos de dados e vulnerabilidades de segurança. Aplique segurança em nível de locatário ou de dados em seu armazenamento e em qualquer lugar onde haja possibilidade de vazamento de dados.

Múltiplos inquilinos

Muitos sistemas atuais de armazenamento e análise oferecem suporte à multilocação de várias maneiras. Os engenheiros de dados podem optar por armazenar os dados de muitos clientes em tabelas comuns para permitir uma visão unificada para análise interna e ML. Esses dados são apresentados externamente a clientes individuais por meio de exibições lógicas com controles e filtros definidos adequadamente. Cabe aos engenheiros de dados entender as minúcias da multilocação nos sistemas que implantam para garantir segurança e isolamento absolutos dos dados.

Aprendizado de máquina

O surgimento e o sucesso do ML é uma das revoluções tecnológicas mais empolgantes. Quando as organizações atingem um alto nível de maturidade de dados, elas podem começar a identificar problemas passíveis de ML e começar a organizar uma prática em torno disso.

As responsabilidades dos engenheiros de dados se sobrepõem significativamente em análise e ML, e os limites entre engenharia de dados, engenharia de ML e engenharia de análise podem ser confusos. Por exemplo, um engenheiro de dados pode precisar oferecer suporte a clusters Spark que facilitam pipelines analíticos e treinamento de modelo de ML. Eles também podem precisar fornecer um sistema que orquestre as tarefas entre as equipes e dê suporte a metadados e sistemas de catalogação que rastreiem o histórico e a linhagem dos dados. Definir esses domínios de responsabilidade e as estruturas de relatórios relevantes é uma decisão organizacional crítica.

A loja de recursos é uma ferramenta desenvolvida recentemente que combina engenharia de dados e engenharia de ML. Os repositórios de recursos são projetados para reduzir a carga operacional dos engenheiros de ML, mantendo o histórico e as versões dos recursos, suportando o compartilhamento de recursos entre as equipes e fornecendo recursos operacionais e de orquestração básicos, como preenchimento. Na prática, os engenheiros de dados fazem parte da equipe principal de suporte para lojas de recursos para oferecer suporte à engenharia de ML.

Um engenheiro de dados deve estar familiarizado com ML? Certamente ajuda. Independentemente do limite operacional entre engenharia de dados, engenharia de ML, análise de negócios e assim por diante, os engenheiros de dados devem manter o conhecimento operacional sobre suas equipes. Um bom engenheiro de dados está familiarizado com as técnicas fundamentais de ML e os requisitos de processamento de dados relacionados, os casos de uso de modelos em sua empresa e as responsabilidades das várias equipes de análise da organização. Isso ajuda a manter uma comunicação eficiente e facilitar a colaboração. Idealmente, os engenheiros de dados criariam ferramentas em parceria com outras equipes que nenhuma delas pode fazer independentemente.

Este livro não pode cobrir ML em profundidade. Um crescente ecossistema de livros, vídeos, artigos e comunidades está disponível se você estiver interessado em aprender mais; incluímos algumas sugestões em “[Recursos adicionais](#)” na [página 69](#).

A seguir estão algumas considerações para a fase de exibição de dados específica para ML:

- Os dados são de qualidade suficiente para executar uma engenharia de recursos confiável? Os requisitos e avaliações de qualidade são desenvolvidos em estreita colaboração com as equipes que consomem os dados.
- Os dados podem ser descobertos? Os cientistas de dados e engenheiros de ML podem encontrar facilmente dados valiosos?
- Onde estão os limites técnicos e organizacionais entre a engenharia de dados e a engenharia de ML? Essa questão organizacional tem implicações arquitetônicas significativas.
- O conjunto de dados representa adequadamente a verdade básica? É injustamente tendenciosa?

Embora o ML seja empolgante, nossa experiência mostra que as empresas muitas vezes mergulham prematuramente nele. Antes de investir uma tonelada de recursos em ML, reserve um tempo para construir uma base sólida de dados. Isso significa configurar os melhores sistemas e arquitetura em todo o ciclo de vida de engenharia de dados e ML. Geralmente, é melhor desenvolver competência em análise antes de mudar para ML. Muitas empresas frustraram seus sonhos de ML porque empreenderam iniciativas sem os fundamentos apropriados.

ETL reverso

O ETL reverso há muito é uma realidade prática em dados, visto como um antipadrão sobre o qual não gostamos de falar ou dignificar com um nome. *ETL reverso*pega os dados processados do lado de saída do ciclo de vida da engenharia de dados e os alimenta de volta nos sistemas de origem, conforme mostrado em [Figura 2-6](#). Na realidade, esse fluxo é benéfico e muitas vezes necessário; ETL reverso nos permite obter análises, modelos pontuados, etc., e alimentá-los de volta em sistemas de produção ou plataformas SaaS.

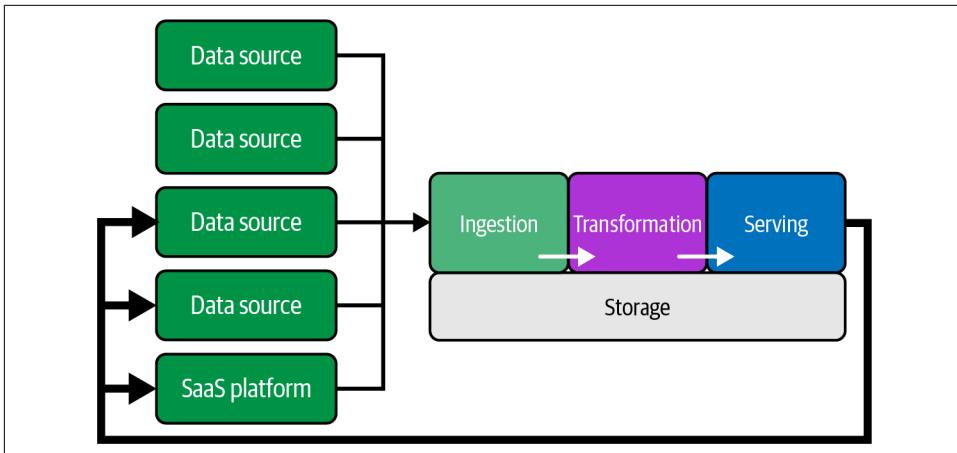


Figura 2-6. ETL reverso

Os analistas de marketing podem calcular lances no Microsoft Excel usando os dados em seu armazenamento de dados e, em seguida, fazer upload desses lances para o Google Ads. Esse processo geralmente era totalmente manual e primitivo.

Enquanto escrevíamos este livro, vários fornecedores adotaram o conceito de ETL reverso e criaram produtos em torno dele, como Hightouch e Census. O ETL reverso continua incipiente como uma prática, mas suspeitamos que veio para ficar.

O ETL reverso tornou-se especialmente importante, pois as empresas dependem cada vez mais de SaaS e plataformas externas. Por exemplo, as empresas podem querer enviar métricas específicas de seu data warehouse para uma plataforma de dados do cliente ou sistema de CRM. As plataformas de publicidade são outro caso de uso diário, como no exemplo do Google Ads. Espere ver mais atividade no ETL reverso, com uma sobreposição na engenharia de dados e na engenharia de ML.

O júri está em dúvida se o termo *ETL reverso* vai ficar. E a prática pode evoluir. Alguns engenheiros afirmam que podemos eliminar o ETL reverso manipulando as transformações de dados em um fluxo de eventos e enviando esses eventos de volta aos sistemas de origem conforme necessário. Perceber a adoção generalizada desse padrão nas empresas é outra questão. A essência é que os dados transformados precisarão ser devolvidos aos sistemas de origem de alguma maneira, idealmente com a linhagem e o processo de negócios corretos associados ao sistema de origem.

Principais subcorrentes em todo o ciclo de vida da engenharia de dados

A engenharia de dados está amadurecendo rapidamente. Considerando que os ciclos anteriores de engenharia de dados simplesmente focaram na camada de tecnologia, a abstração e simplificação contínuas

de ferramentas e práticas mudaram esse foco. A engenharia de dados agora abrange muito mais do que ferramentas e tecnologia. O campo agora está subindo na cadeia de valor, incorporando práticas corporativas tradicionais, como gerenciamento de dados e otimização de custos, e práticas mais recentes, como DataOps.

Nós denominamos essas práticas *correntes ocultas*— segurança, gerenciamento de dados, DataOps, arquitetura de dados, orquestração e engenharia de software — que dão suporte a todos os aspectos do ciclo de vida da engenharia de dados ([Figura 2-7](#)). Nesta seção, damos uma breve visão geral dessas subcorrentes e seus principais componentes, que você verá com mais detalhes ao longo do livro.

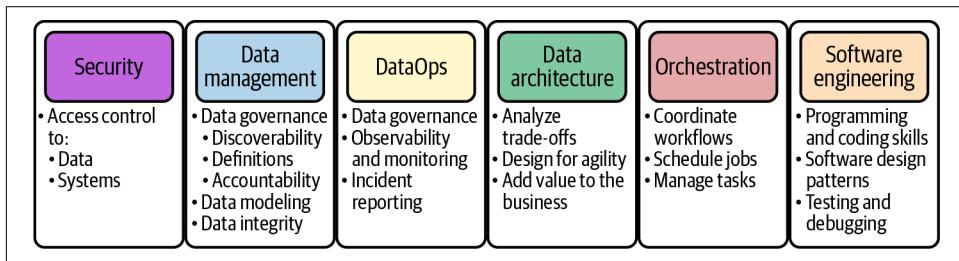


Figura 2-7. As principais subcorrentes da engenharia de dados

Segurança

A segurança deve ser uma prioridade para os engenheiros de dados, e aqueles que a ignoram correm o risco de fazê-lo. É por isso que a segurança é a primeira tendência. Os engenheiros de dados devem entender a segurança dos dados e do acesso, exercendo o princípio do menor privilégio. [O princípio do menor privilégio](#) significa dar a um usuário ou sistema acesso apenas aos dados e recursos essenciais para executar uma função pretendida. Um antipadrão comum que vemos em engenheiros de dados com pouca experiência em segurança é conceder acesso de administrador a todos os usuários. Esta é uma catástrofe esperando para acontecer!

Dê aos usuários apenas o acesso de que precisam para realizar seus trabalhos hoje, nada mais. Não opere a partir de um shell root quando estiver apenas procurando por arquivos visíveis com acesso de usuário padrão. Ao consultar tabelas com uma função menor, não use a função de superusuário em um banco de dados. Impor o princípio do menor privilégio a nós mesmos pode evitar danos acidentais e mantê-lo em uma mentalidade de segurança em primeiro lugar.

As pessoas e a estrutura organizacional são sempre as maiores vulnerabilidades de segurança em qualquer empresa. Quando ouvimos sobre grandes violações de segurança na mídia, muitas vezes descobrimos que alguém na empresa ignorou as precauções básicas, foi vítima de um ataque de phishing ou agiu de forma irresponsável. A primeira linha de defesa para a segurança dos dados é criar uma cultura de segurança que permeie a organização. Todos os indivíduos que têm acesso aos dados devem entender sua responsabilidade na proteção dos dados confidenciais da empresa e de seus clientes.

A segurança dos dados também tem a ver com o tempo – fornecer acesso aos dados exatamente para as pessoas e sistemas que precisam acessá-los *e apenas pelo tempo necessário para realizar seu trabalho*. Os dados devem ser protegidos contra visibilidade indesejada, tanto em trânsito quanto em repouso, usando criptografia, tokenização, mascaramento de dados, ofuscação e controles de acesso simples e robustos.

Os engenheiros de dados devem ser administradores de segurança competentes, já que a segurança cai em seu domínio. Um engenheiro de dados deve entender as práticas recomendadas de segurança para a nuvem e no local. Conhecimento de funções, políticas, grupos, segurança de rede, políticas de senha e criptografia de gerenciamento de acesso de usuário e identidade (IAM) são bons lugares para começar.

Ao longo do livro, destacamos as áreas em que a segurança deve estar em primeiro lugar no ciclo de vida da engenharia de dados. Você também pode obter insights mais detalhados sobre segurança em [Capítulo 10](#).

Gestão de dados

Você provavelmente acha que o gerenciamento de dados soa muito... corporativo. As práticas de gerenciamento de dados da “velha escola” chegam à engenharia de dados e ML. O que é velho é novo de novo. O gerenciamento de dados existe há décadas, mas não ganhou muita força na engenharia de dados até recentemente. As ferramentas de dados estão se tornando mais simples e há menos complexidade para os engenheiros de dados gerenciarem. Como resultado, o engenheiro de dados sobe na cadeia de valor em direção ao próximo degrau das melhores práticas. As melhores práticas de dados antes reservadas para grandes empresas – governança de dados, gerenciamento de dados mestre, gerenciamento de qualidade de dados, gerenciamento de metadados – agora estão sendo filtradas para empresas de todos os tamanhos e níveis de maturidade. Como gostamos de dizer, a engenharia de dados está se tornando “empresarial”. Em última análise, isso é uma grande coisa!

Associação Internacional de Gerenciamento de Dados (DAMA)*Corpo de conhecimento de gerenciamento de dados(DMBoK)*, que consideramos ser o livro definitivo para gerenciamento de dados corporativos, oferece esta definição:

O gerenciamento de dados é o desenvolvimento, execução e supervisão de planos, políticas, programas e práticas que entregam, controlam, protegem e aumentam o valor dos ativos de dados e informações ao longo de seu ciclo de vida.

Isso é um pouco longo, então vamos ver como isso se relaciona com a engenharia de dados. Os engenheiros de dados gerenciam o ciclo de vida dos dados, e o gerenciamento de dados abrange o conjunto de práticas recomendadas que os engenheiros de dados usarão para realizar essa tarefa, técnica e estrategicamente. Sem uma estrutura para gerenciar dados, os engenheiros de dados são simplesmente técnicos operando no vácuo. Os engenheiros de dados precisam de uma perspectiva mais ampla da utilidade dos dados em toda a organização, desde os sistemas de origem até o C-suite e em todos os lugares intermediários.

Por que o gerenciamento de dados é importante? O gerenciamento de dados demonstra que os dados são vitais para as operações diárias, assim como as empresas visualizam recursos financeiros, produtos acabados,

ou imóveis como ativos. As práticas de gerenciamento de dados formam uma estrutura coesa que todos podem adotar para garantir que a organização obtenha valor dos dados e os lide adequadamente.

O gerenciamento de dados tem algumas facetas, incluindo as seguintes:

- Governança de dados, incluindo descoberta e responsabilidade
- Modelagem e design de dados
- Linhagem de dados
- Armazenamento e operações
- Integração e interoperabilidade de dados
- Gerenciamento do ciclo de vida dos dados
- Sistemas de dados para análises avançadas e ML
- Ética e privacidade

Embora este livro não seja um recurso exaustivo sobre gerenciamento de dados, vamos cobrir brevemente alguns pontos importantes de cada área relacionados à engenharia de dados.

Gestão de dados

De acordo com *Governança de dados: o guia definitivo*, "A governança de dados é, antes de mais nada, uma função de gerenciamento de dados para garantir a qualidade, integridade, segurança e usabilidade dos dados coletados por uma organização."¹

Podemos expandir essa definição e dizer que a governança de dados envolve pessoas, processos e tecnologias para maximizar o valor dos dados em uma organização enquanto protege os dados com controles de segurança apropriados. A governança de dados eficaz é desenvolvida com intenção e apoiada pela organização. Quando a governança de dados é acidental e aleatória, os efeitos colaterais podem variar de dados não confiáveis a violações de segurança e tudo mais. Ser intencional sobre a governança de dados maximizará os recursos de dados da organização e o valor gerado a partir dos dados. Também (espero) manterá uma empresa fora das manchetes por práticas de dados questionáveis ou totalmente imprudentes.

Pense no exemplo típico de governança de dados mal feita. Um analista de negócios recebe uma solicitação de relatório, mas não sabe quais dados usar para responder à pergunta. Eles podem passar horas vasculhando dezenas de tabelas em um banco de dados transacional, tentando adivinhar quais campos podem ser úteis. O analista compila um relatório "direcionalmente correto", mas não tem certeza absoluta de que os dados subjacentes do relatório sejam

¹ Evren Eryurek et al., *Governança de dados: o guia definitivo* (Sebastopol, CA: O'Reilly, 2021), 1, <https://oreil.ly/LFT4d>.

preciso ou som. O destinatário do relatório também questiona a validade dos dados. A integridade do analista – e de todos os dados nos sistemas da empresa – é questionada. A empresa está confusa sobre seu desempenho, impossibilitando o planejamento do negócio.

A governança de dados é uma base para práticas de negócios orientadas a dados e uma parte crítica do ciclo de vida da engenharia de dados. Quando a governança de dados é bem praticada, pessoas, processos e tecnologias se alinharam para tratar os dados como um importante impulsionador de negócios; se ocorrerem problemas de dados, eles serão prontamente tratados.

As principais categorias de governança de dados são descoberta, segurança e responsabilidade.² Dentro dessas categorias principais estão subcategorias, como qualidade de dados, metadados e privacidade. Vejamos cada categoria principal por sua vez.

Descoberta. Em uma empresa orientada por dados, os dados devem estar disponíveis e detectáveis. Os usuários finais devem ter acesso rápido e confiável aos dados de que precisam para realizar seus trabalhos. Eles devem saber de onde vêm os dados, como eles se relacionam com outros dados e o que os dados significam.

Algumas áreas-chave de descoberta de dados incluem gerenciamento de metadados e gerenciamento de dados mestre. Vamos descrever brevemente essas áreas.

Metadados. *Metadados* “dados sobre dados” e sustenta cada seção do ciclo de vida da engenharia de dados. Os metadados são exatamente os dados necessários para tornar os dados detectáveis e governáveis.

Dividimos os metadados em duas categorias principais: gerados automaticamente e gerados por humanos. A engenharia de dados moderna gira em torno da automação, mas a coleta de metadados geralmente é manual e sujeita a erros.

A tecnologia pode ajudar nesse processo, removendo grande parte do trabalho sujeito a erros da coleta manual de metadados. Estamos vendo uma proliferação de catálogos de dados, sistemas de rastreamento de linhagem de dados e ferramentas de gerenciamento de metadados. As ferramentas podem rastrear bancos de dados para procurar relacionamentos e monitorar pipelines de dados para rastrear de onde os dados vêm e para onde vão. Uma abordagem manual de baixa fidelidade usa um esforço liderado internamente, onde várias partes interessadas fazem coleta coletiva de metadados dentro da organização. Essas ferramentas de gerenciamento de dados são abordadas em profundidade ao longo do livro, pois prejudicam grande parte do ciclo de vida da engenharia de dados.

Os metadados se tornam um subproduto de dados e processos de dados. No entanto, os principais desafios permanecem. Em particular, ainda faltam interoperabilidade e padrões. As ferramentas de metadados são tão boas quanto seus conectores para sistemas de dados e sua capacidade de compartilhar

2 Eryurek, *Gestão de dados*, 5.

metadados. Além disso, as ferramentas automatizadas de metadados não devem tirar totalmente as pessoas do circuito.

Os dados têm um elemento social; cada organização acumula capital social e conhecimento em torno de processos, conjuntos de dados e pipelines. Os sistemas de metadados orientados para humanos concentram-se no aspecto social dos metadados. Isso é algo que o Airbnb enfatizou em suas várias postagens de blog sobre ferramentas de dados, particularmente seu conceito original de Dataportal.³

Essas ferramentas devem fornecer um local para divulgar proprietários de dados, consumidores de dados e especialistas de domínio. A documentação e as ferramentas wiki internas fornecem uma base fundamental para o gerenciamento de metadados, mas essas ferramentas também devem se integrar à catalogação automatizada de dados. Por exemplo, ferramentas de varredura de dados podem gerar páginas wiki com links para objetos de dados relevantes.

Uma vez que os sistemas e processos de metadados existam, os engenheiros de dados podem consumir metadados de maneiras úteis. Os metadados se tornam uma base para projetar pipelines e gerenciar dados durante todo o ciclo de vida.

DMBO identifica quatro categorias principais de metadados que são úteis para engenheiros de dados:

- Metadados de negócios
- Metadados técnicos
- Metadados operacionais
- Metadados de referência

Vamos descrever brevemente cada categoria de metadados.

Metadados de negócios relaciona-se com a forma como os dados são usados no negócio, incluindo negócios e definições de dados, regras e lógica de dados, como e onde os dados são usados e o(s) proprietário(s) dos dados.

Um engenheiro de dados usa metadados de negócios para responder a perguntas não técnicas sobre quem, o quê, onde e como. Por exemplo, um engenheiro de dados pode ser encarregado de criar um pipeline de dados para análise de vendas do cliente. Mas o que é um cliente? É alguém que comprou nos últimos 90 dias? Ou alguém que comprou a qualquer momento em que o negócio foi aberto? Um engenheiro de dados usaria os dados corretos para se referir aos metadados de negócios (dicionário de dados ou catálogo de dados) para pesquisar como um “cliente” é definido. Os metadados de negócios fornecem ao engenheiro de dados o contexto e as definições corretos para usar os dados adequadamente.

Metadados técnicos descreve os dados criados e usados pelos sistemas em todo o ciclo de vida da engenharia de dados. Inclui o modelo de dados e esquema, linhagem de dados, campo

3 Chris Williams et al., “Democratizing Data at Airbnb,” *O blog de tecnologia do Airbnb*, 12 de maio de 2017, <https://oreil.ly/dM332>.

mapeamentos e fluxos de trabalho de pipeline. Um engenheiro de dados usa metadados técnicos para criar, conectar e monitorar vários sistemas em todo o ciclo de vida da engenharia de dados.

Aqui estão alguns tipos comuns de metadados técnicos que um engenheiro de dados usará:

- Metadados de pipeline (geralmente produzidos em sistemas de orquestração)
- Linhagem de dados
- Esquema

A orquestração é um hub central que coordena o fluxo de trabalho em vários sistemas.

Metadados de pipeline capturado em sistemas de orquestração fornece detalhes da programação do fluxo de trabalho, sistema e dependências de dados, configurações, detalhes de conexão e muito mais.

Metadados de linhagem de dados rastreia a origem e as alterações nos dados e suas dependências ao longo do tempo. À medida que os dados fluem pelo ciclo de vida da engenharia de dados, eles evoluem por meio de transformações e combinações com outros dados. A linhagem de dados fornece uma trilha de auditoria da evolução dos dados conforme eles passam por vários sistemas e fluxos de trabalho.

Metadados do esquema descreve a estrutura dos dados armazenados em um sistema como um banco de dados, um data warehouse, um data lake ou um sistema de arquivos; é um dos principais diferenciadores em diferentes sistemas de armazenamento. Armazenamentos de objetos, por exemplo, não gerenciam metadados de esquema; em vez disso, isso deve ser gerenciado em um *metastore*. Por outro lado, os data warehouses em nuvem gerenciam os metadados do esquema internamente.

Estes são apenas alguns exemplos de metadados técnicos que um engenheiro de dados deve conhecer. Esta não é uma lista completa e abordamos aspectos adicionais de metadados técnicos ao longo do livro.

Metadados operacionais descreve os resultados operacionais de vários sistemas e inclui estatísticas sobre processos, IDs de tarefas, logs de tempo de execução do aplicativo, dados usados em um processo e logs de erros. Um engenheiro de dados usa metadados operacionais para determinar se um processo foi bem-sucedido ou falhou e os dados envolvidos no processo.

Os sistemas de orquestração podem fornecer uma imagem limitada dos metadados operacionais, mas os últimos ainda tendem a estar dispersos em muitos sistemas. A necessidade de metadados operacionais de melhor qualidade e melhor gerenciamento de metadados é uma grande motivação para orquestração de próxima geração e sistemas de gerenciamento de metadados.

Metadados de referências são dados usados para classificar outros dados. Isto também é referido como *dados de pesquisa*. Exemplos padrão de dados de referência são códigos internos, códigos geográficos, unidades de medida e padrões internos de calendário. Observe que muitos dados de referência são totalmente gerenciados internamente, mas itens como códigos geográficos podem vir de referências externas padrão. Os dados de referência são essencialmente um padrão para interpretar outros dados; portanto, se eles mudarem, essa mudança acontecerá lentamente ao longo do tempo.

Prestação de contas de dados. *Prestação de contas de dados* significa designar um indivíduo para controlar uma parte dos dados. A pessoa responsável então coordena as atividades de governança de outras partes interessadas. Gerenciar a qualidade dos dados é difícil se ninguém for responsável pelos dados em questão.

Observe que as pessoas responsáveis pelos dados não precisam ser engenheiros de dados. A pessoa responsável pode ser um engenheiro de software ou gerente de produto, ou atuar em outra função. Além disso, o responsável geralmente não dispõe de todos os recursos necessários para manter a qualidade dos dados. Em vez disso, eles coordenam com todas as pessoas que tocam nos dados, incluindo engenheiros de dados.

A responsabilidade pelos dados pode acontecer em vários níveis; a responsabilidade pode ocorrer no nível de uma tabela ou fluxo de logs, mas pode ser tão refinada quanto uma única entidade de campo que ocorre em várias tabelas. Um indivíduo pode ser responsável por gerenciar um ID de cliente em vários sistemas. Para gerenciamento de dados corporativos, um domínio de dados é o conjunto de todos os valores possíveis que podem ocorrer para um determinado tipo de campo, como neste exemplo de ID. Isso pode parecer excessivamente burocrático e meticoloso, mas pode afetar significativamente a qualidade dos dados.

Qualidade dos dados.

Posso confiar nesses dados?

— Todos no negócio

Qualidade dos dados é a otimização dos dados em direção ao estado desejado e orbita a questão: “O que você obtém em comparação com o que espera?” Os dados devem estar em conformidade com as expectativas nos metadados de negócios. Os dados correspondem à definição acordada pela empresa?

Um engenheiro de dados garante a qualidade dos dados em todo o ciclo de vida da engenharia de dados. Isso envolve a realização de testes de qualidade de dados e a garantia da conformidade dos dados com as expectativas do esquema, integridade e precisão dos dados.

De acordo com *Governança de dados: o guia definitivo*, a qualidade dos dados é definida por três características principais:⁴

Precisão

Os dados coletados são factualmente corretos? Existem valores duplicados? Os valores numéricos são precisos?

Integridade

Os registros estão completos? Todos os campos obrigatórios contêm valores válidos?

⁴ Eryurek, *Gestão de dados*, 113.

pontualidade

Os registros estão disponíveis em tempo hábil?

Cada uma dessas características é bastante nuançada. Por exemplo, como pensamos em bots e web scrapers ao lidar com dados de eventos da web? Se pretendemos analisar a jornada do cliente, devemos ter um processo que nos permita separar os humanos do tráfego gerado por máquinas. Qualquer evento gerado por bot classificado erroneamente como humano apresenta problemas de precisão de dados e vice-versa.

Uma variedade de problemas interessantes surgem em relação à completude e pontualidade. No artigo do Google que apresenta o modelo Dataflow, os autores dão o exemplo de uma plataforma de vídeo off-line que exibe anúncios.⁵ A plataforma baixa vídeos e anúncios enquanto uma conexão está presente, permite que o usuário os assista enquanto estiver off-line e, em seguida, carrega os dados de visualização do anúncio quando uma conexão está presente novamente. Esses dados podem chegar tarde, bem depois de os anúncios serem assistidos. Como a plataforma lida com o faturamento dos anúncios?

Fundamentalmente, este problema não pode ser resolvido por meios puramente técnicos. Em vez disso, os engenheiros precisarão determinar seus padrões para dados atrasados e aplicá-los uniformemente, possivelmente com a ajuda de várias ferramentas tecnológicas.

Gerenciamento de dados mestre

Dados mestres são dados sobre entidades de negócios, como funcionários, clientes, produtos e locais. À medida que as organizações crescem e se tornam mais complexas por meio de aquisições e crescimento orgânico, e colaboram com outros negócios, manter uma imagem consistente de entidades e identidades torna-se cada vez mais desafiador.

Gerenciamento de dados mestre(MDM) é a prática de construir definições de entidades consistentes conhecidas como *discos de ouro*. Os registros de ouro harmonizam os dados da entidade em uma organização e com seus parceiros. MDM é um processo de operações de negócios facilitado pela construção e implantação de ferramentas de tecnologia. Por exemplo, uma equipe MDM pode determinar um formato padrão para endereços e, em seguida, trabalhar com engenheiros de dados para criar uma API para retornar endereços consistentes e um sistema que usa dados de endereço para corresponder aos registros de clientes nas divisões da empresa.

O MDM atinge todo o ciclo de dados em bancos de dados operacionais. Pode cair diretamente sob a alcada da engenharia de dados, mas geralmente é responsabilidade atribuída a uma equipe dedicada que trabalha em toda a organização. Mesmo que não possuam o MDM, os engenheiros de dados devem sempre estar cientes disso, pois podem colaborar em iniciativas de MDM.

5 Tyler Akidau et al., "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost em escala maciça, ilimitada, processamento de dados fora de ordem," *Procedimentos do VLDB Endowment* 8 (2015): 1792-1803, <https://oreil.ly/Z6XYy>.

A qualidade dos dados ultrapassa os limites dos problemas humanos e tecnológicos. Os engenheiros de dados precisam de processos robustos para coletar feedback humano acionável sobre a qualidade dos dados e usar ferramentas de tecnologia para detectar problemas de qualidade preventivamente antes que os usuários downstream os vejam. Cobrimos esses processos de coleta nos capítulos apropriados ao longo deste livro.

Modelagem e design de dados

Para derivar insights de negócios de dados, por meio de análise de negócios e ciência de dados, os dados devem estar em um formato utilizável. O processo de conversão de dados em um formato utilizável é conhecido como *modelagem e design de dados*. Embora tradicionalmente pensemos na modelagem de dados como um problema para administradores de banco de dados (DBAs) e desenvolvedores de ETL, a modelagem de dados pode acontecer em quase qualquer lugar em uma organização. Os engenheiros de firmware desenvolvem o formato de dados de um registro para um dispositivo IoT, ou os desenvolvedores de aplicativos da web projetam a resposta JSON para uma chamada de API ou um esquema de tabela MySQL — todas essas são instâncias de modelagem e design de dados.

A modelagem de dados tornou-se mais desafiadora devido à variedade de novas fontes de dados e casos de uso. Por exemplo, a normalização estrita não funciona bem com dados de evento.

Felizmente, uma nova geração de ferramentas de dados aumenta a flexibilidade dos modelos de dados, mantendo as separações lógicas de medidas, dimensões, atributos e hierarquias. Os data warehouses em nuvem suportam a ingestão de enormes quantidades de dados desnormalizados e semiestruturados, ao mesmo tempo em que suportam padrões comuns de modelagem de dados, como Kimball, Inmon e Data Vault. As estruturas de processamento de dados, como o Spark, podem ingerir todo um espectro de dados, desde registros relacionais estruturados simples até texto bruto não estruturado. Discutimos esses padrões de modelagem e transformação de dados com mais detalhes em [Capítulo 8](#).

Com a grande variedade de dados com os quais os engenheiros devem lidar, existe a tentação de desistir da modelagem de dados. Esta é uma ideia terrível com consequências angustiantes, evidenciadas quando as pessoas murmuram sobre o padrão de acesso escreva uma vez, leia nunca (WORN) ou se referem a um *pântano de dados*. Os engenheiros de dados precisam entender as práticas recomendadas de modelagem, bem como desenvolver a flexibilidade para aplicar o nível e o tipo apropriados de modelagem à fonte de dados e ao caso de uso.

Linhagem de dados

À medida que os dados passam por seu ciclo de vida, como você sabe qual sistema afetou os dados ou do que os dados são compostos à medida que são transmitidos e transformados? *Linhagem de dados* descreve o registro de uma trilha de auditoria de dados ao longo de seu ciclo de vida, rastreando os sistemas que processam os dados e os dados upstream dos quais dependem.

A linhagem de dados ajuda no rastreamento de erros, responsabilidade e depuração de dados e dos sistemas que os processam. Tem o benefício óbvio de fornecer uma trilha de auditoria para o ciclo de vida dos dados e ajuda na conformidade. Por exemplo, se um usuário quiser que seus dados

excluídos de seus sistemas, ter linhagem para esses dados permite que você saiba onde esses dados estão armazenados e suas dependências.

A linhagem de dados existe há muito tempo em grandes empresas com padrões de conformidade rígidos. No entanto, agora está sendo mais amplamente adotado em empresas menores, à medida que o gerenciamento de dados se torna comum. Também notamos que o conceito de Andy Petrella de Desenvolvimento Orientado à Observabilidade de Dados (DODD) está intimamente relacionado com a linhagem de dados. DODD observa dados ao longo de sua linhagem. Este processo é aplicado durante o desenvolvimento, teste e, finalmente, produção para entregar qualidade e conformidade com as expectativas.

Integração e interoperabilidade de dados

Integração e interoperabilidade de dados é o processo de integração de dados entre ferramentas e processos. À medida que nos afastamos de uma abordagem de pilha única para análise e em direção a um ambiente de nuvem heterogêneo no qual várias ferramentas processam dados sob demanda, a integração e a interoperabilidade ocupam uma área cada vez maior do trabalho do engenheiro de dados.

Cada vez mais, a integração ocorre por meio de APIs de uso geral, em vez de conexões de banco de dados personalizadas. Por exemplo, um pipeline de dados pode extrair dados da API Salesforce, armazená-los no Amazon S3, chamar a API Snowflake para carregá-los em uma tabela, chamar a API novamente para executar uma consulta e, em seguida, exportar os resultados para o S3, onde Spark pode consumi-los.

Toda essa atividade pode ser gerenciada com um código Python relativamente simples que se comunica com os sistemas de dados em vez de manipulá-los diretamente. Embora a complexidade da interação com os sistemas de dados tenha diminuído, o número de sistemas e a complexidade dos pipelines aumentaram drasticamente. Os engenheiros que começam do zero superam rapidamente os recursos de scripts sob medida e se deparam com a necessidade de orquestração. A orquestração é uma de nossas correntes ocultas, e a discutimos em detalhes em “Orquestração” na página 64.

Gerenciamento do ciclo de vida dos dados

O advento dos data lakes incentivou as organizações a ignorar o arquivamento e a destruição de dados. Por que descartar dados quando você pode simplesmente adicionar mais armazenamento ad infinitum? Duas mudanças encorajaram os engenheiros a prestar mais atenção ao que acontece no final do ciclo de vida da engenharia de dados.

Primeiro, os dados são cada vez mais armazenados na nuvem. Isso significa que temos custos de armazenamento pagos conforme o uso, em vez de grandes despesas iniciais de capital para um data lake local. Quando cada byte aparece em um extrato mensal da AWS, os CFOs veem oportunidades de economia. Os ambientes de nuvem tornam o arquivamento de dados um processo relativamente simples. Os principais fornecedores de nuvem oferecem classes de armazenamento de objetos específicos para arquivamento que permitem a retenção de dados de longo prazo a um custo extremamente baixo, assumindo acesso muito pouco frequente

(deve-se notar que a recuperação de dados não é tão barata, mas isso é assunto para outra conversa). Essas classes de armazenamento também oferecem suporte a controles de política extras para evitar a exclusão acidental ou deliberada de arquivos críticos.

Em segundo lugar, as leis de privacidade e retenção de dados, como o GDPR e o CCPA, exigem que os engenheiros de dados gerenciem ativamente a destruição de dados para respeitar o “direito de ser esquecido” dos usuários. Os engenheiros de dados devem saber quais dados do consumidor eles retêm e devem ter procedimentos para destruir dados em resposta a solicitações e requisitos de conformidade.

A destruição de dados é direta em um data warehouse na nuvem. A semântica SQL permite a exclusão de linhas em conformidade com uma cláusula. A destruição de dados era mais desafiadora em data lakes, onde gravar uma vez, ler várias vezes era o padrão de armazenamento padrão. Ferramentas como Hive ACID e Delta Lake permitem o gerenciamento fácil de transações de exclusão em grande escala. Novas gerações de gerenciamento de metadados, linhagem de dados e ferramentas de catalogação também simplificam o final do ciclo de vida da engenharia de dados.

Ética e privacidade

Os últimos anos de violações de dados, desinformação e manuseio incorreto de dados deixam uma coisa clara: os dados afetam as pessoas. Os dados costumavam viver no Velho Oeste, livremente coletados e negociados como figurinhas de beisebol. Esses dias já se foram. Embora as implicações éticas e de privacidade dos dados já tenham sido consideradas boas de se ter, como a segurança, elas agora são essenciais para o ciclo de vida geral dos dados. Os engenheiros de dados precisam fazer a coisa certa quando ninguém mais estiver assistindo, porque todo mundo estará assistindo algum dia.⁶

Esperamos que mais organizações incentivem uma cultura de boa ética e privacidade de dados.

Como a ética e a privacidade afetam o ciclo de vida da engenharia de dados? Os engenheiros de dados precisam garantir que os conjuntos de dados ocultem informações de identificação pessoal (PII) e outras informações confidenciais; o viés pode ser identificado e rastreado em conjuntos de dados conforme eles são transformados. Os requisitos regulamentares e as penalidades de conformidade estão crescendo cada vez mais. Certifique-se de que seus ativos de dados estejam em conformidade com um número crescente de regulamentações de dados, como GDPR e CCPA. Por favor, leve isso a sério. Oferecemos dicas ao longo do livro para garantir que você esteja incorporando ética e privacidade ao ciclo de vida da engenharia de dados.

DataOps

O DataOps mapeia as melhores práticas da metodologia Agile, DevOps e controle estatístico de processo (SPC) aos dados. Enquanto o DevOps visa melhorar o lançamento e a qualidade dos produtos de software, o DataOps faz o mesmo com os produtos de dados.

⁶ Defendemos a noção de que o comportamento ético é fazer a coisa certa quando ninguém está olhando, uma ideia que ocorre nos escritos de CS Lewis, Charles Marshall e muitos outros autores.

Os produtos de dados diferem dos produtos de software devido à forma como os dados são usados. Um produto de software fornece funcionalidade específica e recursos técnicos para usuários finais. Por outro lado, um produto de dados é construído em torno de métricas e lógica de negócios sólidas, cujos usuários tomam decisões ou criam modelos que executam ações automatizadas. Um engenheiro de dados deve entender tanto os aspectos técnicos da criação de produtos de software quanto a lógica, a qualidade e as métricas de negócios que criarão excelentes produtos de dados.

Como o DevOps, o DataOps empresta muito da manufatura enxuta e do gerenciamento da cadeia de suprimentos, misturando pessoas, processos e tecnologia para reduzir o tempo de retorno. Como Data Kitchen (especialistas em DataOps) descreve:⁷

DataOps é uma coleção de práticas técnicas, fluxos de trabalho, normas culturais e padrões de arquitetura que permitem:

- Inovação e experimentação rápidas, fornecendo novos insights aos clientes com velocidade crescente
- Qualidade de dados extremamente alta e taxas de erro muito baixas
- Colaboração entre matrizes complexas de pessoas, tecnologia e ambientes
- Medição clara, monitoramento e transparência dos resultados

As práticas enxutas (como redução do tempo de entrega e minimização de defeitos) e as melhorias resultantes na qualidade e produtividade são coisas que estamos felizes em ver ganhando força tanto nas operações de software quanto nas de dados.

Em primeiro lugar, DataOps é um conjunto de hábitos culturais; a equipe de engenharia de dados precisa adotar um ciclo de comunicação e colaboração com os negócios, quebrando silos, aprendendo continuamente com sucessos e erros e iteração rápida. Somente quando esses hábitos culturais são estabelecidos, a equipe pode obter os melhores resultados da tecnologia e das ferramentas.

Dependendo da maturidade dos dados de uma empresa, um engenheiro de dados tem algumas opções para criar DataOps na estrutura do ciclo de vida geral da engenharia de dados. Se a empresa não tiver infraestrutura ou práticas de dados preexistentes, o DataOps é uma oportunidade totalmente nova que pode ser aproveitada desde o primeiro dia. Com um projeto ou infraestrutura existente sem DataOps, um engenheiro de dados pode começar a adicionar DataOps aos fluxos de trabalho. Sugerimos começar com observabilidade e monitoramento para obter uma janela para o desempenho de um sistema e, em seguida, adicionar automação e resposta a incidentes. Um engenheiro de dados pode trabalhar ao lado de uma equipe de DataOps existente para melhorar o ciclo de vida da engenharia de dados em uma empresa com dados maduros. Em todos os casos, um engenheiro de dados deve estar ciente da filosofia e dos aspectos técnicos do DataOps.

⁷ "O que é DataOps", página de perguntas frequentes do DataKitchen, acessada em 5 de maio de 2022, <https://oreil.ly/Ns06w>.

DataOps tem três elementos técnicos principais: automação, monitoramento e observabilidade e resposta a incidentes (Figura 2-8). Vejamos cada uma dessas peças e como elas se relacionam com o ciclo de vida da engenharia de dados.

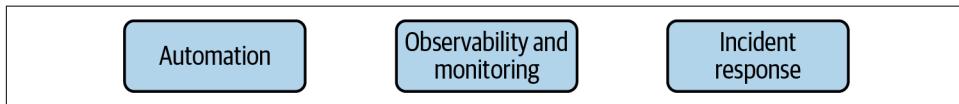


Figura 2-8. Os três pilares do DataOps

Automação

A automação permite confiabilidade e consistência no processo DataOps e permite que os engenheiros de dados implantem rapidamente novos recursos de produtos e melhorias nos fluxos de trabalho existentes. A automação de DataOps tem uma estrutura e fluxo de trabalho semelhantes ao DevOps, consistindo em gerenciamento de mudanças (ambiente, código e controle de versão de dados), integração contínua/implantação contínua (CI/CD) e configuração como código. Como o DevOps, as práticas de DataOps monitoram e mantêm a confiabilidade da tecnologia e dos sistemas (pipelines de dados, orquestração, etc.), com a dimensão adicional de verificação da qualidade dos dados, variação de dados/modelos, integridade de metadados e muito mais.

Vamos discutir brevemente a evolução da automação DataOps dentro de uma organização hipotética. Uma organização com um baixo nível de maturidade de DataOps geralmente tenta agendar vários estágios de processos de transformação de dados usando cron jobs. Isso funciona bem por um tempo. À medida que os pipelines de dados se tornam mais complicados, é provável que várias coisas aconteçam. Se os trabalhos cron estiverem hospedados em uma instância de nuvem, a instância pode ter um problema operacional, fazendo com que os trabalhos parem de ser executados inesperadamente. À medida que o espaçamento entre as tarefas fica mais apertado, uma tarefa acabará sendo executada por muito tempo, fazendo com que uma tarefa subsequente falhe ou produza dados obsoletos. Os engenheiros podem não estar cientes das falhas no trabalho até ouvirem dos analistas que seus relatórios estão desatualizados.

À medida que a maturidade dos dados da organização cresce, os engenheiros de dados normalmente adotam uma estrutura de orquestração, talvez Airflow ou Dagster. Os engenheiros de dados estão cientes de que o Airflow apresenta uma carga operacional, mas os benefícios da orquestração eventualmente superam a complexidade. Os engenheiros migrarão gradualmente seus cron jobs para trabalhos do Airflow. Agora, as dependências são verificadas antes da execução dos jobs. Mais trabalhos de transformação podem ser compactados em um determinado momento porque cada trabalho pode começar assim que os dados upstream estiverem prontos, em vez de em um horário fixo e predeterminado.

A equipe de engenharia de dados ainda tem espaço para melhorias operacionais. Um cientista de dados eventualmente implanta um DAG quebrado, derrubando o servidor web Airflow e deixando a equipe de dados operacionalmente cega. Depois de tantas dores de cabeça, os membros da equipe de engenharia de dados percebem que precisam parar de permitir implantações manuais de DAG. Em sua próxima fase de maturidade operacional, eles adotam a implantação automatizada de DAG. Os DAGs são testados antes da implantação e os processos de monitoramento garantem que os novos DAGs comecem a funcionar corretamente. Além disso, os engenheiros de dados bloqueiam

a implantação de novas dependências do Python até que a instalação seja validada. Depois que a automação é adotada, a equipe de dados fica muito mais feliz e experimenta muito menos dores de cabeça.

Um dos princípios do [Manifesto DataOps](#) é “Abraçar a mudança”. Isso não significa mudança pela mudança, mas sim uma mudança orientada para um objetivo. Em cada estágio de nossa jornada de automação, existem oportunidades para melhoria operacional. Mesmo no alto nível de maturidade que descrevemos aqui, ainda há espaço para melhorias. Os engenheiros podem adotar uma estrutura de orquestração de última geração que incorpora melhores recursos de metadados. Ou eles podem tentar desenvolver uma estrutura que construa DAGs automaticamente com base em especificações de linhagem de dados. O ponto principal é que os engenheiros buscam constantemente implementar melhorias na automação que irão reduzir sua carga de trabalho e aumentar o valor que entregam ao negócio.

Observabilidade e monitoramento

Como dizemos aos nossos clientes, “os dados são um assassino silencioso”. Vimos inúmeros exemplos de dados incorretos que persistem em relatórios por meses ou anos. Os executivos podem tomar decisões importantes a partir desses dados incorretos, descobrindo o erro apenas muito mais tarde. Os resultados geralmente são ruins e às vezes catastróficos para o negócio. Iniciativas são minadas e destruídas, anos de trabalho desperdiçados. Em alguns dos piores casos, dados incorretos podem levar as empresas à ruína financeira.

Outra história de horror ocorre quando os sistemas que criam os dados param de funcionar aleatoriamente, resultando em relatórios atrasados por vários dias. A equipe de dados não sabe até que as partes interessadas perguntam por que os relatórios estão atrasados ou produzindo informações obsoletas. Eventualmente, várias partes interessadas perdem a confiança nos recursos da equipe de dados principais e iniciam suas próprias equipes fragmentadas. O resultado são muitos sistemas instáveis diferentes, relatórios inconsistentes e silos.

Se você não está observando e monitorando seus dados e os sistemas que produzem os dados, você inevitavelmente vivenciará sua própria história de horror de dados. Observabilidade, monitoramento, registro, alerta e rastreamento são essenciais para antecipar quaisquer problemas ao longo do ciclo de vida da engenharia de dados. Recomendamos que você incorpore o SPC para entender se os eventos monitorados estão fora de linha e a qual incidente vale a pena responder.

O método DODD de Petrella mencionado anteriormente neste capítulo fornece uma excelente estrutura para pensar sobre a observabilidade de dados. O DODD é muito parecido com o desenvolvimento orientado a testes (TDD) na engenharia de software:⁸

⁸ Andy Petrella, “Data Observability Driven Development: The Perfect Analogy for Beginners,” Kensu, acessado 5 de maio de 2022, <https://oreil.ly/MxvSX>.

O objetivo do DODD é dar a todos os envolvidos na visibilidade da cadeia de dados os dados e aplicativos de dados para que todos os envolvidos na cadeia de valor de dados tenham a capacidade de identificar alterações nos dados ou aplicativos de dados em cada etapa - desde a ingestão até a transformação para análise — para ajudar a solucionar problemas ou evitar problemas de dados. O DODD se concentra em tornar a observabilidade de dados uma consideração de primeira classe no ciclo de vida da engenharia de dados.

Cobrimos muitos aspectos de monitoramento e observabilidade em todo o ciclo de vida da engenharia de dados em capítulos posteriores.

Resposta a incidentes

Uma equipe de dados altamente funcional usando DataOps poderá enviar novos produtos de dados rapidamente. Mas erros inevitavelmente acontecerão. Um sistema pode ter tempo de inatividade, um novo modelo de dados pode interromper os relatórios downstream, um modelo de ML pode se tornar obsoleto e fornecer previsões ruins – inúmeros problemas podem interromper o ciclo de vida da engenharia de dados. *Resposta a incidentes* trata-se de usar os recursos de automação e observabilidade mencionados anteriormente para identificar rapidamente as causas principais de um incidente e resolvê-lo da maneira mais confiável e rápida possível.

A resposta a incidentes não envolve apenas tecnologia e ferramentas, embora sejam benéficas; trata-se também de uma comunicação aberta e isenta de culpa, tanto na equipe de engenharia de dados quanto em toda a organização. Como Werner Vogels, CTO da Amazon Web Services, é famoso por dizer: “Tudo quebra o tempo todo”. Os engenheiros de dados devem estar preparados para um desastre e prontos para responder da maneira mais rápida e eficiente possível.

Os engenheiros de dados devem encontrar problemas proativamente antes que a empresa os relate. A falha acontece e, quando as partes interessadas ou usuários finais veem problemas, eles os apresentam. Eles ficarão infelizes em fazer isso. A sensação é diferente quando eles vão levantar essas questões para uma equipe e veem que já estão sendo trabalhados ativamente para resolver. Em qual estado de equipe você confiaría mais como usuário final? A confiança leva muito tempo para ser construída e pode ser perdida em minutos. A resposta a incidentes é tanto uma resposta retroativa a incidentes quanto uma abordagem proativa antes que eles aconteçam.

Resumo de DataOps

Neste ponto, DataOps ainda é um trabalho em andamento. Os profissionais fizeram um bom trabalho adaptando os princípios do DevOps ao domínio de dados e mapeando uma visão inicial por meio do DataOps Manifesto e outros recursos. Os engenheiros de dados fariam bem em tornar as práticas de DataOps uma alta prioridade em todo o seu trabalho. O esforço inicial terá um retorno significativo a longo prazo por meio da entrega mais rápida de produtos, melhor confiabilidade e precisão dos dados e maior valor geral para os negócios.

O estado das operações na engenharia de dados ainda é bastante imaturo em comparação com a engenharia de software. Muitas ferramentas de engenharia de dados, especialmente monólitos legados, são

não automação em primeiro lugar. Um movimento recente surgiu para adotar as melhores práticas de automação em todo o ciclo de vida da engenharia de dados. Ferramentas como o Airflow abriram caminho para uma nova geração de ferramentas de automação e gerenciamento de dados. As práticas gerais que descrevemos para DataOps são ambiciosas e sugerimos que as empresas tentem adotá-las ao máximo, dadas as ferramentas e o conhecimento disponíveis hoje.

Arquitetura de dados

Uma arquitetura de dados reflete o estado atual e futuro dos sistemas de dados que suportam as necessidades e estratégias de dados de longo prazo de uma organização. Como os requisitos de dados de uma organização provavelmente mudarão rapidamente e novas ferramentas e práticas parecem chegar quase diariamente, os engenheiros de dados devem entender uma boa arquitetura de dados.[Capítulo 3](#) abrange a arquitetura de dados em profundidade, mas queremos destacar aqui que a arquitetura de dados é uma subcorrente do ciclo de vida da engenharia de dados.

Um engenheiro de dados deve primeiro entender as necessidades do negócio e reunir requisitos para novos casos de uso. Em seguida, um engenheiro de dados precisa traduzir esses requisitos para projetar novas maneiras de capturar e fornecer dados, balanceados para custo e simplicidade operacional. Isso significa conhecer as compensações com padrões de design, tecnologias e ferramentas em sistemas de origem, ingestão, armazenamento, transformação e serviço de dados.

Isso não significa que um engenheiro de dados seja um arquiteto de dados, pois geralmente são duas funções separadas. Se um engenheiro de dados trabalha ao lado de um arquiteto de dados, o engenheiro de dados deve ser capaz de entregar os designs do arquiteto de dados e fornecer feedback de arquitetura.

Orquestração

Achamos que a orquestração é importante porque a vemos realmente como o centro de gravidade tanto da plataforma de dados quanto do ciclo de vida dos dados, o ciclo de vida de desenvolvimento de software no que se refere aos dados.

— Nick Schrock, fundador da Elementl⁹

A orquestração não é apenas um processo central de DataOps, mas também uma parte crítica do fluxo de engenharia e implantação para trabalhos de dados. Então, o que é orquestração?

Orquestração é o processo de coordenar muitos trabalhos para serem executados da maneira mais rápida e eficiente possível em uma cadência programada. Por exemplo, as pessoas costumam se referir a ferramentas de orquestração como o Apache Airflow como *agendadores*. Isso não é muito preciso. Um planejador puro, como o cron, está ciente apenas do tempo; um mecanismo de orquestração cria metadados nas dependências do trabalho, geralmente na forma de um gráfico acíclico direcionado

⁹ Ternary Data, "An Introduction to Dagster: The Orchestrator for the Full Data Lifecycle - UDEM June 2021," Vídeo do YouTube, 1:09:40, <https://oreil.ly/HyGMh>.

(DAG). O DAG pode ser executado uma vez ou programado para ser executado em um intervalo fixo de diariamente, semanalmente, a cada hora, a cada cinco minutos, etc.

Conforme discutimos a orquestração ao longo deste livro, presumimos que um sistema de orquestração permaneça online com alta disponibilidade. Isso permite que o sistema de orquestração detecte e monitore constantemente sem intervenção humana e execute novos trabalhos sempre que forem implantados. Um sistema de orquestração monitora os trabalhos que gerencia e inicia novas tarefas à medida que as dependências internas do DAG são concluídas. Ele também pode monitorar sistemas e ferramentas externas para observar a chegada de dados e os critérios a serem atendidos. Quando certas condições ultrapassam os limites, o sistema também define condições de erro e envia alertas por e-mail ou outros canais. Você pode definir um tempo de conclusão esperado de 10h para pipelines de dados diários noturnos. Se os trabalhos não forem concluídos até esse momento, os alertas serão enviados aos engenheiros de dados e aos consumidores.

Os sistemas de orquestração também criam recursos de histórico de tarefas, visualização e alertas. Mecanismos avançados de orquestração podem preencher novos DAGs ou tarefas individuais à medida que são adicionados a um DAG. Eles também oferecem suporte a dependências em um intervalo de tempo. Por exemplo, um trabalho de relatório mensal pode verificar se um trabalho ETL foi concluído no mês inteiro antes de começar.

A orquestração tem sido um recurso fundamental para o processamento de dados, mas nem sempre era uma prioridade nem acessível a ninguém, exceto às maiores empresas. As empresas usavam várias ferramentas para gerenciar fluxos de trabalho, mas eram caras, fora do alcance de pequenas startups e geralmente não extensíveis. O Apache Oozie era extremamente popular na década de 2010, mas foi projetado para funcionar em um cluster Hadoop e era difícil de usar em um ambiente mais heterogêneo. O Facebook desenvolveu o Dataswarm para uso interno no final dos anos 2000; isso inspirou ferramentas populares como o Airflow, lançado pelo Airbnb em 2014.

O Airflow era de código aberto desde o início e foi amplamente adotado. Ele foi escrito em Python, tornando-o altamente extensível para quase todos os casos de uso imagináveis. Embora existam muitos outros projetos interessantes de orquestração de código aberto, como Luigi e Conductor, o Airflow é indiscutivelmente o líder do mindshare por enquanto. O Airflow chegou no momento em que o processamento de dados estava se tornando mais abstrato e acessível, e os engenheiros estavam cada vez mais interessados em coordenar fluxos complexos em vários processadores e sistemas de armazenamento, especialmente em ambientes de nuvem.

Até o momento, vários projetos nascentes de código aberto visam imitar os melhores elementos do design principal do Airflow, aprimorando-o em áreas-chave. Alguns dos exemplos mais interessantes são o Prefect e o Dagster, que visam melhorar a portabilidade e testabilidade dos DAGs para permitir que os engenheiros passem do desenvolvimento local para a produção com mais facilidade. Argo é um mecanismo de orquestração construído em torno dos primitivos do Kubernetes; Metaflow é um projeto de código aberto da Netflix que visa melhorar a orquestração da ciência de dados.

Devemos salientar que a orquestração é estritamente um conceito de lote. A alternativa de streaming para DAGs de tarefas orquestradas é o DAG de streaming. Os DAGs de streaming continuam difíceis de construir e manter, mas as plataformas de streaming de próxima geração, como a Pulsar, visam reduzir drasticamente a carga operacional e de engenharia. Falamos mais sobre esses desenvolvimentos em [Capítulo 8](#).

Engenharia de software

A engenharia de software sempre foi uma habilidade central para engenheiros de dados. Nos primórdios da engenharia de dados contemporânea (2000–2010), os engenheiros de dados trabalhavam em estruturas de baixo nível e escreviam tarefas MapReduce em C, C++ e Java. No auge da era do big data (meados da década de 2010), os engenheiros começaram a usar estruturas que abstraíam esses detalhes de baixo nível.

Essa abstração continua até hoje. Armazéns de dados em nuvem suportam transformações poderosas usando semântica SQL; ferramentas como o Spark tornaram-se mais fáceis de usar, passando de detalhes de codificação de baixo nível para dataframes fáceis de usar. Apesar dessa abstração, a engenharia de software ainda é crítica para a engenharia de dados. Queremos discutir brevemente algumas áreas comuns da engenharia de software que se aplicam ao ciclo de vida da engenharia de dados.

Código de processamento de dados principais

Embora tenha se tornado mais abstrato e fácil de gerenciar, o código principal de processamento de dados ainda precisa ser escrito e aparece em todo o ciclo de vida da engenharia de dados. Seja na ingestão, transformação ou serviço de dados, os engenheiros de dados precisam ser altamente proficientes e produtivos em estruturas e linguagens como Spark, SQL ou Beam; rejeitamos a noção de que SQL não é código.

Também é imperativo que um engenheiro de dados entenda as metodologias adequadas de teste de código, como unidade, regressão, integração, ponta a ponta e fumaça.

Desenvolvimento de frameworks de código aberto

Muitos engenheiros de dados estão fortemente envolvidos no desenvolvimento de estruturas de código aberto. Eles adotam essas estruturas para resolver problemas específicos no ciclo de vida da engenharia de dados e, em seguida, continuam desenvolvendo o código da estrutura para melhorar as ferramentas para seus casos de uso e contribuir com a comunidade.

Na era do big data, vimos uma explosão cambriana de estruturas de processamento de dados dentro do ecossistema Hadoop. Essas ferramentas focavam principalmente em transformar e servir partes do ciclo de vida da engenharia de dados. A especiação de ferramentas de engenharia de dados não cessou ou diminuiu, mas a ênfase mudou para cima na escada da abstração, longe do processamento direto de dados. Essa nova geração de ferramentas de código aberto ajuda os engenheiros a gerenciar, aprimorar, conectar, otimizar e monitorar dados.

Por exemplo, o Airflow dominou o espaço de orquestração de 2015 até o início dos anos 2020. Agora, um novo lote de concorrentes de código aberto (incluindo Prefect, Dagster e Metaflow) surgiu para corrigir as limitações percebidas do Airflow, fornecendo melhor manuseio de metadados, portabilidade e gerenciamento de dependências. Para onde vai o futuro da orquestração ninguém sabe.

Antes que os engenheiros de dados começem a projetar novas ferramentas internas, eles fariam bem em pesquisar o panorama das ferramentas disponíveis publicamente. Fique de olho no custo total de propriedade (TCO) e no custo de oportunidade associado à implementação de uma ferramenta. Há uma boa chance de que já exista um projeto de código aberto para resolver o problema que eles procuram resolver, e eles fariam bem em colaborar em vez de reinventar a roda.

Transmissão

O processamento de dados de streaming é inherentemente mais complicado do que o batch, e as ferramentas e os paradigmas são indiscutivelmente menos maduros. À medida que o fluxo de dados se torna mais difundido em todos os estágios do ciclo de vida da engenharia de dados, os engenheiros de dados enfrentam problemas interessantes de engenharia de software.

Por exemplo, tarefas de processamento de dados, como junções que consideramos normais no mundo do processamento em lote, geralmente se tornam mais complicadas em tempo real, exigindo uma engenharia de software mais complexa. Os engenheiros também devem escrever código para aplicar uma variedade de janelas/métodos. O janelamento permite que os sistemas em tempo real calculem métricas valiosas, como estatísticas de rastreamento. Os engenheiros têm muitas estruturas para escolher, incluindo várias plataformas de função (OpenFaaS, AWS Lambda, Google Cloud Functions) para lidar com eventos individuais ou processadores de fluxo dedicados (Spark, Beam, Flink ou Pulsar) para analisar fluxos para dar suporte a relatórios e relatórios reais ações de tempo.

Infraestrutura como código

Infraestrutura como código(IaC) aplica práticas de engenharia de software para a configuração e gerenciamento de infraestrutura. A carga de gerenciamento de infraestrutura da era do big data diminuiu à medida que as empresas migraram para sistemas de big data gerenciados – como Databricks e Amazon Elastic MapReduce (EMR) – e data warehouses na nuvem. Quando os engenheiros de dados precisam gerenciar sua infraestrutura em um ambiente de nuvem, eles fazem isso cada vez mais por meio de estruturas de IaC, em vez de criar manualmente instâncias e instalar software. Várias estruturas de uso geral e específicas de plataforma de nuvem permitem a implantação de infraestrutura automatizada com base em um conjunto de especificações. Muitas dessas estruturas podem gerenciar serviços em nuvem, bem como infraestrutura. Existe também uma noção de IaC com containers e Kubernetes, utilizando ferramentas como Helm.

Essas práticas são uma parte vital do DevOps, permitindo o controle de versão e a repetibilidade das implantações. Naturalmente, esses recursos são vitais em todo o ciclo de vida da engenharia de dados, especialmente quando adotamos práticas de DataOps.

Pipelines como código

Pipelines como código é o conceito central dos sistemas de orquestração atuais, que abrangem todos os estágios do ciclo de vida da engenharia de dados. Engenheiros de dados usam código (normalmente Python) para declarar tarefas de dados e dependências entre elas. O mecanismo de orquestração interpreta essas instruções para executar etapas usando os recursos disponíveis.

Resolução de problemas de uso geral

Na prática, independentemente de quais ferramentas de alto nível eles adotam, os engenheiros de dados se deparam com casos extremos ao longo do ciclo de vida da engenharia de dados que exigem que eles resolvam problemas fora dos limites das ferramentas escolhidas e escrevam códigos personalizados. Ao usar estruturas como Fivetran, Airbyte ou Matillion, os engenheiros de dados encontrarão fontes de dados sem conectores existentes e precisarão escrever algo personalizado. Eles devem ser proficientes em engenharia de software para entender APIs, extrair e transformar dados, lidar com exceções e assim por diante.

Conclusão

A maioria das discussões que vimos no passado sobre engenharia de dados envolve tecnologias, mas perde a visão geral do gerenciamento do ciclo de vida dos dados. À medida que as tecnologias se tornam mais abstratas e fazem mais trabalho pesado, um engenheiro de dados tem a oportunidade de pensar e agir em um nível superior. O ciclo de vida da engenharia de dados, apoiado por suas subcorrentes, é um modelo mental extremamente útil para organizar o trabalho da engenharia de dados.

Dividimos o ciclo de vida da engenharia de dados nas seguintes etapas:

- Geração
- Armazenar
- Ingestão
- Transformação
- Dados de serviço

Vários temas também atravessam o ciclo de vida da engenharia de dados. Essas são as subcorrentes do ciclo de vida da engenharia de dados. Em um nível alto, as subcorrentes são as seguintes:

- Segurança
- Gestão de dados
- DataOps
- Arquitetura de dados
- Orquestração
- Engenharia de software

Um engenheiro de dados tem vários objetivos de alto nível em todo o ciclo de vida dos dados: produzir ROI ideal e reduzir custos (financeiros e de oportunidade), reduzir riscos (segurança, qualidade dos dados) e maximizar o valor e a utilidade dos dados.

Os próximos dois capítulos discutem como esses elementos impactam um bom projeto de arquitetura, juntamente com a escolha das tecnologias certas. Se você se sentir confortável com esses dois tópicos, sinta-se à vontade para avançar para a [parte II](#), onde abordamos cada um dos estágios do ciclo de vida da engenharia de dados.

Recursos adicionais

- “Uma comparação de estruturas de processamento de dados” por Ludovic Santos
- Site DAMA Internacional
- “O modelo de fluxo de dados: uma abordagem prática para balancear exatidão, latência e custo em processamento de dados em grande escala, ilimitado e fora de ordem” por Tyler Akidau et al.
- Página da Wikipédia “Processamento de Dados”
- Página da Wikipédia “Transformação de dados”
- “Democratização dos dados no Airbnb” por Chris Williams e outros.
- “Cinco etapas para começar a coletar o valor de seus dados” Página da web Lean-Data
- “Introdução à Automação DevOps” por Jared Murrell
- Página da Atlassian “Gerenciamento de incidentes na era do DevOps”
- Vídeo “Uma introdução ao Dagster: o orquestrador para o ciclo de vida completo dos dados” por Nick Schrock
- “O DevOps está relacionado ao DataOps?” por Carol Jang e Jove Kuang
- Página da Atlassian “Os sete estágios da resposta eficaz a incidentes”
- “Ficar à frente da dívida” por Etai Mizrahi
- “O que são metadados” por Michelle Knight

Projetando uma boa arquitetura de dados

Uma boa arquitetura de dados fornece recursos contínuos em todas as etapas do ciclo de vida dos dados e subcorrente. Começaremos definindo *arquitetura de dados*, em seguida, discutir os componentes e considerações. Em seguida, abordaremos padrões de lote específicos (armazéns de dados, data lakes), padrões de streaming e padrões que unificam lote e streaming. Em todo o processo, enfatizaremos o aproveitamento dos recursos da nuvem para fornecer escalabilidade, disponibilidade e confiabilidade.

O que é arquitetura de dados?

A engenharia de dados bem-sucedida é construída sobre uma arquitetura de dados sólida. Este capítulo tem como objetivo revisar algumas abordagens e estruturas de arquitetura populares e, em seguida, elaborar nossa definição opinativa do que torna a arquitetura de dados “boa”. De fato, não faremos todos felizes. Ainda assim, apresentaremos uma definição de trabalho pragmática, específica de domínio para *arquitetura de dados* que achamos que funcionarão para empresas de escalas, processos de negócios e necessidades muito diferentes.

O que é arquitetura de dados? Quando você para para descompactá-lo, o assunto fica um pouco obscuro; pesquisar a arquitetura de dados produz muitas definições inconsistentes e muitas vezes desatualizadas. É muito parecido com quando definimos *engenharia de dados* em Capítulo 1 - não há consenso. Em um campo que está em constante mudança, isso é de se esperar. Então o que queremos dizer com *arquitetura de dados* para os propósitos deste livro? Antes de definir o termo, é essencial entender o contexto em que ele se insere. Vamos abordar brevemente a arquitetura corporativa, que enquadará nossa definição de arquitetura de dados.

Arquitetura Corporativa Definida

A arquitetura corporativa tem muitos subconjuntos, incluindo negócios, técnicos, aplicativos e dados ([Figura 3-1](#)). Como tal, muitos frameworks e recursos são dedicados à arquitetura corporativa. Na verdade, a arquitetura é um tema surpreendentemente controverso.

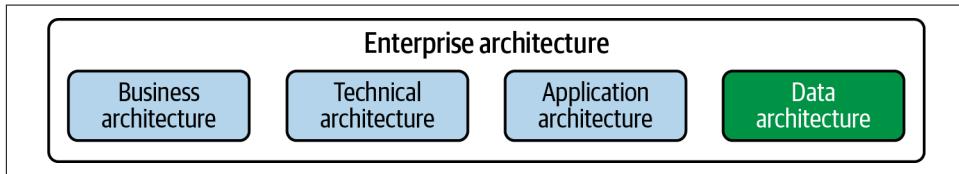


Figura 3-1. A arquitetura de dados é um subconjunto da arquitetura corporativa

O termo *empreendimento* recebe reações mistas. Ele traz à mente escritórios corporativos estéreis, planejamento de comando e controle/em cascata, culturas de negócios estagnadas e bordões vazios. Mesmo assim, podemos aprender algumas coisas aqui.

Antes de definirmos e descrevermos *arquitetura corporativa*, vamos descompactar este termo. Vejamos como a arquitetura corporativa é definida por alguns líderes de pensamento importantes: TOGAF, Gartner e EABOK.

Definição do TOGAF

TOGAF é *A Estrutura de Arquitetura do Grupo Aberto*, um padrão do The Open Group. É apontado como o framework de arquitetura mais usado atualmente. Aqui está a definição do TOGAF:¹

O termo “empresa” no contexto de “arquitetura corporativa” pode denotar uma empresa inteira – abrangendo todos os seus serviços de informação e tecnologia, processos e infraestrutura – ou um domínio específico dentro da empresa. Em ambos os casos, a arquitetura cruza vários sistemas e vários grupos funcionais dentro da empresa.

Definição do Gartner

Gartner é uma empresa global de pesquisa e consultoria que produz artigos de pesquisa e relatórios sobre tendências relacionadas a empresas. Entre outras coisas, é responsável pelo (infamoso) *Gartner Hype Cycle*. A definição do Gartner é a seguinte:²

A arquitetura corporativa (EA) é uma disciplina para liderar de forma proativa e holística as respostas corporativas às forças disruptivas, identificando e analisando a execução da mudança em direção à visão e resultados de negócios desejados. A EA agrupa valor ao apresentar

1 O Grupo Aberto, *TOGAF Versão 9.1*, <https://oreil.ly/A1H67>.

2 Gartner Glossary, sv “Enterprise Architecture (EA),” <https://oreil.ly/SWwQF>.

líderes de negócios e de TI com recomendações prontas para assinatura para ajustar políticas e projetos para alcançar resultados de negócios direcionados que capitalizam interrupções de negócios relevantes.

Definição de EABOK

EABOK é o *Livro de Conhecimento de Arquitetura Corporativa*, uma referência de arquitetura corporativa produzida pela MITRE Corporation. O EABOK foi lançado como um rascunho incompleto em 2004 e não foi atualizado desde então. Embora aparentemente obsoleto, o EABOK é frequentemente referenciado em descrições de arquitetura corporativa; achamos muitas de suas ideias úteis enquanto escrevímos este livro. Aqui está a definição do EABOK:³

Enterprise Architecture (EA) é um modelo organizacional; uma representação abstrata de uma empresa que alinha estratégia, operações e tecnologia para criar um roteiro para o sucesso.

nossa definição

Extraímos alguns tópicos comuns nessas definições de arquitetura empresarial: mudança, alinhamento, organização, oportunidades, resolução de problemas e migração. Aqui está a nossa definição de *arquitetura corporativa*, que consideramos ser mais relevante para o atual cenário de dados dinâmicos:

A arquitetura corporativa é o design de sistemas para dar suporte *mudança na empresa*, alcançado por *decisões flexíveis e reversíveis* alcançado através de cuidado *avaliação de compensações*.

Aqui, abordamos algumas áreas-chave às quais retornaremos ao longo do livro: decisões flexíveis e reversíveis, gerenciamento de mudanças e avaliação de trade-offs. Discutimos cada tema detalhadamente nesta seção e, em seguida, tornamos a definição mais concreta na última parte do capítulo, fornecendo vários exemplos de arquitetura de dados.

Decisões flexíveis e reversíveis são essenciais por dois motivos. Primeiro, o mundo está mudando constantemente e prever o futuro é impossível. Decisões reversíveis permitem que você ajuste o curso à medida que o mundo muda e você coleta novas informações. Em segundo lugar, há uma tendência natural para a ossificação empresarial à medida que as organizações crescem. Adotar uma cultura de decisões reversíveis ajuda a superar essa tendência, reduzindo o risco associado a uma decisão.

Jeff Bezos é creditado com a ideia de portas unidirecionais e bidirecionais.⁴ A porta de mão única é uma decisão quase impossível de reverter. Por exemplo, a Amazon poderia ter

3 Site do Consórcio EABOK, <https://eabok.org>.

4 Jeff Haden, "Fundador da Amazon, Jeff Bezos: É assim que pessoas bem-sucedidas tomam decisões inteligentes", Inc., 3 de dezembro de 2018, <https://oreil.ly/QwIm0>.

decidiu vender a AWS ou desligá-la. Seria quase impossível para a Amazon reconstruir uma nuvem pública com a mesma posição de mercado após tal ação.

Por outro lado, *umporta de mão dupla*é uma decisão facilmente reversível: você entra e continua se gosta do que vê na sala ou volta pela porta se não gosta. A Amazon pode decidir exigir o uso do DynamoDB para um novo banco de dados de microsserviços. Se essa política não funcionar, a Amazon tem a opção de revertê-la e refatorar alguns serviços para usar outros bancos de dados. Como os riscos associados a cada decisão reversível (porta de mão dupla) são baixos, as organizações podem tomar mais decisões, iterar, melhorar e coletar dados rapidamente.

O gerenciamento de mudanças está intimamente relacionado a decisões reversíveis e é um tema central das estruturas de arquitetura corporativa. Mesmo com ênfase em decisões reversíveis, as empresas muitas vezes precisam empreender grandes iniciativas. Estes são idealmente divididos em mudanças menores, cada uma delas uma decisão reversível em si. Voltando à Amazon, notamos um intervalo de cinco anos (2007 a 2012) desde a publicação de um artigo sobre o conceito do DynamoDB até o anúncio de Werner Vogels sobre o serviço DynamoDB na AWS. Nos bastidores, as equipes realizaram inúmeras pequenas ações para tornar o DynamoDB uma realidade concreta para os clientes da AWS. Gerenciar essas pequenas ações está no centro do gerenciamento de mudanças.

Os arquitetos não estão simplesmente mapeando processos de TI e olhando vagamente para um futuro distante e utópico; eles resolvem ativamente problemas de negócios e criam novas oportunidades. As soluções técnicas não existem por si mesmas, mas para apoiar os objetivos de negócios. Os arquitetos identificam problemas no estado atual (baixa qualidade de dados, limites de escalabilidade, linhas de negócios que dão prejuízo), definem estados futuros desejados (melhoria ágil da qualidade dos dados, soluções de dados em nuvem escaláveis, processos de negócios aprimorados) e realizam iniciativas por meio de execução de pequenos passos concretos. Vale a pena repetir:

As soluções técnicas não existem por si mesmas, mas para apoiar os objetivos de negócios.

Encontramos inspiração significativa em *Fundamentos da Arquitetura de Software*por Mark Richards e Neal Ford (O'Reilly). Eles enfatizam que as compensações são inevitáveis e onipresentes no espaço da engenharia. Às vezes, a natureza relativamente fluida do software e dos dados nos leva a acreditar que estamos livres das restrições que os engenheiros enfrentam no mundo físico duro e frio. De fato, isso é parcialmente verdade; corrigir um bug de software é muito mais fácil do que redesenhar e substituir uma asa de avião. No entanto, os sistemas digitais são limitados por limites físicos, como latência, confiabilidade, densidade e consumo de energia. Os engenheiros também enfrentam vários limites não físicos, como características de linguagens e estruturas de programação e restrições práticas no gerenciamento de complexidade, orçamentos, etc. O pensamento mágico culmina em uma engenharia deficiente.

Vamos reiterar um ponto central em nossa definição de arquitetura corporativa: a arquitetura corporativa equilibra flexibilidade e compensações. Este nem sempre é um equilíbrio fácil, e os arquitetos devem constantemente avaliar e reavaliar com o reconhecimento de que o mundo é dinâmico. Dado o ritmo de mudança que as empresas enfrentam, as organizações e sua arquitetura não podem ficar paradas.

Arquitetura de dados definida

Agora que você entende a arquitetura corporativa, vamos nos aprofundar na arquitetura de dados estabelecendo uma definição de trabalho que definirá o cenário para o restante do livro. *Arquitetura de dados* é um subconjunto da arquitetura corporativa, herdando suas propriedades: processos, estratégia, gerenciamento de mudanças e avaliação de trade-offs. Aqui estão algumas definições de arquitetura de dados que influenciam nossa definição.

Definição do TOGAF

O TOGAF define a arquitetura de dados da seguinte forma:⁵

Uma descrição da estrutura e interação dos principais tipos e fontes de dados da empresa, ativos de dados lógicos, ativos de dados físicos e recursos de gerenciamento de dados.

definição do DAMA

o DAMADMBOK define a arquitetura de dados da seguinte forma:⁶

Identificar as necessidades de dados da empresa (independentemente da estrutura) e projetar e manter os planos principais para atender a essas necessidades. Usando blueprints principais para orientar a integração de dados, controlar ativos de dados e alinhar os investimentos em dados com a estratégia de negócios.

nossa definição

Considerando as duas definições anteriores e nossa experiência, elaboramos nossa definição de *arquitetura de dados*:

A arquitetura de dados é o design de sistemas para dar suporte às necessidades de dados em evolução de uma empresa, alcançada por decisões flexíveis e reversíveis alcançadas por meio de uma avaliação cuidadosa dos trade-offs.

Como a arquitetura de dados se encaixa na engenharia de dados? Assim como o ciclo de vida da engenharia de dados é um subconjunto do ciclo de vida dos dados, a arquitetura da engenharia de dados é um subconjunto da arquitetura geral de dados. *Arquitetura de engenharia de dados* são os sistemas e estruturas

5 O Grupo Aberto, *TOGAF Versão 9.1*, <https://oreil.ly/A1H67>.

6 *DAMA - DMBOK: Conhecimento em Gerenciamento de Dados*, 2^a ed. (Publicações Técnicas, 2017).

que compõem as principais seções do ciclo de vida da engenharia de dados. nós vamos usar *arquitetura de dados* intercambiavelmente com *arquitetura de engenharia de dados* ao longo deste livro.

Outros aspectos da arquitetura de dados que você deve conhecer são operacionais e técnicos ([Figura 3-2](#)). *Arquitetura operacional* abrange os requisitos funcionais do que precisa acontecer em relação a pessoas, processos e tecnologia. Por exemplo, quais processos de negócios os dados atendem? Como a organização gerencia a qualidade dos dados? Qual é o requisito de latência desde quando os dados são produzidos até quando ficam disponíveis para consulta? *Arquitetura técnica* descreve como os dados são ingeridos, armazenados, transformados e servidos ao longo do ciclo de vida da engenharia de dados. Por exemplo, como você moverá 10 TB de dados a cada hora de um banco de dados de origem para seu data lake? Em resumo, a arquitetura operacional descreve o que precisa ser feito e os detalhes da arquitetura técnica como isso vai acontecer.

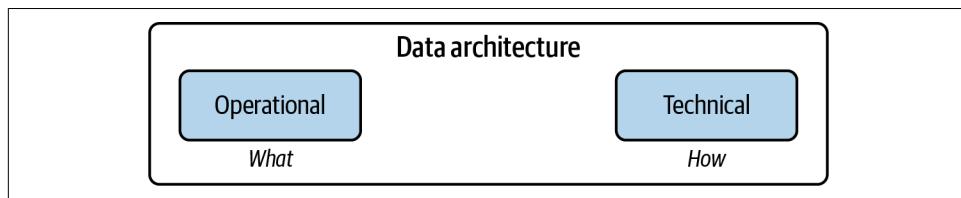


Figura 3-2. Arquitetura de dados operacionais e técnicos

Agora que temos uma definição funcional de arquitetura de dados, vamos abordar os elementos de uma “boa” arquitetura de dados.

Arquitetura de dados “boa”

Nunca atire na melhor arquitetura, mas sim na arquitetura menos pior.

—Mark Richards e Neal Ford⁷

De acordo com [Grady Booch](#), “Arquitetura representa as decisões de design significativas que moldam um sistema, onde *significativo* é medido pelo custo de mudança”. Os arquitetos de dados visam tomar decisões significativas que levarão a uma boa arquitetura em um nível básico.

O que queremos dizer com “boa” arquitetura de dados? Parafraseando um velho clichê, você sabe o que é bom quando o vê. *Boa arquitetura de dados* atende aos requisitos de negócios com um conjunto comum e amplamente reutilizável de blocos de construção, mantendo a flexibilidade e fazendo compensações apropriadas. A arquitetura ruim é autoritária e tenta amontoar um monte de decisões de tamanho único em um [grande bola de lama](#).

⁷ Mark Richards e Neal Ford, *Fundamentos da Arquitetura de Software* (Sebastopol, CA: O'Reilly, 2020), <https://oreil.ly/hpCp0>.

Agilidade é a base para uma boa arquitetura de dados; reconhece que o mundo é fluido. *Uma boa arquitetura de dados é flexível e de fácil manutenção.* Ele evolui em resposta às mudanças nos negócios e às novas tecnologias e práticas que podem agregar ainda mais valor no futuro. As empresas e seus casos de uso de dados estão sempre evoluindo. O mundo é dinâmico e o ritmo de mudança no espaço de dados está se acelerando. A arquitetura de dados do ano passado que serviu bem a você pode não ser suficiente para hoje, muito menos para o próximo ano.

A arquitetura de dados ruim é fortemente acoplada, rígida, excessivamente centralizada ou usa as ferramentas erradas para o trabalho, dificultando o desenvolvimento e o gerenciamento de mudanças. Idealmente, ao projetar a arquitetura com a reversibilidade em mente, as mudanças serão menos dispendiosas.

As tendências do ciclo de vida da engenharia de dados formam a base de uma boa arquitetura de dados para empresas em qualquer estágio de maturidade de dados. Novamente, essas subcorrentes são segurança, gerenciamento de dados, DataOps, arquitetura de dados, orquestração e engenharia de software.

Uma boa arquitetura de dados é uma coisa viva, que respira. Nunca está acabado. Na verdade, de acordo com nossa definição, mudança e evolução são fundamentais para o significado e propósito da arquitetura de dados. Vejamos agora os princípios de uma boa arquitetura de dados.

Princípios da boa arquitetura de dados

Esta seção apresenta uma visão de 10.000 pés da boa arquitetura, concentrando-se nos princípios-chave, ideias úteis na avaliação das principais decisões e práticas arquitetônicas. Tomamos inspiração para nossos princípios de arquitetura de várias fontes, especialmente o AWS Well-Architected Framework e os cinco princípios do Google Cloud para arquitetura nativa da nuvem.

O Estrutura bem arquitetada da AWS é composto por seis pilares:

- Excelência operacional
- Segurança
- Confiabilidade
- Eficiência de desempenho
- Otimização de custos
- Sustentabilidade

do Google Cloud Cinco princípios para arquitetura nativa da nuvem são como segue:

- Projeto para automação.
- Seja esperto com o estado.
- Favorecer serviços gerenciados.

- Pratique a defesa em profundidade.
- Esteja sempre arquitetando.

Aconselhamos você a estudar cuidadosamente ambas as estruturas, identificar ideias valiosas e determinar os pontos de desacordo. Gostaríamos de expandir ou elaborar esses pilares com estes princípios de arquitetura de engenharia de dados:

1. Escolha os componentes comuns com sabedoria.
2. Planeje o fracasso.
3. Arquiteto para escalabilidade.
- 4. Arquitetura é liderança.**
5. Esteja sempre arquitetando.
6. Construir sistemas fracamente acoplados.
7. Tome decisões reversíveis.
8. Priorize a segurança.
9. Abrace o FinOps.

Princípio 1: Escolha componentes comuns com sabedoria

Um dos principais trabalhos de um engenheiro de dados é escolher componentes e práticas comuns que possam ser usados amplamente em uma organização. Quando os arquitetos escolhem bem e lideram de forma eficaz, os componentes comuns tornam-se um tecido que facilita a colaboração da equipe e quebra os silos. Componentes comuns permitem agilidade dentro e entre equipes em conjunto com conhecimentos e habilidades compartilhados.

Componentes comuns podem ser qualquer coisa que tenha ampla aplicabilidade dentro de uma organização. Componentes comuns incluem armazenamento de objetos, sistemas de controle de versão, observabilidade, sistemas de monitoramento e orquestração e mecanismos de processamento. Componentes comuns devem ser acessíveis a todos com um caso de uso apropriado, e as equipes são incentivadas a confiar em componentes comuns já em uso, em vez de reinventar a roda. Os componentes comuns devem oferecer suporte a permissões e segurança robustas para permitir o compartilhamento de ativos entre as equipes e, ao mesmo tempo, impedir o acesso não autorizado.

As plataformas de nuvem são um local ideal para adotar componentes comuns. Por exemplo, a separação de computação e armazenamento em sistemas de dados em nuvem permite que os usuários acessem uma camada de armazenamento compartilhado (mais comumente armazenamento de objetos) usando ferramentas especializadas para acessar e consultar os dados necessários para casos de uso específicos.

Escolher componentes comuns é um ato de equilíbrio. Por um lado, você precisa se concentrar nas necessidades do ciclo de vida e das equipes de engenharia de dados, utilizar componentes comuns que serão úteis para projetos individuais e, simultaneamente, facilitar a interoperação e a colaboração. Por outro lado, os arquitetos devem evitar decisões que prejudiquem a produtividade dos engenheiros que trabalham em problemas específicos do domínio, forçando-os a soluções tecnológicas de tamanho único. [Capítulo 4](#) fornece detalhes adicionais.

Princípio 2: Plano para o fracasso

Tudo falha, o tempo todo.

— Werner Vogels, CTO da Amazon Web Services⁸

O hardware moderno é altamente robusto e durável. Mesmo assim, qualquer componente de hardware falhará, com tempo suficiente. Para construir sistemas de dados altamente robustos, você deve considerar falhas em seus projetos. Aqui estão alguns termos-chave para avaliar cenários de falha; nós os descrevemos com mais detalhes neste capítulo e ao longo do livro:

Disponibilidade

A porcentagem de tempo em que um serviço ou componente de TI está em um estado operável.

Confiabilidade

A probabilidade do sistema de atender aos padrões definidos no desempenho de sua função pretendida durante um intervalo especificado.

Objetivo do tempo de recuperação

O tempo máximo aceitável para uma interrupção de serviço ou sistema. O objetivo de tempo de recuperação (RTO) geralmente é definido determinando o impacto comercial de uma interrupção. Um RTO de um dia pode ser bom para um sistema de relatório interno. Uma interrupção do site de apenas cinco minutos pode ter um impacto adverso significativo nos negócios de um varejista on-line.

Objetivo do ponto de recuperação

O estado aceitável após a recuperação. Em sistemas de dados, os dados geralmente são perdidos durante uma interrupção. Nessa configuração, o objetivo do ponto de recuperação (RPO) refere-se à perda máxima de dados aceitável.

Os engenheiros precisam considerar confiabilidade, disponibilidade, RTO e RPO aceitáveis ao projetar para falhas. Isso orientará suas decisões de arquitetura à medida que avaliam possíveis cenários de falha.

⁸ UberPulse, "Amazon.com CTO: Everything Fails", vídeo do YouTube, 3:03, <https://oreil.ly/vDVIX>.

Princípio 3: Arquiteto para Escalabilidade

A escalabilidade em sistemas de dados abrange dois recursos principais. Primeiro, sistemas escaláveis podem escalar para lidar com quantidades significativas de dados. Pode ser necessário criar um cluster grande para treinar um modelo em um petabyte de dados do cliente ou expandir um sistema de ingestão de streaming para lidar com um pico de carga transitório. Nossa capacidade de escalabilidade nos permite lidar temporariamente com cargas extremas. Em segundo lugar, os sistemas escaláveis podem diminuir gradativamente. Assim que o pico de carga diminuir, devemos remover automaticamente a capacidade para cortar custos. (Isso está relacionado ao princípio 9.) Um sistema elástico pode escalar dinamicamente em resposta à carga, idealmente de forma automatizada.

Alguns sistemas escaláveis também podem escalar para zero: eles desligam completamente quando não estão em uso. Depois que o grande trabalho de treinamento de modelo for concluído, podemos excluir o cluster. Muitos sistemas sem servidor (por exemplo, funções sem servidor e processamento analítico online sem servidor, ou OLAP, bancos de dados) podem escalar automaticamente para zero.

Observe que a implantação de estratégias de dimensionamento inadequadas pode resultar em sistemas supercomplicados e custos elevados. Um banco de dados relacional direto com um nó de failover pode ser apropriado para um aplicativo em vez de um arranjo de cluster complexo. Meça sua carga atual, picos de carga aproximados e carga estimada nos próximos anos para determinar se sua arquitetura de banco de dados é apropriada. Se sua startup crescer muito mais rápido do que o previsto, esse crescimento também deve levar a mais recursos disponíveis para re-arquitetar para escalabilidade.

Princípio 4: Arquitetura é Liderança

Os arquitetos de dados são responsáveis pelas decisões de tecnologia e descrições de arquitetura e pela disseminação dessas escolhas por meio de liderança e treinamento eficazes. Os arquitetos de dados devem ser altamente competentes tecnicamente, mas delegar a maior parte do trabalho individual do colaborador a outros. Fortes habilidades de liderança combinadas com alta competência técnica são raras e extremamente valiosas. Os melhores arquitetos de dados levam essa dualidade a sério.

Observe que a liderança não implica uma abordagem de comando e controle da tecnologia. Não era incomum no passado os arquitetos escolherem uma tecnologia de banco de dados proprietária e forçar cada equipe a armazenar seus dados lá. Nós nos opomos a essa abordagem porque ela pode prejudicar significativamente os projetos de dados atuais. Os ambientes de nuvem permitem que os arquitetos equilibrem escolhas de componentes comuns com flexibilidade que permite a inovação nos projetos.

Voltando à noção de liderança técnica, Martin Fowler descreve um arquétipo específico de um arquiteto de software ideal, bem incorporado em seu colega Dave Rice:⁹

De muitas maneiras, a atividade mais importante da *Architectus Oryzus* é orientar a equipe de desenvolvimento, elevar seu nível para que possam assumir questões mais complexas. Melhorar a capacidade da equipe de desenvolvimento dá ao arquiteto uma alavancagem muito maior do que ser o único tomador de decisões e, portanto, correr o risco de ser um gargalo arquitetônico.

Um arquiteto de dados ideal manifesta características semelhantes. Eles possuem as habilidades técnicas de um engenheiro de dados, mas não praticam mais a engenharia de dados no dia a dia; eles orientam os engenheiros de dados atuais, fazem escolhas tecnológicas cuidadosas em consulta com sua organização e disseminam conhecimento por meio de treinamento e liderança. Eles treinam engenheiros nas melhores práticas e reúnem os recursos de engenharia da empresa para buscar objetivos comuns em tecnologia e negócios.

Como engenheiro de dados, você deve praticar a liderança em arquitetura e buscar orientação de arquitetos. Eventualmente, você mesmo pode ocupar o papel de arquiteto.

Princípio 5: Esteja Sempre Arquitetando

Tomamos emprestado esse princípio diretamente dos cinco princípios do Google Cloud para arquitetura nativa da nuvem. Os arquitetos de dados não servem em sua função simplesmente para manter o estado existente; em vez disso, eles constantemente projetam coisas novas e empolgantes em resposta às mudanças nos negócios e na tecnologia. Por **EABOK**, o trabalho de um arquiteto é desenvolver um conhecimento profundo da *arquitetura básica* (estado atual), desenvolver um *arquitetura alvo*, e mapear um *plano de sequenciamento* para determinar as prioridades e a ordem das mudanças na arquitetura.

Acrescentamos que a arquitetura moderna não deve ser de comando e controle ou cascata, mas colaborativa e ágil. O arquiteto de dados mantém uma arquitetura de destino e planos de sequenciamento que mudam com o tempo. A arquitetura-alvo torna-se um alvo em movimento, ajustado em resposta às mudanças de negócios e tecnologia interna e mundial. O plano de sequenciamento determina prioridades imediatas para entrega.

Princípio 6: Construir Sistemas Fracamente Acoplados

Quando a arquitetura do sistema é projetada para permitir que as equipes testem, implementem e alterem sistemas sem depender de outras equipes, as equipes precisam de pouca comunicação para realizar o trabalho. Em outras palavras, tanto a arquitetura quanto as equipes são fracamente acopladas.

— Guia de arquitetura tecnológica do Google DevOps¹⁰

⁹ Martin Fowler, "Quem Precisa de um Arquiteto?" *Software IEEE*, julho/agosto de 2003, <https://oreil.ly/wAMmZ>.

¹⁰ Google Cloud, "DevOps Tech: Architecture," Cloud Architecture Center, <https://oreil.ly/j4MT1>.

Em 2002, Bezos escreveu um e-mail para os funcionários da Amazon que ficou conhecido como Bezos API Mandate:¹¹

1. Todas as equipas passarão a expor os seus dados e funcionalidades através de interfaces de serviço.
2. As equipes devem se comunicar por meio dessas interfaces.
3. Não haverá nenhuma outra forma de comunicação entre processos permitida: nenhuma ligação direta, nenhuma leitura direta do armazenamento de dados de outra equipe, nenhum modelo de memória compartilhada, nenhum back-door de qualquer natureza. A única comunicação permitida é através de chamadas de interface de serviço pela rede.
4. Não importa qual tecnologia eles usam. HTTP, Corba, Pubsub, protocolos personalizados — não importa.
5. Todas as interfaces de serviço, sem exceção, devem ser projetadas desde o início para serem externalizáveis. Ou seja, a equipe deve planejar e projetar para poder expor a interface aos desenvolvedores do mundo exterior. Sem exceções.

O advento do API Mandate de Bezos é amplamente visto como um divisor de águas para a Amazon. Colocar dados e serviços atrás de APIs permitiu o acoplamento frrouxo e acabou resultando na AWS como a conhecemos agora. A busca do Google pelo baixo acoplamento permitiu que ele expandisse seus sistemas a uma escala extraordinária.

Para arquitetura de software, um sistema fracamente acoplado tem as seguintes propriedades:

1. Os sistemas são divididos em vários componentes pequenos.
2. Esses sistemas interagem com outros serviços por meio de camadas de abstração, como um barramento de mensagens ou uma API. Essas camadas de abstração ocultam e protegem detalhes internos do serviço, como um back-end de banco de dados ou classes internas e chamadas de método.
3. Como consequência da propriedade 2, mudanças internas em um componente do sistema não requerem mudanças em outras partes. Os detalhes das atualizações de código estão ocultos atrás de APIs estáveis. Cada peça pode evoluir e melhorar separadamente.
4. Como consequência da propriedade 3, não há ciclo de liberação global em cascata para todo o sistema. Em vez disso, cada componente é atualizado separadamente à medida que são feitas alterações e melhorias.

Observe que estamos falando sobre *sistemas técnicos*. Precisamos pensar maior. Vamos traduzir essas características técnicas em características organizacionais:

¹¹ "The Bezos API Mandate: Amazon's Manifesto for Externalization," Nordic APIs, 19 de janeiro de 2021, <https://oreil.ly/vls8m>.

1. Muitas equipes pequenas projetam um sistema grande e complexo. Cada equipe tem a tarefa de projetar, manter e melhorar alguns componentes do sistema.
2. Essas equipes publicam os detalhes abstratos de seus componentes para outras equipes por meio de definições de API, esquemas de mensagens, etc. As equipes não precisam se preocupar com os componentes de outras equipes; eles simplesmente usam a API publicada ou as especificações de mensagem para chamar esses componentes. Eles repetem sua parte para melhorar seu desempenho e recursos ao longo do tempo. Eles também podem publicar novos recursos à medida que são adicionados ou solicitar novos itens de outras equipes. Novamente, o último acontece sem que as equipes precisem se preocupar com os detalhes técnicos internos dos recursos solicitados. As equipes trabalham juntas através *comunicação fraca* e *acoplada*.
3. Como consequência da característica 2, cada equipa pode evoluir e melhorar rapidamente a sua componente independentemente do trabalho das outras equipas.
4. Especificamente, a característica 3 implica que as equipes podem liberar atualizações para seus componentes com o mínimo de tempo de inatividade. As equipes liberam continuamente durante o horário de trabalho regular para fazer alterações no código e testá-las.

O baixo acoplamento de tecnologia e sistemas humanos permitirá que suas equipes de engenharia de dados colaborem com mais eficiência umas com as outras e com outras partes da empresa. Este princípio também facilita diretamente o princípio 7.

Princípio 7: Tome Decisões Reversíveis

O cenário de dados está mudando rapidamente. A tecnologia ou pilha quente de hoje é a reflexão tardia de amanhã. A opinião popular muda rapidamente. Você deve buscar decisões reversíveis, pois elas tendem a simplificar sua arquitetura e mantê-la ágil.

Como Fowler escreveu, “Uma das tarefas mais importantes de um arquiteto é remover a arquitetura encontrando maneiras de eliminar a irreversibilidade nos projetos de software”.¹² O que era verdade quando Fowler escreveu isso em 2003 é tão preciso hoje.

Como dissemos anteriormente, Bezos se refere a decisões reversíveis como “portas de mão dupla”. Como ele diz, “Se você passar e não gostar do que vê do outro lado, não pode voltar para antes. Podemos chamar essas decisões de Tipo 1. Mas a maioria das decisões não é assim – elas são mutáveis, reversíveis – são portas de mão dupla.” Aponte para portas de duas vias sempre que possível.

Dado o ritmo da mudança - e a dissociação/modularização de tecnologias em sua arquitetura de dados - sempre se esforce para escolher as melhores soluções que funcionam hoje. Além disso, esteja preparado para atualizar ou adotar melhores práticas à medida que a paisagem evolui.

¹² Fowler, “Quem Precisa de um Arquiteto?”

Princípio 8: Priorize a Segurança

Todo engenheiro de dados deve assumir a responsabilidade pela segurança dos sistemas que constrói e mantém. Nós nos concentramos agora em duas ideias principais: segurança de confiança zero e o modelo de segurança de responsabilidade compartilhada. Eles se alinham de perto a uma arquitetura nativa da nuvem.

Modelos de segurança de perímetro reforçado e confiança zero

Definir *segurança de confiança zero*, é útil começar entendendo o modelo tradicional de segurança de perímetro rígido e suas limitações, conforme detalhado nos cinco princípios do Google Cloud:¹³

As arquiteturas tradicionais colocam muita fé na segurança do perímetro, grosseiramente um perímetro de rede reforçado com “coisas confiáveis” dentro e “coisas não confiáveis” fora. Infelizmente, essa abordagem sempre foi vulnerável a ataques internos, bem como a ameaças externas, como spear phishing.

O filme de 1996 *Missão Impossível* apresenta um exemplo perfeito do modelo de segurança de perímetro rígido e suas limitações. No filme, a CIA hospeda dados altamente confidenciais em um sistema de armazenamento dentro de uma sala com segurança física extremamente rígida. Ethan Hunt se infiltra na sede da CIA e explora um alvo humano para obter acesso físico ao sistema de armazenamento. Uma vez dentro da sala segura, ele pode exfiltrar dados com relativa facilidade.

Por pelo menos uma década, relatórios alarmantes da mídia nos alertaram sobre a crescente ameaça de violações de segurança que exploram alvos humanos dentro de perímetros de segurança organizacional reforçados. Mesmo quando os funcionários trabalham em redes corporativas altamente seguras, eles permanecem conectados ao mundo externo por meio de e-mail e dispositivos móveis. Ameaças externas efetivamente se tornam ameaças internas.

Em um ambiente nativo de nuvem, a noção de um perímetro reforçado se desgasta ainda mais. Todos os ativos estão conectados ao mundo exterior em algum grau. Embora as redes virtuais de nuvem privada (VPC) possam ser definidas sem conectividade externa, o plano de controle da API que os engenheiros usam para definir essas redes ainda está voltado para a Internet.

O modelo de responsabilidade compartilhada

Amazônia enfatiza o **modelo de responsabilidade compartilhada**, que divide a segurança na segurança *de* nuvem e a segurança *em* nuvem. A AWS é responsável pela segurança da nuvem:¹⁴

13 Tom Grey, “5 Principles for Cloud-Native Architecture—What It Is and How to Master It,” Google Cloud blog, 19 de junho de 2019, <https://oreil.ly/4NkGf>.

14 Amazon Web Services, “Segurança no AWS WAF”, documentação do AWS WAF, <https://oreil.ly/rEFoU>.

A AWS é responsável por proteger a infraestrutura que executa os serviços da AWS na Nuvem AWS. A AWS também fornece serviços que você pode usar com segurança.

Os usuários da AWS são responsáveis pela segurança na nuvem:

Sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos de sua organização e as leis e regulamentos aplicáveis.

Em geral, todos os provedores de nuvem operam em alguma forma desse modelo de responsabilidade compartilhada. Eles protegem seus serviços de acordo com as especificações publicadas. Ainda assim, em última análise, é responsabilidade do usuário projetar um modelo de segurança para seus aplicativos e dados e aproveitar os recursos da nuvem para realizar esse modelo.

Engenheiros de dados como engenheiros de segurança

No mundo corporativo de hoje, uma abordagem de comando e controle para a segurança é bastante comum, na qual as equipes de segurança e rede gerenciam perímetros e práticas gerais de segurança. A nuvem transfere essa responsabilidade para engenheiros que não estão explicitamente em funções de segurança. Devido a essa responsabilidade, em conjunto com a erosão mais geral do perímetro de segurança rígido, todos os engenheiros de dados devem se considerar engenheiros de segurança.

A falha em assumir essas novas responsabilidades implícitas pode levar a consequências terríveis. Numerosas violações de dados resultaram do simples erro de configuração de buckets do Amazon S3 com acesso público.¹⁵ Aquelas que lidam com dados devem assumir que são os responsáveis por protegê-los.

Princípio 9: Adote FinOps

Vamos começar considerando algumas definições de FinOps. Primeiro, a FinOps Foundation oferece isso:¹⁶

O FinOps é uma disciplina e prática cultural de gerenciamento financeiro em nuvem em evolução que permite que as organizações obtenham o máximo valor comercial ajudando as equipes de engenharia, finanças, tecnologia e negócios a colaborar em decisões de gastos baseadas em dados.

Além disso, JR Sorment e Mike Fuller fornecem a seguinte definição em *Nuvem FinOps*¹⁷:

O termo “FinOps” normalmente se refere ao movimento profissional emergente que defende uma relação de trabalho colaborativa entre DevOps e Finanças, resultando em

¹⁵ Ericka Chickowski, “Baldes Furados: 10 Piores Violações do Amazon S3,” Bitdefender *Insights de negócios blog*, 24 de janeiro de 2018, <https://oreil.ly/pFEO>.

¹⁶ Fundação FinOps, “O que é FinOps,” <https://oreil.ly/wJFVn>.

¹⁷ JR Storment e Mike Fuller, *Nuvem FinOps* (Sebastopol, CA: O'Reilly, 2019), <https://oreil.ly/QV6vF>.

um gerenciamento iterativo e orientado por dados dos gastos com infraestrutura (ou seja, redução da economia unitária da nuvem) ao mesmo tempo em que aumenta a eficiência de custos e, em última análise, a lucratividade do ambiente de nuvem.

A estrutura de custo dos dados evolui drasticamente durante a era da nuvem. Em um ambiente local, os sistemas de dados geralmente são adquiridos com um gasto de capital (descrito mais em [Capítulo 4](#)) para um novo sistema a cada poucos anos em um ambiente local. As partes responsáveis precisam equilibrar seu orçamento com a capacidade de computação e armazenamento desejada. A compra excessiva implica em dinheiro desperdiçado, enquanto a compra insuficiente significa impedir futuros projetos de dados e direcionar um tempo significativo da equipe para controlar a carga do sistema e o tamanho dos dados; a subcompra pode exigir ciclos de atualização de tecnologia mais rápidos, com custos extras associados.

Na era da nuvem, a maioria dos sistemas de dados são pré-pagos e facilmente escaláveis. Os sistemas podem ser executados em um modelo de custo por consulta, modelo de custo por capacidade de processamento ou outra variante de um modelo de pagamento conforme o uso. Esta abordagem pode ser muito mais eficiente do que a abordagem de despesas de capital. Agora é possível escalar para alto desempenho e, em seguida, reduzir para economizar dinheiro. No entanto, a abordagem de pagamento conforme o uso torna os gastos muito mais dinâmicos. O novo desafio para os líderes de dados é gerenciar orçamentos, prioridades e eficiência.

As ferramentas de nuvem exigem um conjunto de processos para gerenciar gastos e recursos. No passado, os engenheiros de dados pensavam em termos de engenharia de desempenho — maximizando o desempenho dos processos de dados em um conjunto fixo de recursos e comprando recursos adequados para necessidades futuras. Com o FinOps, os engenheiros precisam aprender a pensar nas estruturas de custo dos sistemas em nuvem. Por exemplo, qual é a combinação apropriada de instâncias spot da AWS ao executar um cluster distribuído? Qual é a abordagem mais apropriada para executar um trabalho diário considerável em termos de custo-benefício e desempenho? Quando a empresa deve mudar de um modelo de pagamento por consulta para capacidade reservada?

O FinOps desenvolve o modelo de monitoramento operacional para monitorar os gastos de forma contínua. Em vez de simplesmente monitorar as solicitações e a utilização da CPU para um servidor da Web, o FinOps pode monitorar o custo contínuo das funções sem servidor que lidam com o tráfego, bem como picos nos alertas de acionamento de gastos. Assim como os sistemas são projetados para falhar graciosamente em tráfego excessivo, as empresas podem considerar a adoção de limites rígidos para gastos, com modos de falha graciosa em resposta a picos de gastos.

As equipes de operações também devem pensar em termos de ataques de custo. Assim como um ataque distribuído de negação de serviço (DDoS) pode bloquear o acesso a um servidor web, muitas empresas descobriram, para seu desgosto, que downloads excessivos de balões S3 podem aumentar os gastos e ameaçar a falência de uma pequena startup. Ao compartilhar dados publicamente, as equipes de dados podem resolver esses problemas definindo políticas de pagamento pelo solicitante ou simplesmente monitorando gastos excessivos com acesso a dados e removendo rapidamente o acesso se os gastos começarem a subir para níveis inaceitáveis.

No momento em que este livro foi escrito, FinOps é uma prática formalizada recentemente. A Fundação FinOps foi iniciada apenas em 2019.¹⁸ No entanto, é altamente recomendável que você comece a pensar em FinOps com antecedência, antes de enfrentar altas contas de nuvem. Comece sua jornada com o Fundação FinOpse O'Reilly's *Nuvem FinOps*. Também sugerimos que os engenheiros de dados se envolvam no processo comunitário de criação de práticas de FinOps para engenharia de dados — em uma área de prática tão nova, ainda há muito território a ser mapeado.

Agora que você tem uma compreensão de alto nível dos bons princípios de arquitetura de dados, vamos nos aprofundar um pouco mais nos principais conceitos necessários para projetar e construir uma boa arquitetura de dados.

Principais Conceitos de Arquitetura

Se você seguir as tendências atuais em dados, parece que novos tipos de ferramentas e arquiteturas de dados estão chegando ao mercado toda semana. Em meio a essa agitação, não podemos perder de vista o objetivo principal de todas essas arquiteturas: pegar dados e transformá-los em algo útil para consumo downstream.

Domínios e Serviços

Domínio: Uma esfera de conhecimento, influência ou atividade. A área de assunto à qual o usuário aplica um programa é o domínio do software.

—Eric Evans¹⁹

Antes de mergulhar nos componentes da arquitetura, vamos abordar brevemente dois termos que você verá com frequência: domínio e serviços. *Adomínio*é a área de assunto do mundo real para a qual você está arquitetando. *Aserviço*é um conjunto de funcionalidades cujo objetivo é realizar uma tarefa. Por exemplo, você pode ter um serviço de processamento de pedidos de vendas cuja tarefa é processar os pedidos à medida que são criados. A única tarefa do serviço de processamento de pedidos de vendas é processar os pedidos; ele não fornece outra funcionalidade, como gerenciamento de inventário ou atualização de perfis de usuário.

Um domínio pode conter vários serviços. Por exemplo, você pode ter um domínio de vendas com três serviços: pedidos, faturamento e produtos. Cada serviço tem tarefas específicas que suportam o domínio de vendas. Outros domínios também podem compartilhar serviços (Figura 3-3). Nesse caso, o domínio contábil é responsável pelas funções contábeis básicas: faturamento, folha de pagamento e contas a receber (AR). Observe que o domínio de contabilidade compartilha o serviço de fatura com o domínio de vendas, pois uma venda gera uma fatura,

¹⁸ "FinOps Foundation sobe para 300 membros e apresenta novos níveis de parceiros para provedores de serviços em nuvem e fornecedores," Business Wire, 17 de junho de 2019, <https://oreil.ly/XcwYO>.

¹⁹ Eric Evans, *Referência de design orientado a domínio: definições e resumos de padrões* (março de 2015), <https://oreil.ly/pQ9oq>.

e a contabilidade deve acompanhar as faturas para garantir que o pagamento seja recebido. Vendas e contabilidade possuem seus respectivos domínios.

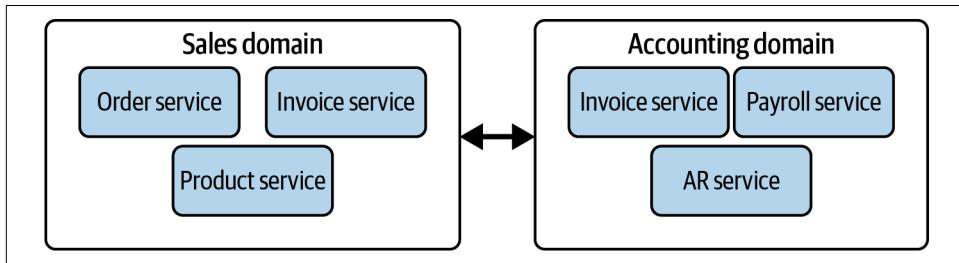


Figura 3-3. Dois domínios (vendas e contabilidade) compartilham um serviço comum (faturas), e vendas e contabilidade possuem seus respectivos domínios

Ao pensar sobre o que constitui um domínio, concentre-se no que o domínio representa no mundo real e trabalhe de trás para frente. No exemplo anterior, o domínio de vendas deve representar o que acontece com a função de vendas em sua empresa. Ao arquitetar o domínio de vendas, evite copiar e colar do que outras empresas fazem. A função de vendas da sua empresa provavelmente possui aspectos exclusivos que exigem serviços específicos para que funcione da maneira que sua equipe de vendas espera.

Identifique o que deve entrar no domínio. Ao determinar o que o domínio deve abranger e quais serviços incluir, o melhor conselho é simplesmente conversar com usuários e partes interessadas, ouvir o que eles estão dizendo e criar os serviços que os ajudarão a realizar seu trabalho. Evite a armadilha clássica de arquitetar no vácuo.

Sistemas distribuídos, escalabilidade e design para falhas

A discussão nesta seção está relacionada ao nosso segundo e terceiro princípios de arquitetura de engenharia de dados discutidos anteriormente: planejar para falhas e arquitetar para escalabilidade. Como engenheiros de dados, estamos interessados em quatro características estreitamente relacionadas dos sistemas de dados (a disponibilidade e a confiabilidade foram mencionadas anteriormente, mas as reiteraremos aqui para fins de integridade):

Escalabilidade

Nos permite aumentar a capacidade de um sistema para melhorar o desempenho e lidar com a demanda. Por exemplo, podemos querer dimensionar um sistema para lidar com uma alta taxa de consultas ou processar um grande conjunto de dados.

Elasticidade

A capacidade de um sistema escalável escalar dinamicamente; um sistema altamente elástico pode aumentar e diminuir automaticamente com base na carga de trabalho atual. A expansão é crítica à medida que a demanda aumenta, enquanto a redução economiza dinheiro em um ambiente de nuvem. Às vezes, os sistemas modernos são dimensionados para zero, o que significa que podem desligar automaticamente quando ociosos.

Disponibilidade

A porcentagem de tempo em que um serviço ou componente de TI está em um estado operável.

Confiabilidade

A probabilidade do sistema de atender aos padrões definidos no desempenho de sua função pretendida durante um intervalo especificado.



Ver PagerDuty's "[Por que a disponibilidade e a confiabilidade são cruciais?](#)" página da [Internet](#) para definições e informações sobre disponibilidade e confiabilidade.

Como essas características estão relacionadas? Se um sistema não atender aos requisitos de desempenho durante um intervalo especificado, ele pode deixar de responder. Assim, a baixa confiabilidade pode levar à baixa disponibilidade. Por outro lado, o dimensionamento dinâmico ajuda a garantir um desempenho adequado sem intervenção manual dos engenheiros – a elasticidade melhora a confiabilidade.

A escalabilidade pode ser realizada de várias maneiras. Para seus serviços e domínios, uma única máquina lida com tudo? Uma única máquina pode ser dimensionada verticalmente; você pode aumentar os recursos (CPU, disco, memória, E/S). Mas há limites rígidos para possíveis recursos em uma única máquina. Além disso, o que acontece se esta máquina morrer? Com tempo suficiente, alguns componentes acabarão falhando. Qual é o seu plano para backup e failover? Máquinas individuais geralmente não podem oferecer alta disponibilidade e confiabilidade.

Utilizamos um sistema distribuído para obter maior capacidade de dimensionamento geral e maior disponibilidade e confiabilidade. *Escala horizontal* permite adicionar mais máquinas para atender aos requisitos de carga e recursos ([Figura 3-4](#)). Os sistemas dimensionados horizontalmente comuns têm um nó líder que atua como o principal ponto de contato para a instanciação, o progresso e a conclusão das cargas de trabalho. Quando uma carga de trabalho é iniciada, o nó líder distribui tarefas para os nós do trabalhador em seu sistema, concluindo as tarefas e retornando os resultados para o nó líder. Arquiteturas distribuídas modernas típicas também são construídas em redundância. Os dados são replicados para que, se uma máquina morrer, as outras máquinas possam continuar de onde o servidor ausente parou; o cluster pode adicionar mais máquinas para restaurar a capacidade.

Os sistemas distribuídos são difundidos nas várias tecnologias de dados que você usará em sua arquitetura. Quase todos os sistemas de armazenamento de objetos de armazenamento de dados em nuvem que você usa têm alguma noção de distribuição sob o capô. Os detalhes de gerenciamento do sistema distribuído geralmente são abstraídos, permitindo que você se concentre na arquitetura de alto nível em vez de encanamentos de baixo nível. No entanto, é altamente recomendável que você aprenda mais sobre sistemas distribuídos porque esses detalhes podem ser extremamente úteis para entender e melhorar o desempenho de seus pipelines; de Martin Kleppmann [Projetando aplicativos com uso intensivo de dados](#) (O'Reilly) é um excelente recurso.

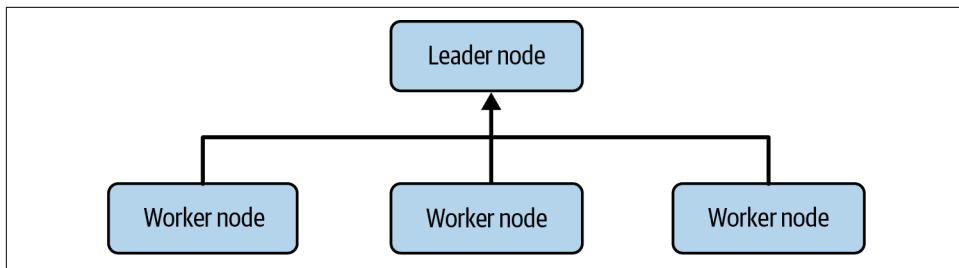


Figura 3-4. Um sistema distribuído horizontal simples utilizando uma arquitetura líder-seguidor, com um nó líder e três nós trabalhadores

Acoplamento rígido versus flexível: camadas, monólitos e microsserviços

Ao projetar uma arquitetura de dados, você escolhe quanta interdependência deseja incluir em seus vários domínios, serviços e recursos. Em uma extremidade do espectro, você pode optar por ter dependências e fluxos de trabalho extremamente centralizados. Cada parte de um domínio e serviço é vitalmente dependente de todos os outros domínios e serviços. Este padrão é conhecido como *fortemente acoplado*.

No outro extremo do espectro, você tem domínios e serviços descentralizados que não dependem estritamente uns dos outros, em um padrão conhecido como *acoplamento solto*. Em um cenário fracamente acoplado, é fácil para equipes descentralizadas criar sistemas cujos dados podem não ser utilizáveis por seus pares. Certifique-se de atribuir padrões comuns, propriedade, responsabilidade e prestação de contas às equipes proprietárias de seus respectivos domínios e serviços. Projetar uma arquitetura de dados “boa” depende de compensações entre o acoplamento rígido e flexível de domínios e serviços.

Vale a pena notar que muitas das ideias nesta seção se originam no desenvolvimento de software. Tentaremos reter o contexto da intenção original dessas grandes ideias e mantê-las agnósticas em relação aos dados - enquanto explicamos posteriormente algumas diferenças das quais você deve estar ciente ao aplicar esses conceitos especificamente aos dados.

Camadas de arquitetura

À medida que você desenvolve sua arquitetura, é útil estar ciente das camadas de arquitetura. Sua arquitetura tem camadas — dados, aplicativos, lógica de negócios, apresentação e assim por diante — e você precisa saber como dissociar essas camadas. Como o acoplamento rígido de modalidades apresenta vulnerabilidades óbvias, lembre-se de como você estrutura as camadas de sua arquitetura para obter o máximo de confiabilidade e flexibilidade. Vejamos a arquitetura de camada única e multicamada.

Camada Única.Em uma *arquitetura de camada única*, seu banco de dados e aplicativo estão fortemente acoplados, residindo em um único servidor (Figura 3-5). Esse servidor pode ser seu laptop ou uma única máquina virtual (VM) na nuvem. A natureza fortemente acoplada significa que se

o servidor, o banco de dados ou o aplicativo falhar, toda a arquitetura falhará. Embora as arquiteturas de camada única sejam boas para prototipagem e desenvolvimento, elas não são recomendadas para ambientes de produção devido aos riscos óbvios de falha.

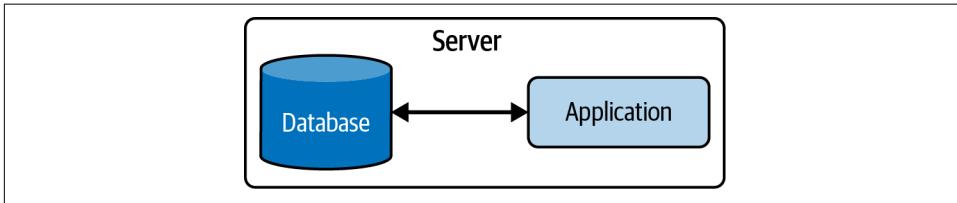


Figura 3-5. Arquitetura de camada única

Mesmo quando arquiteturas de camada única construídas em redundância (por exemplo, uma réplica de failover), elas apresentam limitações significativas de outras maneiras. Por exemplo, muitas vezes não é prático (e não é aconselhável) executar consultas analíticas em bancos de dados de aplicativos de produção. Fazer isso corre o risco de sobrecarregar o banco de dados e fazer com que o aplicativo fique indisponível. Uma arquitetura de camada única é adequada para testar sistemas em uma máquina local, mas não é recomendada para uso em produção.

Multitier. Os desafios de uma arquitetura de camada única fortemente acoplada são resolvidos ao desacoplar os dados e o aplicativo. A *multicamada* (também conhecido como *nível n*) é composta de camadas separadas: dados, aplicação, lógica de negócios, apresentação etc. Essas camadas são de baixo para cima e hierárquicas, o que significa que a camada inferior não depende necessariamente das camadas superiores; as camadas superiores dependem das camadas inferiores. A ideia é separar os dados do aplicativo e o aplicativo da apresentação.

Uma arquitetura multicamada comum é uma arquitetura de três camadas, um projeto cliente-servidor amplamente utilizado. A *arquitetura de três camadas* consiste em dados, lógica de aplicação e camadas de apresentação (Figura 3-6). Cada camada é isolada da outra, permitindo a separação de preocupações. Com uma arquitetura de três camadas, você pode usar qualquer tecnologia de sua preferência em cada camada, sem a necessidade de focar monologicamente.

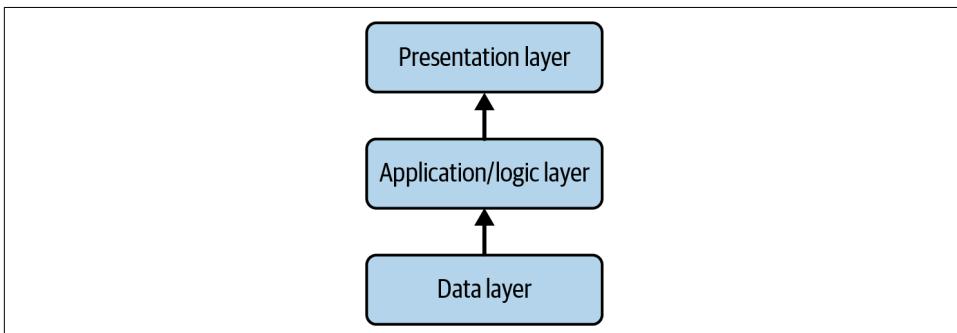


Figura 3-6. Uma arquitetura de três camadas

Vimos muitas arquiteturas de camada única em produção. As arquiteturas de camada única oferecem simplicidade, mas também limitações severas. Eventualmente, uma organização ou aplicativo supera esse arranjo; funciona bem até que não funcione. Por exemplo, em uma arquitetura de camada única, as camadas de dados e lógica compartilham e competem por recursos (disco, CPU e memória) de maneiras que são simplesmente evitadas em uma arquitetura de várias camadas. Os recursos estão distribuídos em vários níveis. Os engenheiros de dados devem usar camadas para avaliar sua arquitetura em camadas e a maneira como as dependências são tratadas. Mais uma vez, comece simples e vá evoluindo para camadas adicionais à medida que sua arquitetura se torna mais complexa.

Em uma arquitetura multicamadas, você precisa considerar a separação de suas camadas e a maneira como os recursos são compartilhados nas camadas ao trabalhar com um sistema distribuído. Os sistemas distribuídos sob o capô potencializam muitas tecnologias que você encontrará em todo o ciclo de vida da engenharia de dados. Primeiro, pense se você deseja contenção de recursos com seus nós. Se não, exerce uma *arquitetura sem compartilhamento*: um único nó lida com cada solicitação, o que significa que outros nós não compartilham recursos como memória, disco ou CPU com este nó ou entre si. Dados e recursos são isolados no nó. Como alternativa, vários nós podem lidar com várias solicitações e compartilhar recursos, mas corre o risco de contenção de recursos. Outra consideração é se os nós devem compartilhar o mesmo disco e memória acessível a todos os nós. Isso é chamado de *arquitetura de disco compartilhado* e é comum quando você deseja recursos compartilhados se ocorrer uma falha de nó aleatório.

Monólitos

A noção geral de um monólito inclui tanto quanto possível sob o mesmo teto; em sua versão mais extrema, um monólito consiste em uma única base de código executada em uma única máquina que fornece a lógica do aplicativo e a interface do usuário.

O acoplamento dentro de monólitos pode ser visto de duas maneiras: acoplamento técnico e acoplamento de domínio. *acoplamento técnico* refere-se a camadas arquitetônicas, enquanto *acoplamento de domínio* refere-se à maneira como os domínios são acoplados. Um monólito tem vários graus de acoplamento entre tecnologias e domínios. Você pode ter um aplicativo com várias camadas desacopladas em uma arquitetura multicamadas, mas ainda compartilhar vários domínios. Ou você pode ter uma arquitetura de camada única atendendo a um único domínio.

O acoplamento apertado de um monólito implica na falta de modularidade de seus componentes. Trocar ou atualizar componentes em um monólito geralmente é um exercício de troca de uma dor por outra. Devido à natureza fortemente acoplada, a reutilização de componentes em toda a arquitetura é difícil ou impossível. Ao avaliar como melhorar uma arquitetura monolítica, muitas vezes é um jogo de whack-a-mole: um componente é melhorado, muitas vezes à custa de consequências desconhecidas com outras áreas do monólito.

As equipes de dados geralmente ignoram a solução da crescente complexidade de seu monólito, permitindo que ele se transforme em um grande bola de lama.

Capítulo 4 fornece uma discussão mais extensa comparando monólitos com tecnologias distribuídas. Também discutimos o *monólito distribuído*, um estranho híbrido que surge quando os engenheiros constroem sistemas distribuídos com acoplamento rígido excessivo.

Microsserviços

Comparados com os atributos de um monólito – serviços interligados, centralização e forte acoplamento entre os serviços – os microsserviços são o oposto. *Arquitetura de microsserviços* compreende serviços separados, descentralizados e fracamente acoplados. Cada serviço tem uma função específica e é dissociado de outros serviços que operam em seu domínio. Se um serviço ficar temporariamente inativo, isso não afetará a capacidade de outros serviços continuarem funcionando.

Uma questão que surge com frequência é como converter seu monólito em vários microsserviços ([Figura 3-7](#)). Isso depende completamente da complexidade do seu monólito e de quanto esforço será necessário para começar a extraír serviços dele. É totalmente possível que seu monólito não possa ser desmembrado; nesse caso, você desejará começar a criar uma nova arquitetura paralela que tenha os serviços desacoplados de maneira amigável aos microsserviços. Não sugerimos uma refatoração inteira, mas, em vez disso, dividir os serviços. O monólito não surgiu da noite para o dia e é uma questão de tecnologia como organizacional. Certifique-se de obter a adesão das partes interessadas do monólito se planeja separá-lo.

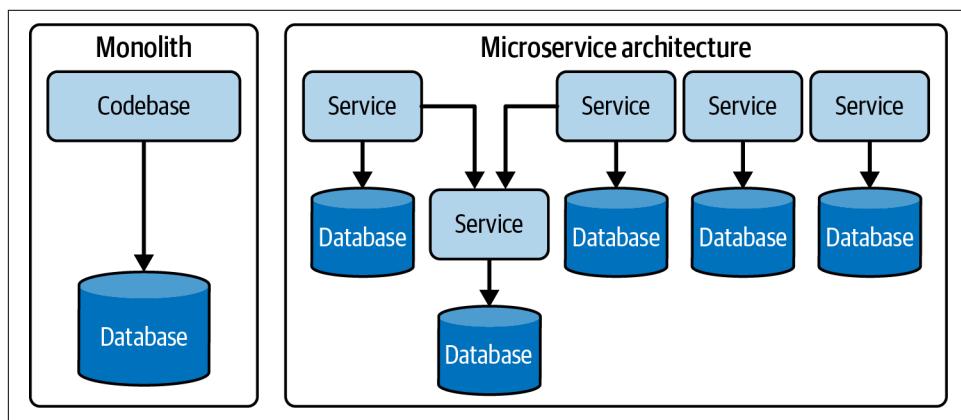


Figura 3-7. Uma arquitetura extremamente monólica executa todas as funcionalidades dentro de uma única base de código, potencialmente colocando um banco de dados no mesmo servidor host

Se você quiser aprender mais sobre como quebrar um monólito, sugerimos a leitura do fantástico e pragmático guia [Arquitetura de software: as partes difíceis](#) por Neal Ford e cols. (O'Reilly).

Considerações para arquitetura de dados

Como mencionamos no início desta seção, os conceitos de acoplamento rígido versus acoplamento flexível decorrem do desenvolvimento de software, com alguns desses conceitos datando de mais de 20 anos. Embora as práticas arquitetônicas em dados agora estejam adotando aquelas do desenvolvimento de software, ainda é comum ver arquiteturas de dados muito monolíticas e fortemente acopladas. Parte disso se deve à natureza das tecnologias de dados existentes e à forma como elas se integram.

Por exemplo, pipelines de dados podem consumir dados de muitas fontes ingeridas em um data warehouse central. O data warehouse central é inherentemente monolítico. Um movimento em direção a um equivalente de microsserviços com um data warehouse é desacoplar o fluxo de trabalho com pipelines de dados específicos de domínio conectando-se a data warehouses específicos de domínio correspondentes. Por exemplo, o pipeline de dados de vendas se conecta ao data warehouse específico de vendas e os domínios de estoque e produto seguem um padrão semelhante.

Em vez de pregar microsserviços dogmaticamente sobre monólitos (entre outros argumentos), sugerimos que você use pragmaticamente o acoplamento flexível como um ideal, enquanto reconhece o estado e as limitações das tecnologias de dados que está usando em sua arquitetura de dados. Incorpore opções de tecnologia reversíveis que permitem modularidade e acoplamento flexível sempre que possível.

Como você pode ver em [Figura 3-7](#), você separa os componentes de sua arquitetura em diferentes camadas de interesse de forma vertical. Embora uma arquitetura multicamada resolva os desafios técnicos de desacoplar recursos compartilhados, ela não aborda a complexidade de compartilhar domínios. Ao longo das linhas de arquitetura única versus multicamada, você também deve considerar como separa os domínios de sua arquitetura de dados. Por exemplo, sua equipe de analistas pode contar com dados de vendas e estoque. Os domínios de vendas e estoque são diferentes e devem ser vistos separadamente.

Uma abordagem para esse problema é a centralização: uma única equipe é responsável por coletar dados de todos os domínios e reconciliá-los para consumo em toda a organização. (Essa é uma abordagem comum no armazenamento de dados tradicional.) Outra abordagem é a *malha de dados*. Com a malha de dados, cada equipe de software é responsável por preparar seus dados para consumo no restante da organização. Falaremos mais sobre a malha de dados mais adiante neste capítulo.

Nosso conselho: os monólitos não são necessariamente ruins e pode fazer sentido começar com um sob certas condições. Às vezes, você precisa se mover rapidamente e é muito mais simples começar com um monólito. Apenas esteja preparado para quebrá-lo em pedaços menores eventualmente; não fique muito confortável.

Acesso do usuário: único versus multilocatário

Como engenheiro de dados, você precisa tomar decisões sobre o compartilhamento de sistemas entre várias equipes, organizações e clientes. De certa forma, todos os serviços em nuvem são

multitenant, embora essa multilocação ocorra em vários grãos. Por exemplo, uma instância de computação em nuvem geralmente está em um servidor compartilhado, mas a própria VM fornece algum grau de isolamento. O armazenamento de objetos é um sistema multilocatário, mas os fornecedores de nuvem garantem segurança e isolamento, desde que os clientes configurem suas permissões corretamente.

Os engenheiros frequentemente precisam tomar decisões sobre multilocação em uma escala muito menor. Por exemplo, vários departamentos em uma grande empresa compartilham o mesmo data warehouse? A organização compartilha dados para vários grandes clientes na mesma tabela?

Temos dois fatores a considerar na multilocação: desempenho e segurança. Com vários locatários grandes em um sistema de nuvem, o sistema oferecerá suporte a um desempenho consistente para todos os locatários ou haverá um problema de vizinho barulhento? (Ou seja, o alto uso de um inquilino prejudicará o desempenho de outros inquilinos?) Em relação à segurança, os dados de diferentes inquilinos devem ser devidamente isolados. Quando uma empresa tem vários locatários de clientes externos, esses locatários não devem estar cientes uns dos outros e os engenheiros devem evitar o vazamento de dados. As estratégias para isolamento de dados variam de acordo com o sistema. Por exemplo, muitas vezes é perfeitamente aceitável usar tabelas multitenant e isolar dados por meio de exibições. No entanto, você deve certificar-se de que essas exibições não possam vazar dados. Leia a documentação do fornecedor ou do projeto para entender as estratégias e os riscos apropriados.

Arquitetura orientada a eventos

Sua empresa raramente é estática. Muitas vezes acontecem coisas em seu negócio, como conseguir um novo cliente, um novo pedido de um cliente ou um pedido de um produto ou serviço. Estes são todos exemplos de *eventos* que são amplamente definidos como algo que aconteceu, normalmente uma mudança no estado de alguma coisa. Por exemplo, um novo pedido pode ser criado por um cliente ou um cliente pode posteriormente fazer uma atualização nesse pedido.

Um fluxo de trabalho orientado a eventos ([Figura 3-8](#)) abrange a capacidade de criar, atualizar e mover eventos de forma assíncrona em várias partes do ciclo de vida da engenharia de dados. Esse fluxo de trabalho se resume a três áreas principais: produção de eventos, roteamento e consumo. Um evento deve ser produzido e encaminhado para algo que o consuma sem dependências fortemente acopladas entre o produtor, o roteador de eventos e o consumidor.



Figura 3-8. Em um fluxo de trabalho orientado a eventos, um evento é produzido, roteado e consumido

Uma arquitetura orientada a eventos ([Figura 3-9](#)) adota o fluxo de trabalho orientado a eventos e o usa para se comunicar em vários serviços. A vantagem de uma arquitetura orientada a eventos é que ela distribui o estado de um evento por vários serviços. Isso é útil se um serviço ficar offline, um nó falhar em um sistema distribuído ou se você quiser que vários consumidores ou serviços accessem os mesmos eventos. Sempre que você tiver serviços fracamente acoplados, esse é um candidato à arquitetura orientada a eventos. Muitos dos exemplos que descrevemos mais adiante neste capítulo incorporam alguma forma de arquitetura orientada a eventos.

Você aprenderá mais sobre streaming orientado a eventos e sistemas de mensagens em[capítulo 5](#).

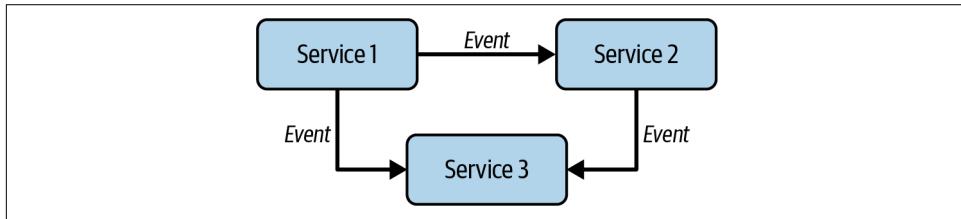


Figura 3-9. Em uma arquitetura orientada a eventos, os eventos são transmitidos entre serviços fracamente acoplados

Projetos brownfield versus projetos greenfield

Antes de projetar seu projeto de arquitetura de dados, você precisa saber se está começando do zero ou reprojetando uma arquitetura existente. Cada tipo de projeto requer a avaliação de compensações, embora com diferentes considerações e abordagens. Os projetos se enquadram em dois grupos: brownfield e greenfield.

projetos brownfield

projetos brownfield geralmente envolvem refatoração e reorganização de uma arquitetura existente e são limitados pelas escolhas do presente e do passado. Como uma parte fundamental da arquitetura é o gerenciamento de mudanças, você deve descobrir uma maneira de contornar essas limitações e projetar um caminho a seguir para atingir seus novos objetivos comerciais e técnicos. Projetos brownfield requerem um entendimento completo da arquitetura legada e a interação de várias tecnologias antigas e novas. Com muita frequência, é fácil criticar o trabalho e as decisões de uma equipe anterior, mas é muito melhor se aprofundar, fazer perguntas e entender por que as decisões foram tomadas. A empatia e o contexto ajudam muito a diagnosticar problemas com a arquitetura existente, identificar oportunidades e reconhecer armadilhas.

Você precisará apresentar sua nova arquitetura e tecnologias e depreciar o material antigo em algum momento. Vejamos algumas abordagens populares. Muitas equipes mergulham de cabeça em uma revisão completa ou big-bang da arquitetura antiga, muitas vezes descobrindo a depreciação à medida que avançam. Embora popular, não recomendamos essa abordagem

devido aos riscos associados e à falta de um plano. Esse caminho muitas vezes leva ao desastre, com muitas decisões irreversíveis e caras. Seu trabalho é tomar decisões reversíveis e de alto ROI.

Uma alternativa popular para uma reescrita direta é o padrão estrangulador: novos sistemas substituem lenta e gradualmente os componentes de uma arquitetura legada.²⁰ Eventualmente, a arquitetura legada é completamente substituída. A atração pelo padrão estrangulador é sua abordagem direcionada e cirúrgica de desaprovar uma peça de um sistema por vez. Isso permite decisões flexíveis e reversíveis ao avaliar o impacto da descontinuação em sistemas dependentes.

É importante observar que a depreciação pode ser um conselho de “torre de marfim” e não é prática ou alcançável. Erradicar tecnologia ou arquitetura legada pode ser impossível se você estiver em uma grande organização. Alguém, em algum lugar, está usando esses componentes legados. Como alguém disse uma vez: “Legado é uma maneira condescendente de descrever algo que gera dinheiro”.

Se você puder desaprovar, entenda que existem várias maneiras de desaprovar sua arquitetura antiga. É fundamental demonstrar valor na nova plataforma aumentando gradualmente sua maturidade para mostrar evidências de sucesso e, em seguida, seguir um plano de saída para desligar os sistemas抗igos.

Projetos Greenfield

No extremo oposto do espectro, um projeto *greenfield* permite que você seja pioneiro em um novo começo, sem restrições pela história ou legado de uma arquitetura anterior. Projetos greenfield tendem a ser mais fáceis do que projetos brownfield, e muitos arquitetos e engenheiros de dados os consideram mais divertidos! Você tem a oportunidade de experimentar as ferramentas e os padrões arquitetônicos mais novos e interessantes. O que poderia ser mais emocionante?

Você deve estar atento a algumas coisas antes de se deixar levar. Vemos as equipes ficarem excessivamente exuberantes com a síndrome do objeto brilhante. Eles se sentem compelidos a buscar a última e maior moda tecnológica sem entender como isso afetará o valor do projeto. Há também uma tentação de fazer *desenvolvimento orientado por currículo*, acumulando novas tecnologias impressionantes sem priorizar os objetivos finais do projeto.²¹ Sempre priorize os requisitos ao invés de construir algo legal.

Esteja você trabalhando em um projeto brownfield ou greenfield, sempre concentre-se nos princípios da “boa” arquitetura de dados. Avalie os trade-offs, tome decisões flexíveis e reversíveis e busque um ROI positivo.

20 Martin Fowler, “StranglerFigApplication”, 29 de junho de 2004, <https://oreil.ly/PmqxB>.

21 Mike Loukides, “Resume Driven Development,” *Radar O’Reilly*, 13 de outubro de 2004, <https://oreil.ly/BUHa8>.

Agora, veremos exemplos e tipos de arquiteturas – algumas estabelecidas por décadas (o data warehouse), algumas totalmente novas (o data lakehouse) e algumas que surgiram e desapareceram rapidamente, mas ainda influenciam os padrões de arquitetura atuais (Lambda arquitetura).

Exemplos e tipos de arquitetura de dados

Como a arquitetura de dados é uma disciplina abstrata, ajuda raciocinar pelo exemplo. Nesta seção, descrevemos exemplos proeminentes e tipos de arquitetura de dados que são populares atualmente. Embora este conjunto de exemplos não seja de forma alguma exaustivo, a intenção é expô-lo a alguns dos padrões de arquitetura de dados mais comuns e fazê-lo pensar sobre a flexibilidade necessária e a análise de trade-off necessária ao projetar uma boa arquitetura para seu caso de uso .

Armazém de dados

Armazém de dados é um hub de dados central usado para relatórios e análises. Os dados em um data warehouse geralmente são altamente formatados e estruturados para casos de uso de análise. Está entre as arquiteturas de dados mais antigas e bem estabelecidas.

Em 1989, Bill Inmon originou a noção de data warehouse, que ele descreveu como “uma coleção de dados orientada por assunto, integrada, não volátil e variável no tempo para apoiar as decisões da administração”.²² Embora os aspectos técnicos do data warehouse tenham evoluído significativamente, sentimos que essa definição original ainda mantém seu peso hoje.

No passado, os data warehouses eram amplamente usados em empresas com orçamentos significativos (geralmente na casa dos milhões de dólares) para adquirir sistemas de dados e pagar equipes internas para fornecer suporte contínuo para manter o data warehouse. Isso era caro e trabalhoso. Desde então, o modelo escalável de pagamento conforme o uso tornou os armazéns de dados em nuvem acessíveis até mesmo para pequenas empresas. Como um provedor terceirizado gerencia a infraestrutura do data warehouse, as empresas podem fazer muito mais com menos pessoas, mesmo com o aumento da complexidade de seus dados.

Vale a pena observar dois tipos de arquitetura de data warehouse: organizacional e técnica. O *arquitetura de data warehouse organizacional* organiza os dados associados a determinadas estruturas e processos da equipe de negócios. O *arquitetura de data warehouse técnica* reflete a natureza técnica do data warehouse, como o MPP. Uma empresa pode ter um data warehouse sem um sistema MPP ou executar um sistema MPP que não esteja organizado como um data warehouse. No entanto, as arquiteturas técnica e organizacional existiram em um ciclo virtuoso e são frequentemente identificadas umas com as outras.

22 HW Inmon, *Construindo o Data Warehouse*(Hoboken: Wiley, 2005).

A arquitetura de data warehouse organizacional tem duas características principais:

Separa o processamento analítico online (OLAP) dos bancos de dados de produção (processamento de transações online)

Essa separação é crítica à medida que os negócios crescem. Mover dados para um sistema físico separado direciona a carga para longe dos sistemas de produção e melhora o desempenho analítico.

Centraliza e organiza os dados

Tradicionalmente, um data warehouse extrai dados de sistemas de aplicativos usando ETL. A fase de extração extrai dados dos sistemas de origem. A fase de transformação limpa e padroniza os dados, organizando e impondo a lógica de negócios de forma altamente modelada. ([Capítulo 8](#) sobre transformações e modelos de dados.) A fase de carregamento empurra os dados para o sistema de banco de dados de destino do data warehouse. Os dados são carregados em vários data marts que atendem às necessidades analíticas de linhas ou negócios e departamentos específicos. [Figura 3-10](#) mostra o fluxo de trabalho geral. O data warehouse e o ETL andam de mãos dadas com estruturas de negócios específicas, incluindo equipes de desenvolvimento de DBA e ETL que implementam a direção dos líderes de negócios para garantir que os dados para relatórios e análises correspondam aos processos de negócios.

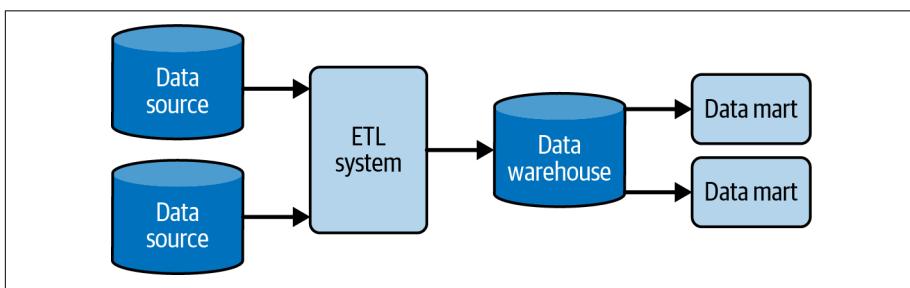


Figura 3-10. Armazém de dados básico com ETL

Em relação à arquitetura de data warehouse técnico, os primeiros sistemas MPP no final da década de 1970 tornaram-se populares na década de 1980. Os MPPs suportam essencialmente a mesma semântica SQL usada em bancos de dados de aplicativos relacionais. Ainda assim, eles são otimizados para digitalizar grandes quantidades de dados em paralelo e, assim, permitir agregação de alto desempenho e cálculos estatísticos. Nos últimos anos, os sistemas MPP mudaram cada vez mais de uma arquitetura baseada em linhas para uma arquitetura colunar para facilitar dados e consultas ainda maiores, especialmente em data warehouses na nuvem. Os MPPs são indispensáveis para a execução de consultas de alto desempenho para grandes empresas à medida que as necessidades de dados e relatórios aumentam.

Uma variação do ETL é o ELT. Com a arquitetura de data warehouse ELT, os dados são movidos mais ou menos diretamente dos sistemas de produção para uma área de preparação no data warehouse. A preparação nesta configuração indica que os dados estão em um formato bruto. Em vez de usar um sistema externo, as transformações são tratadas diretamente no data warehouse. A intenção é aproveitar o enorme poder computacional dos dados em nuvem.

armazéns e ferramentas de processamento de dados. Os dados são processados em lotes e a saída transformada é gravada em tabelas e exibições para análise. Figura 3-11 mostra o processo geral. O ELT também é popular em um arranjo de streaming, pois os eventos são transmitidos de um processo CDC, armazenados em uma área de preparação e posteriormente transformados no data warehouse.

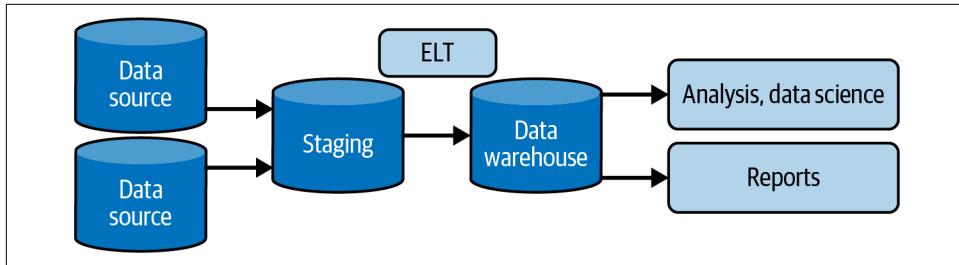


Figura 3-11. ELT—extrair, carregar e transformar

Uma segunda versão do ELT foi popularizada durante o crescimento de big data no ecossistema Hadoop. Isso é *ELT de transformação na leitura*, que discutimos em “[Data Lake](#)” na página 101.

O armazenamento de dados na nuvem

Armazéns de dados na nuvem representam uma evolução significativa da arquitetura de data warehouse local e, portanto, levaram a mudanças significativas na arquitetura organizacional. O Amazon Redshift deu início à revolução do armazenamento de dados na nuvem. Em vez de precisar dimensionar adequadamente um sistema MPP para os próximos anos e assinar um contrato multimilionário para adquirir o sistema, as empresas tiveram a opção de ativar um cluster Redshift sob demanda, ampliando-o ao longo do tempo à medida que a demanda de dados e análises crescia. Eles podiam até criar novos clusters Redshift sob demanda para atender a cargas de trabalho específicas e excluir clusters rapidamente quando não fossem mais necessários.

Google BigQuery, Snowflake e outros concorrentes popularizaram a ideia de separar a computação do armazenamento. Nessa arquitetura, os dados são armazenados no armazenamento de objetos, permitindo armazenamento virtualmente ilimitado. Isso também oferece aos usuários a opção de aumentar o poder de computação sob demanda, fornecendo recursos de big data ad hoc sem o custo de longo prazo de milhares de nós.

Os armazéns de dados em nuvem expandem os recursos dos sistemas MPP para cobrir muitos casos de uso de big data que exigiam um cluster Hadoop no passado muito recente. Eles podem processar prontamente petabytes de dados em uma única consulta. Eles normalmente suportam estruturas de dados que permitem o armazenamento de dezenas de megabytes de dados de texto bruto por linha ou documentos JSON extremamente ricos e complexos. À medida que os data warehouses em nuvem (e data lakes) amadurecem, a linha entre o data warehouse e o data lake continuará a se confundir.

O impacto dos novos recursos oferecidos pelos data warehouses em nuvem é tão significativo que podemos considerar descartar o termo *armazém de dados* completamente. Em vez disso, esses serviços estão evoluindo para uma nova plataforma de dados com recursos muito mais amplos do que os oferecidos por um sistema MPP tradicional.

data marts

A *data mart* é um subconjunto mais refinado de um warehouse projetado para servir análises e relatórios, focado em uma única suborganização, departamento ou linha de negócios; cada departamento tem seu próprio data mart, específico para suas necessidades. Isso contrasta com o data warehouse completo que atende a uma organização ou negócio mais amplo.

Os data marts existem por dois motivos. Primeiro, um data mart torna os dados mais facilmente acessíveis a analistas e desenvolvedores de relatórios. Em segundo lugar, os data marts fornecem um estágio adicional de transformação além daquele fornecido pelos pipelines ETL ou ELT iniciais. Isso pode melhorar significativamente o desempenho se relatórios ou consultas analíticas exigirem junções e agregações complexas de dados, especialmente quando os dados brutos são grandes. Os processos de transformação podem preencher o data mart com dados unidos e agrupados para melhorar o desempenho de consultasativas. [Figura 3-12](#) mostra o fluxo de trabalho geral. Discutimos data marts e dados de modelagem para data marts, em [Capítulo 8](#).

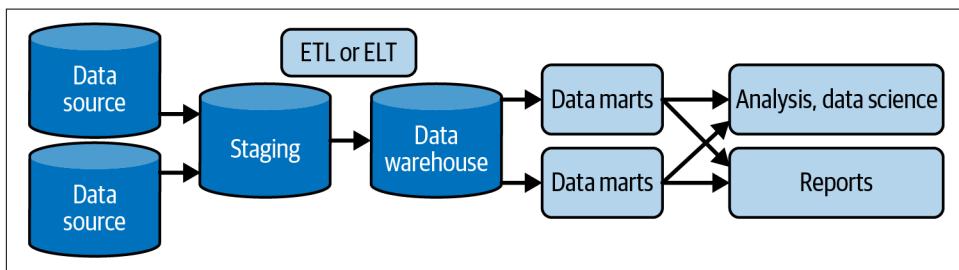


Figura 3-12. ETL ou ELT mais data marts

Data Lake

Entre as arquiteturas mais populares que surgiram durante a era do big data está a *lago de dados*. Em vez de impor limitações estruturais rígidas aos dados, por que não simplesmente despejar todos os seus dados — estruturados e não estruturados — em um local central? O data lake prometia ser uma força democratizadora, liberando os negócios para beber de uma fonte de dados ilimitados. O data lake de primeira geração, “data lake 1.0”, fez contribuições sólidas, mas geralmente não cumpriu sua promessa.

Data lake 1.0 começou com HDFS. À medida que a nuvem cresceu em popularidade, esses data lakes mudaram para o armazenamento de objetos baseado em nuvem, com custos de armazenamento extremamente baratos e capacidade de armazenamento virtualmente ilimitada. Em vez de depender de um data warehouse monolítico onde o armazenamento e a computação estão fortemente acoplados, o data lake permite o armazenamento de uma imensa quantidade de dados de qualquer tamanho e tipo. Quando esses dados precisam ser consultados ou

transformado, você tem acesso a um poder de computação quase ilimitado girando um cluster sob demanda e pode escolher sua tecnologia de processamento de dados favorita para a tarefa em questão - MapReduce, Spark, Ray, Presto, Hive, etc.

Apesar da promessa e do hype, o data lake 1.0 tinha sérias deficiências. O data lake se tornou um depósito de lixo; termos como *pântano de dados*, *dados obscuros*, e *DESGASTADO* foram cunhados como projetos de dados outrora promissores falharam. Os dados cresceram para tamanhos incontroláveis, com pouco gerenciamento de esquema, catalogação de dados e ferramentas de descoberta. Além disso, o conceito original de data lake era essencialmente somente gravação, criando grandes dores de cabeça com a chegada de regulamentações como o GDPR, que exigiam a exclusão direcionada de registros de usuários.

O processamento de dados também foi desafiador. Transformações de dados relativamente banais, como junções, eram uma grande dor de cabeça para codificar como tarefas MapReduce. Estruturas posteriores, como Pig e Hive, melhoraram um pouco a situação do processamento de dados, mas fizeram pouco para resolver os problemas básicos de gerenciamento de dados. As operações simples de linguagem de manipulação de dados (DML) comuns em SQL — excluir ou atualizar linhas — eram difíceis de implementar, geralmente obtidas com a criação de tabelas totalmente novas. Embora os engenheiros de big data irradiassem um desdém particular por seus equivalentes em armazenamento de dados, os últimos poderiam apontar que os armazéns de dados forneciam recursos básicos de gerenciamento de dados prontos para uso e que o SQL era uma ferramenta eficiente para escrever consultas e transformações complexas e de alto desempenho.

O data lake 1.0 também falhou em cumprir outra promessa central do movimento de big data. O software de código aberto no ecossistema Apache foi apresentado como um meio de evitar contratos multimilionários para sistemas MPP proprietários. Hardware barato e pronto para uso substituiria soluções personalizadas de fornecedores. Na realidade, os custos de big data aumentaram à medida que as complexidades do gerenciamento de clusters Hadoop forçaram as empresas a contratar grandes equipes de engenheiros com altos salários. Muitas vezes, as empresas optam por comprar versões licenciadas e personalizadas do Hadoop de fornecedores para evitar os fios expostos e as bordas afiadas da base de código Apache bruta e adquirir um conjunto de ferramentas de andaime para tornar o Hadoop mais fácil de usar. Mesmo as empresas que evitavam gerenciar clusters Hadoop usando armazenamento em nuvem tiveram que gastar muito em talentos para escrever trabalhos MapReduce.

Devemos ter cuidado para não subestimar a utilidade e o poder dos data lakes de primeira geração. Muitas organizações encontraram valor significativo em data lakes – especialmente grandes empresas de tecnologia do Vale do Silício, fortemente focadas em dados, como Netflix e Facebook. Essas empresas tinham os recursos para criar práticas de dados bem-sucedidas e criar suas ferramentas e aprimoramentos personalizados baseados em Hadoop. Mas, para muitas organizações, os data lakes se transformaram em um superfundo interno de desperdício, decepção e custos crescentes.

Convergência, data lakes de próxima geração e a plataforma de dados

Em resposta às limitações dos data lakes de primeira geração, vários participantes procuraram aprimorar o conceito para cumprir plenamente sua promessa. Por exemplo, Databricks

introduziu a noção de *data lakehouse*. O lakehouse incorpora os controles, o gerenciamento de dados e as estruturas de dados encontrados em um data warehouse enquanto ainda hospeda os dados no armazenamento de objetos e oferece suporte a uma variedade de mecanismos de consulta e transformação. Em particular, o data lakehouse oferece suporte a transações de atomicidade, consistência, isolamento e durabilidade (ACID), uma grande diferença do data lake original, onde você simplesmente despeja dados e nunca os atualiza ou exclui. O termo *data lakehouse* sugere uma convergência entre data lakes e data warehouses.

A arquitetura técnica dos data warehouses em nuvem evoluiu para ser muito semelhante a uma arquitetura de data lake. Os data warehouses em nuvem separam a computação do armazenamento, oferecem suporte a consultas em escala de petabytes, armazenam uma variedade de dados não estruturados e objetos semiestruturados e se integram a tecnologias de processamento avançadas, como Spark ou Beam.

Acreditamos que a tendência de convergência só vai continuar. O data lake e o data warehouse ainda existirão como arquiteturas diferentes. Na prática, seus recursos irão convergir para que poucos usuários percebam um limite entre eles em seu trabalho diário. Agora vemos vários fornecedores oferecendo *plataformas de dados* que combinam recursos de data lake e data warehouse. Do nosso ponto de vista, AWS, Azure, [Google Cloud](#), [Floco de neve](#), e Databricks são líderes de classe, cada um oferecendo uma constelação de ferramentas totalmente integradas para trabalhar com dados, variando de relacional a completamente não estruturada. Em vez de escolher entre uma arquitetura de data lake ou data warehouse, os futuros engenheiros de dados terão a opção de escolher uma plataforma de dados convergentes com base em vários fatores, incluindo fornecedor, ecossistema e abertura relativa.

pilha de dados moderna

O pilha de dados moderna ([Figura 3-13](#)) é atualmente uma arquitetura de análise moderna que destaca o tipo de abstração que esperamos ver mais amplamente usado nos próximos anos. Considerando que as pilhas de dados anteriores dependiam de conjuntos de ferramentas monolíticos e caros, o principal objetivo da pilha de dados moderna é usar componentes prontos para uso, plug-and-play, fáceis de usar e baseados em nuvem para criar um sistema modular e de baixo custo. - arquitetura de dados eficaz. Esses componentes incluem pipelines de dados, armazenamento, transformação, gerenciamento/governança de dados, monitoramento, visualização e exploração. O domínio ainda está em fluxo e as ferramentas específicas estão mudando e evoluindo rapidamente, mas o objetivo principal permanecerá o mesmo: reduzir a complexidade e aumentar a modularização. Observe que a noção de uma pilha de dados moderna se integra bem com a ideia de plataforma de dados convergentes da seção anterior.

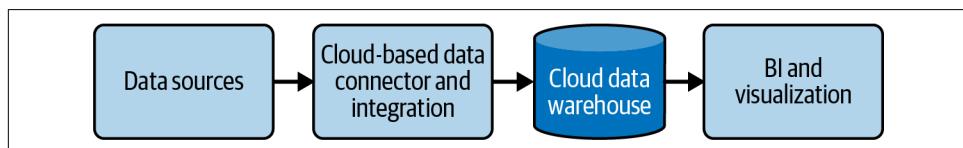


Figura 3-13. Componentes básicos da pilha de dados moderna

Os principais resultados da pilha de dados moderna são autoatendimento (análise e pipelines), gerenciamento ágil de dados e uso de ferramentas de código aberto ou ferramentas proprietárias simples com estruturas de preços claras. A comunidade também é um aspecto central da pilha de dados moderna. Ao contrário dos produtos do passado, que tinham lançamentos e roteiros amplamente ocultos dos usuários, os projetos e as empresas que operam no espaço moderno da pilha de dados normalmente têm bases de usuários fortes e comunidades ativas que participam do desenvolvimento usando o produto antecipadamente, sugerindo recursos e enviando pull pedidos para melhorar o código.

Independentemente de onde o “moderno” vá (compartilhamos nossas ideias em [Capítulo 11](#)), acreditamos que o conceito-chave de modularidade plug-and-play com preços e implementação fáceis de entender é o caminho do futuro. Especialmente na engenharia analítica, a pilha de dados moderna é e continuará sendo a escolha padrão da arquitetura de dados. Ao longo do livro, a arquitetura a que nos referimos contém partes da pilha de dados moderna, como componentes modulares plug-and-play e baseados em nuvem.

Arquitetura lambda

Nos “velhos tempos” (início a meados da década de 2010), a popularidade de trabalhar com dados de streaming explodiu com o surgimento do Kafka como uma fila de mensagens altamente escalável e estruturas como Apache Storm e Samza para streaming/análise em tempo real. Isso. Essas tecnologias permitiram que as empresas executassem novos tipos de análise e modelagem em grandes quantidades de dados, agregação e classificação de usuários e recomendações de produtos. Os engenheiros de dados precisavam descobrir como reconciliar dados em lote e streaming em uma única arquitetura. A arquitetura Lambda foi uma das primeiras respostas populares a esse problema.

Em uma *arquitetura lambda* ([Figura 3-14](#)), você tem sistemas operando independentemente uns dos outros — lote, streaming e serviço. O sistema de origem é idealmente imutável e apenas anexado, enviando dados para dois destinos para processamento: fluxo e lote. O processamento in-stream pretende servir os dados com a menor latência possível em uma camada de “velocidade”, geralmente um banco de dados NoSQL. Na camada batch, os dados são processados e transformados em um sistema como um data warehouse, criando visões pré-computadas e agregadas dos dados. A camada de serviço fornece uma visualização combinada agregando os resultados da consulta das duas camadas.

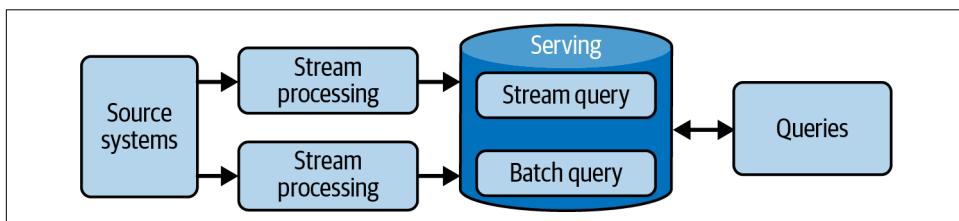


Figura 3-14. arquitetura lambda

A arquitetura Lambda tem sua parcela de desafios e críticas. Gerenciar vários sistemas com diferentes bases de código é tão difícil quanto parece, criando sistemas propensos a erros com códigos e dados extremamente difíceis de reconciliar.

Mencionamos a arquitetura Lambda porque ela ainda chama a atenção e é popular nos resultados de mecanismos de pesquisa para arquitetura de dados. O Lambda não é nossa primeira recomendação se você estiver tentando combinar streaming e dados em lote para análise. A tecnologia e as práticas mudaram.

Em seguida, vamos ver uma reação à arquitetura Lambda, a arquitetura Kappa.

Arquitetura Kappa

Como resposta às deficiências da arquitetura Lambda, Jay Kreps propôs uma alternativa chamada *arquitetura Kappa* (Figura 3-15).²³ A tese central é esta: por que não usar apenas uma plataforma de processamento de fluxo como a espinha dorsal para todo o tratamento de dados – ingestão, armazenamento e serviço? Isso facilita uma verdadeira arquitetura baseada em eventos. O processamento em tempo real e em lote pode ser aplicado perfeitamente aos mesmos dados, lendo o fluxo de eventos ao vivo diretamente e reproduzindo grandes blocos de dados para processamento em lote.

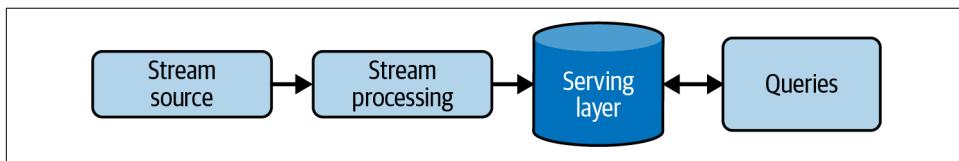


Figura 3-15. arquitetura Kappa

Embora o artigo original de arquitetura Kappa tenha sido publicado em 2014, não o vimos amplamente adotado. Pode haver algumas razões para isso. Primeiro, o streaming em si ainda é um mistério para muitas empresas; é fácil falar, mas mais difícil do que o esperado para executar. Em segundo lugar, a arquitetura Kappa acaba sendo complicada e cara na prática. Embora alguns sistemas de streaming possam ser dimensionados para grandes volumes de dados, eles são complexos e caros; o armazenamento e o processamento em lote permanecem muito mais eficientes e econômicos para enormes conjuntos de dados históricos.

O modelo de fluxo de dados e lote e streaming unificados

Tanto o Lambda quanto o Kappa procuraram abordar as limitações do ecossistema Hadoop da década de 2010, tentando unir ferramentas complicadas que provavelmente não eram ajustes naturais em primeiro lugar. O desafio central de unificar dados em lote e streaming permaneceu, e Lambda e Kappa forneceram inspiração e base para o progresso contínuo nessa busca.

²³ Jay Kreps, "Questionando a Arquitetura Lambda," *Radar O'Reilly*, 2 de julho de 2014, <https://oreil.ly/wWR3n>.

Um dos problemas centrais do gerenciamento de processamento em lote e fluxo é a unificação de vários caminhos de código. Embora a arquitetura Kappa dependa de uma camada unificada de enfileiramento e armazenamento, ainda é necessário confrontar o uso de diferentes ferramentas para coletar estatísticas em tempo real ou executar tarefas de agregação em lote. Hoje, os engenheiros procuram resolver isso de várias maneiras. O Google deixou sua marca ao desenvolver o [modelo de fluxo de dados](#) a [Apache Beam](#) framework que implementa este modelo.

A ideia central do modelo Dataflow é visualizar todos os dados como eventos, pois a agregação é realizada em vários tipos de janelas. Fluxos de eventos contínuos em tempo real são *dados ilimitados*. Os lotes de dados são simplesmente fluxos de eventos limitados e os limites fornecem uma janela natural. Os engenheiros podem escolher entre várias janelas para agregação em tempo real, como deslizar ou cair. O processamento em lote e em tempo real ocorre no mesmo sistema usando código quase idêntico.

A filosofia de “lote como um caso especial de streaming” agora é mais difundida. Vários frameworks, como Flink e Spark, adotaram uma abordagem semelhante.

Arquitetura para IoT

O *Internet das Coisas*(IoT) é a coleção distribuída de dispositivos, também conhecida como *coisas*— computadores, sensores, dispositivos móveis, dispositivos domésticos inteligentes e qualquer outra coisa com conexão à Internet. Em vez de gerar dados de entrada humana direta (pense na entrada de dados de um teclado), os dados da IoT são gerados a partir de dispositivos que coletam dados periodicamente ou continuamente do ambiente circundante e os transmitem a um destino. Os dispositivos IoT geralmente são de baixa potência e operam em ambientes com poucos recursos/ baixa largura de banda.

Embora o conceito de dispositivos IoT remonte há pelo menos algumas décadas, a revolução dos smartphones criou um enorme enxame de IoT praticamente da noite para o dia. Desde então, surgiram várias novas categorias de IoT, como termostatos inteligentes, sistemas de entretenimento automotivo, TVs inteligentes e alto-falantes inteligentes. A IoT evoluiu de uma fantasia futurista para um enorme domínio de engenharia de dados. Esperamos que a IoT se torne uma das formas dominantes de geração e consumo de dados, e esta seção é um pouco mais profunda do que as outras que você leu.

Ter uma compreensão superficial da arquitetura IoT ajudará você a entender as tendências mais amplas da arquitetura de dados. Vejamos brevemente alguns conceitos de arquitetura de IoT.

Dispositivos

Dispositivos(também conhecido como *coisas*) são o hardware físico conectado à internet, detectando o ambiente ao seu redor e coletando e transmitindo dados para um destino downstream. Esses dispositivos podem ser usados em aplicativos de consumo, como câmera de campainha, smartwatch ou termostato. O dispositivo pode ser uma câmera com inteligência artificial que monitora uma linha de montagem em busca de componentes defeituosos, um rastreador GPS para

registre as localizações dos veículos ou um Raspberry Pi programado para baixar os tweets mais recentes e preparar seu café. Qualquer dispositivo capaz de coletar dados de seu ambiente é um dispositivo IoT.

Os dispositivos devem ser minimamente capazes de coletar e transmitir dados. No entanto, o dispositivo também pode processar dados ou executar ML nos dados coletados antes de enviá-los para o downstream — edge computing e edge machine learning, respectivamente.

Um engenheiro de dados não precisa necessariamente conhecer os detalhes internos dos dispositivos IoT, mas deve saber o que o dispositivo faz, os dados que ele coleta, quaisquer cálculos de borda ou ML executados antes de transmitir os dados e com que frequência os envia. Também ajuda a conhecer as consequências de um dispositivo ou interrupção da Internet, fatores ambientais ou outros fatores externos que afetam a coleta de dados e como eles podem afetar a coleta de dados a jusante do dispositivo.

Interface com dispositivos

Um dispositivo não é benéfico a menos que você possa obter seus dados. Esta seção aborda alguns dos principais componentes necessários para interagir com dispositivos IoT em estado selvagem.

gateway IoT. Um *gateway IoT* é um hub para conectar dispositivos e rotear dispositivos com segurança para os destinos apropriados na Internet. Embora você possa conectar um dispositivo diretamente à Internet sem um gateway IoT, o gateway permite que os dispositivos se conectem usando extremamente pouca energia. Ele atua como uma estação intermediária para retenção de dados e gerencia uma conexão com a Internet para o destino final dos dados.

Novos padrões WiFi de baixo consumo de energia são projetados para tornar os gateways de IoT menos críticos no futuro, mas eles estão sendo lançados agora. Normalmente, um enxame de dispositivos utilizará muitos gateways IoT, um em cada local físico onde os dispositivos estão presentes ([Figura 3-16](#)).

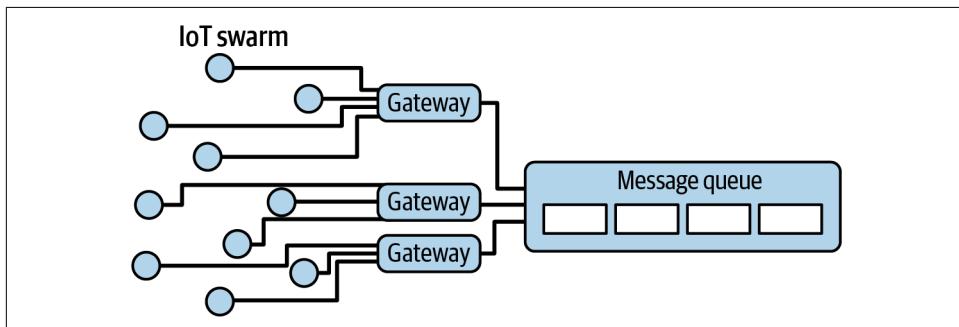


Figura 3-16. Um enxame de dispositivos (círculos), gateways IoT e fila de mensagens com mensagens (retângulos dentro da fila)

Ingestão. Ingestão começa com um gateway IoT, conforme discutido anteriormente. A partir daí, eventos e medições podem fluir para uma arquitetura de ingestão de eventos.

Claro, outros padrões são possíveis. Por exemplo, o gateway pode acumular dados e carregá-los em lotes para processamento analítico posterior. Em ambientes físicos remotos, os gateways podem não ter conectividade com uma rede na maior parte do tempo. Eles podem fazer upload de todos os dados somente quando estiverem dentro do alcance de uma rede celular ou Wi-Fi. O ponto é que a diversidade de sistemas e ambientes de IoT apresenta complicações – por exemplo, dados atrasados, estrutura de dados e disparidades de esquema, corrupção de dados e interrupção de conexão – que os engenheiros devem levar em conta em suas arquiteturas e análises downstream.

Armazenar. Os requisitos de armazenamento dependerão muito do requisito de latência para os dispositivos IoT no sistema. Por exemplo, para sensores remotos que coletam dados científicos para análise posterior, o armazenamento de objetos em lote pode ser perfeitamente aceitável. No entanto, respostas quase em tempo real podem ser esperadas de um back-end do sistema que analisa constantemente os dados em uma solução de monitoramento e automação residencial. Nesse caso, uma fila de mensagens ou banco de dados de série temporal é mais apropriado. Discutimos os sistemas de armazenamento com mais detalhes em [Capítulo 6](#).

Servindo. Os padrões de atendimento são incrivelmente diversos. Em um aplicativo científico em lote, os dados podem ser analisados usando um data warehouse na nuvem e, em seguida, exibidos em um relatório. Os dados serão apresentados e servidos de várias maneiras em um aplicativo de monitoramento doméstico. Os dados serão analisados no tempo próximo usando um mecanismo de processamento de fluxo ou consultas em um banco de dados de séries temporais para procurar eventos críticos, como incêndio, queda de energia ou arrombamento. A detecção de uma anomalia acionará alertas para o proprietário, corpo de bombeiros ou outra entidade. Também existe um componente de análise de lote - por exemplo, um relatório mensal sobre o estado da casa.

Um padrão de serviço significativo para IoT se parece com ETL reverso ([Figura 3-17](#)), embora tendemos a não usar esse termo no contexto da IoT. Pense neste cenário: dados de sensores em dispositivos de fabricação são coletados e analisados. Os resultados dessas medições são processados para buscar otimizações que permitirão que o equipamento opere com mais eficiência. Os dados são enviados de volta para reconfigurar os dispositivos e otimizá-los.

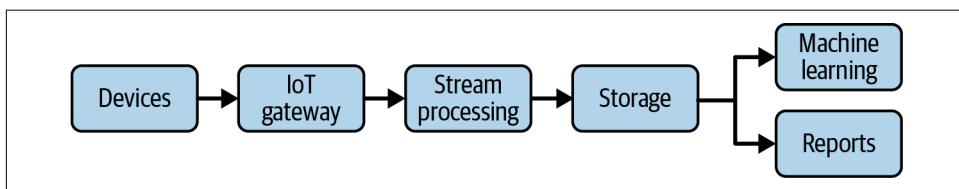


Figura 3-17. Padrão de serviço de IoT para casos de uso downstream

Arranhando a superfície da IoT

Os cenários de IoT são incrivelmente complexos, e a arquitetura e os sistemas de IoT também são menos familiares aos engenheiros de dados que podem ter passado suas carreiras trabalhando com dados de negócios. Esperamos que esta introdução encoraje os engenheiros de dados interessados a aprender mais sobre esta especialização fascinante e em rápida evolução.

Malha de dados

O malha de dados é uma resposta recente às plataformas de dados monolíticos em expansão, como data lakes centralizados e data warehouses, e “a grande divisão de dados”, em que a paisagem é dividida entre dados operacionais e dados analíticos.²⁴ A malha de dados tenta inverter os desafios da arquitetura de dados centralizada, pegando os conceitos de design orientado por domínio (comumente usado em arquiteturas de software) e aplicando-os à arquitetura de dados. Como a malha de dados atraiu muita atenção recentemente, você deve estar ciente disso.

Uma grande parte da malha de dados é a descentralização, como Zhamak Dehghani observou em seu artigo inovador sobre o assunto:²⁵

Para descentralizar a plataforma de dados monolítica, precisamos inverter a forma como pensamos sobre os dados, sua localidade e propriedade. Em vez de fluir os dados dos domínios para um data lake ou plataforma de propriedade central, os domínios precisam hospedar e servir seus conjuntos de dados de domínio de maneira facilmente consumível.

Mais tarde, Dehghani identificou quatro componentes principais da malha de dados:²⁶

- Propriedade e arquitetura de dados descentralizados orientados para o domínio
- Dados como um produto
- Infraestrutura de dados de autoatendimento como uma plataforma
- Governança computacional federada

Figura 3-18 mostra uma versão simplificada de uma arquitetura de malha de dados. Você pode aprender mais sobre malha de dados no livro de Dehghani *Malha de dados* (O'Reilly).

24 Zhamak Dehghani, “Princípios de malha de dados e arquitetura lógica,” MartinFowler.com, 3 de dezembro de 2020, <https://oreil.ly/ezWE7>.

25 Zhamak Dehghani, “How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh,” MartinFowler.com, 20 de maio de 2019, <https://oreil.ly/SqMe8>.

26 Zhamak Dehghani, “Princípios de malha de dados e arquitetura lógica”.

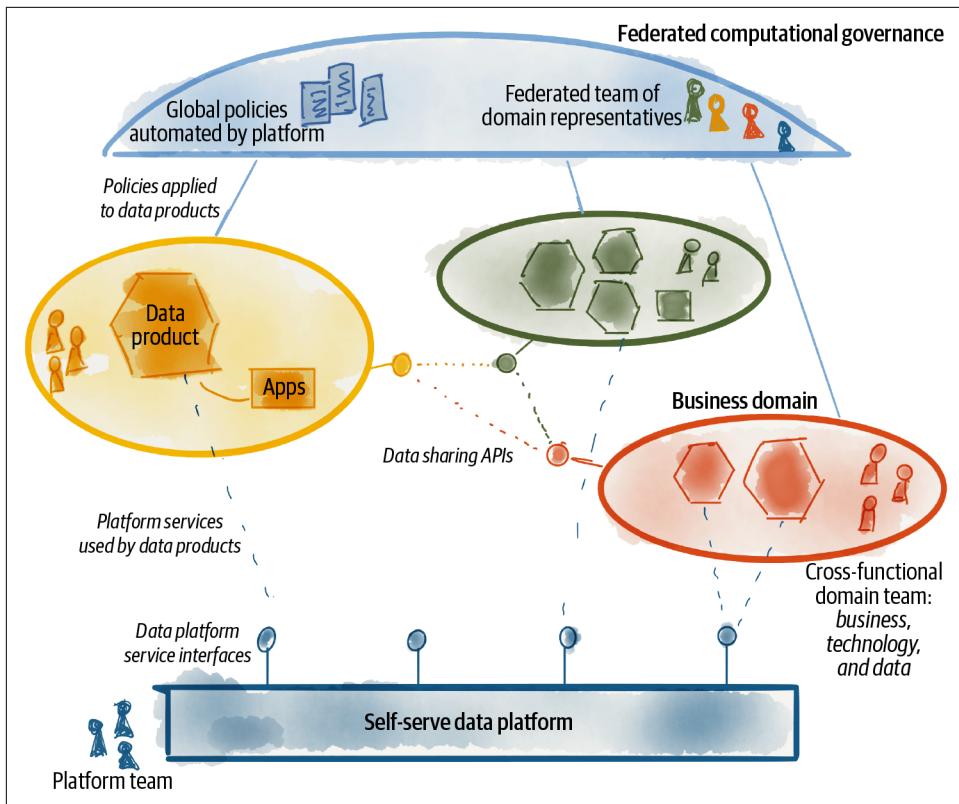


Figura 3-18. Exemplo simplificado de uma arquitetura de malha de dados. Retirado de Malha de dados, por Zhamak Dehghani. Direitos autorais © 2022 Zhamak Dehghani. Publicado por O'Reilly Media, Inc. Usado com permissão.

Outros exemplos de arquitetura de dados

As arquiteturas de dados têm inúmeras outras variações, como data fabric, data hub, [arquitetura em escala](#), [arquitetura de metadados primeiro](#), arquitetura orientada a eventos, pilha de dados ao vivo ([Capítulo 11](#)), e muitos mais. E novas arquiteturas continuarão a surgir à medida que as práticas se consolidam e amadurecem, e as ferramentas são simplificadas e aprimoradas. Nós nos concentraremos em alguns dos padrões de arquitetura de dados mais críticos que são extremamente bem estabelecidos, evoluindo rapidamente ou ambos.

Como engenheiro de dados, preste atenção em como as novas arquiteturas podem ajudar sua organização. Fique a par dos novos desenvolvimentos cultivando uma consciência de alto nível dos desenvolvimentos do ecossistema de engenharia de dados. Tenha a mente aberta e não se apegue emocionalmente a uma abordagem. Depois de identificar o valor potencial, aprofunde seu aprendizado e tome decisões concretas. Quando bem feito, pequenos ajustes – ou grandes revisões – em sua arquitetura de dados podem impactar positivamente os negócios.

Quem está envolvido no projeto de uma arquitetura de dados?

A arquitetura de dados não é projetada no vácuo. As empresas maiores ainda podem雇用 arquitetos de dados, mas esses arquitetos precisarão estar fortemente sintonizados e atualizados com o estado da tecnologia e dos dados. Já se foram os dias da arquitetura de dados da torre de marfim. No passado, a arquitetura era em grande parte ortogonal à engenharia. Esperamos que essa distinção desapareça à medida que a engenharia de dados e a engenharia em geral evoluam rapidamente, tornando-se mais ágeis, com menos separação entre engenharia e arquitetura.

Idealmente, um engenheiro de dados trabalhará ao lado de um arquiteto de dados dedicado. No entanto, se uma empresa for pequena ou com baixo nível de maturidade de dados, um engenheiro de dados pode trabalhar em dobro como arquiteto. Como a arquitetura de dados é uma subcorrente do ciclo de vida da engenharia de dados, um engenheiro de dados deve entender a arquitetura “boa” e os vários tipos de arquitetura de dados.

Ao projetar a arquitetura, você trabalhará junto com as partes interessadas do negócio para avaliar os trade-offs. Quais são as compensações inerentes à adoção de um data warehouse em nuvem versus um data lake? Quais são as vantagens e desvantagens de várias plataformas de nuvem? Quando uma estrutura unificada de lote/streaming (Beam, Flink) pode ser uma escolha apropriada? Estudar essas escolhas em abstrato irá prepará-lo para tomar decisões concretas e valiosas.

Conclusão

Você aprendeu como a arquitetura de dados se encaixa no ciclo de vida da engenharia de dados e o que torna a arquitetura de dados “boa”, e viu vários exemplos de arquiteturas de dados. Como a arquitetura é um alicerce fundamental para o sucesso, incentivamos você a investir tempo para estudá-la profundamente e entender as vantagens e desvantagens inerentes a qualquer arquitetura. Você estará preparado para mapear a arquitetura que corresponde aos requisitos exclusivos de sua organização.

A seguir, veremos algumas abordagens para escolher as tecnologias certas a serem usadas na arquitetura de dados e em todo o ciclo de vida da engenharia de dados.

Recursos adicionais

- “[AnemicDomainModel](#)” por Martin Fowler
- “[Arquiteturas de Big Data](#)” Documentação do Azure
- “[Contexto Delimitado](#)” por Martin Fowler
- “[Uma Breve Introdução a Duas Arquiteturas de Processamento de Dados—Lambda e Kappa para Big Data](#)” por Iman Samizadeh
- “[Os blocos de construção de uma plataforma de dados moderna](#)” por Prukalpa
- “[Escolher Abrir com Sabedoria](#)” por Benoit Dageville et al.

- “Escolhendo a arquitetura certa para distribuição global de dados” Página da Web da arquitetura do Google Cloud
- Página da Wikipédia “DBMS Orientado a Colunas”
- “Uma comparação de estruturas de processamento de dados” por Ludovik Santos
- “O custo da nuvem, um paradoxo de trilhões de dólares” por Sarah Wang e Martin Casado
- “A Maldição do Monstro do Data Lake” por Kiran Prakash e Lucy Chambers
- *Arquitetura de dados: uma cartilha para o cientista de dados* por WH Inmon et al. (Imprensa Acadêmica)
- “Arquitetura de Dados: Complexa x Complicada” por Dave Wells
- “Dados como um produto x Dados como um serviço” por Justin Gage
- “A dicotomia dos dados: repensando a maneira como tratamos dados e serviços” por Ben Stopford
- “A arquitetura do Data Fabric é a chave para modernizar o gerenciamento e a integração de dados” por Ashutosh Gupta
- “Definição da malha de dados” por James Serra
- “Plataforma de equipe de dados” por dados do GitLab
- “Arquitetura de Data Warehouse: Visão Geral” por Roelant Vos
- Tutorial “Data Warehouse Architecture” no Javatpoint
- Página da web “Definindo Arquitetura” ISO/IEC/IEEE 42010
- “O Projeto e Implementação de Sistemas de Banco de Dados Orientados a Colunas Modernos” por Daniel Abadi et al.
- “Desastres que vi em um mundo de microsserviços” por João Alves
- “DomainDrivenDesign” por Martin Fowler
- “Abaixo o endividamento: apresentando grandes expectativas” pelo projeto Grandes Esperanças
- *EABOKfrascunho*, editado por Paula Hagan
- Site da EABOK
- “EagerReadDerivation” por Martin Fowler
- “Orquestração de ETL sem servidor de ponta a ponta na AWS: um guia” por Rittika Jindal
- Definição do Glossário de “Arquitetura Corporativa” do Gartner
- “O Papel da Arquitetura Corporativa na Construção de uma Organização Orientada a Dados” por Ashutosh Gupta
- “Fornecimento de eventos” por Martin Fowler

- “Apaixonando-se novamente por pipelines de dados”por Sean Knapp
- “Cinco princípios para a arquitetura nativa da nuvem: o que é e como dominá-la” por Tom Gray
- “Foco em Eventos”por Martin Fowler
- “Engenharia de dados funcionais: um paradigma moderno para processamento de dados em lote” por Maxime Beauchemin
- “Estrutura de arquitetura do Google Cloud” Página da Web de arquitetura do Google Cloud
- “Como Vencer o Teorema do Cap”por Nathan Marz
- “Como construir uma arquitetura de dados para impulsionar a inovação — hoje e amanhã” por Antonio Castro e cols.
- “Como o TOGAF define a arquitetura corporativa (EA)”por Avancier Limited
- O site do Corpo de Conhecimento de Gerenciamento de Informações
- “Apresentando o Dagster: uma biblioteca Python de código aberto para criar aplicativos de dados”por Nick Schrock
- “O registro: o que todo engenheiro de software deve saber sobre a abstração unificada de dados em tempo real”por Jay Kreps
- Documentação “Arquitetura de Referência IoT do Microsoft Azure”
- da Microsoft“Centro de Arquitetura Azul”
 - “A CI moderna é muito complexa e mal direcionada”por Gregory Szorc
 - “A Pilha de Dados Moderna: Passado, Presente e Futuro”por Tristan Handy
 - “Indo além do lote versus streaming”por David Yaffe
- “Uma implementação pessoal da arquitetura de dados moderna: colocando dados do Strava no Google Cloud Platform”por Matthew Reeve
- “Persistência Poliglota”por Martin Fowler
- “Ciência de Dados Potemkin”por Michael Correll
- “Princípios de Engenharia de Dados, Parte I: Visão Geral da Arquitetura”por Hussein Danish
- “Questionando a Arquitetura Lambda”por Jay Kreps
- “Troca confiável de dados de microsserviços com o padrão Outbox”por Gunnar Morling
- “ReportingDatabase”por Martin Fowler
- “A ascensão do lago de metadados”por Prukalpa
- “Gerencie sua equipe de dados como uma equipe de produto”por Emilie Schario e Taylor A. Murphy

- “Separando a Utilidade do Valor Agregado”por Ross Pettit
- “Os seis princípios da arquitetura de dados moderna”por Joshua Klahr
- floco de nevePágina da Web "O que é arquitetura de data warehouse"
- “Infraestrutura de software 2.0: uma lista de desejos”por Erik Bernhardsson
- “Ficar à frente da dívida de dados”por Eta Mizrahi
- “Táticas vs. Estratégia: SOA e o Tarpit da Irrelevância”por Neal Ford
- “Teste a qualidade dos dados em escala com Deequ”por Dustin Lange e outros.
- “Arquitetura de três camadas”por IBM Educação
- site da estrutura TOGAF
- “As 5 principais tendências de dados para CDOs a serem observadas em 2021”por Prukalpa
- “240 Tabelas e Nenhuma Documentação?”por Alexey Makhotkin
- “A Lista de Verificação Definitiva de Observabilidade de Dados”por Molly Vorwerck
- “Análise unificada: onde o lote e o streaming se unem; SQL e além”
Roteiro do Apache Flink
- “Utilidade Vs Dicotomia Estratégica”por Martin Fowler
- “O que é um Data Lakehouse?”por Ben Lorica et al.
- “O que é arquitetura de dados? Uma estrutura para gerenciar dados”por Thor Olavsrud
- “O que é o ecossistema de dados abertos e por que veio para ficar”por Casber Wang
- “O que há de errado com MLOps?”por Laszlo Sragner
- “O que diabos é malha de dados”por Chris Riccomini
- “Quem Precisa de um Arquiteto”por Martin Fowler
- Página da Wikipédia “Zachman Framework”

Escolhendo tecnologias entre os dados

Ciclo de vida de engenharia

A engenharia de dados hoje em dia sofre de um embaraço de riquezas. Não faltam tecnologias para resolver diversos tipos de problemas de dados. As tecnologias de dados estão disponíveis como ofertas prontas para uso em quase todos os aspectos - código aberto, código aberto gerenciado, software proprietário, serviço proprietário e muito mais. No entanto, é fácil ser pego em busca de tecnologia de ponta enquanto perde de vista o objetivo principal da engenharia de dados: projetar sistemas robustos e confiáveis para transportar dados por todo o ciclo de vida e atendê-los de acordo com as necessidades dos usuários finais. Assim como os engenheiros estruturais escolhem cuidadosamente tecnologias e materiais para realizar a visão de um arquiteto para um edifício, os engenheiros de dados têm a tarefa de fazer escolhas tecnológicas apropriadas para conduzir os dados ao longo do ciclo de vida para atender aos aplicativos e usuários de dados.

Capítulo 3 discutiu a "boa" arquitetura de dados e por que ela é importante. Agora explicamos como escolher as tecnologias certas para atender a essa arquitetura. Os engenheiros de dados devem escolher boas tecnologias para criar o melhor produto de dados possível. Acreditamos que o critério para escolher uma boa tecnologia de dados é simples: ela agrupa valor a um produto de dados e ao negócio em geral?

Muita gente confunde arquitetura e ferramentas. Arquitetura é *estratégico*; ferramentas são *tático*. Às vezes ouvimos: "Nossa arquitetura de dados são ferramentas X, Y, e Z." Esta é a maneira errada de pensar sobre arquitetura. Arquitetura é o design de alto nível, roteiro e projeto de sistemas de dados que satisfazem os objetivos estratégicos do negócio. Arquitetura é *o que, por que, quando*. As ferramentas são usadas para tornar a arquitetura uma realidade; as ferramentas são *as como*.

Muitas vezes vemos equipes "saindo dos trilhos" e escolhendo tecnologias antes de mapear uma arquitetura. As razões variam: síndrome do objeto brilhante,

desenvolvimento e falta de experiência em arquitetura. Na prática, essa priorização da tecnologia geralmente significa que eles montam uma espécie de máquina de fantasia do Dr. Seuss, em vez de uma verdadeira arquitetura de dados. Aconselhamos fortemente a não escolher a tecnologia antes de acertar sua arquitetura. Arquitetura em primeiro lugar, tecnologia em segundo.

Este capítulo discute nosso plano tático para fazer escolhas tecnológicas assim que tivermos um projeto de arquitetura estratégica. A seguir estão algumas considerações para escolher tecnologias de dados em todo o ciclo de vida da engenharia de dados:

- Tamanho e capacidades da equipe
- Velocidade para o mercado
- Interoperabilidade
- Otimização de custos e valor comercial
- Hoje versus futuro: tecnologias imutáveis versus transitórias
- Localização (nuvem, no local, nuvem híbrida, multicloud)
- Construir versus comprar
- Monólito versus modular
- Sem servidor versus servidores
- Otimização, desempenho e as guerras de benchmark
- As tendências do ciclo de vida da engenharia de dados

Tamanho e capacidades da equipe

A primeira coisa que você precisa avaliar é o tamanho de sua equipe e suas capacidades com tecnologia. Você está em uma equipe pequena (talvez uma equipe de um) de pessoas que devem desempenhar muitos papéis, ou a equipe é grande o suficiente para que as pessoas trabalhem em funções especializadas? Um punhado de pessoas será responsável por vários estágios do ciclo de vida da engenharia de dados ou as pessoas cobrem nichos específicos? O tamanho da sua equipe influenciará os tipos de tecnologias que você adotar.

Há um continuum de tecnologias simples a complexas, e o tamanho de uma equipe determina aproximadamente a quantidade de largura de banda que sua equipe pode dedicar a soluções complexas. Às vezes, vemos pequenas equipes de dados lendo postagens de blog sobre uma nova tecnologia de ponta em uma empresa gigante de tecnologia e, em seguida, tentando emular essas mesmas tecnologias e práticas extremamente complexas. Nós chamamos isso de *engenharia de culto à carga*, e geralmente é um grande erro que consome muito tempo e dinheiro valiosos, geralmente com pouco ou nada para mostrar em troca. Especialmente para equipes pequenas ou com habilidades técnicas mais fracas, use o máximo possível de ferramentas gerenciadas e SaaS e dedique sua largura de banda limitada para resolver os problemas complexos que agregam valor diretamente aos negócios.

Faça um inventário das habilidades de sua equipe. As pessoas se inclinam para ferramentas de baixo código ou preferem abordagens que priorizam o código? As pessoas são fortes em certas linguagens como Java, Python ou Go? As tecnologias estão disponíveis para atender a todas as preferências no espectro de código baixo a código pesado. Mais uma vez, sugerimos usar tecnologias e fluxos de trabalho com os quais a equipe esteja familiarizada. Vimos equipes de dados investirem muito tempo aprendendo a nova e brilhante tecnologia, linguagem ou ferramenta de dados, apenas para nunca usá-la na produção. Aprender novas tecnologias, idiomas e ferramentas é um investimento de tempo considerável, portanto, faça esses investimentos com sabedoria.

Velocidade para o mercado

Em tecnologia, a velocidade para o mercado vence. Isso significa escolher as tecnologias certas que ajudam você a fornecer recursos e dados mais rapidamente, mantendo padrões de alta qualidade e segurança. Também significa trabalhar em um ciclo de feedback estreito de lançamento, aprendizado, iteração e melhorias.

O perfeito é inimigo do bom. Algumas equipes de dados irão deliberar sobre as escolhas de tecnologia por meses ou anos sem chegar a nenhuma decisão. Decisões e resultados lentos são o beijo da morte para as equipes de dados. Vimos mais do que algumas equipes de dados se dissolverem por se moverem muito devagar e não conseguirem entregar o valor para o qual foram contratadas.

Entregue valor com antecedência e com frequência. Como mencionamos, use o que funciona. Os membros de sua equipe provavelmente obterão melhor aproveitamento com as ferramentas que já conhecem. Evite o trabalho pesado indiferenciado que envolve sua equipe em trabalhos desnecessariamente complexos que agregam pouco ou nenhum valor. Escolha ferramentas que ajudem você a se mover de forma rápida, confiável e segura.

Interoperabilidade

Raramente você usará apenas uma tecnologia ou sistema. Ao escolher uma tecnologia ou sistema, você precisará garantir que ele interaja e opere com outras tecnologias.

Interoperabilidadedescreve como várias tecnologias ou sistemas se conectam, trocam informações e interagem.

Digamos que você esteja avaliando duas tecnologias, A e B. Com que facilidade a tecnologia A se integra à tecnologia B quando se pensa em interoperabilidade? Isso geralmente é um espectro de dificuldade, variando de contínuo a demorado. A integração perfeita já está incorporada em cada produto, facilitando a configuração? Ou você precisa fazer muitas configurações manuais para integrar essas tecnologias?

Freqüentemente, fornecedores e projetos de código aberto têm como alvo plataformas e sistemas específicos para interoperar. A maioria das ferramentas de visualização e ingestão de dados possui integrações integradas com data warehouses e data lakes populares. Além disso, a popular ingestão de dados

as ferramentas se integrarão a APIs e serviços comuns, como CRMs, software de contabilidade e muito mais.

Às vezes, existem padrões para interoperabilidade. Quase todos os bancos de dados permitem conexões via Java Database Connectivity (JDBC) ou Open Database Connectivity (ODBC), o que significa que você pode se conectar facilmente a um banco de dados usando esses padrões. Em outros casos, a interoperabilidade ocorre na ausência de padrões. A transferência de estado representacional (REST) não é realmente um padrão para APIs; toda API REST tem suas peculiaridades. Nesses casos, cabe ao fornecedor ou projeto de software de código aberto (OSS) garantir uma integração suave com outras tecnologias e sistemas.

Esteja sempre ciente de como será simples conectar suas várias tecnologias ao longo do ciclo de vida da engenharia de dados. Conforme mencionado em outros capítulos, sugerimos projetar para a modularidade e dar a si mesmo a capacidade de trocar facilmente as tecnologias à medida que novas práticas e alternativas se tornam disponíveis.

Otimização de custos e valor comercial

Em um mundo perfeito, você experimentaria todas as tecnologias mais recentes e interessantes sem considerar custo, investimento de tempo ou valor agregado ao negócio. Na realidade, os orçamentos e o tempo são finitos, e o custo é uma grande restrição para a escolha das arquiteturas e tecnologias de dados corretas. Sua organização espera um ROI positivo de seus projetos de dados, portanto, você deve entender os custos básicos que pode controlar. A tecnologia é um importante direcionador de custos, portanto, suas escolhas de tecnologia e estratégias de gerenciamento afetarão significativamente seu orçamento. Analisamos os custos por meio de três lentes principais: custo total de propriedade, custo de oportunidade e FinOps.

Custo Total de Propriedade

Custo total de propriedade(TCO) é o custo total estimado de uma iniciativa, incluindo os custos diretos e indiretos dos produtos e serviços utilizados. *Custos diretos* podem ser diretamente atribuídos a uma iniciativa. Exemplos são os salários de uma equipe que trabalha na iniciativa ou a conta da AWS por todos os serviços consumidos. *Custos indiretos*, também conhecido como *a sobrecarga*, são independentes da iniciativa e devem ser pagos independentemente do local de atribuição.

Além dos custos diretos e indiretos, *como algo é comprado* impacta a forma como os custos são contabilizados. As despesas se dividem em dois grandes grupos: despesas de capital e despesas operacionais.

Despesas de capital, também conhecido como *Capex*, exigem um investimento inicial. O pagamento é obrigatório *hoje*. Antes da nuvem existir, as empresas normalmente compravam hardware e software antecipadamente por meio de grandes contratos de aquisição. Além disso, foram necessários investimentos significativos para hospedar hardware em salas de servidores, data centers e instalações de colocation. Esses investimentos iniciais - geralmente centenas de milhares

a milhões de dólares ou mais - seriam tratados como ativos e depreciariam lentamente ao longo do tempo. Do ponto de vista orçamentário, o capital era necessário para financiar toda a compra. Isso é capex, um desembolso de capital significativo com um plano de longo prazo para obter um ROI positivo sobre o esforço e as despesas realizadas.

Despesas operacionais, também conhecido como opex, são o oposto do capex em certos aspectos. Opex é gradual e espalhado ao longo do tempo. Enquanto o capex é focado no longo prazo, o opex é de curto prazo. Opex pode ser pré-pago ou similar e permite muita flexibilidade. Opex está mais próximo de um custo direto, facilitando a atribuição a um projeto de dados.

Até recentemente, opex não era uma opção para grandes projetos de dados. Os sistemas de dados geralmente exigiam contratos multimilionários. Isso mudou com o advento da nuvem, pois os serviços de plataforma de dados permitem que os engenheiros paguem em um modelo baseado no consumo. Em geral, o opex permite uma capacidade muito maior para as equipes de engenharia escolherem seu software e hardware. Os serviços baseados em nuvem permitem que os engenheiros de dados interajam rapidamente com várias configurações de software e tecnologia, muitas vezes de forma econômica.

Os engenheiros de dados precisam ser pragmáticos em relação à flexibilidade. O cenário de dados está mudando muito rapidamente para investir em hardware de longo prazo que inevitavelmente fica obsoleto, não pode ser dimensionado com facilidade e pode dificultar a flexibilidade de um engenheiro de dados para tentar coisas novas. Dada a vantagem da flexibilidade e dos baixos custos iniciais, instamos os engenheiros de dados a adotar uma abordagem de primeiro plano centrada na nuvem e em tecnologias flexíveis de pagamento conforme o uso.

Custo total de oportunidade de propriedade

Qualquer escolha exclui inherentemente outras possibilidades. *Custo total de oportunidade de propriedade* (TOCO) é o custo das oportunidades perdidas que incorremos na escolha de uma tecnologia, arquitetura ou processo.¹ Observe que a propriedade nesta configuração não requer compras de longo prazo de hardware ou licenças. Mesmo em um ambiente de nuvem, possuímos efetivamente uma tecnologia, uma pilha ou um pipeline, uma vez que se torna uma parte essencial de nossos processos de dados de produção e é difícil abandoná-lo. Os engenheiros de dados geralmente não avaliam o TOCO ao realizar um novo projeto; em nossa opinião, este é um grande ponto cego.

Se você escolher a pilha de dados A, terá escolhido os benefícios da pilha de dados A sobre todas as outras opções, excluindo efetivamente as pilhas de dados B, C e D. Você está comprometido com a pilha de dados A e tudo o que ela envolve - a equipe para dar suporte ele, treinamento, configuração e manutenção. O que acontece se a pilha de dados A for uma escolha ruim? O que acontece quando a pilha de dados A se torna obsoleta? Você ainda pode mover para outras pilhas de dados?

Com que rapidez e economia você pode mudar para algo mais novo e melhor? Esta é uma questão crítica no espaço de dados, onde novas tecnologias e produtos parecem

1 Para mais detalhes, consulte "Total Opportunity Cost of Ownership" de Joseph Reis em *97 coisas que todo engenheiro de dados Deveria saber*(O'Reilly).

aparecem em um ritmo cada vez mais rápido. O conhecimento que você construiu na pilha de dados A se traduz na próxima onda? Ou você pode trocar os componentes da pilha de dados A e ganhar algum tempo e opções?

O primeiro passo para minimizar o custo de oportunidade é avaliá-lo com os olhos bem abertos. Vimos inúmeras equipes de dados ficarem presas a tecnologias que pareciam boas na época e não são flexíveis para crescimento futuro ou simplesmente obsoletas. Tecnologias de dados inflexíveis são muito parecidas com armadilhas para ursos. Eles são fáceis de entrar e extremamente dolorosos de escapar.

FinOps

Já falamos sobre FinOps em “[Princípio 9: Adote FinOps](#)” na página 85. Como discutimos, os gastos típicos com a nuvem são inherentemente opex: as empresas pagam por serviços para executar processos de dados críticos, em vez de fazer compras antecipadas e recuperar o valor ao longo do tempo. O objetivo do FinOps é operacionalizar totalmente a responsabilidade financeira e o valor comercial, aplicando as práticas do tipo DevOps de monitoramento e ajuste dinâmico de sistemas.

Neste capítulo, queremos enfatizar uma coisa sobre FinOps que está bem incorporada nesta citação:²

Se parece que o FinOps é para economizar dinheiro, pense novamente. FinOps é sobre ganhar dinheiro. Os gastos com a nuvem podem gerar mais receita, sinalizar o crescimento da base de clientes, permitir maior velocidade de lançamento de produtos e recursos ou até mesmo ajudar a fechar um data center.

Em nossa configuração de engenharia de dados, a capacidade de iterar rapidamente e escalar dinamicamente é inestimável para criar valor comercial. Essa é uma das principais motivações para transferir as cargas de trabalho de dados para a nuvem.

Hoje versus futuro: tecnologias imutáveis versus transitórias

Em um domínio empolgante como a engenharia de dados, é muito fácil focar em um futuro em rápida evolução, ignorando as necessidades concretas do presente. A intenção de construir um futuro melhor é nobre, mas muitas vezes leva à superarquitetura e à superengenharia. As ferramentas escolhidas para o futuro podem estar obsoletas e desatualizadas quando esse futuro chegar; o futuro freqüentemente se parece pouco com o que imaginamos anos antes.

Como muitos coaches de vida diriam a você, concentre-se no presente. Você deve escolher a melhor tecnologia para o momento e para o futuro próximo, mas de uma forma que apoie as incógnitas futuras e a evolução. Pergunte a si mesmo: onde você está hoje e quais são seus objetivos

² JR Storment e Mike Fuller, *Nuvem FinOps* (Sebastopol, CA: O'Reilly, 2019), 6, <https://oreil.ly/RvRvX>.

para o futuro? Suas respostas a essas perguntas devem informar suas decisões sobre sua arquitetura e, portanto, as tecnologias usadas nessa arquitetura. Isso é feito compreendendo o que provavelmente mudará e o que tende a permanecer o mesmo.

Temos duas classes de ferramentas a considerar: imutáveis e transitórias. *Tecnologias imutáveis* podem ser componentes que sustentam a nuvem ou linguagens e paradigmas que resistiram ao teste do tempo. Na nuvem, exemplos de tecnologias imutáveis são armazenamento de objetos, redes, servidores e segurança. O armazenamento de objetos, como Amazon S3 e Azure Blob Storage, existirá de hoje até o final da década e provavelmente por muito mais tempo. Armazenar seus dados no armazenamento de objetos é uma escolha sábia. O armazenamento de objetos continua a melhorar de várias maneiras e oferece constantemente novas opções, mas seus dados estarão seguros e utilizáveis no armazenamento de objetos, independentemente da rápida evolução da tecnologia como um todo.

Para idiomas, SQL e bash existem há muitas décadas e não os vemos desaparecendo tão cedo. As tecnologias imutáveis se beneficiam do efeito Lindy: quanto mais tempo uma tecnologia estiver estabelecida, mais tempo ela será usada. Pense na rede elétrica, nos bancos de dados relacionais, na linguagem de programação C ou na arquitetura do processador x86. Sugerimos aplicar o efeito Lindy como um teste decisivo para determinar se uma tecnologia é potencialmente imutável.

tecnologias transitórias são aqueles que vêm e vão. A trajetória típica começa com muito hype, seguido por um crescimento meteórico em popularidade, e então uma lenta queda na obscuridade. O cenário front-end do JavaScript é um exemplo clássico. Quantas estruturas de front-end JavaScript surgiram e desapareceram entre 2010 e 2020? Backbone.js, Ember.js e Knockout eram populares no início dos anos 2010, e React e Vue.js têm grande popularidade hoje. Qual será o popular framework JavaScript de front-end daqui a três anos? Quem sabe.

Novos entrantes bem financiados e projetos de código aberto chegam na frente de dados todos os dias. Todo fornecedor dirá que seu produto mudará a indústria e "Faça do mundo um lugar melhor". A maioria dessas empresas e projetos não obtém tração de longo prazo e desaparece lentamente na obscuridade. Os principais VCs estão fazendo grandes apostas, sabendo que a maioria de seus investimentos em ferramentas de dados falhará. Se os VCs que despejam milhões (ou bilhões) em investimentos em ferramentas de dados não conseguem acertar, como você pode saber em quais tecnologias investir para sua arquitetura de dados? É difícil. Basta considerar o número de tecnologias nas (in)famosas representações de Matt Turck do [Cenário de ML, IA e dados \(MAD\)](#) que introduzimos em [Capítulo 1\(Figura 4-1\)](#).

Mesmo tecnologias relativamente bem-sucedidas muitas vezes desaparecem rapidamente na obscuridade, após alguns anos de rápida adoção, uma vítima de seu sucesso. Por exemplo, no início da década de 2010, o Hive teve uma aceitação rápida porque permitia que analistas e engenheiros consultassem conjuntos de dados massivos sem codificar trabalhos complexos de MapReduce manualmente. Inspirado

pelo sucesso do Hive, mas desejando melhorar suas deficiências, os engenheiros desenvolveram o Presto e outras tecnologias. O Hive agora aparece principalmente em implantações legadas. Quase toda tecnologia segue esse caminho inevitável de declínio.

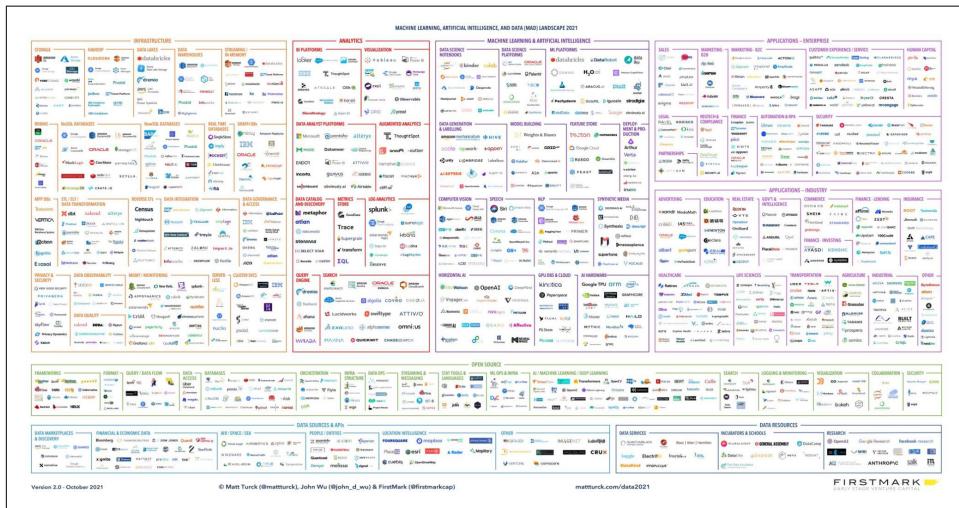


Figura 4-1. 2021 de Matt Turck Cenário de dados MAD

Nosso conselho

Dado o ritmo acelerado das mudanças nas ferramentas e nas melhores práticas, sugerimos avaliar as ferramentas a cada dois anos (Figura 4-2). Sempre que possível, encontre as tecnologias imutáveis ao longo do ciclo de vida da engenharia de dados e use-as como sua base. Construa ferramentas transitórias em torno dos imutáveis.

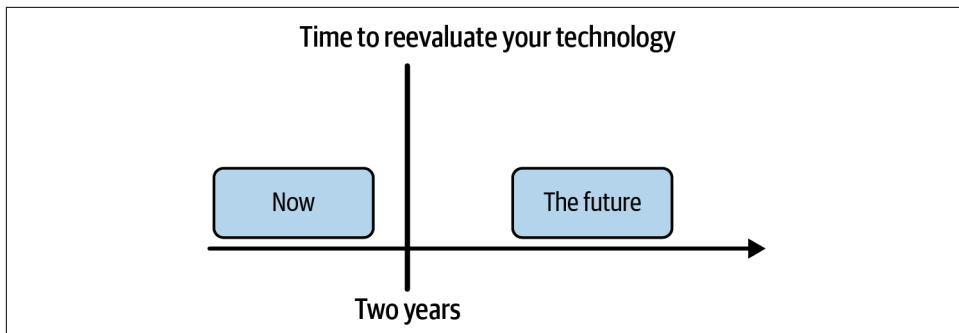


Figura 4-2. Use um horizonte temporal de dois anos para reavaliar suas escolhas tecnológicas

Dada a probabilidade razoável de falha para muitas tecnologias de dados, você precisa considerar como é fácil fazer a transição de uma tecnologia escolhida. Quais são as barreiras para sair? Conforme mencionado anteriormente em nossa discussão sobre custo de oportunidade, evite

“armadilhas para ursos”. Entre em uma nova tecnologia com os olhos bem abertos, sabendo que o projeto pode ser abandonado, a empresa pode não ser viável ou a tecnologia simplesmente não é mais adequada.

Localização

As empresas agora têm várias opções ao decidir onde executar suas pilhas de tecnologia. Uma lenta mudança em direção à nuvem culmina em uma verdadeira debandada de empresas girando cargas de trabalho na AWS, Azure e Google Cloud Platform (GCP). Na última década, muitos CTOs passaram a ver suas decisões sobre hospedagem de tecnologia como tendo um significado existencial para suas organizações. Se eles se movem muito devagar, correm o risco de serem deixados para trás por sua competição mais ágil; por outro lado, uma migração para a nuvem mal planejada pode levar a falhas tecnológicas e custos catastróficos.

Vejamos os principais locais para executar sua pilha de tecnologia: local, nuvem, nuvem híbrida e multicloud.

No local

Enquanto novas startups nascem cada vez mais na nuvem, os sistemas locais ainda são o padrão para empresas estabelecidas. Essencialmente, essas empresas são proprietárias de seu hardware, que pode residir em data centers próprios ou em espaços de colocation alugados. Em ambos os casos, as empresas são operacionalmente responsáveis por seu hardware e pelo software executado nele. Se o hardware falhar, eles terão que repará-lo ou substituí-lo. Eles também precisam gerenciar os ciclos de atualização a cada poucos anos, conforme o hardware novo e atualizado é lançado e o hardware mais antigo envelhece e se torna menos confiável. Eles devem garantir que tenham hardware suficiente para lidar com picos; para um varejista on-line, isso significa hospedar capacidade suficiente para lidar com os picos de carga da Black Friday. Para engenheiros de dados responsáveis por sistemas locais,

Por um lado, as empresas estabelecidas estabeleceram práticas operacionais que as serviram bem. Suponha que uma empresa que depende da tecnologia da informação esteja no mercado há algum tempo. Isso significa que ela conseguiu conciliar os requisitos de custo e pessoal para executar seu hardware, gerenciar ambientes de software, implantar código de equipes de desenvolvimento e executar bancos de dados e sistemas de big data.

Por outro lado, as empresas estabelecidas veem sua concorrência mais jovem e ágil escalando rapidamente e aproveitando os serviços gerenciados em nuvem. Eles também veem concorrentes estabelecidos fazendo incursões na nuvem, permitindo-lhes aumentar temporariamente o enorme poder de computação para trabalhos de dados massivos ou o pico de compras da Black Friday.

As empresas de setores competitivos geralmente não têm a opção de ficar paradas. A concorrência é acirrada e sempre há a ameaça de ser “interrompido” por uma concorrência mais ágil, muitas vezes apoiada por uma grande pilha de dólares de capital de risco. Toda empresa deve manter seus sistemas existentes funcionando de forma eficiente enquanto decide quais movimentos fazer a seguir. Isso pode envolver a adoção de práticas de DevOps mais recentes, como contêineres, Kubernetes, microsserviços e implantação contínua, mantendo o hardware em execução no local. Isso pode envolver uma migração completa para a nuvem, conforme discutido a seguir.

Nuvem

A nuvem vira o modelo local de cabeça para baixo. Em vez de comprar hardware, você simplesmente aluga hardware e serviços gerenciados de um provedor de nuvem (como AWS, Azure ou Google Cloud). Esses recursos muitas vezes podem ser reservados em uma base de prazo extremamente curto; As VMs são ativadas em menos de um minuto e o uso subsequente é cobrado em incrementos por segundo. Isso permite que os usuários da nuvem dimensionem dinamicamente recursos que seriam inconcebíveis com servidores locais.

Em um ambiente de nuvem, os engenheiros podem lançar projetos e experimentar rapidamente sem se preocupar com planejamento de hardware de longo prazo. Eles podem começar a executar servidores assim que seu código estiver pronto para implantação. Isso torna o modelo de nuvem extremamente atraente para startups com orçamento e tempo apertados.

A era inicial da nuvem foi dominada pela oferta de produtos de infraestrutura como serviço (IaaS), como VMs e discos virtuais, que são essencialmente pedaços de hardware alugados. Lentamente, vimos uma mudança em direção à plataforma como serviço (PaaS), enquanto os produtos SaaS continuam a crescer rapidamente.

PaaS inclui produtos IaaS, mas adiciona serviços gerenciados mais sofisticados para dar suporte a aplicativos. Exemplos são bancos de dados gerenciados, como Amazon Relational Database Service (RDS) e Google Cloud SQL, plataformas de streaming gerenciadas, como Amazon Kinesis e Simple Queue Service (SQS), e Kubernetes gerenciados, como Google Kubernetes Engine (GKE) e Azure Serviço Kubernetes (AKS). Os serviços de PaaS permitem que os engenheiros ignorem os detalhes operacionais do gerenciamento de máquinas individuais e da implantação de estruturas em sistemas distribuídos. Eles fornecem acesso pronto para uso a sistemas complexos de escalonamento automático com sobrecarga operacional mínima.

As ofertas de SaaS avançam um degrau adicional na escada da abstração. O SaaS normalmente fornece uma plataforma de software empresarial totalmente funcional com pouco gerenciamento operacional. Exemplos de SaaS incluem Salesforce, Google Workspace, Microsoft 365, Zoom e Fivetrans. Tanto as principais nuvens públicas quanto terceiros oferecem plataformas SaaS. O SaaS abrange todo um espectro de domínios corporativos, incluindo videoconferência, gerenciamento de dados, tecnologia de anúncios, aplicativos de escritório e sistemas de CRM.

Este capítulo também discute serverless, cada vez mais importante em ofertas de PaaS e SaaS. Os produtos sem servidor geralmente oferecem escalonamento automatizado de zero a taxas de uso extremamente altas. Eles são cobrados com base no pagamento conforme o uso e permitem que os engenheiros operem sem conhecimento operacional dos servidores subjacentes. Muitas pessoas discordam do termo *sem servidor*, afinal, o código deve ser executado em algum lugar. Na prática, sem servidor geralmente significa *muitos servidores invisíveis*.

Os serviços em nuvem tornaram-se cada vez mais atraentes para empresas estabelecidas com data centers e infraestrutura de TI existentes. O dimensionamento dinâmico e contínuo é extremamente valioso para empresas que lidam com a sazonalidade (por exemplo, empresas de varejo lidando com a carga da Black Friday) e picos de carga de tráfego da web. O advento do COVID-19 em 2020 foi um dos principais impulsionadores da adoção da nuvem, pois as empresas reconheceram o valor de aumentar rapidamente os processos de dados para obter insights em um clima de negócios altamente incerto; as empresas também tiveram que lidar com um aumento substancial da carga devido a um aumento nas compras on-line, no uso de aplicativos da web e no trabalho remoto.

Antes de discutirmos as nuances da escolha de tecnologias na nuvem, vamos primeiro discutir por que a migração para a nuvem requer uma mudança dramática de pensamento, especificamente na frente de preço; isso está intimamente relacionado ao FinOps, introduzido em “[FinOps” na página 120](#). As empresas que migram para a nuvem geralmente cometem grandes erros de implantação por não adaptarem adequadamente suas práticas ao modelo de preços da nuvem.

Um breve desvio na economia da nuvem

Para entender como usar os serviços em nuvem de forma eficiente por meio de [arquitetura nativa da nuvem](#), você precisa saber como as nuvens ganham dinheiro. Este é um conceito extremamente complexo e sobre o qual os provedores de nuvem oferecem pouca transparência. Considere esta barra lateral um ponto de partida para sua pesquisa, descoberta e desenvolvimento de processos.

Serviços Cloud e Credit Default Swaps

Vamos sair um pouco pela tangente sobre os swaps de inadimplência de crédito. Não se preocupe, isso fará sentido daqui a pouco. Lembre-se de que os swaps de inadimplência de crédito tornaram-se infames após a crise financeira global de 2007. Um swap de inadimplência de crédito era um mecanismo para vender diferentes níveis de risco associados a um ativo (por exemplo, uma hipoteca). Não é nossa intenção apresentar essa ideia em detalhes, mas oferecer uma analogia em que muitos serviços em nuvem são semelhantes a derivativos financeiros; Os provedores de nuvem não apenas dividem os ativos de hardware em pequenos pedaços por meio da virtualização, mas também vendem esses pedaços com características técnicas e riscos associados variados. Embora os provedores sejam extremamente discretos sobre os detalhes de seus sistemas internos, há grandes oportunidades de otimização e dimensionamento ao entender os preços da nuvem e trocar notas com outros usuários.

Veja o exemplo de armazenamento em nuvem de arquivamento. No momento da redação deste artigo, o GCP admite abertamente que seu armazenamento de classe de arquivamento é executado nos mesmos clusters do armazenamento em nuvem padrão, mas o preço por gigabyte por mês de armazenamento de arquivamento é aproximadamente 1/17 do armazenamento padrão. Como isso é possível?

Aqui está o nosso palpite. Ao comprar armazenamento em nuvem, cada disco em um cluster de armazenamento possui três ativos que os provedores e consumidores de nuvem usam. Primeiro, ele tem uma certa capacidade de armazenamento, digamos, 10 TB. Em segundo lugar, ele oferece suporte a um determinado número de operações de entrada/saída (IOPs) por segundo — digamos, 100. Em terceiro lugar, os discos suportam uma determinada largura de banda máxima, a velocidade máxima de leitura para arquivos organizados de maneira otimizada. Uma unidade magnética pode ser capaz de ler a 200 MB/s.

Qualquer um desses limites (IOPs, capacidade de armazenamento, largura de banda) é um gargalo potencial para um provedor de nuvem. Por exemplo, o provedor de nuvem pode ter um disco armazenando 3 TB de dados, mas atingindo IOPs máximos. Uma alternativa para deixar os 7 TB restantes vazios é vender o espaço vazio sem vender IOPs. Ou, mais especificamente, vender espaço de armazenamento barato e IOPs caros para desencorajar leituras.

Assim como os comerciantes de derivativos financeiros, os fornecedores de nuvem também lidam com riscos. No caso do armazenamento de arquivos, os vendedores estão vendendo um tipo de seguro, mas que paga para a seguradora e não para o comprador da apólice no caso de uma catástrofe. Embora os custos de armazenamento de dados por mês sejam extremamente baratos, arrisco pagar um preço alto se precisar recuperar dados. Mas este é um preço que pagarei com prazer em uma verdadeira emergência.

Considerações semelhantes se aplicam a praticamente qualquer serviço de nuvem. Embora os servidores locais sejam vendidos essencialmente como hardware de commodity, o modelo de custo na nuvem é mais sutil. Em vez de apenas cobrar por núcleos de CPU, memória e recursos, os fornecedores de nuvem monetizam características como durabilidade, confiabilidade, longevidade e previsibilidade; uma variedade de plataformas de computação oferecem descontos em suas ofertas para cargas de trabalho que são **efêmeras** ou pode ser **interrompido arbitrariamente** quando a capacidade é necessária em outro lugar.

Nuvem ≠ No local

Esse título pode parecer uma tautologia boba, mas a crença de que os serviços em nuvem são como servidores locais familiares é um erro cognitivo generalizado que afeta as migrações para a nuvem e leva a contas horríveis. Isso demonstra um problema mais amplo em tecnologia que nos referimos como *a maldição da familiaridade*. Muitos novos produtos de tecnologia são projetados intencionalmente para se parecer com algo familiar para facilitar o uso e acelerar a adoção. Mas, qualquer produto de nova tecnologia tem sutilezas e rugas que os usuários devem aprender a identificar, acomodar e otimizar.

Mover servidores locais um por um para VMs na nuvem — conhecido como *simples/levante e mude* — é uma estratégia perfeitamente razoável para a fase inicial da migração para a nuvem, especialmente quando uma empresa está enfrentando algum tipo de precipício financeiro, como a necessidade de assinar um novo contrato de aluguel ou hardware significativo se o hardware existente não for desativado. No entanto, as empresas que deixam seus ativos de nuvem nesse estado inicial correm o risco de

um choque grosso. Em uma base de comparação direta, os servidores de longa duração na nuvem são significativamente mais caros do que seus equivalentes locais.

A chave para encontrar valor na nuvem é entender e otimizar o modelo de especificação da nuvem. Em vez de implantar um conjunto de servidores de longa duração capazes de lidar com cargas de pico completas, use o escalonamento automático para permitir que as cargas de trabalho diminuam para uma infraestrutura mínima quando as cargas são leves e para clusters massivos durante os horários de pico. Para obter descontos por meio de cargas de trabalho mais efêmeras e menos duráveis, use instâncias reservadas ou pontuais ou use funções sem servidor no lugar de servidores.

Muitas vezes pensamos que essa otimização leva a custos mais baixos, mas também devemos nos esforçar para aumentar o valor do negócio explorando a natureza dinâmica da nuvem.³ Os engenheiros de dados podem criar um novo valor na nuvem realizando coisas que seriam impossíveis em seu ambiente local. Por exemplo, é possível criar rapidamente clusters de computação massivos para executar transformações complexas em escalas inacessíveis para hardware local.

Gravidade de Dados

Além de erros básicos, como seguir práticas operacionais locais na nuvem, os engenheiros de dados precisam estar atentos a outros aspectos de preços e incentivos da nuvem que frequentemente pegam os usuários desprevenidos.

Os fornecedores querem trancá-lo em suas ofertas. Colocar dados na plataforma é barato ou gratuito na maioria das plataformas de nuvem, mas obter dados pode ser extremamente caro. Esteja ciente das taxas de saída de dados e seus impactos de longo prazo em seus negócios antes de ser pego de surpresa por uma conta grande. *Gravidade dos dados* é real: quando os dados chegam a uma nuvem, o custo para extraí-los e migrar os processos pode ser muito alto.

nuvem híbrida

À medida que empresas mais estabelecidas migram para a nuvem, o modelo de nuvem híbrida ganha importância. Praticamente nenhuma empresa pode migrar todas as suas cargas de trabalho da noite para o dia. O modelo de nuvem híbrida pressupõe que uma organização manterá indefinidamente algumas cargas de trabalho fora da nuvem.

Há muitos motivos para considerar um modelo de nuvem híbrida. As organizações podem acreditar que alcançaram a excelência operacional em determinadas áreas, como sua pilha de aplicativos e hardware associado. Assim, eles podem migrar apenas para cargas de trabalho específicas, onde veem benefícios imediatos no ambiente de nuvem. Por exemplo, uma pilha Spark local é migrada para clusters de nuvem efêmeros, reduzindo a carga operacional de gerenciamento de software e hardware para a equipe de engenharia de dados e permitindo o dimensionamento rápido para grandes trabalhos de dados.

³ Este é um ponto importante de ênfase em Storment e Fuller, *Nuvem FinOps*.

Esse padrão de colocar análises na nuvem é bonito porque os dados fluem principalmente em uma direção, minimizando os custos de saída de dados ([Figura 4-3](#)). Ou seja, os aplicativos locais geram dados de eventos que podem ser enviados para a nuvem essencialmente de graça. A maior parte dos dados permanece na nuvem onde é analisada, enquanto quantidades menores de dados são enviadas de volta para o local para implantação de modelos em aplicativos, ETL reverso, etc.

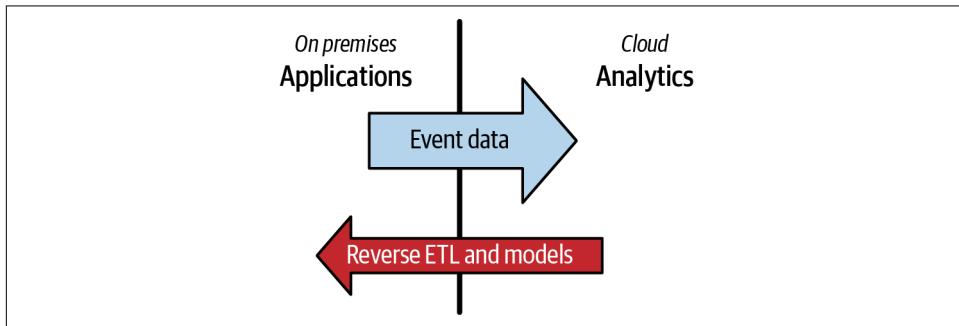


Figura 4-3. Um modelo de fluxo de dados em nuvem híbrida minimizando os custos de saída

Uma nova geração de ofertas de serviço de nuvem híbrida gerenciada também permite que os clientes localizem servidores gerenciados em nuvem em seus data centers.⁴ Isso dá aos usuários a capacidade de incorporar os melhores recursos em cada nuvem junto com a infraestrutura local.

Multicloud

Multicloud refere-se simplesmente à implantação de cargas de trabalho em várias nuvens públicas. As empresas podem ter várias motivações para implementações multicloud. As plataformas SaaS geralmente desejam oferecer serviços próximos às cargas de trabalho de nuvem existentes do cliente. Snowflake e Databricks fornecem suas ofertas de SaaS em várias nuvens por esse motivo. Isso é especialmente crítico para aplicativos com uso intensivo de dados, em que a latência da rede e as limitações de largura de banda prejudicam o desempenho e os custos de saída de dados podem ser proibitivos.

Outra motivação comum para empregar uma abordagem multicloud é aproveitar os melhores serviços em várias nuvens. Por exemplo, uma empresa pode querer gerenciar seus dados do Google Ads e Analytics no Google Cloud e implantar o Kubernetes por meio do GKE. E a empresa também pode adotar o Azure especificamente para cargas de trabalho da Microsoft. Além disso, a empresa pode gostar da AWS porque possui vários serviços de ponta (por exemplo, AWS Lambda) e desfruta de grande aceitação, tornando relativamente fácil a contratação de engenheiros proficientes na AWS. Qualquer combinação de vários serviços de provedor de nuvem é possível. Dada a intensa concorrência entre os principais provedores de nuvem, espere que eles ofereçam mais serviços de ponta, tornando a multicloud mais atraente.

4 Exemplos incluem [Google Cloud Anthos](#) e [Postos avançados da AWS](#).

Uma metodologia multicloud tem várias desvantagens. Como acabamos de mencionar, os custos de saída de dados e os gargalos de rede são críticos. Tornar-se multicloud pode apresentar uma complexidade significativa. As empresas agora devem gerenciar uma variedade estonteante de serviços em várias nuvens; a integração e a segurança entre nuvens apresentam um desafio considerável; a rede multicloud pode ser diabolicamente complicada.

Uma nova geração de serviços de “nuvem de nuvens” visa facilitar a multicloud com complexidade reduzida, oferecendo serviços entre nuvens e replicando dados perfeitamente entre nuvens ou gerenciando cargas de trabalho em várias nuvens por meio de um único painel de vidro. Para citar um exemplo, uma conta do Snowflake é executada em uma única região de nuvem, mas os clientes podem criar facilmente outras contas no GCP, AWS ou Azure. O Snowflake fornece replicação de dados agendada simples entre essas várias contas de nuvem. A interface do Snowflake é essencialmente a mesma em todas essas contas, eliminando a carga de treinamento de alternar entre os serviços de dados nativos da nuvem.

O espaço “nuvem das nuvens” está evoluindo rapidamente; poucos anos após a publicação deste livro, muitos outros serviços estarão disponíveis. Engenheiros e arquitetos de dados fariam bem em manter a consciência desse cenário de nuvem em rápida mudança.

Descentralizado: Blockchain e o Edge

Embora não seja amplamente usado agora, vale a pena mencionar brevemente uma nova tendência que pode se tornar popular na próxima década: a computação descentralizada. Considerando que os aplicativos de hoje são executados principalmente no local e na nuvem, o surgimento do blockchain, da Web 3.0 e da computação de ponta pode inverter esse paradigma. No momento, as plataformas descentralizadas se mostraram extremamente populares, mas não tiveram um impacto significativo no espaço de dados; mesmo assim, vale a pena ficar de olho nessas plataformas ao avaliar as decisões de tecnologia.

Nosso conselho

Do nosso ponto de vista, ainda estamos no início da transição para a nuvem. Assim, as evidências e os argumentos em torno da colocação e migração da carga de trabalho estão em fluxo. A própria nuvem está mudando, com uma mudança do modelo IaaS criado em torno do Amazon EC2 que impulsionou o crescimento inicial da AWS e, de maneira mais geral, em direção a ofertas de serviços mais gerenciados, como AWS Glue, Google BigQuery e Snowflake.

Também vimos o surgimento de novas abstrações de posicionamento de carga de trabalho. Os serviços locais estão se tornando mais parecidos com a nuvem e abstratos. Os serviços de nuvem híbrida permitem que os clientes executem serviços totalmente gerenciados dentro de suas paredes, facilitando a integração entre ambientes locais e remotos. Além disso, a “nuvem das nuvens” está começando a tomar forma, alimentada por serviços terceirizados e fornecedores de nuvem pública.

Escolha tecnologias para o presente, mas olhe para o futuro

Como mencionamos em “Hoje versus futuro: tecnologias imutáveis versus transitórias” na página 120, você precisa ficar de olho no presente enquanto planeja as incógnitas. Agora é um momento difícil para planejar colocações e migrações de carga de trabalho. Devido ao ritmo acelerado da concorrência e das mudanças no setor de nuvem, o espaço de decisão parecerá muito diferente em cinco a dez anos. É tentador levar em consideração todas as possíveis permutações futuras de arquitetura.

Acreditamos que é fundamental evitar essa armadilha sem fim de análise. Em vez disso, planeje para o presente. Escolha as melhores tecnologias para suas necessidades atuais e planos concretos para o futuro próximo. Escolha sua plataforma de implantação com base nas necessidades reais de negócios, concentrando-se na simplicidade e na flexibilidade.

Em particular, não escolha uma estratégia complexa de multicloud ou nuvem híbrida, a menos que haja um motivo convincente. Você precisa fornecer dados próximos aos clientes em várias nuvens? Os regulamentos do setor exigem que você hospede determinados dados em seus data centers? Você tem uma necessidade de tecnologia atraente para serviços específicos em duas nuvens diferentes? Escolha uma estratégia de implantação de nuvem única se esses cenários não se aplicarem a você.

Por outro lado, tenha um plano de fuga. Como enfatizamos antes, toda tecnologia – até mesmo software de código aberto – vem com algum grau de bloqueio. Uma estratégia de nuvem única tem vantagens significativas de simplicidade e integração, mas vem com um bloqueio significativo associado. Neste caso, estamos falando de flexibilidade mental, a flexibilidade para avaliar o estado atual do mundo e imaginar alternativas. Idealmente, seu plano de fuga permanecerá trancado atrás de um vidro, mas preparar esse plano o ajudará a tomar melhores decisões no presente e a fornecer uma saída se as coisas derem errado no futuro.

Argumentos de repatriação na nuvem

Enquanto escrevíamos este livro, Sarah Wang e Martin Casado publicaram “O custo da nuvem, um paradoxo de trilhões de dólares”, um artigo que gerou som e fúria significativos no espaço tecnológico. Os leitores interpretaram amplamente o artigo como um apelo à repatriação de cargas de trabalho em nuvem para servidores locais. Eles fazem um argumento um pouco mais sutil de que as empresas devem gastar recursos significativos para controlar os gastos com a nuvem e devem considerar a repatriação como uma opção possível.

Queremos dedicar um momento para dissecar uma parte de sua discussão. Wang e Casado citam a repatriação do Dropbox de cargas de trabalho significativas da AWS para servidores de propriedade do Dropbox como um estudo de caso para empresas que consideram movimentos de repatriação semelhantes.

Você não é Dropbox, nem Cloudflare

Acreditamos que este estudo de caso é freqüentemente usado sem contexto apropriado e é um exemplo convincente do *falsa equivalência* falácia lógica. O Dropbox oferece serviços específicos nos quais a propriedade de hardware e data centers pode oferecer uma vantagem competitiva. As empresas não devem confiar excessivamente no exemplo do Dropbox ao avaliar as opções de implantação local e na nuvem.

Primeiro, é importante entender que o Dropbox armazena enormes quantidades de dados. A empresa está de boca fechada sobre exatamente quantos dados ela hospeda, mas diz que são muitos exabytes e continua a crescer.

Em segundo lugar, o Dropbox lida com uma grande quantidade de tráfego de rede. Sabemos que seu consumo de largura de banda em 2017 foi significativo o suficiente para a empresa adicionar “centenas de gigabits de conectividade de internet com provedores de trânsito (ISPs regionais e globais) e centenas de novos parceiros de peering (onde trocamos tráfego diretamente em vez de do que através de um ISP).”⁵ Os custos de saída de dados seriam extremamente altos em um ambiente de nuvem pública.

Em terceiro lugar, o Dropbox é essencialmente um fornecedor de armazenamento em nuvem, mas com um produto de armazenamento altamente especializado que combina características de armazenamento de objetos e blocos. A principal competência do Dropbox é um sistema diferencial de atualização de arquivos que pode sincronizar com eficiência arquivos editados entre os usuários, minimizando o uso da rede e da CPU. O produto não é adequado para armazenamento de objetos, armazenamento em blocos ou outras ofertas de nuvem padrão. Em vez disso, o Dropbox se beneficiou da criação de uma pilha personalizada e altamente integrada de software e hardware.⁶

Quarto, embora o Dropbox tenha transferido seu produto principal para seu hardware, ele continuou desenvolvendo outras cargas de trabalho da AWS. Isso permite que o Dropbox se concentre na criação de um serviço de nuvem altamente ajustado em uma escala extraordinária, em vez de tentar substituir vários serviços. O Dropbox pode se concentrar em sua competência principal em armazenamento em nuvem e sincronização de dados enquanto descarrega o gerenciamento de software e hardware em outras áreas, como análise de dados.⁷

Outras histórias de sucesso frequentemente citadas que as empresas construíram fora da nuvem incluem Backblaze e Cloudflare, mas oferecem lições semelhantes. Backblaze começou como um produto pessoal de backup de dados em nuvem, mas desde então começou a oferecer B2, um serviço de armazenamento de objetos semelhante ao Amazon S3. Atualmente, o Backblaze armazena mais de um exabyte de dados. Cloudflare afirma fornecer serviços para mais de 25 milhões de propriedades da Internet,

5 Raghav Bhargava, “Evolution of Dropbox’s Edge Network,” Dropbox.Tech, 19 de junho de 2017, <https://oreil.ly/RAwPf>.

6 Akhil Gupta, “Scaling to Exabytes and Beyond,” Dropbox.Tech, 14 de março de 2016, <https://oreil.ly/5XPkV>.

7 “Dropbox Migrates 34 PB of Data to an Amazon S3 Data Lake for Analytics,” website da AWS, 2020, <https://oreil.ly/wpVOM>.

com pontos de presença em mais de 200 cidades e 51 terabits por segundo (Tbps) de capacidade total de rede.

A Netflix oferece mais um exemplo útil. A empresa é famosa por executar sua pilha de tecnologia na AWS, mas isso é apenas parcialmente verdade. A Netflix executa a transcodificação de vídeo na AWS, respondendo por aproximadamente 70% de suas necessidades de computação em 2017.⁸ A Netflix também executa seu back-end de aplicativos e análise de dados na AWS. No entanto, em vez de usar a rede de distribuição de conteúdo da AWS, a Netflix construiu um **CDN personalizado** em colaboração com provedores de serviços de Internet, utilizando uma combinação altamente especializada de software e hardware. Para uma empresa que consome uma fatia substancial de todo o tráfego da Internet,⁹ a construção dessa infra-estrutura crítica permitiu que ela fornecesse vídeo de alta qualidade para uma enorme base de clientes de maneira econômica.

Esses estudos de caso sugerem que faz sentido para as empresas gerenciar seu próprio hardware e conexões de rede em circunstâncias específicas. As maiores histórias de sucesso modernas de empresas que constroem e mantêm hardware envolvem escala extraordinária (exabytes de dados, terabits por segundo de largura de banda etc.) Além disso, todas essas empresas consomem grande largura de banda de rede, sugerindo que as cobranças de saída de dados seriam um grande custo se optassem por operar totalmente em uma nuvem pública.

Considere continuar a executar cargas de trabalho no local ou repatriar cargas de trabalho na nuvem se você executar um serviço verdadeiramente em escala de nuvem. O que é escala de nuvem? Você pode estar em escala de nuvem se estiver armazenando um exabyte de dados ou lidando com terabits por segundo de tráfego de rede para a internet. (Atingindo um terabit por segundo de *internet* tráfego de rede é bastante fácil.) Além disso, considere possuir seus servidores se os custos de saída de dados forem um fator importante para o seu negócio. Para dar um exemplo concreto de cargas de trabalho em escala de nuvem que poderiam se beneficiar da repatriação, a Apple pode obter uma vantagem financeira e de desempenho significativa ao migrar o armazenamento do iCloud para seus próprios servidores.¹⁰

Construir versus comprar

Construir versus comprar é um debate antigo em tecnologia. O argumento para a construção é que você tem controle de ponta a ponta sobre a solução e não está à mercê de um fornecedor ou comunidade de código aberto. O argumento que sustenta a compra se resume a restrições de recursos e expertise; você tem a experiência para construir um

8 Todd Hoff, "The Eternal Cost Savings of Netflix's Internal Spot Market," *High Scalability*, 4 de dezembro de 2017, <https://oreil.ly/LLoFt>.

9 Todd Spangler, "Consumo de largura de banda da Netflix eclipsado por aplicativos de streaming de mídia na Web," *Variedade*, 10 de setembro de 2019, <https://oreil.ly/tTm3k>.

10 Amir Efrati e Kevin McLaughlin, "Gastos da Apple no Google Cloud Storage a caminho de aumentar 50% neste Ano," *A informação*, 29 de junho de 2021, <https://oreil.ly/OIfyR>.

melhor solução do que algo já disponível? Qualquer decisão se resume a TCO, TOCO e se a solução oferece uma vantagem competitiva para sua organização.

Se você captou algum tema do livro até aqui, é que sugerimos investir na construção e customização *ao fazê-lo, proporcionará uma vantagem competitiva* para seu negócio. Caso contrário, fique sobre os ombros de gigantes e usar o que já está disponível no mercado. Dado o número de serviços de código aberto e pagos - ambos podem ter comunidades de voluntários ou equipes altamente bem pagas de engenheiros incríveis - você é um tolo em construir tudo sozinho.

Como costumamos perguntar: "Quando você precisa de pneus novos para o seu carro, você obtém as matérias-primas, cria os pneus do zero e os instala você mesmo?" Como a maioria das pessoas, você provavelmente está comprando pneus e tendo alguém para instalá-los. O mesmo argumento se aplica a construir versus comprar. Vimos equipes que construíram seus bancos de dados do zero. Um simples RDBMS de código aberto teria atendido muito melhor às suas necessidades após uma inspeção mais detalhada. Imagine a quantidade de tempo e dinheiro investidos neste banco de dados caseiro. Fale sobre baixo ROI para TCO e custo de oportunidade.

É aqui que a distinção entre o engenheiro de dados do tipo A e do tipo B é útil. Como apontamos anteriormente, as funções do tipo A e do tipo B geralmente estão incorporadas ao mesmo engenheiro, especialmente em uma pequena organização. Sempre que possível, incline-se para o comportamento do tipo A; evite o trabalho pesado indiferenciado e abrace a abstração. Use estruturas de código aberto ou, se isso for muito problemático, procure comprar uma solução proprietária ou gerenciada adequada. Muitos serviços modulares excelentes estão disponíveis para escolha em ambos os casos.

Vale a pena mencionar a realidade mutante de como as empresas adotam software. Enquanto no passado a TI costumava tomar a maior parte das decisões de compra e adoção de software de maneira descendente, hoje em dia a tendência é a adoção de software ascendente em uma empresa, impulsionada por desenvolvedores, engenheiros de dados, cientistas de dados, e outras funções técnicas. A adoção de tecnologia nas empresas está se tornando um processo orgânico e contínuo.

Vejamos algumas opções de soluções proprietárias e de código aberto.

Software livre

Software livre(OSS) é um modelo de distribuição de software no qual o software e a base de código subjacente são disponibilizados para uso geral, geralmente sob termos de licenciamento específicos. Freqüentemente, o OSS é criado e mantido por uma equipe distribuída de colaboradores. O OSS é livre para usar, alterar e distribuir na maior parte do tempo, mas com ressalvas específicas. Por exemplo, muitas licenças exigem que o código-fonte do software derivado de código-fonte aberto seja incluído quando o software é distribuído.

As motivações para criar e manter OSS variam. Às vezes, o OSS é orgânico, surgindo da mente de um indivíduo ou de uma pequena equipe que cria uma nova solução e opta por liberá-la para uso público. Outras vezes, uma empresa pode disponibilizar uma ferramenta ou tecnologia específica ao público sob uma licença OSS.

OSS tem dois tipos principais: OSS gerenciado pela comunidade e OSS comercial.

OSS gerenciado pela comunidade

Os projetos OSS são bem-sucedidos com uma comunidade forte e uma base de usuários vibrante. *OSS gerenciado pela comunidade* é um caminho predominante para projetos OSS. A comunidade abre altas taxas de inovações e contribuições de desenvolvedores em todo o mundo com projetos OSS populares.

A seguir estão os fatores a serem considerados com um projeto de OSS gerenciado pela comunidade:

Mindshare

Evite adotar projetos OSS que não tenham tração e popularidade. Observe o número de estrelas do GitHub, bifurcações e volume de confirmação e atualidade. Outra coisa a prestar atenção é a atividade da comunidade em grupos de bate-papo e fóruns relacionados. O projeto tem um forte senso de comunidade? Uma comunidade forte cria um ciclo virtuoso de forte adoção. Isso também significa que você terá mais facilidade em obter assistência técnica e encontrar talentos qualificados para trabalhar com o framework.

Maturidade

Há quanto tempo o projeto existe, quão ativo ele está hoje e quão utilizável as pessoas o encontram na produção? A maturidade de um projeto indica que as pessoas o consideram útil e desejam incorporá-lo em seus fluxos de trabalho de produção.

Solução de problemas

Como você terá que lidar com os problemas se eles surgirem? Você está sozinho para solucionar problemas ou a comunidade pode ajudá-lo a resolver seu problema?

Gerenciamento de projetos

Veja os problemas do Git e a maneira como eles são abordados. Eles são abordados rapidamente? Em caso afirmativo, qual é o processo para enviar um problema e resolvê-lo?

Equipe

Uma empresa está patrocinando o projeto OSS? Quem são os principais contribuintes?

Relacionamento com desenvolvedores e gerenciamento da comunidade

O que o projeto está fazendo para incentivar a aceitação e adoção? Existe uma comunidade de bate-papo vibrante (por exemplo, no Slack) que oferece incentivo e suporte?

Contribuindo

O projeto incentiva e aceita pull requests? Quais são os processos e cronogramas para que as solicitações pull sejam aceitas e incluídas na base de código principal?

Roteiro

Existe um roteiro de projeto? Em caso afirmativo, é claro e transparente?

Auto-hospedagem e manutenção

Você tem os recursos para hospedar e manter a solução OSS? Em caso afirmativo, qual é o TCO e o TOCO em comparação com a compra de um serviço gerenciado do fornecedor de OSS?

Retribuindo à comunidade

Se você gosta do projeto e o está usando ativamente, considere investir nele. Você pode contribuir com a base de código, ajudar a corrigir problemas e dar conselhos nos fóruns e bate-papos da comunidade. Se o projeto permitir doações, considere fazer uma. Muitos projetos OSS são essencialmente projetos de serviço comunitário, e os mantenedores geralmente têm empregos em tempo integral, além de ajudar no projeto OSS. Infelizmente, muitas vezes é um trabalho de amor que não oferece ao mantenedor um salário digno. Se você puder doar, por favor, faça.

OSS comercial

Às vezes, o OSS tem algumas desvantagens. Ou seja, você precisa hospedar e manter a solução em seu ambiente. Isso pode ser trivial ou extremamente complicado e complicado, dependendo do aplicativo OSS. Os fornecedores comerciais tentam resolver essa dor de cabeça de gerenciamento hospedando e gerenciando a solução OSS para você, geralmente como uma oferta de SaaS em nuvem. Exemplos de tais fornecedores incluem Databricks (Spark), Confluent (Kafka), DBT Labs (dbt) e existem muitos, muitos outros.

Este modelo é chamado *OSS comercial*(COSS). Normalmente, um fornecedor oferece o “núcleo” do OSS gratuitamente enquanto cobra por melhorias, distribuições de código com curadoria ou serviços totalmente gerenciados.

Um fornecedor geralmente é afiliado ao projeto OSS da comunidade. À medida que um projeto OSS se torna mais popular, os mantenedores podem criar um negócio separado para uma versão gerenciada do OSS. Isso normalmente se torna uma plataforma SaaS em nuvem criada em torno de uma versão gerenciada do código-fonte aberto. Esta é uma tendência generalizada: um projeto OSS torna-se popular, uma empresa afiliada levanta caminhões de dinheiro de capital de risco (VC) para comercializar o projeto OSS e a empresa cresce como um foguete em movimento rápido.

Neste ponto, o engenheiro de dados tem duas opções. Você pode continuar usando a versão OSS gerenciada pela comunidade, que você precisa continuar mantendo por conta própria (atualizações, manutenção de servidor/contêiner, solicitações pull para correções de bugs, etc.). Ou você pode pagar ao fornecedor e deixá-lo cuidar do gerenciamento administrativo do produto COSS.

A seguir estão os fatores a serem considerados com um projeto OSS comercial:

Valor

O fornecedor está oferecendo um valor melhor do que se você mesmo gerenciasse a tecnologia OSS? Alguns fornecedores adicionarão muitos sinos e assobios às suas ofertas gerenciadas que não estão disponíveis na versão OSS da comunidade. Essas adições são atraentes para você?

modelo de entrega

Como você acessa o serviço? O produto está disponível por download, API ou interface de usuário da Web/móvel? Certifique-se de que você pode acessar facilmente a versão inicial e as versões subsequentes.

Apoiar

O suporte não pode ser subestimado e geralmente é opaco para o comprador. Qual é o modelo de suporte para o produto e há um custo extra para suporte? Freqüentemente, os fornecedores vendem suporte por uma taxa adicional. Certifique-se de entender claramente os custos de obtenção de suporte. Além disso, entenda o que é coberto pelo suporte e o que não é coberto. Qualquer coisa que não esteja coberta pelo suporte será de sua responsabilidade possuir e gerenciar.

Lançamentos e correções de bugs

O fornecedor é transparente sobre o cronograma de lançamento, melhorias e correções de bugs? Essas atualizações estão facilmente disponíveis para você?

Ciclo de vendas e preços

Freqüentemente, um fornecedor oferece preços sob demanda, especialmente para um produto SaaS, e oferece um desconto se você se comprometer com um contrato estendido. Certifique-se de entender as vantagens e desvantagens de pagar conforme o uso versus pagar adiantado. Vale a pena pagar uma quantia fixa ou é melhor gastar seu dinheiro em outro lugar?

finanças da empresa

A empresa é viável? Se a empresa levantou fundos de VC, você pode verificar o financiamento em sites como o Crunchbase. Quanta pista a empresa tem e ainda estará no mercado daqui a alguns anos?

Logotipos versus receita

A empresa está focada em aumentar o número de clientes (logotipos) ou está tentando aumentar a receita? Você pode se surpreender com o número de empresas preocupadas principalmente em aumentar sua contagem de clientes, estrelas do GitHub ou membros do canal Slack sem a receita para estabelecer finanças sólidas.

Supporte da comunidade

A empresa está realmente apoiando a versão comunitária do projeto OSS? Quanto a empresa está contribuindo para a base de código OSS da comunidade? Controvérsias surgiram com certos fornecedores cooptando projetos de OSS e subsequentemente

fornecendo pouco valor de volta para a comunidade. Qual a probabilidade de o produto permanecer viável como um código aberto apoiado pela comunidade se a empresa fechar?

Observe também que as nuvens oferecem seus próprios produtos gerenciados de código aberto. Se um fornecedor de nuvem vê tração com um determinado produto ou projeto, espere que esse fornecedor ofereça sua versão. Isso pode variar de exemplos simples (Linux de código aberto oferecido em VMs) a serviços gerenciados extremamente complexos (Kafka totalmente gerenciado). A motivação para essas ofertas é simples: as nuvens ganham dinheiro com o consumo. Mais ofertas em um ecossistema de nuvem significam uma chance maior de “permanência” e aumento dos gastos do cliente.

Jardins murados proprietários

Embora o OSS seja onipresente, também existe um grande mercado para tecnologias não OSS. Algumas das maiores empresas do setor de dados vendem produtos de código fechado. Vejamos dois tipos principais de *jardins murados proprietários*, empresas independentes e ofertas de plataforma de nuvem.

ofertas independentes

O cenário de ferramentas de dados teve um crescimento exponencial nos últimos anos. Todos os dias surgem novas ofertas independentes para ferramentas de dados. Com a capacidade de levantar fundos de VCs cheios de capital, essas empresas de dados podem escalar e contratar grandes equipes de engenharia, vendas e marketing. Isso apresenta uma situação em que os usuários têm ótimas opções de produtos no mercado, enquanto precisam percorrer vendas intermináveis e confusão de marketing. No momento em que escrevo este livro, os bons tempos de capital disponível gratuitamente para empresas de dados estão chegando ao fim, mas essa é outra longa história cujas consequências ainda estão se desenrolando.

Freqüentemente, uma empresa que vende uma ferramenta de dados não a lança como OSS, mas oferece uma solução proprietária. Embora você não tenha a transparência de uma solução OSS pura, uma solução proprietária independente pode funcionar muito bem, especialmente como um serviço totalmente gerenciado na nuvem.

A seguir estão as coisas a considerar com uma oferta independente:

Interoperabilidade

Certifique-se de que a ferramenta interopere com outras ferramentas que você escolheu (OSS, outras independentes, ofertas de nuvem, etc.). A interoperabilidade é fundamental, portanto, certifique-se de experimentá-lo antes de comprar.

Mindshare e participação de mercado

A solução é popular? Ele comanda uma presença no mercado? Ele gosta de comentários positivos dos clientes?

Documentação e suporte

Problemas e perguntas inevitavelmente surgirão. Está claro como resolver seu problema, seja por meio de documentação ou suporte?

Preços

O preço é compreensível? Mapeie cenários de baixa, média e alta probabilidade de uso, com os respectivos custos. Você é capaz de negociar um contrato, juntamente com um desconto? Vale a pena? Quanta flexibilidade você perde ao assinar um contrato, tanto na negociação quanto na capacidade de experimentar novas opções? Você é capaz de obter compromissos contratuais sobre preços futuros?

Longevidade

A empresa sobreviverá tempo suficiente para você obter valor de seu produto? Se a empresa levantou dinheiro, procure sua situação de financiamento. Veja as avaliações dos usuários. Pergunte a amigos e poste perguntas nas redes sociais sobre as experiências dos usuários com o produto. Certifique-se de saber no que está se metendo.

Ofertas de serviços proprietários da plataforma de nuvem

Os fornecedores de nuvem desenvolvem e vendem seus serviços proprietários para armazenamento, bancos de dados e muito mais. Muitas dessas soluções são ferramentas internas usadas pelas respectivas empresas irmãs. Por exemplo, a Amazon criou o banco de dados DynamoDB para superar as limitações dos bancos de dados relacionais tradicionais e lidar com grandes quantidades de dados de usuários e pedidos à medida que a Amazon.com se tornava um gigante. Posteriormente, a Amazon ofereceu o serviço DynamoDB exclusivamente na AWS; agora é um produto de primeira linha usado por empresas de todos os portes e níveis de maturidade. Os fornecedores de nuvem geralmente agrupam seus produtos para funcionarem bem juntos. Cada nuvem pode criar aderência com sua base de usuários, criando um forte ecossistema integrado.

A seguir estão os fatores a serem considerados com uma oferta de nuvem proprietária:

Comparações de desempenho versus preço

A oferta de nuvem é substancialmente melhor do que uma versão independente ou OSS? Qual é o TCO de escolher uma oferta de nuvem?

Considerações de compra

O preço sob demanda pode ser caro. Você pode reduzir seus custos adquirindo capacidade reservada ou firmando um contrato de compromisso de longo prazo?

Nosso conselho

Construir versus comprar volta a conhecer sua vantagem competitiva e onde faz sentido investir recursos para customização. Em geral, favorecemos o OSS e o COSS por padrão, o que o libera para se concentrar em melhorar as áreas em que essas opções são insuficientes. Concentre-se em algumas áreas em que construir algo agregará valor significativo ou reduzirá substancialmente o atrito.

Não trate a sobrecarga operacional interna como um custo irrecuperável. Há um valor excelente em aprimorar sua equipe de dados existente para criar sistemas sofisticados em plataformas gerenciadas, em vez de cuidar de servidores locais. Além disso, pense em como uma empresa ganha dinheiro, especialmente suas equipes de vendas e experiência do cliente, que geralmente indicam como você é tratado durante o ciclo de vendas e quando é um cliente pagante.

Finalmente, quem é o responsável pelo orçamento na sua empresa? Como essa pessoa decide os projetos e tecnologias que serão financiados? Antes de fazer o caso de negócios para COSS ou serviços gerenciados, faz sentido tentar usar o OSS primeiro? A última coisa que você deseja é que sua escolha de tecnologia fique presa no limbo enquanto aguarda a aprovação do orçamento. Como diz o velho ditado, *o tempo mata negócios*. No seu caso, mais tempo gasto no limbo significa uma probabilidade maior de que a aprovação do seu orçamento seja interrompida. Saiba de antemão quem controla o orçamento e o que será aprovado com sucesso.

Monólito Versus Modular

Monólitos versus sistemas modulares é outro debate de longa data no espaço da arquitetura de software. Os sistemas monolíticos são autocontidos, muitas vezes realizando múltiplas funções em um único sistema. O acampamento monolítico favorece a simplicidade de ter tudo em um só lugar. É mais fácil raciocinar sobre uma única entidade e você pode se mover mais rápido porque há menos partes móveis. *Omodularcamp* inclina-se para as melhores tecnologias desacopladas, executando tarefas nas quais são excepcionalmente excelentes. Especialmente considerando a taxa de mudança de produtos no mundo dos dados, o argumento é que você deve buscar a interoperabilidade entre uma gama de soluções em constante mudança.

Qual abordagem você deve adotar em sua pilha de engenharia de dados? Vamos explorar as compensações.

Monólito

O *monólito* ([Figura 4-4](#)) tem sido um dos pilares da tecnologia por décadas. Os velhos tempos da cascata significavam que os lançamentos de software eram enormes, fortemente acoplados e movidos em uma cadência lenta. Grandes equipes trabalharam juntas para fornecer uma única base de código funcional. Os sistemas de dados monolíticos continuam até hoje, com fornecedores de software mais antigos, como Informatica, e estruturas de código aberto, como Spark.

Os prós do monólito são fáceis de raciocinar e requer uma carga cognitiva menor e mudança de contexto, já que tudo é independente. Em vez de lidar com dezenas de tecnologias, você lida com “uma” tecnologia e normalmente uma linguagem de programação principal. Monólitos são uma excelente opção se você deseja simplicidade no raciocínio sobre sua arquitetura e processos.

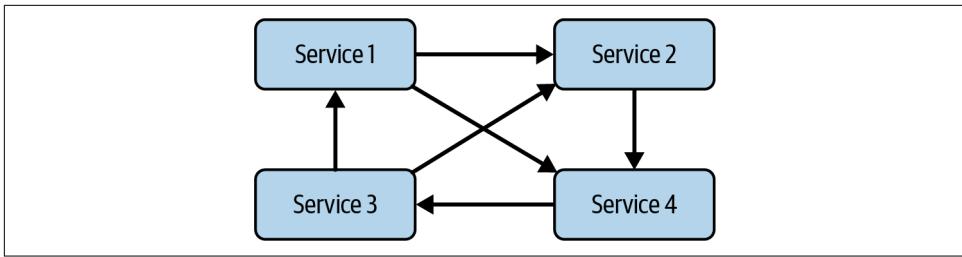


Figura 4-4. O monólito acopla fortemente seus serviços

Claro, o monólito tem contras. Por um lado, os monólitos são frágeis. Devido ao grande número de partes móveis, atualizações e lançamentos demoram mais e tendem a assar na “pia da cozinha”. Se o sistema tiver um bug - espera-se que o software tenha sido totalmente testado antes do lançamento! - ele pode danificar todo o sistema.

Problemas induzidos pelo usuário também acontecem com monólitos. Por exemplo, vimos um pipeline ETL monolítico que levou 48 horas para ser executado. Se algo quebrasse em algum ponto do pipeline, todo o processo precisava ser reiniciado. Enquanto isso, usuários corporativos ansiosos aguardavam seus relatórios, que já estavam atrasados dois dias por padrão e geralmente chegavam muito depois. As quebras eram tão comuns que o sistema monolítico acabou sendo descartado.

A multilocação em um sistema monolítico também pode ser um problema significativo. Pode ser um desafio isolar as cargas de trabalho de vários usuários. Em um data warehouse local, uma função definida pelo usuário pode consumir CPU suficiente para desacelerar o sistema para outros usuários. Conflitos entre dependências e contenção de recursos são fontes frequentes de dores de cabeça.

Outra desvantagem dos monólitos é que mudar para um novo sistema será doloroso se o fornecedor ou o projeto de código aberto morrer. Como todos os seus processos estão contidos no monólito, sair desse sistema e colocá-lo em uma nova plataforma custará tempo e dinheiro.

Modularidade

Modularidade(Figura 4-5) é um conceito antigo em engenharia de software, mas os sistemas distribuídos modulares realmente entraram em voga com o surgimento dos microsserviços. Em vez de depender de um monólito maciço para lidar com suas necessidades, por que não separar sistemas e processos em suas áreas de interesse independentes? Os microsserviços podem se comunicar por meio de APIs, permitindo que os desenvolvedores se concentrem em seus domínios enquanto tornam seus aplicativos acessíveis a outros microsserviços. Essa é a tendência na engenharia de software e cada vez mais vista nos sistemas de dados modernos.

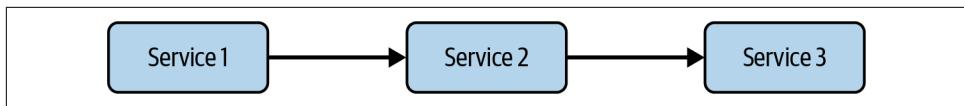


Figura 4-5. Com a modularidade, cada serviço é desacoplado de outro

As principais empresas de tecnologia têm sido os principais impulsionadores do movimento de microsserviços. A famosa API de Bezos diminui o acoplamento entre aplicativos, permitindo refatoração e decomposição. Bezos também impôs a regra das duas pizzas (nenhuma equipe deve ser tão grande que duas pizzas não possam alimentar todo o grupo). Efetivamente, isso significa que uma equipe terá no máximo cinco membros. Esse limite também limita a complexidade do domínio de responsabilidade de uma equipe — em particular, a base de código que ela pode gerenciar. Considerando que um aplicativo monolítico extenso pode envolver um grupo de cem pessoas, dividir os desenvolvedores em pequenos grupos de cinco requer que esse aplicativo seja dividido em partes pequenas, gerenciáveis e fracamente acopladas.

Em um ambiente de microsserviço modular, os componentes são intercambiáveis e é possível criar *um polyglota* (linguagem de multiprogramação) aplicação; um serviço Java pode substituir um serviço escrito em Python. Os clientes do serviço precisam se preocupar apenas com as especificações técnicas da API do serviço, não com os detalhes da implementação nos bastidores.

As tecnologias de processamento de dados mudaram para um modelo modular, fornecendo forte suporte para interoperabilidade. Os dados são armazenados no armazenamento de objetos em um formato padrão, como Parquet, em data lakes e lakehouses. Qualquer ferramenta de processamento que suporte o formato pode ler os dados e gravar os resultados processados de volta no lake para processamento por outra ferramenta. Os data warehouses em nuvem oferecem suporte à interoperação com armazenamento de objetos por meio de importação/exportação usando formatos padrão e tabelas externas, ou seja, consultas executadas diretamente nos dados em um data lake.

Novas tecnologias entram em cena em um ritmo vertiginoso no ecossistema de dados de hoje, e a maioria fica obsoleta e obsoleta rapidamente. Enxague e repita. A capacidade de trocar ferramentas conforme a tecnologia muda é inestimável. Vemos a modularidade de dados como um paradigma mais poderoso do que a engenharia de dados monolítica. A modularidade permite que os engenheiros escolham a melhor tecnologia para cada trabalho ou etapa ao longo do pipeline.

Os contras da modularidade são que há mais motivos para raciocinar. Em vez de lidar com um único sistema de preocupação, agora você tem potencialmente inúmeros sistemas para entender e operar. A interoperabilidade é uma dor de cabeça em potencial; esperamos que todos esses sistemas funcionem bem juntos.

Esse mesmo problema nos levou a separar a orquestração como uma subcorrente separada, em vez de colocá-la sob o gerenciamento de dados. A orquestração também é importante para arquiteturas de dados monolíticas; testemunhe o sucesso de ferramentas como o Control-M da BMC Software no espaço de armazenamento de dados tradicional. Mas orquestrar cinco ou dez

ferramentas é dramaticamente mais complexo do que orquestrar um. A orquestração torna-se a cola que une os módulos da pilha de dados.

O Padrão Monolítico Distribuído

O *padrão de monólito distribuído* é uma arquitetura distribuída que ainda sofre com muitas das limitações da arquitetura monolítica. A ideia básica é que se execute um sistema distribuído com diferentes serviços para realizar diferentes tarefas. Ainda assim, serviços e nós compartilham um conjunto comum de dependências ou uma base de código comum.

Um exemplo padrão é um cluster Hadoop tradicional. Um cluster Hadoop pode hospedar simultaneamente vários frameworks, como Hive, Pig ou Spark. O cluster também tem muitas dependências internas. Além disso, o cluster executa os principais componentes do Hadoop: bibliotecas comuns do Hadoop, HDFS, YARN e Java. Na prática, um cluster geralmente possui uma versão de cada componente instalado.

Um sistema Hadoop local padrão envolve o gerenciamento de um ambiente comum que funciona para todos os usuários e todos os trabalhos. Gerenciar atualizações e instalações é um desafio significativo. Forçar trabalhos para atualizar dependências corre o risco de quebrá-los; manter duas versões de um framework envolve complexidade extra.

Algumas tecnologias modernas de orquestração baseadas em Python — por exemplo, Apache Airflow — também sofrem desse problema. Embora utilizem uma arquitetura altamente desacoplada e assíncrona, cada serviço executa a mesma base de código com as mesmas dependências. Qualquer executor pode executar qualquer tarefa, portanto, uma biblioteca cliente para uma única tarefa executada em um DAG deve ser instalada em todo o cluster. A orquestração de muitas ferramentas envolve a instalação de bibliotecas de cliente para um host de APIs. Conflitos de dependência são um problema constante.

Uma solução para os problemas do monólito distribuído é a infraestrutura efêmera em um ambiente de nuvem. Cada trabalho obtém seu próprio servidor temporário ou cluster instalado com dependências. Cada cluster permanece altamente monolítico, mas a separação de tarefas reduz drasticamente os conflitos. Por exemplo, esse padrão agora é bastante comum para Spark com serviços como Amazon EMR e Google Cloud Dataproc.

Uma segunda solução é decompor adequadamente o monólito distribuído em vários ambientes de software usando contêineres. Temos mais a dizer sobre contêineres em "[Sem servidor versus servidores](#)" na página 143.

Nosso conselho

Embora os monólitos sejam atraentes devido à facilidade de compreensão e complexidade reduzida, isso tem um alto custo. O custo é a perda potencial de flexibilidade, custo de oportunidade e ciclos de desenvolvimento de alto atrito.

Aqui estão algumas coisas a considerar ao avaliar monólitos versus opções modulares:

Interoperabilidade

Arquiteto para compartilhamento e interoperabilidade.

Evitando a “armadilha do urso”

Algo que é fácil de entrar pode ser doloroso ou impossível de escapar.

Flexibilidade

As coisas estão se movendo tão rápido no espaço de dados agora. Comprometer-se com um monólito reduz a flexibilidade e as decisões reversíveis.

Sem servidor versus servidores

Uma grande tendência para provedores de nuvem é *sem servidor*, permitindo que desenvolvedores e engenheiros de dados executem aplicativos sem gerenciar servidores nos bastidores. Serverless fornece um tempo rápido para valorizar os casos de uso corretos. Para outros casos, pode não ser um bom ajuste. Vejamos como avaliar se o serverless é adequado para você.

sem servidor

Embora o serverless já exista há algum tempo, a tendência serverless começou com força total com o AWS Lambda em 2014. Com a promessa de executar pequenos blocos de código conforme necessário sem ter que gerenciar um servidor, o serverless explodiu em popularidade. As principais razões para sua popularidade são custo e conveniência. Em vez de pagar o custo de um servidor, por que não pagar apenas quando seu código for evocado?

Serverless tem muitos sabores. Embora a função como serviço (FaaS) seja extremamente popular, os sistemas sem servidor são anteriores ao advento do AWS Lambda. Por exemplo, o BigQuery do Google Cloud é sem servidor, pois os engenheiros de dados não precisam gerenciar a infraestrutura de back-end, e o sistema é dimensionado para zero e expandido automaticamente para lidar com consultas grandes. Basta carregar os dados no sistema e começar a consultar. Você paga pela quantidade de dados que sua consulta consome e um pequeno custo para armazenar seus dados. Esse modelo de pagamento – pagar por consumo e armazenamento – está se tornando mais comum.

Quando o serverless faz sentido? Como acontece com muitos outros serviços em nuvem, depende; e os engenheiros de dados fariam bem em entender os detalhes do preço da nuvem para prever quando as implantações sem servidor se tornarão caras. Olhando especificamente para o caso do AWS Lambda, vários engenheiros encontraram hacks para executar cargas de trabalho em lote com custos reduzidos.¹¹ Por outro lado, as funções sem servidor sofrem de uma ineficiência de sobrecarga inherente. Lidar com um evento por chamada de função em uma taxa de evento alta pode ser catastroficamente caro, especialmente quando abordagens mais simples como multithreading ou multiprocessamento são ótimas alternativas.

11 Evan Sangaline, “Running FFmpeg on AWS Lambda for 1.9% the Cost of AWS Elastic Transcoder,” Intoli blog, 2 de maio de 2018, <https://oreil.ly/myzOv>.

Assim como em outras áreas de operações, é fundamental monitorar e modelar. Monitor para determinar o custo por evento em um ambiente do mundo real e a duração máxima da execução sem servidor, e modelo usando esse custo por evento para determinar os custos gerais à medida que as taxas de eventos aumentam. A modelagem também deve incluir os piores cenários — o que acontece se meu site for atingido por um enxame de bots ou ataque DDoS?

Containers

Em conjunto com serviços sem servidor e microsserviços, containers são uma das tecnologias operacionais de tendências mais poderosas até o momento em que este livro foi escrito. Os contêineres desempenham um papel nos microsserviços e sem servidor.

Os contêineres são geralmente chamados de *máquinas virtuais leves*. Enquanto uma VM tradicional envolve um sistema operacional inteiro, um contêiner empacota um espaço de usuário isolado (como um sistema de arquivos e alguns processos); muitos desses contêineres podem coexistir em um único sistema operacional host. Isso fornece alguns dos principais benefícios da virtualização (ou seja, dependência e isolamento de código) sem a sobrecarga de carregar todo o kernel do sistema operacional.

Um único nó de hardware pode hospedar vários contêineres com alocações de recursos refinadas. No momento da redação deste artigo, os contêineres continuam a crescer em popularidade, juntamente com o Kubernetes, um sistema de gerenciamento de contêineres. Os ambientes sem servidor geralmente são executados em contêineres nos bastidores. De fato, o Kubernetes é uma espécie de ambiente sem servidor, pois permite que desenvolvedores e equipes de operações implantem microsserviços sem se preocupar com os detalhes das máquinas em que são implantados.

Os contêineres fornecem uma solução parcial para os problemas do monólito distribuído mencionados anteriormente neste capítulo. Por exemplo, o Hadoop agora oferece suporte a contêineres, permitindo que cada trabalho tenha suas próprias dependências isoladas.



Os clusters de contêineres não fornecem a mesma segurança e isolamento que as VMs completas oferecem. *Fuga de contêiner* — em termos gerais, uma classe de exploits em que o código em um contêiner ganha privilégios fora do contêiner no nível do sistema operacional — é comum e suficiente para ser considerado um risco para multilocação. Embora o Amazon EC2 seja um ambiente verdadeiramente multilocatário com VMs de muitos clientes hospedados no mesmo hardware, um cluster Kubernetes deve hospedar código apenas em um ambiente de confiança mútua (por exemplo, dentro das paredes de uma única empresa). Além disso, os processos de revisão de código e verificação de vulnerabilidade são essenciais para garantir que um desenvolvedor não introduza uma falha de segurança.

Vários tipos de plataformas de contêiner adicionam recursos adicionais sem servidor. Plataformas de função conteinerizadas executam contêineres como unidades efêmeras acionadas por eventos em vez de

do que serviços persistentes.¹² Isso oferece aos usuários a simplicidade do AWS Lambda com a flexibilidade total de um ambiente de contêiner em vez do tempo de execução do Lambda altamente restritivo. E serviços como AWS Fargate e Google App Engine executam contêineres sem gerenciar um cluster de computação necessário para o Kubernetes. Esses serviços também isolam totalmente os contêineres, evitando os problemas de segurança associados à multilocação.

A abstração continuará trabalhando na pilha de dados. Considere o impacto do Kubernetes no gerenciamento de cluster. Embora você possa gerenciar seu cluster do Kubernetes – e muitas equipes de engenharia o fazem – até mesmo o Kubernetes está amplamente disponível como um serviço gerenciado. O que vem depois do Kubernetes? Estamos tão ansiosos quanto você para descobrir.

Como avaliar servidor versus sem servidor

Por que você deseja executar seus próprios servidores em vez de usar sem servidor? Há algumas razões. O custo é um grande fator. Serverless faz menos sentido quando o uso e o custo excedem o custo contínuo de execução e manutenção de um servidor ([Figura 4-6](#)). No entanto, em certa escala, os benefícios econômicos do serverless podem diminuir e a execução de servidores se torna mais atraente.

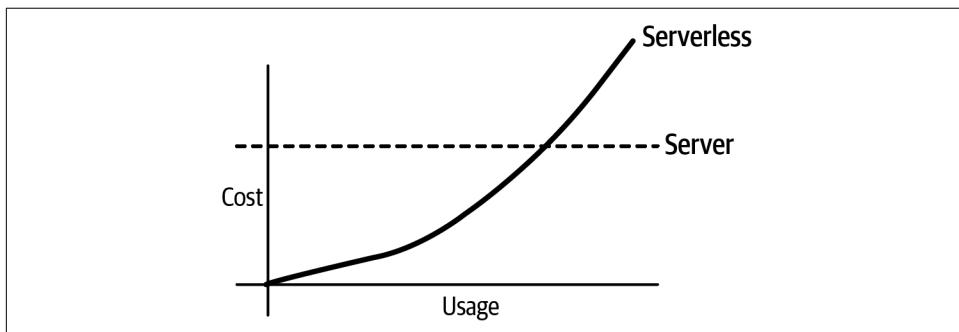


Figura 4-6. Custo sem servidor versus utilização de um servidor

Personalização, poder e controle são outros motivos importantes para favorecer servidores em vez de sem servidor. Algumas estruturas sem servidor podem ser insuficientes ou limitadas para determinados casos de uso. Aqui estão algumas coisas a considerar ao usar servidores, especialmente na nuvem, onde os recursos do servidor são efêmeros:

Espere que os servidores falhem.

A falha do servidor acontecerá. Evite usar um servidor “floco de neve especial” que seja excessivamente personalizado e frágil, pois isso introduz uma vulnerabilidade gritante em sua arquitetura. Em vez disso, trate os servidores como recursos efêmeros que você pode criar conforme necessário e depois excluir. Se seu aplicativo requer código específico para ser instalado no

12 Exemplos incluem [OpenFaaS](#), [Knative](#), e [Google Cloud Run](#).

servidor, use um script de inicialização ou crie uma imagem. Implante o código no servidor por meio de um pipeline de CI/CD.

Use clusters e escalonamento automático.

Aproveite a capacidade da nuvem de aumentar e reduzir os recursos de computação sob demanda. À medida que seu aplicativo aumenta seu uso, agrupe seus servidores de aplicativos e use os recursos de escalonamento automático para dimensionar automaticamente horizontalmente seu aplicativo à medida que a demanda aumenta.

Trate sua infraestrutura como código.

A automação não se aplica apenas a servidores e deve se estender à sua infraestrutura sempre que possível. Implante sua infraestrutura (servidores ou não) usando gerenciadores de implantação como Terraform, AWS CloudFormation e Google Cloud Deployment Manager.

Utilize recipientes.

Para cargas de trabalho mais sofisticadas ou pesadas com dependências instaladas complexas, considere o uso de contêineres em um único servidor ou Kubernetes.

Nosso conselho

Aqui estão algumas considerações importantes para ajudá-lo a determinar se o serverless é ideal para você:

Tamanho e complexidade da carga de trabalho

Serverless funciona melhor para tarefas e cargas de trabalho simples e discretas. Não é tão adequado se você tiver muitas peças móveis ou precisar de muita potência de computação ou memória. Nesse caso, considere o uso de contêineres e uma estrutura de orquestração de fluxo de trabalho de contêiner, como o Kubernetes.

Frequência e duração da execução

Quantas solicitações por segundo seu aplicativo sem servidor processará? Quanto tempo cada solicitação levará para ser processada? As plataformas sem servidor em nuvem têm limites de frequência de execução, simultaneidade e duração. Se seu aplicativo não pode funcionar perfeitamente dentro desses limites, é hora de considerar uma abordagem orientada a contêineres.

Solicitações e networking

As plataformas sem servidor geralmente utilizam alguma forma de rede simplificada e não oferecem suporte a todos os recursos de rede virtual em nuvem, como VPCs e firewalls.

Linguagem

Que linguagem você costuma usar? Se não for um dos idiomas oficialmente suportados pela plataforma sem servidor, você deve considerar os contêineres.

Limitações de tempo de execução

As plataformas sem servidor não fornecem abstrações completas do sistema operacional. Em vez disso, você está limitado a uma imagem de tempo de execução específica.

Custo

As funções sem servidor são incrivelmente convenientes, mas potencialmente caras. Quando sua função sem servidor processa apenas alguns eventos, seus custos são baixos; os custos aumentam rapidamente à medida que a contagem de eventos aumenta. Esse cenário é uma fonte frequente de contas inesperadas da nuvem.

No final, a abstração tende a vencer. Sugerimos usar primeiro o serverless e depois os servidores - com contêineres e orquestração, se possível - depois que você superar as opções sem servidor.

Otimização, desempenho e as guerras de benchmark

Imagine que você é um bilionário comprando um novo transporte. Você reduziu sua escolha a duas opções:

- Jato Executivo 787
 - Alcance: 9.945 milhas náuticas (com 25 passageiros)
 - Velocidade máxima: 0,90 Mach
 - Velocidade de cruzeiro: 0,85 Mach
 - Capacidade de combustível: 101.323 quilos
 - Peso máximo de decolagem: 227.930 quilos
 - Empuxo máximo: 128.000 libras
- Plaid Tesla Model S
 - Alcance: 560 quilômetros
 - Velocidade máxima: 322 quilômetros/hora
 - 0-100 quilômetros/hora: 2,1 segundos
 - Capacidade da bateria: 100 quilowatts-hora
 - Tempo da volta em Nürburgring: 7 minutos, 30,9 segundos
 - Potência: 1020
 - Torque: 1050 lb-ft

Qual dessas opções oferece melhor desempenho? Você não precisa saber muito sobre carros ou aeronaves para reconhecer que essa é uma comparação idiota. Uma opção é um jato particular de fuselagem larga projetado para operação intercontinental, enquanto a outra é um supercarro elétrico.

Vemos essas comparações de maçãs com laranjas feitas o tempo todo no espaço do banco de dados. Os benchmarks comparam bancos de dados otimizados para casos de uso completamente diferentes ou usam cenários de teste que não têm nenhuma semelhança com as necessidades do mundo real.

Recentemente, vimos uma nova rodada de guerras de benchmark surgir entre os principais fornecedores no espaço de dados. Aplaudimos os benchmarks e estamos felizes em ver muitos fornecedores de banco de dados finalmente retirando as cláusulas DeWitt de seus contratos com clientes.¹³ Mesmo assim, deixe o comprador tomar cuidado: o espaço de dados está cheio de benchmarks sem sentido.¹⁴ Aqui estão alguns truques comuns usados para colocar um polegar na escala de referência.

Big Data... para a década de 1990

Os produtos que afirmam oferecer suporte a “big data” em escala de petabytes geralmente usam conjuntos de dados de referência pequenos o suficiente para caber facilmente no armazenamento de seu smartphone. Para sistemas que dependem de camadas de cache para fornecer desempenho, os conjuntos de dados de teste residem totalmente na unidade de estado sólido (SSD) ou na memória, e os benchmarks podem mostrar desempenho ultra-alto consultando repetidamente os mesmos dados. Um pequeno conjunto de dados de teste também minimiza os custos de RAM e SSD ao comparar preços.

Para comparar os casos de uso do mundo real, você deve simular os dados do mundo real antecipados e o tamanho da consulta. Avalie o desempenho da consulta e os custos de recursos com base em uma avaliação detalhada de suas necessidades.

Comparações de custos absurdas

Comparações de custo sem sentido são um truque padrão ao analisar preço/desempenho ou TCO. Por exemplo, muitos sistemas MPP não podem ser criados e excluídos imediatamente, mesmo quando residem em um ambiente de nuvem; esses sistemas funcionam por anos a fio depois de configurados. Outros bancos de dados suportam um modelo de computação dinâmico e cobram por consulta ou por segundo de uso. Comparar sistemas efêmeros e não efêmeros com base no custo por segundo não faz sentido, mas vemos isso o tempo todo em benchmarks.

Otimização assimétrica

O engano da otimização assimétrica aparece de várias formas, mas aqui está um exemplo. Freqüentemente, um fornecedor compara um sistema MPP baseado em linha com um banco de dados colunar usando um benchmark que executa consultas de junção complexas em dados altamente normalizados. O modelo de dados normalizado é ideal para o sistema baseado em linhas, mas o modelo colunar

13 Justin Olsson e Reynold Xin, “Eliminating the Anti-competitive DeWitt Clause for Database Benchmarking,” Databricks, 8 de novembro de 2021, <https://oreil.ly/3lFOE>.

14 Para um clássico do gênero, veja William McKnight e Jake Dolezal, “Data Warehouse in the Cloud Benchmarks,” GigaOm, 7 de fevereiro de 2019, <https://oreil.ly/QjCmA>.

sistema realizaria todo o seu potencial apenas com algumas mudanças de esquema. Para piorar a situação, os fornecedores aprimoram seus sistemas com uma injeção extra de otimização de junção (por exemplo, pré-indexação de junções) sem aplicar ajustes comparáveis no banco de dados concorrente (por exemplo, colocar junções em uma visualização materializada).

Caveat Emptor

Como acontece com todas as coisas em tecnologia de dados, deixe o comprador tomar cuidado. Faça sua lição de casa antes de confiar cegamente em benchmarks de fornecedores para avaliar e escolher a tecnologia.

Subcorrentes e seus impactos na escolha de tecnologias

Como visto neste capítulo, um engenheiro de dados tem muito a considerar ao avaliar tecnologias. Seja qual for a tecnologia que você escolher, certifique-se de entender como ela oferece suporte às correntes ocultas do ciclo de vida da engenharia de dados. Vamos revisá-los brevemente novamente.

Gestão de dados

O gerenciamento de dados é uma área ampla e, no que diz respeito às tecnologias, nem sempre é claro se uma tecnologia adota o gerenciamento de dados como preocupação principal. Por exemplo, nos bastidores, um fornecedor terceirizado pode usar as melhores práticas de gerenciamento de dados – conformidade regulatória, segurança, privacidade, qualidade de dados e governança – mas ocultar esses detalhes por trás de uma camada de interface do usuário limitada. Nesse caso, ao avaliar o produto, ajuda perguntar à empresa sobre suas práticas de gerenciamento de dados. Aqui estão alguns exemplos de perguntas que você deve fazer:

- Como você está protegendo os dados contra violações, tanto externas quanto internas?
- Qual é a conformidade do seu produto com GDPR, CCPA e outros regulamentos de privacidade de dados?
- Você permite que eu hospede meus dados para cumprir esses regulamentos?
- Como você garante a qualidade dos dados e se estou visualizando os dados corretos em sua solução?

Há muitas outras perguntas a serem feitas, e essas são apenas algumas das maneiras de pensar sobre o gerenciamento de dados no que se refere à escolha das tecnologias certas. Essas mesmas perguntas também devem se aplicar às soluções OSS que você está considerando.

DataOps

Problemas acontecerão. Eles simplesmente vão. Um servidor ou banco de dados pode morrer, uma região de nuvem pode ter uma interrupção, você pode implantar código com bugs, dados incorretos podem ser introduzidos em seu data warehouse e outros problemas imprevistos podem ocorrer.

Ao avaliar uma nova tecnologia, quanto controle você tem sobre a implantação do novo código, como você será alertado se houver um problema e como reagirá quando houver um problema? A resposta depende muito do tipo de tecnologia que você está considerando. Se a tecnologia for OSS, você provavelmente será responsável por configurar o monitoramento, hospedagem e implantação de código. Como você lidará com os problemas? Qual é a sua resposta ao incidente?

Muitas das operações estão fora de seu controle se você estiver usando uma oferta gerenciada. Considere o SLA do fornecedor, a forma como ele o alerta sobre os problemas e se eles são transparentes sobre como estão lidando com o caso, incluindo o fornecimento de um ETA para uma correção.

Arquitetura de dados

Conforme discutido em [Capítulo 3](#), uma boa arquitetura de dados significa avaliar compensações e escolher as melhores ferramentas para o trabalho, mantendo suas decisões reversíveis. Com o cenário de dados se transformando em alta velocidade, *o melhor ferramenta* depois o trabalho é um alvo em movimento. Os principais objetivos são evitar bloqueios desnecessários, garantir a interoperabilidade na pilha de dados e produzir alto ROI. Escolha suas tecnologias de acordo.

Exemplo de Orquestração: Airflow

Durante a maior parte deste capítulo, evitamos ativamente discutir qualquer tecnologia em particular de forma muito extensa. Abrimos uma exceção para orquestração porque o espaço é atualmente dominado por uma tecnologia de código aberto, o Apache Airflow.

Maxime Beauchemin deu início ao projeto Airflow no Airbnb em 2014. O Airflow foi desenvolvido desde o início como um projeto de código aberto não comercial. A estrutura rapidamente cresceu significativamente fora do Airbnb, tornando-se um projeto da Apache Incubator em 2016 e um projeto totalmente patrocinado pela Apache em 2019.

O Airflow possui muitas vantagens, principalmente por causa de sua posição dominante no mercado de código aberto. Primeiro, o projeto de código aberto Airflow é extremamente ativo, com uma alta taxa de confirmações e um tempo de resposta rápido para bugs e problemas de segurança, e o projeto lançou recentemente o Airflow 2, um importante refatorador da base de código. Em segundo lugar, o Airflow desfruta de um enorme reconhecimento. O Airflow tem uma comunidade vibrante e ativa em muitas plataformas de comunicação, incluindo Slack, Stack Overflow e GitHub. Os usuários podem facilmente encontrar respostas para perguntas e problemas. Em terceiro lugar, o Airflow está disponível comercialmente como um serviço gerenciado ou distribuição de software por meio de muitos fornecedores, incluindo GCP, AWS e Astronomer.io.

O fluxo de ar também tem algumas desvantagens. O Airflow depende de alguns componentes principais não escaláveis (o agendador e o banco de dados de back-end) que podem se tornar gargalos para desempenho, escala e confiabilidade; as partes escaláveis do Airflow ainda seguem um padrão monolítico distribuído. (Ver “[Monólito Versus Modular](#)” na [página 139](#).) Finalmente, fluxo de ar

carece de suporte para muitas construções nativas de dados, como gerenciamento de esquema, linhagem e catalogação; e é desafiador desenvolver e testar fluxos de trabalho do Airflow.

Não tentamos uma discussão exaustiva das alternativas do Airflow aqui, mas apenas mencionamos alguns dos principais candidatos à orquestração no momento em que escrevemos. Prefect e Dagster visam resolver alguns dos problemas discutidos anteriormente, repensando os componentes da arquitetura do Airflow. Haverá outras estruturas de orquestração e tecnologias não discutidas aqui? Planeje isso.

Recomendamos enfaticamente que qualquer pessoa que escolha uma tecnologia de orquestração estude as opções discutidas aqui. Eles também devem se familiarizar com a atividade no espaço, pois novos desenvolvimentos certamente ocorrerão quando você ler isso.

Engenharia de software

Como engenheiro de dados, você deve buscar simplificação e abstração em toda a pilha de dados. Compre ou use soluções de código aberto pré-construídas sempre que possível. Eliminar o levantamento de peso indiferenciado deve ser seu grande objetivo. Concentre seus recursos – codificação personalizada e ferramentas – em áreas que lhe dão uma sólida vantagem competitiva. Por exemplo, codificar manualmente uma conexão de banco de dados entre seu banco de dados de produção e seu data warehouse na nuvem é uma vantagem competitiva para você? Provavelmente não. Este é um problema muito resolvido. Em vez disso, escolha uma solução pronta para uso (código aberto ou SaaS gerenciado). O mundo não precisa do milionésimo +1 conector de data warehouse de banco de dados para nuvem.

Por outro lado, por que os clientes compram de você? Sua empresa provavelmente tem algo especial na maneira como faz as coisas. Talvez seja um algoritmo específico que alimenta sua plataforma fintech. Ao abstrair muitos fluxos de trabalho e processos redundantes, você pode continuar eliminando, refinando e personalizando as coisas que movem a agulha para os negócios.

Conclusão

Escolher as tecnologias certas não é tarefa fácil, especialmente quando novas tecnologias e padrões surgem diariamente. Hoje é possivelmente o momento mais confuso da história para avaliar e selecionar tecnologias. A escolha de tecnologias é um equilíbrio entre caso de uso, custo, construção versus compra e modularização. Sempre aborde a tecnologia da mesma forma que a arquitetura: avalie os trade-offs e busque decisões reversíveis.

Recursos adicionais

- *Nuvem FinOps*por JR Storment e Mike Fuller (O'Reilly)
- “*Infraestrutura de nuvem: o guia definitivo para iniciantes*”por Matthew Smith

- “O custo da nuvem, um paradoxo de trilhões de dólares” por Sarah Wang e Martin Casado
- Fundação FinOps [Página da web “O que é FinOps”](#)
- “Red Hot: O cenário de aprendizado de máquina, IA e dados (MAD) de 2021” por Matt Turck
- Dados Ternários “O que vem a seguir para bancos de dados analíticos? com Jordan Tigani (Mãe-Pato)” vídeo
- “A promessa não cumprida de sem servidor” por Corey Quinn
- “O que é a pilha de dados moderna?” por Charles Wang

PARTE II

A Engenharia de Dados

Ciclo de vida em profundidade

Geração de dados em sistemas de origem

Bem-vindo ao primeiro estágio do ciclo de vida da engenharia de dados: geração de dados em sistemas de origem. Como descrevemos anteriormente, o trabalho de um engenheiro de dados é pegar dados de sistemas de origem, fazer algo com eles e torná-los úteis para servir a casos de uso downstream. Mas antes de obter dados brutos, você deve entender onde os dados existem, como são gerados e suas características e peculiaridades.

Este capítulo aborda alguns padrões de sistemas de origem operacional populares e os tipos significativos de sistemas de origem. Existem muitos sistemas de origem para geração de dados e não estamos cobrindo todos eles exaustivamente. Consideraremos os dados gerados por esses sistemas e as coisas que você deve considerar ao trabalhar com sistemas de origem. Também discutimos como as subcorrentes da engenharia de dados se aplicam a esta primeira fase do ciclo de vida da engenharia de dados (Figura 5-1).

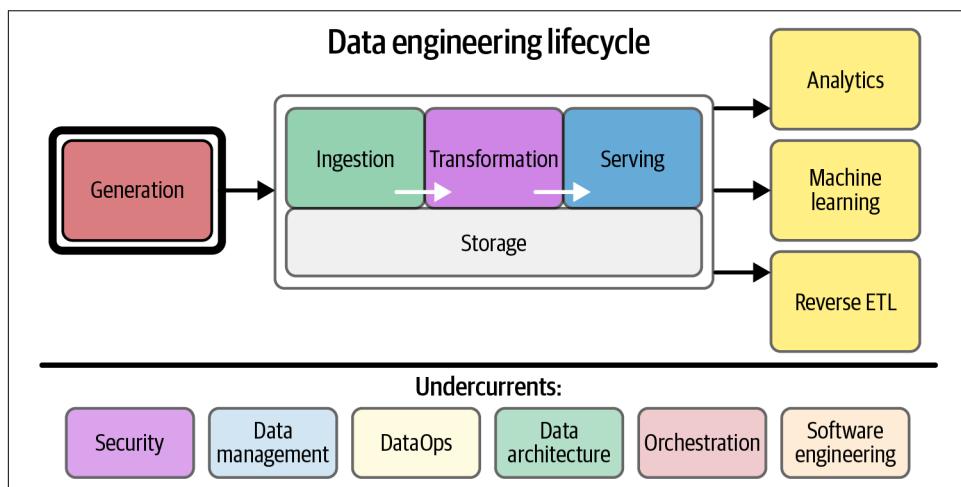


Figura 5-1. Os sistemas de origem geram os dados para o restante do ciclo de vida da engenharia de dados

À medida que os dados proliferam, especialmente com o aumento do compartilhamento de dados (discutido a seguir), esperamos que a função de um engenheiro de dados mude fortemente para entender a interação entre fontes e destinos de dados. As tarefas básicas de encanamento da engenharia de dados – mover dados de A para B – serão simplificadas drasticamente. Por outro lado, continuará sendo fundamental entender a natureza dos dados conforme são criados nos sistemas de origem.

Fontes de dados: como os dados são criados?

Conforme você aprende sobre os vários padrões operacionais subjacentes dos sistemas que geram dados, é essencial entender como os dados são criados. Os dados são uma coleção desorganizada e sem contexto de fatos e números. Pode ser criado de várias maneiras, tanto analógicas quanto digitais.

Dados analógicos criação ocorre no mundo real, como fala vocal, linguagem de sinais, escrita no papel ou tocar um instrumento. Esses dados analógicos geralmente são transitórios; com que frequência você teve uma conversa verbal cujo conteúdo se perdeu no éter após o término da conversa?

Dados digitais é criado pela conversão de dados analógicos em formato digital ou é o produto nativo de um sistema digital. Um exemplo de analógico para digital é um aplicativo de mensagens de texto móvel que converte fala analógica em texto digital. Um exemplo de criação de dados digitais é uma transação de cartão de crédito em uma plataforma de comércio eletrônico. Um cliente faz um pedido, a transação é cobrada em seu cartão de crédito e as informações da transação são salvas em vários bancos de dados.

Utilizaremos alguns exemplos comuns neste capítulo, como dados criados ao interagir com um site ou aplicativo móvel. Mas, na verdade, os dados estão em todo o mundo ao nosso redor. Capturamos dados de dispositivos IoT, terminais de cartão de crédito, sensores de telescópio, negociações de ações e muito mais.

Familiarize-se com seu sistema de origem e como ele gera dados. Esforce-se para ler a documentação do sistema de origem e entender seus padrões e peculiaridades. Se seu sistema de origem for um RDBMS, aprenda como ele opera (gravações, commits, consultas, etc.); aprenda os meandros do sistema de origem que podem afetar sua capacidade de ingerir dele.

Sistemas de Origem: Ideias Principais

Os sistemas de origem produzem dados de várias maneiras. Esta seção discute as principais ideias que você encontrará frequentemente ao trabalhar com sistemas de origem.

Arquivos e dados não estruturados

Arquivo é uma sequência de bytes, normalmente armazenada em um disco. Os aplicativos geralmente gravam dados em arquivos. Os arquivos podem armazenar parâmetros locais, eventos, logs, imagens e áudio.

Além disso, os arquivos são um meio universal de troca de dados. Por mais que os engenheiros de dados desejem obter dados programaticamente, grande parte do mundo ainda envia e recebe arquivos. Por exemplo, se você estiver obtendo dados de uma agência governamental, há uma excelente chance de baixar os dados como um arquivo Excel ou CSV ou receber o arquivo por e-mail.

Os principais tipos de formato de arquivo de origem que você encontrará como engenheiro de dados - arquivos originados manualmente ou como uma saída de um processo do sistema de origem - são Excel, CSV, TXT, JSON e XML. Esses arquivos têm suas peculiaridades e podem ser estruturados (Excel, CSV), semiestruturados (JSON, XML, CSV) ou não estruturados (TXT, CSV). Embora você use determinados formatos fortemente como engenheiro de dados (como Parquet, ORC e Avro), abordaremos esses formatos mais tarde e colocaremos o foco aqui nos arquivos de sistema de origem. [Capítulo 6](#) cobre os detalhes técnicos dos arquivos.

APIs

Interfaces de programação de aplicativos(APIs) são uma forma padrão de troca de dados entre sistemas. Em teoria, as APIs simplificam a tarefa de ingestão de dados para engenheiros de dados. Na prática, muitas APIs ainda expõem uma grande complexidade de dados para os engenheiros gerenciarem. Mesmo com o surgimento de vários serviços e estruturas e serviços para automatizar a ingestão de dados de API, os engenheiros de dados geralmente devem investir bastante energia na manutenção de conexões de API personalizadas. Discutimos APIs com mais detalhes posteriormente neste capítulo.

Bancos de dados de aplicativos (sistemas OLTP)

Um banco de dados do aplicativo armazena o estado de um aplicativo. Um exemplo padrão é um banco de dados que armazena saldos de contas bancárias. À medida que as transações e os pagamentos dos clientes acontecem, o aplicativo atualiza os saldos das contas bancárias.

Normalmente, um banco de dados de aplicativo é um *processamento de transações on-line*(OLTP) systemum banco de dados que lê e grava registros de dados individuais em uma taxa alta. Os sistemas OLTP são muitas vezes referidos como *bancos de dados transacionais*, mas isso não implica necessariamente que o sistema em questão suporta *transações atômicas*.

De forma mais geral, os bancos de dados OLTP suportam baixa latência e alta simultaneidade. Um banco de dados RDBMS pode selecionar ou atualizar uma linha em menos de um milissegundo (sem levar em conta a latência da rede) e lidar com milhares de leituras e gravações por segundo. Um cluster de banco de dados de documentos pode gerenciar taxas de confirmação de documentos ainda mais altas às custas de possíveis inconsistências. Alguns bancos de dados gráficos também podem lidar com casos de uso transacionais.

Fundamentalmente, os bancos de dados OLTP funcionam bem como back-ends de aplicativos quando milhares ou mesmo milhões de usuários podem estar interagindo com o aplicativo simultaneamente,

atualizando e gravando dados simultaneamente. Os sistemas OLTP são menos adequados para casos de uso orientados por análises em escala, em que uma única consulta deve verificar uma grande quantidade de dados.

ÁCIDO

O suporte para transações atômicas faz parte de um conjunto crítico de características de banco de dados conhecidas juntas como ACID (como você deve se lembrar de [Capítulo 3](#), isso significa *atomicidade, consistência, isolamento, durabilidade*). *Consistência* significa que qualquer leitura de banco de dados retornará a última versão escrita do item recuperado. *Isolamento* implica que, se duas atualizações estiverem em andamento simultaneamente para a mesma coisa, o estado final do banco de dados será consistente com a execução sequencial dessas atualizações na ordem em que foram enviadas. *Durabilidade* indica que os dados confirmados nunca serão perdidos, mesmo em caso de perda de energia.

Observe que as características do ACID não são necessárias para oferecer suporte a back-ends de aplicativos, e relaxar essas restrições pode ser um benefício considerável para o desempenho e a escala. No entanto, as características do ACID garantem que o banco de dados manterá uma imagem consistente do mundo, simplificando drasticamente a tarefa do desenvolvedor do aplicativo.

Todos os engenheiros (dados ou não) devem entender a operação com e sem ACID. Por exemplo, para melhorar o desempenho, alguns bancos de dados distribuídos usam restrições de consistência relaxada, como *eventual consistência*, para melhorar o desempenho. Entender o modelo de consistência com o qual você está trabalhando ajuda a evitar desastres.

transações atômicas

Uma *transação atômica* é um conjunto de várias alterações que são confirmadas como uma unidade. No exemplo em [Figura 5-2](#), um aplicativo bancário tradicional em execução em um RDBMS executa uma instrução SQL que verifica os saldos de duas contas, uma na Conta A (a origem) e outra na Conta B (o destino). O dinheiro é então movido da Conta A para a Conta B se houver fundos suficientes na Conta A. A transação inteira deve ser executada com atualizações nos saldos de ambas as contas ou falha sem atualizar o saldo de nenhuma das contas. Ou seja, toda a operação deve acontecer como uma *transação*.

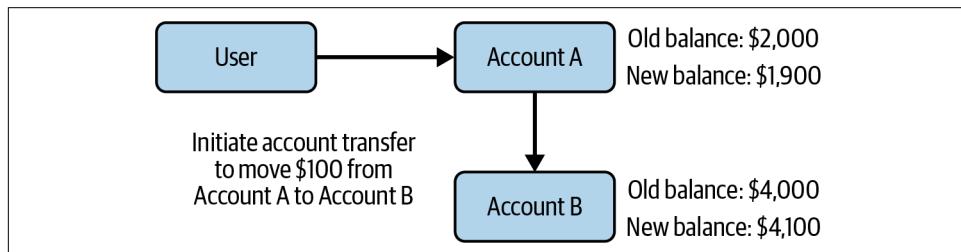


Figura 5-2. Exemplo de uma transação atômica: uma transferência de conta bancária usando OLTP

OLTP e análise

Frequentemente, pequenas empresas executam análises diretamente em um OLTP. Esse padrão funciona a curto prazo, mas não é escalável. Em algum momento, a execução de consultas analíticas no OLTP gera problemas de desempenho devido a limitações estruturais do OLTP ou contenção de recursos com cargas de trabalho transacionais concorrentes. Os engenheiros de dados devem entender o funcionamento interno do OLTP e dos back-ends de aplicativos para configurar integrações apropriadas com sistemas analíticos sem degradar o desempenho do aplicativo de produção.

À medida que as empresas oferecem mais recursos analíticos em aplicativos SaaS, a necessidade de recursos híbridos — atualizações rápidas com recursos analíticos combinados — criou novos desafios para os engenheiros de dados. Usaremos o termo *aplicativo de dados* para se referir a aplicativos que hibridizam cargas de trabalho transacionais e analíticas.

Sistema de Processamento Analítico Online

Ao contrário de um sistema OLTP, um *processamento analítico online* (OLAP) é construído para executar grandes consultas analíticas e normalmente é ineficiente ao lidar com pesquisas de registros individuais. Por exemplo, bancos de dados de colunas modernos são otimizados para escanear grandes volumes de dados, dispensando índices para melhorar a escalabilidade e o desempenho de escaneamento. Qualquer consulta geralmente envolve a verificação de um bloco de dados mínimo, geralmente de 100 MB ou mais. Tentar procurar milhares de itens individuais por segundo em tal sistema o deixará de joelhos, a menos que seja combinado com uma camada de cache projetada para este caso de uso.

Observe que estamos usando o termo *OLAP* para se referir a qualquer sistema de banco de dados que suporte consultas analíticas interativas de alta escala; não estamos nos limitando a sistemas que suportam cubos OLAP (arrays multidimensionais de dados). O *on-line* parte do OLAP implica que o sistema escuta constantemente as consultas recebidas, tornando os sistemas OLAP adequados para análises interativas.

Embora este capítulo cubra os sistemas de origem, os OLAPs geralmente são sistemas de armazenamento e consulta para análises. Por que estamos falando sobre eles em nosso capítulo sobre sistemas de origem? Em casos de uso prático, os engenheiros geralmente precisam ler dados de um sistema OLAP. Por exemplo, um data warehouse pode fornecer dados usados para treinar um modelo de ML. Ou, um sistema OLAP pode servir a um fluxo de trabalho ETL reverso, onde os dados derivados em um sistema analítico são enviados de volta para um sistema de origem, como um CRM, plataforma SaaS ou aplicativo transacional.

Alterar captura de dados

Alterar captura de dados (CDC) é um método para extrair cada evento de alteração (inserir, atualizar, excluir) que ocorre em um banco de dados. O CDC é frequentemente aproveitado para replicar entre bancos de dados quase em tempo real ou criar um fluxo de eventos para processamento downstream.

O CDC é tratado de forma diferente, dependendo da tecnologia do banco de dados. Bancos de dados relacionais geralmente geram um log de eventos armazenado diretamente no servidor de banco de dados que pode ser processado para criar um fluxo. (Ver “[Logs do banco de dados](#)” na página 161.) Muitos bancos de dados NoSQL em nuvem podem enviar um log ou fluxo de eventos para um local de armazenamento de destino.

Histórico

A *registro* captura informações sobre eventos que ocorrem nos sistemas. Por exemplo, um log pode capturar padrões de tráfego e uso em um servidor web. O sistema operacional do seu computador desktop (Windows, macOS, Linux) registra eventos quando o sistema inicializa e quando os aplicativos iniciam ou travam, por exemplo.

Os logs são uma fonte de dados rica, potencialmente valiosa para análise de dados downstream, ML e automação. Aqui estão algumas fontes familiares de logs:

- Sistemas operacionais
- Formulários
- Servidores
- Containers
- Redes
- dispositivos IoT

Todos os logs rastreiam eventos e metadados de eventos. No mínimo, um log deve capturar quem, o quê e quando:

Quem

O humano, sistema ou conta de serviço associado ao evento (por exemplo, um agente de usuário do navegador da web ou um ID de usuário)

O que aconteceu

O evento e os metadados relacionados

Quando

O timestamp do evento

Codificação de registro

Os logs são codificados de algumas maneiras:

Registros codificados em binário

Eles codificam dados em um formato compacto personalizado para economia de espaço e E/S rápida. Logs de banco de dados, discutidos em “[Logs do banco de dados](#)” na página 161, são um exemplo padrão.

Registros semiestruturados

Eles são codificados como texto em um formato de serialização de objeto (JSON, na maioria das vezes). Logs semiestruturados são legíveis por máquina e portáteis. No entanto, eles são muito menos eficientes do que os logs binários. E embora sejam nominalmente legíveis por máquina, extraír valor deles geralmente requer um código personalizado significativo.

Logs de texto simples (não estruturados)

Eles basicamente armazenam a saída do console do software. Como tal, não existem padrões de uso geral. Esses logs podem fornecer informações úteis para cientistas de dados e engenheiros de ML, embora a extração de informações úteis dos dados de texto brutos possa ser complicada.

resolução de registro

Os logs são criados em várias resoluções e níveis de log. O registro *resolução* refere-se à quantidade de dados de eventos capturados em um log. Por exemplo, os logs do banco de dados capturam informações suficientes dos eventos do banco de dados para permitir a reconstrução do estado do banco de dados a qualquer momento.

Por outro lado, capturar todas as alterações de dados em logs para um sistema de big data geralmente não é prático. Em vez disso, esses logs podem observar apenas que um determinado tipo de evento de confirmação ocorreu. O nível de registro refere-se às condições necessárias para registrar uma entrada de log, especificamente em relação a erros e depuração. O software geralmente é configurável para registrar todos os eventos ou registrar apenas erros, por exemplo.

Latência de registro: Lote ou tempo real

Os logs de lote geralmente são gravados continuamente em um arquivo. Entradas de log individuais podem ser gravadas em um sistema de mensagens como Kafka ou Pulsar para aplicativos em tempo real.

Registros do banco de dados

Registros do banco de dados são essenciais o suficiente para merecerem uma cobertura mais detalhada. Logs write-ahead — normalmente, arquivos binários armazenados em um formato nativo de banco de dados específico — desempenham um papel crucial nas garantias e capacidade de recuperação do banco de dados. O servidor de banco de dados recebe solicitações de gravação e atualização para uma tabela de banco de dados (consulte [Figura 5-3](#)), armazenando cada operação no log antes de confirmar a solicitação. A confirmação vem com uma garantia associada ao log: mesmo que o servidor falhe, ele pode recuperar seu estado na reinicialização completando o trabalho inacabado dos logs.

Os logs do banco de dados são extremamente úteis na engenharia de dados, especialmente para o CDC gerar fluxos de eventos a partir de alterações no banco de dados.

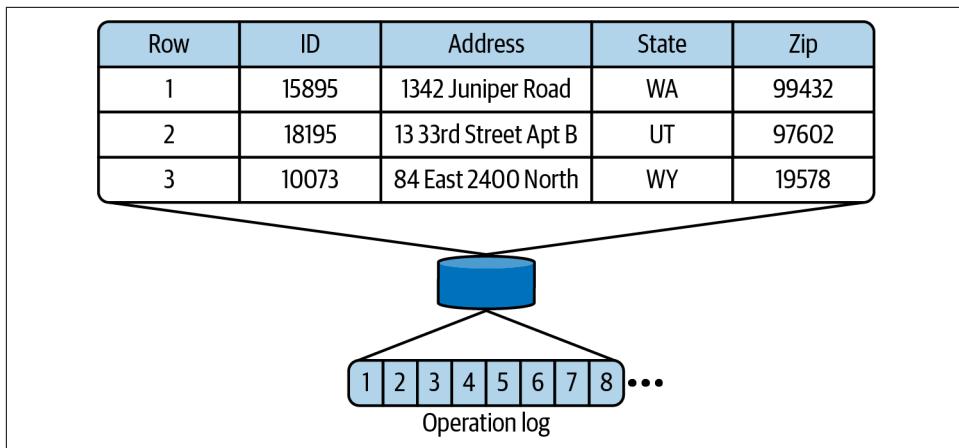


Figura 5-3. Operações de registro de logs de banco de dados em uma tabela

CRUD

CRUD, que significa *criar, ler, atualizar, e excluir*, é um padrão transacional comumente usado em programação e representa as quatro operações básicas de armazenamento persistente. CRUD é o padrão mais comum para armazenar o estado do aplicativo em um banco de dados. Um princípio básico do CRUD é que os dados devem ser criados antes de serem usados. Depois que os dados foram criados, os dados podem ser lidos e atualizados. Finalmente, os dados podem precisar ser destruídos. O CRUD garante que essas quatro operações ocorrerão nos dados, independentemente de seu armazenamento.

O CRUD é um padrão amplamente usado em aplicativos de software, e você normalmente encontrará o CRUD usado em APIs e bancos de dados. Por exemplo, um aplicativo da Web fará uso intenso de CRUD para solicitações HTTP RESTful e armazenará e recuperará dados de um banco de dados.

Como em qualquer banco de dados, podemos usar a extração baseada em instantâneo para obter dados de um banco de dados onde nosso aplicativo aplica operações CRUD. Por outro lado, a extração de eventos com CDC nos fornece um histórico completo das operações e potencialmente permite análises quase em tempo real.

Somente Inserção

O padrão somente de inserção tem o histórico diretamente em uma tabela contendo dados. Em vez de atualizar registros, novos registros são inseridos com um carimbo de data/hora indicando quando foram criados (Tabela 5-1). Por exemplo, suponha que você tenha uma tabela de endereços de clientes. Seguindo um padrão CRUD, você simplesmente atualizaria o registro caso o cliente mudasse de endereço. Com o padrão somente inserção, um novo registro de endereço é inserido com o mesmo ID do cliente. Para ler o endereço do cliente atual por ID do cliente, procure o registro mais recente sob esse ID.

Tabela 5-1. Um padrão somente de inserção produz várias versões de um registro

ID do registro	Valor	carimbo de data/hora
1	40	2021-09-19T00:10:23+00:00
1	51	2021-09-30T00:12:00+00:00

De certo modo, o padrão somente de inserção mantém um log do banco de dados diretamente na própria tabela, tornando-o especialmente útil se o aplicativo precisar de acesso ao histórico. Por exemplo, o padrão somente de inserção funcionaria bem para um aplicativo bancário projetado para apresentar o histórico de endereço do cliente.

Um padrão somente de inserção de analítica separado geralmente é usado com tabelas de aplicativos CRUD regulares. No padrão ETL somente de inserção, os pipelines de dados inserem um novo registro na tabela analítica de destino sempre que ocorre uma atualização na tabela CRUD.

Inserir apenas tem algumas desvantagens. Primeiro, as tabelas podem crescer bastante, especialmente se os dados forem alterados com frequência, pois cada alteração é inserida na tabela. Às vezes, os registros são limpos com base em uma data de expiração do registro ou em um número máximo de versões de registro para manter o tamanho da tabela razoável. A segunda desvantagem é que as pesquisas de registro incorrem em sobrecarga extra porque a pesquisa do estado atual envolve a execução MAX (criado_timestamp). Se centenas ou milhares de registros estiverem em um único ID, essa operação de pesquisa será cara de executar.

Mensagens e fluxos

Relacionados à arquitetura orientada a eventos, dois termos que você frequentemente verá usados de forma intercambiável são *fila de mensagens* e *plataforma de streaming*, mas existe uma diferença sutil, mas essencial, entre os dois. Vale a pena definir e contrastar esses termos, pois eles abrangem muitas grandes ideias relacionadas a sistemas de origem, práticas e tecnologias que abrangem todo o ciclo de vida da engenharia de dados.

A *mensagem* é dados brutos comunicados em dois ou mais sistemas ([Figura 5-4](#)). Por exemplo, temos o Sistema 1 e o Sistema 2, onde o Sistema 1 envia uma mensagem para o Sistema 2. Esses sistemas podem ser diferentes microsserviços, um servidor enviando uma mensagem para uma função serverless etc. A *fila de mensagens* é um editor para um consumidor e, uma vez entregue, a mensagem é removida da fila.

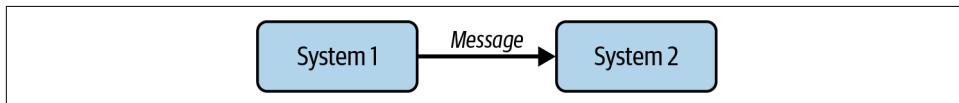


Figura 5-4. Uma mensagem passada entre dois sistemas

As mensagens são sinais discretos e singulares em um sistema orientado a eventos. Por exemplo, um dispositivo IoT pode enviar uma mensagem com a leitura de temperatura mais recente para uma mensagem

fila. Essa mensagem é então ingerida por um serviço que determina se o forno deve ser ligado ou desligado. Este serviço envia uma mensagem para um controlador de forno que executa a ação apropriada. Depois que a mensagem é recebida e a ação é executada, a mensagem é removida da fila de mensagens.

Em contraste, um *fluxo* é um log somente anexado de registros de eventos. (Streams são ingeridos e armazenados em *plataformas de streaming de eventos*, que discutimos mais detalhadamente em “[Mensagens e fluxos](#)” na página 163.) À medida que os eventos ocorrem, eles são acumulados em uma sequência ordenada ([Figura 5-5](#)); um carimbo de data/hora ou um ID pode ordenar eventos. (Observe que os eventos nem sempre são entregues na ordem exata devido às sutilezas dos sistemas distribuídos.)

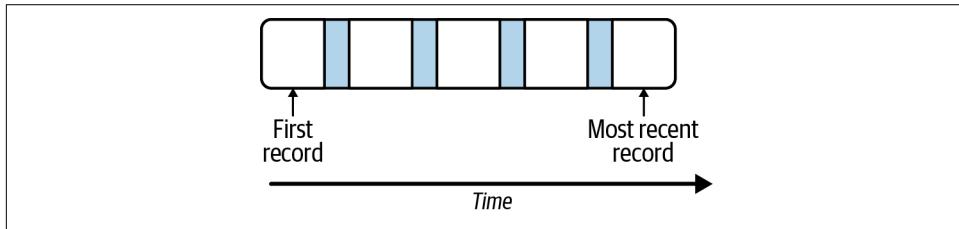


Figura 5-5. Um fluxo, que é um log de registros somente anexado ordenado

Você usará streams quando se importar com o que aconteceu em muitos eventos. Devido à natureza somente de anexação dos fluxos, os registros em um fluxo persistem por uma longa janela de retenção - geralmente semanas ou meses - permitindo operações complexas em registros, como agregações em vários registros ou a capacidade de retroceder para um ponto no tempo dentro do fluxo.

Vale a pena notar que sistemas que processam streams podem processar mensagens, e plataformas de streaming são freqüentemente usadas para passagem de mensagens. Muitas vezes, acumulamos mensagens em fluxos quando queremos realizar análises de mensagens. Em nosso exemplo de IoT, as leituras de temperatura que acionam o forno para ligar ou desligar também podem ser analisadas posteriormente para determinar estatísticas e tendências de temperatura.

Tipos de Tempo

Embora o tempo seja uma consideração essencial para toda a ingestão de dados, ele se torna muito mais crítico e útil no contexto de streaming, onde vemos os dados como contínuos e esperamos consumi-los logo após serem produzidos. Vejamos os principais tipos de tempo que você encontrará ao ingerir dados: a hora em que o evento é gerado, quando é ingerido e processado e quanto tempo levou o processamento ([Figura 5-6](#)).

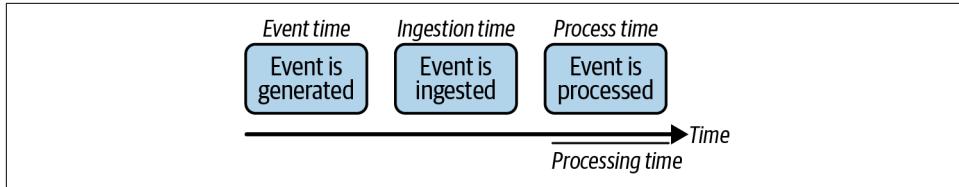


Figura 5-6. Evento, ingestão, processo e tempo de processamento

Hora do evento indica quando um evento é gerado em um sistema de origem, incluindo o timestamp do próprio evento original. Um intervalo de tempo indeterminado ocorrerá na criação do evento, antes que o evento seja ingerido e processado downstream. Sempre inclua carimbos de data/hora para cada fase pela qual um evento passa. Registre eventos à medida que ocorrem e em cada estágio do tempo — quando são criados, ingeridos e processados. Use esses logs de carimbo de data/hora para rastrear com precisão o movimento de seus dados por meio de seus pipelines de dados.

Depois que os dados são criados, eles são ingeridos em algum lugar. *tempo de ingestão* indica quando um evento é ingerido dos sistemas de origem para uma fila de mensagens, cache, memória, armazenamento de objetos, um banco de dados ou qualquer outro local em que os dados sejam armazenados (consulte [Capítulo 6](#)). Após a ingestão, os dados podem ser processados imediatamente; ou dentro de minutos, horas ou dias; ou simplesmente persistem no armazenamento indefinidamente.

Tempo de processamento ocorre após o tempo de ingestão, quando os dados são processados (normalmente, uma transformação). *Tempo de processamento* é quanto tempo os dados levaram para processar, medido em segundos, minutos, horas, etc.

Você vai querer registrar essas várias vezes, de preferência de forma automatizada. Configure o monitoramento ao longo de seus fluxos de trabalho de dados para capturar quando os eventos ocorrem, quando são ingeridos e processados e quanto tempo leva para processar eventos.

Detalhes práticos do sistema de origem

Esta seção discute os detalhes práticos da interação com sistemas de origem modernos. Vamos nos aprofundar nos detalhes de bancos de dados, APIs e outros aspectos comumente encontrados. Essas informações terão vida útil mais curta do que as principais ideias discutidas anteriormente; estruturas de API populares, bancos de dados e outros detalhes continuarão a mudar rapidamente.

No entanto, esses detalhes são conhecimentos críticos para engenheiros de dados em atividade. Sugerimos que você estude essas informações como conhecimento básico, mas leia bastante para ficar a par dos desenvolvimentos em andamento.

bancos de dados

Nesta seção, veremos tecnologias comuns de banco de dados de sistemas de origem que você encontrará como engenheiro de dados e considerações de alto nível para trabalhar com esses sistemas. Existem tantos tipos de bancos de dados quanto casos de uso para dados.

Principais considerações para entender as tecnologias de banco de dados

Aqui, apresentamos as principais ideias que ocorrem em uma variedade de tecnologias de banco de dados, incluindo aquelas que respaldam aplicativos de software e aquelas que oferecem suporte a casos de uso de análise:

Sistema de gerenciamento de banco de dados

Um sistema de banco de dados usado para armazenar e servir dados. Abreviado como DBMS, consiste em um mecanismo de armazenamento, otimizador de consulta, recuperação de desastres e outros componentes-chave para gerenciar o sistema de banco de dados.

Pesquisas

Como o banco de dados encontra e recupera dados? Os índices podem ajudar a acelerar as pesquisas, mas nem todos os bancos de dados possuem índices. Saiba se seu banco de dados usa índices; em caso afirmativo, quais são os melhores padrões para projetá-los e mantê-los? Entenda como aproveitar para uma extração eficiente. Também ajuda ter um conhecimento básico dos principais tipos de índices, incluindo árvore B e árvores de mesclagem estruturadas em log (LSM).

Otimizador de consultas

O banco de dados utiliza um otimizador? Quais são suas características?

Dimensionamento e distribuição

O banco de dados escala com a demanda? Qual estratégia de dimensionamento ela implementa? Ele escala horizontalmente (mais nós de banco de dados) ou verticalmente (mais recursos em uma única máquina)?

Padrões de modelagem

Quais padrões de modelagem funcionam melhor com o banco de dados (por exemplo, normalização de dados ou tabelas amplas)? (Ver [Capítulo 8](#) para nossa discussão sobre modelagem de dados.)

CRUD

Como os dados são consultados, criados, atualizados e excluídos no banco de dados? Cada tipo de banco de dados lida com operações CRUD de maneira diferente.

Consistência

O banco de dados é totalmente consistente ou suporta um modelo de consistência relaxado (por exemplo, consistência eventual)? O banco de dados oferece suporte a modos de consistência opcionais para leituras e gravações (por exemplo, leituras fortemente consistentes)?

Dividimos os bancos de dados em categorias relacionais e não relacionais. Na verdade, a categoria não relacional é muito mais diversificada, mas os bancos de dados relacionais ainda ocupam um espaço significativo nos backends de aplicativos.

Bancos de dados relacionais

As *sistemas de gerenciamento de banco de dados relacional* (RDBMS) é um dos back-ends de aplicativos mais comuns. Os bancos de dados relacionais foram desenvolvidos na IBM na década de 1970 e popularizados pela Oracle na década de 1980. O crescimento da Internet viu o surgimento da pilha LAMP (Linux, servidor web Apache, MySQL, PHP) e uma explosão de opções de fornecedores e RDBMS de código aberto. Mesmo com o surgimento dos bancos de dados NoSQL (descritos na seção a seguir), os bancos de dados relacionais permaneceram extremamente populares.

Os dados são armazenados em uma tabela de *relações* (linhas), e cada relação contém múltiplas *Campos* (colunas); ver [Figura 5-7](#). Note que usamos os termos *coluna* e *campo* indistintamente ao longo deste livro. Cada relação na tabela tem o mesmo *esquema* (uma sequência de colunas com tipos estáticos atribuídos, como string, integer ou float). As linhas são normalmente armazenadas como uma sequência contígua de bytes no disco.

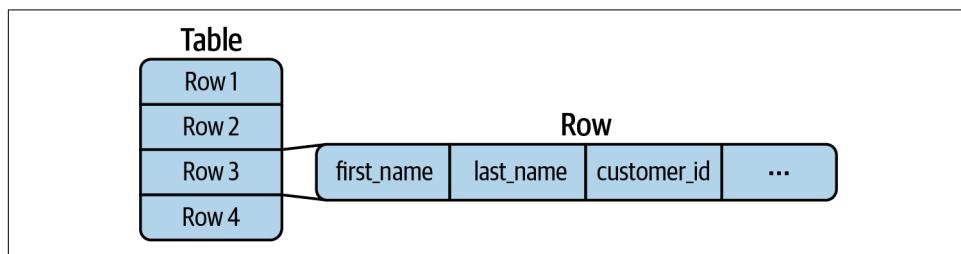


Figura 5-7. O RDBMS armazena e recupera dados em um nível de linha

As tabelas são normalmente indexadas por uma *chave primária*, um campo exclusivo para cada linha da tabela. A estratégia de indexação para a chave primária está intimamente ligada ao layout da tabela no disco.

As tabelas também podem ter vários *chaves estrangeiras*—campos com valores conectados com os valores de chaves primárias em outras tabelas, facilitando junções e permitindo esquemas complexos que espalham dados por várias tabelas. Em particular, é possível projetar um *esquema normalizado*. A normalização é uma estratégia para garantir que os dados nos registros não sejam duplicados em vários locais, evitando assim a necessidade de atualizar os estados em vários locais ao mesmo tempo e evitando inconsistências (consulte [Capítulo 8](#)).

Os sistemas RDBMS são normalmente compatíveis com ACID. A combinação de um esquema normalizado, conformidade com ACID e suporte para altas taxas de transação torna os sistemas de banco de dados relacionais ideais para armazenar estados de aplicativos que mudam rapidamente. O desafio para os engenheiros de dados é determinar como capturar informações de estado ao longo do tempo.

Uma discussão completa da teoria, história e tecnologia do RDBMS está além do escopo deste livro. Incentivamos você a estudar sistemas RDBMS, álgebra relacional e estratégias de normalização porque eles são amplamente difundidos e você os encontrará com frequência. Ver “[Recursos adicionais](#)” na [página 188](#) para livros sugeridos.

Bancos de dados não relacionais: NoSQL

Embora os bancos de dados relacionais sejam ótimos para muitos casos de uso, eles não são uma solução única para todos. Muitas vezes vemos que as pessoas começam com um banco de dados relacional com a impressão de que é um dispositivo universal e uma calçadeira em uma tonelada de casos de uso e cargas de trabalho. À medida que os requisitos de dados e consultas se transformam, o banco de dados relacional colapsa sob seu peso. Nesse ponto, você desejará usar um banco de dados apropriado para a carga de trabalho específica sob pressão. Insira bancos de dados não relacionais ou NoSQL. *NoSQL*, que significa *não só SQL*, refere-se a toda uma classe de bancos de dados que abandonam o paradigma relacional.

Por um lado, eliminar as restrições relacionais pode melhorar o desempenho, a escalabilidade e a flexibilidade do esquema. Mas, como sempre na arquitetura, existem compensações. Os bancos de dados NoSQL também geralmente abandonam várias características do RDBMS, como consistência forte, junções ou um esquema fixo.

Um grande tema deste livro é que a inovação de dados é constante. Vamos dar uma olhada rápida na história do NoSQL, pois é útil obter uma perspectiva sobre por que e como as inovações de dados afetam seu trabalho como engenheiro de dados. No início dos anos 2000, empresas de tecnologia como Google e Amazon começaram a superar seus bancos de dados relacionais e foram pioneiras em novos bancos de dados não relacionais distribuídos para dimensionar suas plataformas da web.

enquanto o termo *NoSQL* apareceu pela primeira vez em 1998, a versão moderna foi cunhada por Eric Evans nos anos 2000.¹ Ele conta a história em [um postagem no blog de 2009](#):

Passei os últimos dias em [nosqleaste](#) um dos tópicos quentes aqui é o nome “nosql”. Compreensivelmente, muitas pessoas se preocupam com o fato de o nome ser ruim, de enviar uma mensagem inadequada ou imprecisa. Embora eu não reivindique a ideia, tenho que aceitar alguma culpa pelo que está sendo chamado agora. Como é isso? Johan Oskarsson estava organizando o primeiro encontro e fez a pergunta “Qual é um bom nome?” no IRC; foi uma das três ou quatro sugestões que dei no espaço de 45 segundos, sem pensar.

Meu arrependimento, no entanto, não é sobre o que o nome diz; é sobre o que não faz. Quando Johan originalmente teve a ideia para o primeiro encontro, ele parecia estar pensando em Big Data e sistemas distribuídos linearmente escaláveis, mas o nome é tão vago que abriu a porta para falar de submissões para literalmente qualquer coisa que armazenasse dados, e não era um RDBMS.

¹ Keith D. Foote, “A Brief History of Non-Relational Databases,” Dataversity, 19 de junho de 2018, <https://oreil.ly/5Ukg2>.

O NoSQL permanece vago em 2022, mas foi amplamente adotado para descrever um universo de bancos de dados “nova escola”, alternativas aos bancos de dados relacionais.

Existem vários tipos de banco de dados NoSQL projetados para quase todos os casos de uso imagináveis. Como há muitos bancos de dados NoSQL para cobrir exaustivamente nesta seção, consideramos os seguintes tipos de banco de dados: valor-chave, documento, coluna larga, gráfico, pesquisa e série temporal. Esses bancos de dados são muito populares e têm ampla adoção. Um engenheiro de dados deve entender esses tipos de bancos de dados, incluindo considerações de uso, a estrutura dos dados que eles armazenam e como aproveitar cada um deles no ciclo de vida da engenharia de dados.

Armazenamentos de chave-valor. *Abanco de dados chave-valoré* um banco de dados não relacional que recupera registros usando uma chave que identifica exclusivamente cada registro. Isso é semelhante ao mapa de hash ou estruturas de dados de dicionário apresentadas em muitas linguagens de programação, mas potencialmente mais escaláveis. Os armazenamentos de valor-chave abrangem vários tipos de banco de dados NoSQL — por exemplo, armazenamento de documentos e bancos de dados de colunas largas (discutido a seguir).

Diferentes tipos de bancos de dados de valor-chave oferecem uma variedade de características de desempenho para atender a várias necessidades de aplicativos. Por exemplo, os bancos de dados de valor-chave na memória são populares para armazenar dados de sessão em cache para aplicativos da web e móveis, onde são necessárias pesquisas ultrarrápidas e alta simultaneidade. O armazenamento nesses sistemas geralmente é temporário; se o banco de dados for encerrado, os dados desaparecem. Esses caches podem reduzir a pressão no banco de dados principal do aplicativo e fornecer respostas rápidas.

Obviamente, os armazenamentos de valor-chave também podem atender a aplicativos que exigem persistência de alta durabilidade. Um aplicativo de comércio eletrônico pode precisar salvar e atualizar grandes quantidades de alterações de estado de evento para um usuário e seus pedidos. Um usuário faz login no aplicativo de comércio eletrônico, clica em várias telas, adiciona itens a um carrinho de compras e, em seguida, finaliza a compra. Cada evento deve ser armazenado de forma durável para recuperação. Os armazenamentos de valor-chave geralmente persistem dados no disco e em vários nós para oferecer suporte a esses casos de uso.

Lojas de documentos. Como mencionado anteriormente, um*loja de documentosé* um armazenamento de valor-chave especializado. Neste contexto, um*documentoé* um objeto aninhado; geralmente podemos pensar em cada documento como um objeto JSON para fins práticos. Os documentos são armazenados em coleções e recuperados por chave. A*coleçãooé* aproximadamente equivalente a uma tabela em um banco de dados relacional (consulte [Tabela 5-2](#)).

Tabela 5-2. Comparação de RDBMS e terminologia de documento

RDBMS	banco de dados de documentos
Mesa	Coleção
Linha	Documento, itens, entidade

Uma diferença fundamental entre bancos de dados relacionais e armazenamentos de documentos é que o último não oferece suporte a junções. Isso significa que os dados não podem ser facilmente *normalizado*, ou seja, dividido em várias tabelas. (Aplicativos ainda podem ingressar manualmente. O código pode pesquisar um documento, extrair uma propriedade e, em seguida, recuperar outro documento.) Idealmente, todos os dados relacionados podem ser armazenados no mesmo documento.

Em muitos casos, os mesmos dados devem ser armazenados em vários documentos espalhados por várias coleções; os engenheiros de software devem ter o cuidado de atualizar uma propriedade em qualquer lugar em que ela esteja armazenada. (Muitos repositórios de documentos oferecem suporte a uma noção de transações para facilitar isso.)

Os bancos de dados de documentos geralmente abrangem toda a flexibilidade do JSON e não impõem esquemas ou tipos; isso é uma bênção e uma maldição. Por um lado, isso permite que o esquema seja altamente flexível e expressivo. O esquema também pode evoluir à medida que um aplicativo cresce. Por outro lado, vimos bancos de dados de documentos se tornarem pesadelos absolutos para gerenciar e consultar. Se os desenvolvedores não tiverem cuidado ao gerenciar a evolução do esquema, os dados podem se tornar inconsistentes e inchados com o tempo. A evolução do esquema também pode interromper a ingestão de downstream e causar dores de cabeça aos engenheiros de dados se não for comunicada em tempo hábil (antes da implantação).

Veja a seguir um exemplo de dados armazenados em uma coleção chamada `Usuários`. A chave da coleção é `aeu ia`. Nós também temos `um nome` (juntamente com `primeiro` e `durar` como elementos filho) e um array das bandas favoritas do usuário dentro de cada documento:

```
{  
  "Usuários": [  
    {  
      "eu ia": "1234",  
      "nome": {  
        "primeiro": "João",  
        "durar": "Reis"  
      },  
      "bandas_favoritas": [  
        "AC/DC",  
        "Assassino",  
        "Clã Wu-Tang",  
        "Ação Bronson"  
      ],  
      {  
        "eu ia": "1235",  
        "nome": {  
          "primeiro": "Mate",  
          "durar": "Housley"  
        },  
        "bandas_favoritas": [  
          "Banda de Dave Matthews",  
          "Crença",  
          "Nickelback"  
        ]  
      }  
    }  
  ]  
}
```

```
        ]
    }
]
}
```

Para consultar os dados neste exemplo, você pode recuperar registros por chave. Observe que a maioria dos bancos de dados de documentos também oferece suporte à criação de índices e tabelas de pesquisa para permitir a recuperação de documentos por propriedades específicas. Isso geralmente é inestimável no desenvolvimento de aplicativos quando você precisa pesquisar documentos de várias maneiras. Por exemplo, você pode definir um índice em nome.

Outro detalhe técnico crítico para engenheiros de dados é que os armazenamentos de documentos geralmente não são compatíveis com ACID, ao contrário dos bancos de dados relacionais. O conhecimento técnico em um armazenamento de documentos específico é essencial para entender o desempenho, ajuste, configuração, efeitos relacionados em gravações, consistência, durabilidade, etc. Por exemplo, muitos armazenamentos de documentos são *eventualmente consistente*. Permitir a distribuição de dados em um cluster é uma vantagem para escalabilidade e desempenho, mas pode levar a catástrofes quando engenheiros e desenvolvedores não entendem as implicações.²

Para executar análises em armazenamentos de documentos, os engenheiros geralmente devem executar uma verificação completa para extraírem todos os dados de uma coleção ou empregar uma estratégia de CDC para enviar eventos a um fluxo de destino. A abordagem de varredura completa pode ter implicações de desempenho e custo. A verificação geralmente torna o banco de dados mais lento enquanto ele é executado, e muitas ofertas de nuvem sem servidor cobram uma taxa significativa por cada verificação completa. Em bancos de dados de documentos, geralmente é útil criar um índice para ajudar a acelerar as consultas. Discutimos índices e padrões de consulta em [Capítulo 8](#).

Coluna larga. Um banco de dados de coluna larga é otimizado para armazenar grandes quantidades de dados com altas taxas de transação e latência extremamente baixa. Esses bancos de dados podem ser dimensionados para taxas de gravação extremamente altas e grandes quantidades de dados. Especificamente, os bancos de dados de colunas largas podem suportar petabytes de dados, milhões de solicitações por segundo e latência abaixo de 10ms. Essas características tornaram os bancos de dados de colunas largas populares em aplicativos de comércio eletrônico, fintech, ad tech, IoT e personalização em tempo real. Os engenheiros de dados devem estar cientes das características operacionais dos bancos de dados de colunas largas com os quais trabalham para definir uma configuração adequada, projetar o esquema e escolher uma chave de linha apropriada para otimizar o desempenho e evitar problemas operacionais comuns.

Esses bancos de dados suportam varreduras rápidas de grandes quantidades de dados, mas não suportam consultas complexas. Eles têm apenas um único índice (a chave de linha) para pesquisas. Os engenheiros de dados geralmente devem extraírem dados e enviá-los para um sistema analítico secundário para executar consultas complexas para lidar com essas limitações. Isso pode ser feito executando varreduras grandes para a extração ou empregando CDC para capturar um fluxo de eventos.

² A excelente série de Nimal Dalal sobre [A história do MongoDB](#) relata algumas histórias angustiantes de abuso de banco de dados e suas consequências para startups iniciantes.

Bancos de dados gráficos. Bancos de dados gráficos armazenam dados explicitamente com uma estrutura de gráfico matemático (como um conjunto de nós e arestas).³ O Neo4j provou ser extremamente popular, enquanto Amazon, Oracle e outros fornecedores oferecem seus produtos de banco de dados de gráficos. Grosso modo, os bancos de dados gráficos são uma boa opção quando você deseja analisar a conectividade entre os elementos.

Por exemplo, você pode usar um banco de dados de documentos para armazenar um documento para cada usuário que descreve suas propriedades. Você pode adicionar um elemento de array para conexões que contém IDs de usuários diretamente conectados em um contexto de mídia social. É muito fácil determinar o número de conexões diretas que um usuário possui, mas suponha que você queira saber quantos usuários podem ser alcançados percorrendo duas conexões diretas. Você poderia responder a essa pergunta escrevendo um código complexo, mas cada consulta seria executada lentamente e consumiria recursos significativos. O armazenamento de documentos simplesmente não é otimizado para este caso de uso.

Os bancos de dados gráficos são projetados precisamente para esse tipo de consulta. Suas estruturas de dados permitem consultas baseadas na conectividade entre os elementos; os bancos de dados de grafos são indicados quando nos preocupamos em entender as travessias complexas entre os elementos. Na linguagem dos gráficos, armazenamos *nós* (usuários no exemplo anterior) e *arestas* (conexões entre usuários). Bancos de dados gráficos suportam modelos de dados ricos para nós e arestas. Dependendo do mecanismo de banco de dados gráfico subjacente, os bancos de dados gráficos utilizam linguagens de consulta especializadas, como SPARQL, Resource Description Framework (RDF), Graph Query Language (GQL) e Cypher.

Como exemplo de grafo, considere uma rede de quatro usuários. O usuário 1 segue o usuário 2, que segue o usuário 3 e o usuário 4; O usuário 3 também segue o usuário 4 ([Figura 5-8](#)).

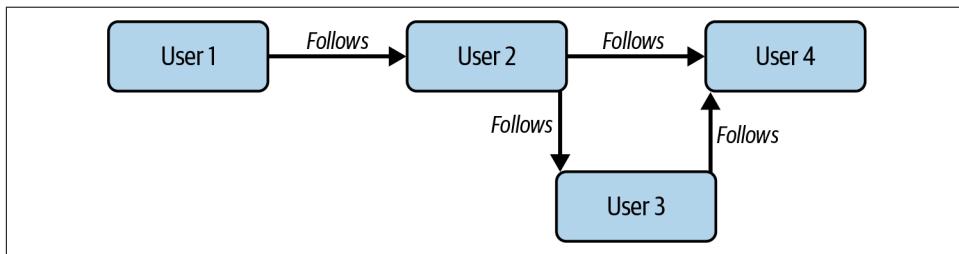


Figura 5-8. Um gráfico de rede social

Prevemos que os aplicativos de banco de dados de gráficos crescerão dramaticamente fora das empresas de tecnologia; análises de mercado também prevêem crescimento rápido.⁴ Claro, bancos de dados gráficos

³Martin Kleppmann, *Projetando aplicativos com uso intensivo de dados* (Sebastopol, CA: O'Reilly, 2017), 49, <https://oreil.ly/v1NhG>.

⁴Aashish Mehra, "Mercado de bancos de dados gráficos vale US\$ 5,1 bilhões até 2026: relatório exclusivo da MarketsandMarkets," Cision PR Newswire, 30 de julho de 2021, <https://oreil.ly/mGVkY>.

são benéficos de uma perspectiva operacional e suportam os tipos de relações sociais complexas críticas para aplicações modernas. As estruturas gráficas também são fascinantes do ponto de vista da ciência de dados e ML, potencialmente revelando insights profundos sobre interações e comportamentos humanos.

Isso apresenta desafios exclusivos para engenheiros de dados que podem estar mais acostumados a lidar com relações estruturadas, documentos ou dados não estruturados. Os engenheiros devem escolher se querem fazer o seguinte:

- Mapeie os dados do gráfico do sistema de origem em um de seus paradigmas preferenciais existentes
- Analisar dados do gráfico dentro do próprio sistema de origem
- Adote ferramentas analíticas específicas para gráficos

Os dados do gráfico podem ser recodificados em linhas em um banco de dados relacional, o que pode ser uma solução adequada dependendo do caso de uso analítico. Bancos de dados de gráficos transacionais também são projetados para análise, embora consultas grandes possam sobrecarregar os sistemas de produção. Bancos de dados de gráficos contemporâneos baseados em nuvem oferecem suporte a análises de gráficos de leitura pesada em grandes quantidades de dados.

Procurar. *Abanco de dados de pesquisa*é um banco de dados não relacional usado para pesquisar as características semânticas e estruturais complexas e diretas de seus dados. Existem dois casos de uso proeminentes para um banco de dados de pesquisa: pesquisa de texto e análise de log. Vamos cobrir cada um deles separadamente.

*Pesquisa de texto*envolve a pesquisa de palavras-chave ou frases em um corpo de texto, correspondências exatas, difusas ou semanticamente semelhantes. *Análise de log* é normalmente usado para detecção de anomalias, monitoramento em tempo real, análise de segurança e análise operacional. As consultas podem ser otimizadas e aceleradas com o uso de índices.

Dependendo do tipo de empresa em que você trabalha, você pode usar bancos de dados de pesquisa regularmente ou não. Independentemente disso, é bom estar ciente de que eles existem, caso você os encontre na natureza. Os bancos de dados de pesquisa são populares para pesquisa e recuperação rápidas e podem ser encontrados em vários aplicativos; um site de comércio eletrônico pode potencializar sua pesquisa de produtos usando um banco de dados de pesquisa. Como engenheiro de dados, pode-se esperar que você traga dados de um banco de dados de pesquisa (como Elasticsearch, Apache Solr ou Lucene ou Algolia) para relatórios de KPI downstream ou algo semelhante.

Séries temporais. *Asérie temporal*é uma série de valores organizados por tempo. Por exemplo, os preços das ações podem se mover à medida que as negociações são executadas ao longo do dia, ou um sensor meteorológico medirá as temperaturas atmosféricas a cada minuto. Quaisquer eventos registrados ao longo do tempo – regular ou esporadicamente – são dados de séries temporais. *Abanco de dados de séries temporais*é otimizado para recuperação e processamento estatístico de dados de séries temporais.

Embora dados de séries temporais, como pedidos, remessas, logs e assim por diante, tenham sido armazenados em bancos de dados relacionais por muito tempo, esses tamanhos e volumes de dados geralmente eram minúsculos. À medida que os dados cresciam mais rapidamente e maiores, novos bancos de dados para fins especiais eram necessários. Os bancos de dados de séries temporais atendem às necessidades de volumes crescentes de dados de alta velocidade de IoT, logs de eventos e aplicativos, ad tech e fintech, entre muitos outros casos de uso. Muitas vezes, essas cargas de trabalho são pesadas para gravação. Como resultado, os bancos de dados de séries temporais geralmente utilizam buffer de memória para suportar gravações e leituras rápidas.

Devemos distinguir entre medição e dados baseados em eventos, comuns em bancos de dados de séries temporais. *Dados de medição* é gerado regularmente, como sensores de temperatura ou qualidade do ar. *Dados baseados em eventos* é irregular e criado sempre que ocorre um evento - por exemplo, quando um sensor de movimento detecta movimento.

O esquema para uma série temporal geralmente contém um carimbo de data/hora e um pequeno conjunto de campos. Como os dados dependem do tempo, os dados são ordenados pelo carimbo de data/hora. Isso torna os bancos de dados de série temporal adequados para análise operacional, mas não ótimos para casos de uso de BI. Junções não são comuns, embora alguns bancos de dados quase de séries temporais, como o Apache Druid, suportem junções. Muitos bancos de dados de séries temporais estão disponíveis, tanto como opções de código aberto quanto pagas.

APIs

As APIs são agora uma forma padrão e generalizada de troca de dados na nuvem, para plataformas SaaS e entre sistemas internos da empresa. Existem muitos tipos de interfaces de API na web, mas estamos interessados principalmente naquelas construídas em torno de HTTP, o tipo mais popular na web e na nuvem.

DESCANSAR

Falaremos primeiro sobre REST, atualmente o paradigma de API dominante. Conforme observado em [Capítulo 4, DESCANSAR apoia a transferência de estado representacional](#). Este conjunto de práticas e filosofias para construir APIs da Web HTTP foi apresentado por Roy Fielding em 2000 em uma dissertação de doutorado. REST é construído em torno de verbos HTTP, como GET e PUT; na prática, o REST moderno usa apenas alguns dos mapeamentos verbais descritos na dissertação original.

Uma das principais ideias do REST é que as interações não têm estado. Ao contrário de uma sessão de terminal Linux, não há noção de uma sessão com variáveis de estado associadas, como um diretório de trabalho; cada chamada REST é independente. As chamadas REST podem alterar o estado do sistema, mas essas alterações são globais, aplicando-se ao sistema completo, e não a uma sessão atual.

Os críticos apontam que REST não é de forma alguma uma especificação completa.⁵ O REST estipula propriedades básicas de interações, mas os desenvolvedores que utilizam uma API devem obter uma quantidade significativa de conhecimento de domínio para criar aplicativos ou extrair dados de forma eficaz.

Vemos uma grande variação nos níveis de abstração da API. Em alguns casos, as APIs são apenas um invólucro fino sobre os componentes internos que fornecem a funcionalidade mínima necessária para proteger o sistema das solicitações do usuário. Em outros exemplos, uma API de dados REST é uma obra-prima da engenharia que prepara dados para aplicativos analíticos e oferece suporte a relatórios avançados.

Alguns desenvolvimentos simplificaram a configuração de pipelines de ingestão de dados de APIs REST. Primeiro, os provedores de dados frequentemente fornecem bibliotecas cliente em várias linguagens, especialmente em Python. As bibliotecas de cliente removem grande parte do trabalho clichê de criar o código de interação da API. As bibliotecas cliente lidam com detalhes críticos, como autenticação e mapeiam métodos fundamentais em classes acessíveis.

Em segundo lugar, vários serviços e bibliotecas de código aberto surgiram para interagir com APIs e gerenciar a sincronização de dados. Muitos fornecedores de SaaS e de software livre fornecem conectores prontos para uso para APIs comuns. As plataformas também simplificam o processo de criação de conectores personalizados conforme necessário.

Existem inúmeras APIs de dados sem bibliotecas de cliente ou suporte de conector pronto para uso. Como enfatizamos ao longo do livro, os engenheiros fariam bem em reduzir o trabalho pesado indiferenciado usando ferramentas comuns. No entanto, de baixo nível *encanamento* tarefas ainda consomem muitos recursos. Em praticamente qualquer grande empresa, os engenheiros de dados precisarão lidar com o problema de escrever e manter um código personalizado para extrair dados de APIs, o que requer entender a estrutura dos dados fornecidos, desenvolver um código de extração de dados apropriado e determinar um conjunto de dados adequado. estratégia de sincronização.

GraphQLGenericName

GraphQLGenericName foi criado no Facebook como uma linguagem de consulta para dados de aplicativos e uma alternativa para APIs REST genéricas. Enquanto as APIs REST geralmente restringem suas consultas a um modelo de dados específico, o GraphQL abre a possibilidade de recuperar vários modelos de dados em uma única solicitação. Isso permite consultas mais flexíveis e expressivas do que com REST. GraphQL é construído em torno de JSON e retorna dados em uma forma semelhante à consulta JSON.

Há uma espécie de guerra santa entre REST e GraphQL, com algumas equipes de engenharia partidárias de um ou de outro e algumas usando ambos. Na realidade, os engenheiros encontrarão ambos ao interagir com os sistemas de origem.

5 Para obter um exemplo, consulte Michael S. Mikowski, "RESTful APIs: The Big Lie," 10 de agosto de 2015, <https://oreil.ly/rqja3>.

Webhooks

*Webhook*s são um padrão simples de transmissão de dados baseado em eventos. A fonte de dados pode ser um back-end de aplicativo, uma página da Web ou um aplicativo móvel. Quando eventos especificados acontecem no sistema de origem, isso aciona uma chamada para um terminal HTTP hospedado pelo consumidor de dados. Observe que a conexão vai do sistema de origem para o coletor de dados, o oposto das APIs típicas. Por esse motivo, os webhooks costumam ser chamados de *APIs reversas*.

O endpoint pode fazer várias coisas com os dados do evento POST, possivelmente acionando um processo downstream ou armazenando os dados para uso futuro. Para fins de análise, estamos interessados em coletar esses eventos. Os engenheiros geralmente usam filas de mensagens para ingerir dados em alta velocidade e volume. Falaremos sobre filas de mensagens e fluxos de eventos mais adiante neste capítulo.

RPC e gRPC

A *chamada de procedimento remoto*(RPC) é comumente usado em computação distribuída. Ele permite que você execute um procedimento em um sistema remoto.

*gRPC*é uma biblioteca de chamada de procedimento remoto desenvolvida internamente no Google em 2015 e posteriormente lançada como um padrão aberto. Seu uso apenas no Google seria suficiente para merecer inclusão em nossa discussão. Muitos serviços do Google, como Google Ads e GCP, oferecem APIs gRPC. O gRPC é construído em torno do padrão de serialização de dados abertos Protocol Buffers, também desenvolvido pelo Google.

O gRPC enfatiza a troca bidirecional eficiente de dados por HTTP/2.*Eficiência* refere-se a aspectos como utilização da CPU, consumo de energia, duração da bateria e largura de banda. Como o GraphQL, o gRPC impõe padrões técnicos muito mais específicos do que o REST, permitindo assim o uso de bibliotecas de clientes comuns e permitindo que os engenheiros desenvolvam um conjunto de habilidades que se aplicará a qualquer código de interação do gRPC.

Compartilhamento de dados

O conceito central do compartilhamento de dados em nuvem é que um sistema multitenant oferece suporte a políticas de segurança para compartilhamento de dados entre locatários. Concretamente, qualquer sistema de armazenamento de objetos em nuvem pública com um sistema de permissão refinado pode ser uma plataforma para compartilhamento de dados. As plataformas populares de armazenamento de dados em nuvem também oferecem suporte a recursos de compartilhamento de dados. Obviamente, os dados também podem ser compartilhados por meio de download ou troca por e-mail, mas um sistema multilocatário torna o processo muito mais fácil.

Muitas plataformas de compartilhamento modernas (especialmente data warehouses em nuvem) suportam filtragem de linha, coluna e dados confidenciais. O compartilhamento de dados também simplifica a noção do *mercado de dados*, disponível em várias nuvens e plataformas de dados populares. Os mercados de dados fornecem um local centralizado para o comércio de dados, onde os provedores de dados podem anunciar suas ofertas e vendê-las sem se preocupar com os detalhes de gerenciamento de acesso à rede para sistemas de dados.

O compartilhamento de dados também pode simplificar os pipelines de dados dentro de uma organização. O compartilhamento de dados permite que as unidades de uma organização gerenciem seus dados e os compartilhem seletivamente com outras unidades, enquanto ainda permite que unidades individuais gerenciem seus custos de computação e consulta separadamente, facilitando a descentralização dos dados. Isso facilita padrões de gerenciamento de dados descentralizados, como malha de dados.⁶

O compartilhamento de dados e a malha de dados se alinham estreitamente com nossa filosofia de componentes de arquitetura comuns. Escolha componentes comuns (consulte [Capítulo 3](#)) que permitem o intercâmbio simples e eficiente de dados e conhecimentos, em vez de adotar a tecnologia mais empolgante e sofisticada.

Fontes de dados de terceiros

A consumerização da tecnologia significa que toda empresa agora é essencialmente uma empresa de tecnologia. A consequência é que essas empresas – e cada vez mais agências governamentais – desejam disponibilizar seus dados para seus clientes e usuários, seja como parte de seu serviço ou como uma assinatura separada. Por exemplo, o Bureau of Labor Statistics dos EUA publica várias estatísticas sobre o mercado de trabalho dos EUA. A Administração Nacional de Aeronáutica e Espaço (NASA) publica vários dados de suas iniciativas de pesquisa. O Facebook compartilha dados com empresas que anunciam em sua plataforma.

Por que as empresas querem disponibilizar seus dados? Os dados são fixos e um volante é criado permitindo que os usuários integrem e estendam seu aplicativo no aplicativo de um usuário. Maior adoção e uso do usuário significa mais dados, o que significa que os usuários podem integrar mais dados em seus aplicativos e sistemas de dados. O efeito colateral é que agora existem fontes quase infinitas de dados de terceiros.

O acesso direto a dados de terceiros geralmente é feito por meio de APIs, por meio de compartilhamento de dados em uma plataforma de nuvem ou por download de dados. As APIs geralmente fornecem recursos de integração profunda, permitindo que os clientes extraiam e enviem dados. Por exemplo, muitos CRMs oferecem APIs que seus usuários podem integrar em seus sistemas e aplicativos. Vemos um fluxo de trabalho comum para obter dados de um CRM, combinar os dados do CRM por meio do modelo de pontuação do cliente e, em seguida, usar o ETL reverso para enviar esses dados de volta ao CRM para que os vendedores entrem em contato com leads mais qualificados.

Filas de mensagens e plataformas de streaming de eventos

As arquiteturas orientadas a eventos são difundidas em aplicativos de software e estão prontas para aumentar ainda mais sua popularidade. Primeiro, as filas de mensagens e as plataformas de fluxo de eventos - camadas críticas em arquiteturas orientadas a eventos - são mais fáceis de configurar e gerenciar

⁶ Martin Fowler, "How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh," Martin-Fowler.com, 20 de maio de 2019, <https://oreil.ly/TEdJF>.

em um ambiente de nuvem. Em segundo lugar, a ascensão dos aplicativos de dados — aplicativos que integram diretamente a análise em tempo real — está crescendo cada vez mais. As arquiteturas orientadas a eventos são ideais nesse cenário porque os eventos podem acionar o trabalho no aplicativo e alimentar análises quase em tempo real.

Observe que os dados de streaming (neste caso, mensagens e streams) atravessam muitos estágios do ciclo de vida da engenharia de dados. Ao contrário de um RDBMS, que geralmente é anexado diretamente a um aplicativo, as linhas de dados de streaming às vezes são menos definidas. Esses sistemas são usados como sistemas de origem, mas geralmente atravessam o ciclo de vida da engenharia de dados devido à sua natureza transitória. Por exemplo, você pode usar uma plataforma de fluxo de eventos para passagem de mensagens em um aplicativo orientado a eventos, um sistema de origem. A mesma plataforma de streaming de eventos pode ser usada no estágio de ingestão e transformação para processar dados para análise em tempo real.

Como sistemas de origem, as filas de mensagens e as plataformas de streaming de eventos são usadas de várias maneiras, desde o roteamento de mensagens entre microsserviços que ingerem milhões de eventos por segundo de dados de eventos de aplicativos da Web, móveis e IoT. Vamos examinar as filas de mensagens e as plataformas de streaming de eventos um pouco mais de perto.

filas de mensagens

A *fila de mensagens* é um mecanismo para enviar dados de forma assíncrona (geralmente como pequenas mensagens individuais, em kilobytes) entre sistemas discretos usando um modelo de publicação e assinatura. Os dados são publicados em uma fila de mensagens e entregues a um ou mais assinantes ([Figura 5-9](#)). O assinante признается o recebimento da mensagem, retirando-a da fila.

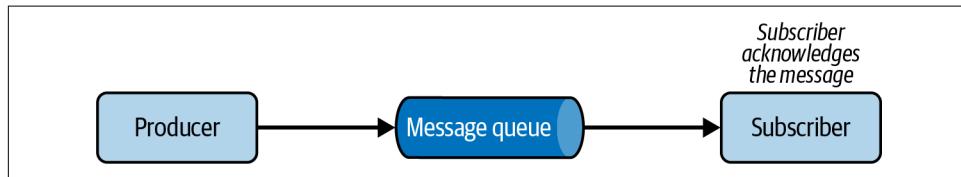


Figura 5-9. Uma fila de mensagens simples

As filas de mensagens permitem que aplicativos e sistemas sejam desacoplados uns dos outros e são amplamente utilizados em arquiteturas de microsserviços. A fila de mensagens armazena mensagens em buffer para lidar com picos de carga transitórios e torna as mensagens duráveis por meio de uma arquitetura distribuída com replicação.

As filas de mensagens são um ingrediente crítico para microsserviços desacoplados e arquiteturas orientadas a eventos. Algumas coisas a serem lembradas com as filas de mensagens são a frequência de entrega, a ordem das mensagens e a escalabilidade.

Encomenda e entrega de mensagens. A ordem na qual as mensagens são criadas, enviadas e recebidas pode afetar significativamente os assinantes downstream. Em geral, ordem em

filas de mensagens distribuídas é um problema complicado. As filas de mensagens geralmente aplicam uma noção difusa de ordem e primeiro a entrar, primeiro a sair (FIFO). FIFO estrito significa que se a mensagem A for ingerida antes da mensagem B, a mensagem A sempre será entregue antes da mensagem B. Na prática, as mensagens podem ser publicadas e recebidas fora de ordem, especialmente em sistemas de mensagens altamente distribuídas.

Por exemplo, Amazon SQS⁷ faz o melhor esforço para preservar a ordem das mensagens. A SQS também oferece Filas FIFO, que oferecem garantias muito mais fortes ao custo de sobrecarga extra.

Em geral, não presuma que suas mensagens serão entregues em ordem, a menos que sua tecnologia de fila de mensagens garanta isso. Normalmente, você precisa projetar para entrega de mensagens fora de ordem.

Frequência de entrega. As mensagens podem ser enviadas exatamente uma vez ou pelo menos uma vez. Se uma mensagem for enviada *exatamente uma vez*, depois que o assinante confirmar a mensagem, a mensagem desaparecerá e não será entregue novamente.⁷ Mensagens enviadas *pelo menos uma vez* pode ser consumido por vários assinantes ou pelo mesmo assinante mais de uma vez. Isso é ótimo quando duplicações ou redundância não importam.

Idealmente, os sistemas devem ser *idempotente*. Em um sistema idempotente, o resultado do processamento de uma mensagem uma vez é idêntico ao resultado do processamento várias vezes. Isso ajuda a explicar uma variedade de cenários sutis. Por exemplo, mesmo que nosso sistema possa garantir a entrega exatamente uma vez, um consumidor pode processar totalmente uma mensagem, mas falhar antes de confirmar o processamento. A mensagem será efetivamente processada duas vezes, mas um sistema idempotente lida com esse cenário normalmente.

Escalabilidade. As filas de mensagens mais populares utilizadas em aplicativos orientados a eventos são escalonáveis horizontalmente, executando em vários servidores. Isso permite que essas filas aumentem e diminuam dinamicamente, armazenem mensagens quando os sistemas ficam para trás e armazenem mensagens de forma durável para resiliência contra falhas. No entanto, isso pode criar uma variedade de complicações, conforme mencionado anteriormente (múltiplas entregas e pedidos difusos).

Plataformas de streaming de eventos

De certa forma, uma *plataforma de streaming de eventos* é uma continuação de uma fila de mensagens em que as mensagens são passadas de produtores para consumidores. Conforme discutido anteriormente neste capítulo, a grande diferença entre mensagens e fluxos é que uma fila de mensagens é usada principalmente para rotear mensagens com certas garantias de entrega. Em contraste, uma plataforma de streaming de eventos é usada para ingerir e processar dados em um log ordenado de

⁷ Se *exatamente uma vez* é possível é um debate semântico. Tecnicamente, exatamente quando a entrega é impossível de garantia, como ilustra o [Problema de dois generais](#).

registros. Em uma plataforma de streaming de eventos, os dados são retidos por um tempo e é possível reproduzir mensagens de um ponto no tempo passado.

Vamos descrever um evento relacionado a uma plataforma de streaming de eventos. Como mencionado em [Capítulo 3](#), um evento é “algo que aconteceu, normalmente uma mudança no estado de alguma coisa.” Um evento tem os seguintes recursos: uma chave, um valor e um carimbo de data/hora. Vários carimbos de data/hora de valor-chave podem estar contidos em um único evento. Por exemplo, um evento para um pedido de comércio eletrônico pode ter esta aparência:

```
{  
    "Chave": "Pedido nº 12345",  
    "Valor": "SKU 123, preço de compra de US$ 100",  
    "Carimbo de data/hora": "2023-01-02 06:01:00"  
}
```

Vejamos algumas das características críticas de uma plataforma de streaming de eventos que você deve conhecer como engenheiro de dados.

Tópicos. Em uma plataforma de streaming de eventos, um produtor transmite eventos para um tópico, uma coleção de eventos relacionados. Um tópico pode conter alertas de fraude, pedidos de clientes ou leituras de temperatura de dispositivos IoT, por exemplo. Um tópico pode ter zero, um ou vários produtores e clientes na maioria das plataformas de transmissão de eventos.

Usando o exemplo de evento anterior, um tópico pode ser gerado pelo sistema de processamento de pedidos na web. Além disso, vamos enviar este tópico para alguns consumidores, como marketing e cumprimento. Este é um excelente exemplo de linhas tênues entre análise e um sistema orientado a eventos. O cumprimento assinante usará eventos para acionar um processo de atendimento, enquanto o marketing executa análises em tempo real ou treina e executa modelos de ML para ajustar campanhas de marketing ([Figura 5-10](#)).

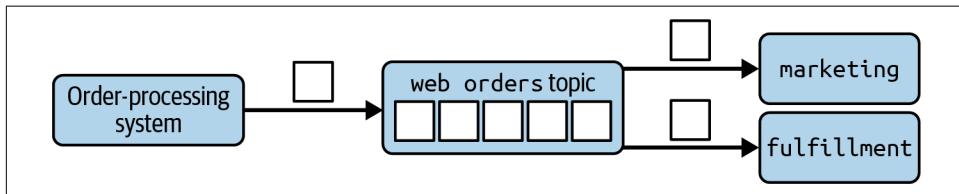


Figura 5-10. Um sistema de processamento de pedidos gera eventos (pequenos quadrados) e os publica no tópico web orders topic. Dois assinantes—marketing e cumprimento—puxam eventos do tópico.

Partições de fluxo. Partições de fluxo são subdivisões de um fluxo em vários fluxos. Uma boa analogia é uma rodovia com várias pistas. Ter várias pistas permite paralelismo e maior rendimento. As mensagens são distribuídas entre as partições por *chave de partição*. Mensagens com a mesma chave de partição sempre terminarão na mesma partição.

Em [Figura 5-11](#), por exemplo, cada mensagem tem um ID numérico — mostrado dentro do círculo que representa a mensagem — que usamos como uma chave de partição. Para determinar a partição, dividimos por 3 e pegamos o resto. Indo de baixo para cima, as partícões têm resto 0, 1 e 2, respectivamente.

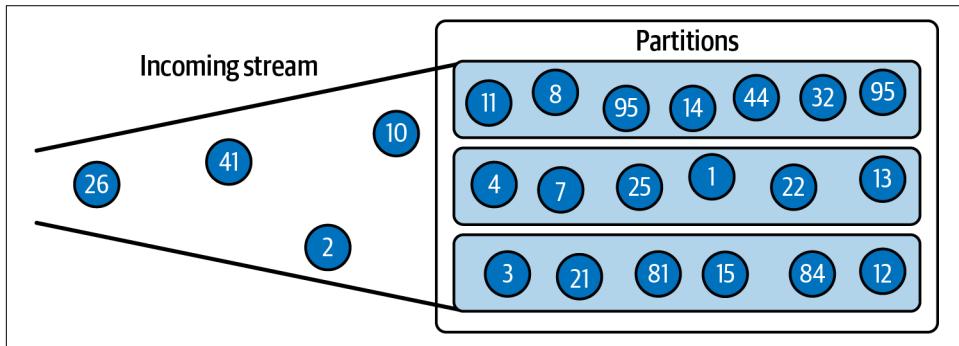


Figura 5-11. Um fluxo de mensagens recebidas dividido em três partícões

Defina uma chave de partição para que as mensagens que devem ser processadas juntas tenham a mesma chave de partição. Por exemplo, é comum em configurações de IoT querer enviar todas as mensagens de um determinado dispositivo para o mesmo servidor de processamento. Podemos conseguir isso usando um ID de dispositivo como a chave de partição e, em seguida, configurando um servidor para consumir de cada partição.

Uma preocupação importante com o particionamento de fluxo é garantir que sua chave de partição não gere ponto de acesso — um número desproporcional de mensagens entregues a uma partição. Por exemplo, se cada dispositivo IoT estivesse localizado em um determinado estado dos EUA, poderíamos usar o estado como a chave de partição. Dada uma distribuição de dispositivos proporcional à população do estado, as partícões contendo Califórnia, Texas, Flórida e Nova York podem ficar sobrecarregadas, com outras partícões relativamente subutilizadas. Certifique-se de que sua chave de partição distribuirá as mensagens uniformemente entre as partícões.

Tolerância a falhas e resiliência. Plataformas de streaming de eventos são tipicamente sistemas distribuídos, com streams armazenados em vários nós. Se um nó ficar inativo, outro nó o substituirá e o fluxo ainda estará acessível. Isso significa que os registros não são perdidos; você pode optar por excluir registros, mas isso é outra história. Essa tolerância a falhas e resiliência tornam as plataformas de streaming uma boa escolha quando você precisa de um sistema que possa produzir, armazenar e ingerir dados de eventos de maneira confiável.

Com quem você trabalhará

Ao acessar os sistemas de origem, é essencial entender as pessoas com quem você trabalhará. Em nossa experiência, boa diplomacia e relacionamento com o

as partes interessadas dos sistemas de origem são uma parte subestimada e crucial da engenharia de dados bem-sucedida.

Quem são essas partes interessadas? Normalmente, você lidará com duas categorias de stakeholders: stakeholders de sistemas e de dados ([Figura 5-12](#)). *Aparte interessada dos sistemas* constrói e mantém os sistemas de origem; podem ser engenheiros de software, desenvolvedores de aplicativos e terceiros. As partes interessadas dos dados possuem e controlam o acesso aos dados que você deseja, geralmente manipulados por TI, um grupo de governança de dados ou terceiros. As partes interessadas em sistemas e dados geralmente são pessoas ou equipes diferentes; às vezes, eles são os mesmos.

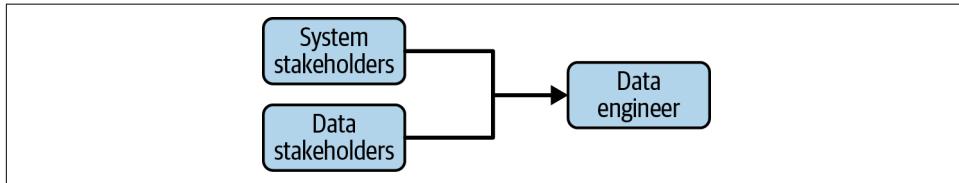


Figura 5-12. As partes interessadas upstream do engenheiro de dados

Muitas vezes, você está à mercê da capacidade da parte interessada de seguir as práticas corretas de engenharia de software, gerenciamento de banco de dados e desenvolvimento. Idealmente, as partes interessadas estão fazendo DevOps e trabalhando de maneira ágil. Sugerimos a criação de um ciclo de feedback entre os engenheiros de dados e as partes interessadas dos sistemas de origem para criar consciência de como os dados são consumidos e usados. Essa é uma das áreas mais negligenciadas em que os engenheiros de dados podem obter muito valor. Quando algo acontece com os dados de origem upstream - e algo acontecerá, seja um esquema ou alteração de dados, um servidor ou banco de dados com falha ou outros eventos importantes - você deseja ter certeza de que está ciente do impacto que esses problemas terão em seus sistemas de engenharia de dados.

Pode ser útil ter um contrato de dados em vigor com os proprietários do sistema de origem upstream. O que é um contrato de dados? James Denmore oferece esta definição:⁸

Um contrato de dados é um acordo por escrito entre o proprietário de um sistema de origem e a equipe que ingere dados desse sistema para uso em um pipeline de dados. O contrato deve indicar quais dados estão sendo extraídos, por meio de qual método (completo, incremental), com que frequência e quem (pessoa, equipe) são os contatos do sistema de origem e da ingestão. Os contratos de dados devem ser armazenados em um local conhecido e fácil de encontrar, como um repositório GitHub ou site de documentação interna. Se possível, formate os contratos de dados em um formato padronizado para que possam ser integrados ao processo de desenvolvimento ou consultados programaticamente.

⁸James Denmore, *Referência de Bolso de Pipelines de Dados* (Sebastopol, CA: O'Reilly), <https://oreil.ly/8Qdkj>. Leia o livro para obter mais informações sobre como um contrato de dados deve ser escrito.

Além disso, considere estabelecer um SLA com provedores upstream. Um SLA fornece expectativas do que você pode esperar dos sistemas de origem nos quais você confia. Um exemplo de um SLA pode ser “os dados dos sistemas de origem estarão disponíveis de forma confiável e de alta qualidade”. Um objetivo de nível de serviço (SLO) mede o desempenho em relação ao que você concordou no SLA. Por exemplo, dado o seu exemplo de SLA, um SLO pode ser “os sistemas de origem terão 99% de tempo de atividade”. Se um contrato de dados ou SLA/SLO parecer muito formal, pelo menos defina verbalmente as expectativas de garantias do sistema de origem para tempo de atividade, qualidade de dados e qualquer outra coisa importante para você. Os proprietários upstream de sistemas de origem precisam entender seus requisitos para que possam fornecer os dados de que você precisa.

Subcorrentes e seu impacto nos sistemas de origem

Ao contrário de outras partes do ciclo de vida da engenharia de dados, os sistemas de origem geralmente estão fora do controle do engenheiro de dados. Há uma suposição implícita (alguns podem chamá-la de *esperança*) que as partes interessadas e proprietários dos sistemas de origem - e os dados que eles produzem - estão seguindo as melhores práticas relacionadas ao gerenciamento de dados, DataOps (e DevOps), DODD (mencionado em [Capítulo 2](#)) arquitetura de dados, orquestração e engenharia de software. O engenheiro de dados deve obter o máximo de suporte upstream possível para garantir que as subcorrentes sejam aplicadas quando os dados forem gerados nos sistemas de origem. Isso fará com que o restante das etapas do ciclo de vida da engenharia de dados ocorra com muito mais facilidade.

Como as subcorrentes afetam os sistemas de origem? Vamos dar uma olhada.

Segurança

A segurança é crítica e a última coisa que você deseja é criar acidentalmente um ponto de vulnerabilidade em um sistema de origem. Aqui estão algumas áreas a serem consideradas:

- O sistema de origem foi arquitetado para que os dados sejam seguros e criptografados, tanto com dados em repouso quanto durante a transmissão?
- Você precisa acessar o sistema de origem pela Internet pública ou está usando uma rede privada virtual (VPN)?
- Mantenha senhas, tokens e credenciais para o sistema de origem trancados com segurança. Por exemplo, se você estiver usando chaves Secure Shell (SSH), use um gerenciador de chaves para proteger suas chaves; a mesma regra se aplica a senhas - use um gerenciador de senhas ou um provedor de logon único (SSO).
- Você confia no sistema de origem? Certifique-se sempre de confiar, mas verifique se o sistema de origem é legítimo. Você não quer receber dados de um ator mal-intencionado.

Gestão de dados

O gerenciamento de dados de sistemas de origem é um desafio para os engenheiros de dados. Na maioria dos casos, você terá apenas controle periférico - se houver algum controle - sobre os sistemas de origem e os dados que eles produzem. Na medida do possível, você deve entender como os dados são gerenciados nos sistemas de origem, pois isso influenciará diretamente como você ingere, armazena e transforma os dados.

Aqui estão algumas áreas a serem consideradas:

Gestão de dados

Os dados e sistemas upstream são controlados de forma confiável e fácil de entender? Quem gerencia os dados?

Qualidade dos dados

Como você garante a qualidade e a integridade dos dados em sistemas upstream? Trabalhe com as equipes do sistema de origem para definir as expectativas sobre dados e comunicação.

Esquema

Espere que os esquemas upstream mudem. Sempre que possível, colabore com as equipes do sistema de origem para ser notificado sobre alterações de esquema iminentes.

Gerenciamento de dados mestre

A criação de registros upstream é controlada por uma prática ou sistema de gerenciamento de dados mestre?

Privacidade e ética

Você tem acesso aos dados brutos ou os dados serão ofuscados? Quais são as implicações dos dados de origem? Quanto tempo ele é retido? Ele muda de local com base em políticas de retenção?

Regulatório

Com base nos regulamentos, você deve acessar os dados?

DataOps

Excelência operacional — DevOps, DataOps, MLOps,xOps - deve estender para cima e para baixo toda a pilha e suportar a engenharia de dados e o ciclo de vida. Embora isso seja ideal, muitas vezes não é totalmente realizado.

Como você está trabalhando com as partes interessadas que controlam os sistemas de origem e os dados que eles produzem, você precisa garantir que pode observar e monitorar o tempo de atividade e o uso dos sistemas de origem e responder quando ocorrerem incidentes. Por exemplo, quando o banco de dados do aplicativo do qual você depende para o CDC excede sua capacidade de E/S e precisa ser redimensionado, como isso afetará sua capacidade de receber dados desse sistema? Você poderá acessar os dados ou eles ficarão indisponíveis até que o banco de dados seja redimensionado? Como isso afetará os relatórios? Em outro exemplo, se o software

equipe de engenharia está implantando continuamente, uma alteração de código pode causar falhas imprevistas no próprio aplicativo. Como a falha afetará sua capacidade de acessar os bancos de dados que alimentam o aplicativo? Os dados estarão atualizados?

Configure uma cadeia de comunicação clara entre a engenharia de dados e as equipes de suporte aos sistemas de origem. Idealmente, essas equipes de partes interessadas incorporaram o DevOps em seu fluxo de trabalho e cultura. Isso ajudará muito a atingir os objetivos do DataOps (um irmão do DevOps), para abordar e reduzir os erros rapidamente. Como mencionamos anteriormente, os engenheiros de dados precisam se inserir nas práticas de DevOps das partes interessadas e vice-versa. O DataOps bem-sucedido funciona quando todas as pessoas estão a bordo e se concentram em fazer os sistemas funcionarem de forma holística.

Algumas considerações de DataOps são as seguintes:

Automação

Há a automação que afeta o sistema de origem, como atualizações de código e novos recursos.

Depois, há a automação DataOps que você configurou para seus fluxos de trabalho de dados.

Um problema na automação do sistema de origem afeta a automação do fluxo de trabalho de dados? Nesse caso, considere desacoplar esses sistemas para que eles possam executar a automação de forma independente.

Observabilidade

Como você saberá quando houver um problema com um sistema de origem, como uma interrupção ou um problema de qualidade de dados? Configure o monitoramento do tempo de atividade do sistema de origem (ou use o monitoramento criado pela equipe proprietária do sistema de origem). Configure verificações para garantir que os dados do sistema de origem estejam em conformidade com as expectativas de uso downstream. Por exemplo, os dados são de boa qualidade? O esquema é compatível? Os registros dos clientes são consistentes? Os dados são hash conforme estipulado pela política interna?

Resposta a incidentes

Qual é o seu plano se algo ruim acontecer? Por exemplo, como seu pipeline de dados se comportará se um sistema de origem ficar offline? Qual é o seu plano para preencher os dados "perdidos" assim que o sistema de origem estiver online novamente?

Arquitetura de dados

Semelhante ao gerenciamento de dados, a menos que você esteja envolvido no projeto e na manutenção da arquitetura do sistema de origem, você terá pouco impacto na arquitetura do sistema de origem upstream. Você também deve entender como a arquitetura upstream é projetada e seus pontos fortes e fracos. converse frequentemente com as equipes responsáveis pelos sistemas de origem para entender os fatores discutidos nesta seção e garantir que seus sistemas atendam às suas expectativas. Saber onde a arquitetura funciona bem e onde não tem impacto sobre como você projeta seu pipeline de dados.

Aqui estão algumas coisas a considerar em relação às arquiteturas do sistema de origem:

Confiabilidade

Todos os sistemas sofrem de entropia em algum ponto, e as saídas irão desviar do que é esperado. Bugs são introduzidos e falhas aleatórias acontecem. O sistema produz saídas previsíveis? Com que frequência podemos esperar que o sistema falhe? Qual é o tempo médio de reparo para que o sistema volte a ter confiabilidade suficiente?

Durabilidade

Tudo falha. Um servidor pode morrer, uma zona ou região da nuvem pode ficar offline ou outros problemas podem surgir. Você precisa considerar como uma falha ou interrupção inevitável afetará seus sistemas de dados gerenciados. Como o sistema de origem lida com a perda de dados por falhas de hardware ou interrupções de rede? Qual é o plano para lidar com interrupções por um período prolongado e limitar o raio de explosão de uma interrupção?

Disponibilidade

O que garante que o sistema de origem esteja funcionando e disponível quando deveria?

Pessoas

Quem é responsável pelo design do sistema de origem e como você saberá se foram feitas alterações significativas na arquitetura? Um engenheiro de dados precisa trabalhar com as equipes que mantêm os sistemas de origem e garantir que esses sistemas sejam arquitetados de maneira confiável. Crie um SLA com a equipe do sistema de origem para definir as expectativas sobre possíveis falhas do sistema.

Orquestração

Ao orquestrar em seu fluxo de trabalho de engenharia de dados, você se preocupará principalmente em garantir que sua orquestração possa acessar o sistema de origem, o que requer acesso à rede, autenticação e autorização corretos.

Aqui estão algumas coisas para se pensar sobre a orquestração de sistemas de origem:

Cadênci a e frequência

Os dados estão disponíveis em um horário fixo ou você pode acessar novos dados quando quiser?

Estruturas comuns

Os engenheiros de software e dados usam o mesmo gerenciador de contêineres, como o Kubernetes? Faria sentido integrar aplicações e cargas de trabalho de dados no mesmo cluster Kubernetes? Se você estiver usando uma estrutura de orquestração como o Airflow, faz sentido integrá-la à equipe de aplicativos upstream? Não há uma resposta correta aqui, mas você precisa equilibrar os benefícios da integração com os riscos do acoplamento rígido.

Engenharia de software

À medida que o cenário de dados muda para ferramentas que simplificam e automatizam o acesso aos sistemas de origem, você provavelmente precisará escrever código. Aqui estão algumas considerações ao escrever código para acessar um sistema de origem:

rede

Certifique-se de que seu código será capaz de acessar a rede onde reside o sistema de origem. Além disso, sempre pense em redes seguras. Você está acessando um URL HTTPS pela Internet pública, SSH ou VPN?

Autenticação e autorização

Você tem as credenciais adequadas (tokens, nome de usuário/senhas) para acessar o sistema de origem? Onde você armazenará essas credenciais para que não apareçam em seu código ou controle de versão? Você tem as funções IAM corretas para executar as tarefas codificadas?

padrões de acesso

Como você está acessando os dados? Você está usando uma API e como está lidando com solicitações REST/GraphQL, volumes de dados de resposta e paginação? Se você estiver acessando dados por meio de um driver de banco de dados, o driver é compatível com o banco de dados que você está acessando? Para qualquer padrão de acesso, como coisas como novas tentativas e tempos limite são tratados?

Orquestração

Seu código se integra a uma estrutura de orquestração e pode ser executado como um fluxo de trabalho orquestrado?

paralelização

Como você está gerenciando e dimensionando o acesso paralelo aos sistemas de origem?

Implantação

Como você está lidando com a implantação das alterações no código-fonte?

Conclusão

Os sistemas de origem e seus dados são vitais no ciclo de vida da engenharia de dados. Os engenheiros de dados tendem a tratar os sistemas de origem como “problema de outra pessoa” — faça isso por sua conta e risco! Os engenheiros de dados que abusam dos sistemas de origem podem precisar procurar outro emprego quando a produção cair.

Se tem pau, tem cenoura também. Uma melhor colaboração com as equipes do sistema de origem pode levar a dados de maior qualidade, resultados mais bem-sucedidos e melhores produtos de dados. Crie um fluxo bidirecional de comunicações com seus colegas nessas equipes; configurar processos para notificar alterações de esquema e aplicativo que afetam

análise e ML. Comunique suas necessidades de dados proativamente para auxiliar as equipes de aplicativos no processo de engenharia de dados.

Esteja ciente de que a integração entre engenheiros de dados e equipes de sistemas de origem está crescendo. Um exemplo é o ETL reverso, que viveu por muito tempo nas sombras, mas recentemente ganhou destaque. Também discutimos que a plataforma de transmissão de eventos poderia desempenhar um papel em arquiteturas e análises orientadas a eventos; um sistema de origem também pode ser um sistema de engenharia de dados. Construir sistemas compartilhados onde faz sentido fazê-lo.

Procure oportunidades para criar produtos de dados voltados para o usuário. converse com as equipes de aplicativos sobre análises que gostariam de apresentar aos usuários ou locais onde o ML poderia melhorar a experiência do usuário. Torne as equipes de aplicativos partes interessadas na engenharia de dados e encontre maneiras de compartilhar seus sucessos.

Agora que você entende os tipos de sistemas de origem e os dados que eles geram, veremos a seguir maneiras de armazenar esses dados.

Recursos adicionais

- Confluentes [Documentação “Evolução e Compatibilidade do Esquema”](#)
- *Internos do banco de dados* por Alex Petrov (O'Reilly)
- *Conceitos do sistema de banco de dados* por Abraham (Avi) Silberschatz et al. (Mc Graw Hill)
- “O registro: o que todo engenheiro de software deve saber sobre a abstração unificada de dados em tempo real” por Jay Kreps
- “Modernizando a indexação de dados de negócios” por Benjamin Douglas e Mohammad Mohtasham
- “NoSQL: o que há em um nome” por Eric Evans
- “Teste a qualidade dos dados em escala com Deequ” por Dustin Lange e outros.
- “O que, por que e quando do design de tabela única com o DynamoDB” por Alex DeBrie

Armazenar

O armazenamento é a pedra angular do ciclo de vida da engenharia de dados (Figura 6-1) e fundamenta seus principais estágios - ingestão, transformação e serviço. Os dados são armazenados muitas vezes ao longo do ciclo de vida. Parafraseando um velho ditado, é armazenamento até o fim. Quer os dados sejam necessários segundos, minutos, dias, meses ou anos depois, eles devem persistir no armazenamento até que os sistemas estejam prontos para consumi-los para processamento e transmissão adicionais. Conhecer o caso de uso dos dados e a forma como você os recuperará no futuro é o primeiro passo para escolher as soluções de armazenamento adequadas para sua arquitetura de dados.

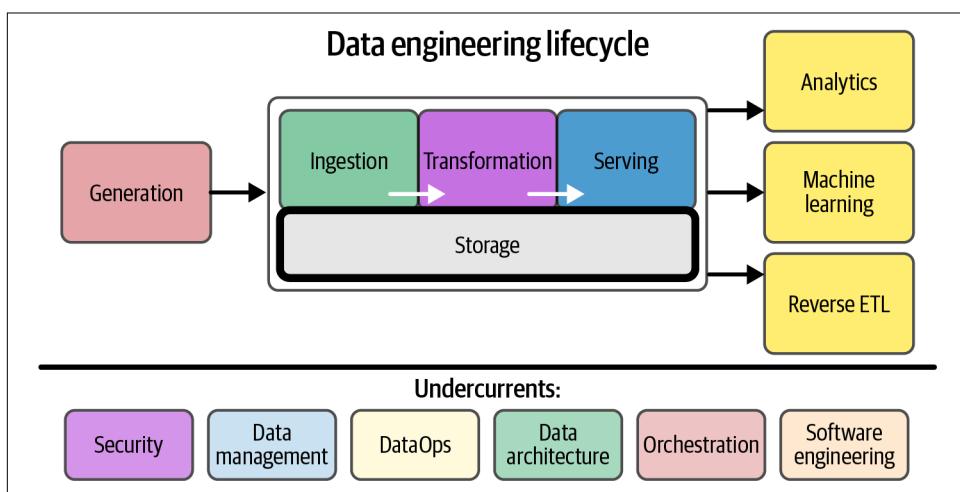


Figura 6-1. O armazenamento desempenha um papel central no ciclo de vida da engenharia de dados

Também discutimos o armazenamento em [capítulo 5](#), mas com uma diferença de foco e domínio de controle. Os sistemas de origem geralmente não são mantidos ou controlados por engenheiros de dados. O armazenamento que os engenheiros de dados lidam diretamente, no qual focaremos neste capítulo, abrange os estágios do ciclo de vida da engenharia de dados, desde a ingestão de dados dos sistemas de origem até o fornecimento de dados para agregar valor com análise, ciência de dados etc. todo o ciclo de vida da engenharia de dados de alguma forma.

Para entender o armazenamento, vamos começar estudando os *ingredientes crus* que compõem os sistemas de armazenamento, incluindo discos rígidos, unidades de estado sólido e memória do sistema (consulte [Figura 6-2](#)). É essencial compreender as características básicas das tecnologias de armazenamento físico para avaliar as compensações inerentes a qualquer arquitetura de armazenamento. Esta seção também discute serialização e compactação, elementos-chave de software de armazenamento prático. (Deferimos uma discussão técnica mais profunda sobre serialização e compactação para [Apêndice A](#).) Também discutimos *cache*, o que é crítico na montagem de sistemas de armazenamento.

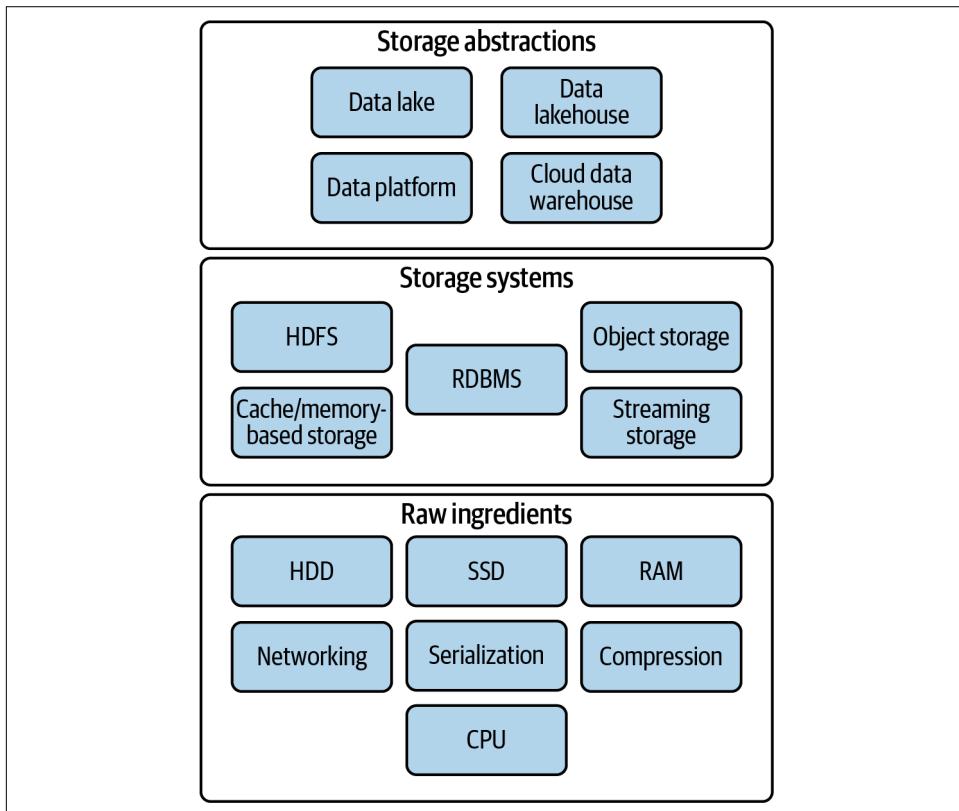


Figura 6-2. Ingredientes brutos, sistemas de armazenamento e abstrações de armazenamento

A seguir, veremos *sistemas de armazenamento*. Na prática, não acessamos diretamente a memória do sistema ou os discos rígidos. Esses componentes de armazenamento físico existem dentro de servidores e clusters que podem receber e recuperar dados usando vários paradigmas de acesso.

Por fim, veremos *abstrações de armazenamento*. Os sistemas de armazenamento são montados em um data warehouse em nuvem, um data lake, etc. Ao criar pipelines de dados, os engenheiros escolhem as abstrações apropriadas para armazenar seus dados à medida que passam pelos estágios de ingestão, transformação e serviço.

Ingredientes brutos de armazenamento de dados

O armazenamento é tão comum que é fácil considerá-lo um dado adquirido. Muitas vezes ficamos surpresos com o número de engenheiros de software e dados que usam o armazenamento todos os dias, mas não têm ideia de como ele funciona nos bastidores ou das vantagens e desvantagens inerentes a várias mídias de armazenamento. Como resultado, vemos o armazenamento sendo usado de algumas maneiras bastante interessantes. Embora os serviços gerenciados atuais liberem potencialmente os engenheiros de dados das complexidades do gerenciamento de servidores, os engenheiros de dados ainda precisam estar cientes das características essenciais dos componentes subjacentes, considerações de desempenho, durabilidade e custos.

Na maioria das arquiteturas de dados, os dados frequentemente passam por armazenamento magnético, SSDs e memória à medida que passam pelas várias fases de processamento de um pipeline de dados. Os sistemas de armazenamento e consulta de dados geralmente seguem receitas complexas envolvendo sistemas distribuídos, vários serviços e várias camadas de armazenamento de hardware. Esses sistemas requerem as matérias-primas certas para funcionar corretamente.

Vejamos alguns dos ingredientes básicos do armazenamento de dados: unidades de disco, memória, rede e CPU, serialização, compactação e cache.

Unidade de disco magnético

discos magnéticos utilizam pratos giratórios revestidos com um filme ferromagnético ([Figura 6-3](#)). Este filme é magnetizado por um cabeçote de leitura/gravação durante as operações de gravação para codificar fisicamente os dados binários. A cabeça de leitura/gravação detecta o campo magnético e emite um fluxo de bits durante as operações de leitura. As unidades de disco magnético existem há séculos. Eles ainda formam a espinha dorsal dos sistemas de armazenamento de dados em massa porque são significativamente mais baratos do que os SSDs por gigabyte de dados armazenados.

Por um lado, esses discos tiveram melhorias extraordinárias em desempenho, densidade de armazenamento e custo. Por outro lado, os SSDs superaram drasticamente os discos magnéticos em várias métricas. Atualmente, as unidades de disco magnético comerciais custam cerca de 3 centavos por gigabyte de capacidade. (Observe que frequentemente usaremos o

1 Andy Klein, "Unidade de disco rígido (HDD) vs. Unidade de estado sólido (SSD): Qual é a diferença?", Backblaze blog, 5 de outubro de 2021, <https://oreil.ly/XBps8>.

abreviaturas *disco rígido* e *SSD* para denotar disco magnético rotativo e unidades de estado sólido, respectivamente.)

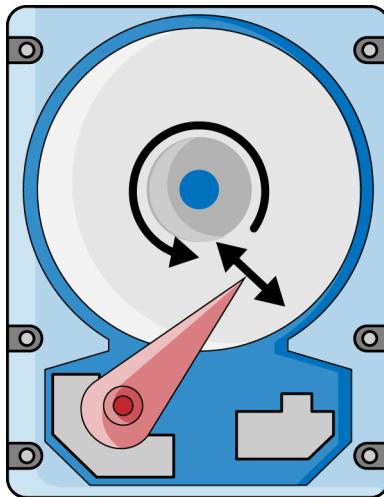


Figura 6-3. O movimento e a rotação da cabeça do disco magnético são essenciais na latência de acesso aleatório

A IBM desenvolveu a tecnologia de unidade de disco magnético na década de 1950. Desde então, as capacidades dos discos magnéticos cresceram constantemente. A primeira unidade de disco magnético comercial, o IBM 350, tinha uma capacidade de 3,75 megabytes. No momento em que este livro foi escrito, as unidades magnéticas que armazenam 20 TB estão disponíveis comercialmente. Na verdade, os discos magnéticos continuam a ter uma rápida inovação, com métodos como gravação magnética assistida por calor (HAMR), gravação magnética shingled (SMR) e compartimentos de disco preenchidos com hélio sendo usados para obter densidades de armazenamento cada vez maiores. Apesar das melhorias contínuas na capacidade do drive, outros aspectos do desempenho do HDD são prejudicados pela física.

Primeiro, *velocidade de transferência de disco*, a taxa na qual os dados podem ser lidos e gravados, não escala proporcionalmente à capacidade do disco. Escalas de capacidade de disco com *densidade de área* (gigabits armazenados por polegada quadrada), enquanto a velocidade de transferência aumenta com *densidade linear* (bits por polegada). Isso significa que, se a capacidade do disco aumentar em um fator de 4, a velocidade de transferência aumentará apenas em um fator de 2. Consequentemente, as unidades de data center atuais suportam velocidades máximas de transferência de dados de 200 a 300 MB/s. Para enquadrar isso de outra forma, leva mais de 20 horas para ler todo o conteúdo de um drive magnético de 30 TB, assumindo uma velocidade de transferência de 300 MB/s.

Uma segunda grande limitação é o tempo de busca. Para acessar os dados, a unidade deve realocar fisicamente os cabeçotes de leitura/gravação para a trilha apropriada no disco. Em terceiro lugar, para encontrar um dado específico no disco, o controlador de disco deve esperar que esses dados girem sob os cabeçotes de leitura/gravação. Isto leva a *latência rotacional*. Típica

unidades comerciais girando a 7.200 revoluções por minuto (RPM) tempo de busca e latência rotacional levam a mais de quatro milissegundos de latência média geral (tempo para acessar um dado selecionado). Uma quarta limitação são as operações de entrada/saída por segundo (IOPS), críticas para bancos de dados transacionais. Uma unidade magnética varia de 50 a 500 IOPS.

Vários truques podem melhorar a latência e a velocidade de transferência. O uso de uma velocidade rotacional mais alta pode aumentar a taxa de transferência e diminuir a latência rotacional. Limitar o raio do prato do disco ou gravar dados apenas em uma faixa estreita do disco reduz o tempo de busca. No entanto, nenhuma dessas técnicas torna as unidades magnéticas remotamente competitivas com os SSDs para pesquisas de acesso aleatório. Os SSDs podem fornecer dados com latência significativamente menor, IOPS mais alto e velocidades de transferência mais altas, em parte porque não há nenhum disco rotativo fisicamente ou cabeçote magnético para esperar.

Conforme mencionado anteriormente, os discos magnéticos ainda são valorizados nos data centers por seus baixos custos de armazenamento de dados. Além disso, as unidades magnéticas podem sustentar taxas de transferência extraordinariamente altas por meio do paralelismo. Esta é a ideia crítica por trás do armazenamento de objetos em nuvem: os dados podem ser distribuídos em milhares de discos em clusters. As taxas de transferência de dados aumentam drasticamente com a leitura de vários discos simultaneamente, limitadas principalmente pelo desempenho da rede e não pela taxa de transferência do disco. Portanto, os componentes de rede e as CPUs também são matérias-primas importantes nos sistemas de armazenamento, e voltaremos a esses tópicos em breve.

Disco de Estado Sólido

Unidades de estado sólido(SSDs) armazenam dados como cargas em células de memória flash. Os SSDs eliminam os componentes mecânicos das unidades magnéticas; os dados são lidos por meios puramente eletrônicos. Os SSDs podem pesquisar dados aleatórios em menos de 0,1 ms (100 microssegundos). Além disso, os SSDs podem dimensionar as velocidades de transferência de dados e IOPS dividindo o armazenamento em partições com vários controladores de armazenamento em execução em paralelo. SSDs comerciais podem suportar velocidades de transferência de muitos gigabytes por segundo e dezenas de milhares de IOPS.

Devido a essas características excepcionais de desempenho, os SSDs revolucionaram os bancos de dados transacionais e são o padrão aceito para implantações comerciais de sistemas OLTP. Os SSDs permitem que bancos de dados relacionais, como PostgreSQL, MySQL e SQL Server, lide com milhares de transações por segundo.

No entanto, os SSDs não são atualmente a opção padrão para armazenamento de dados analíticos em alta escala. Mais uma vez, isso se resume ao custo. Os SSDs comerciais geralmente custam de 20 a 30 centavos (USD) por gigabyte de capacidade, quase 10 vezes o custo por capacidade de uma unidade magnética. Assim, o armazenamento de objetos em discos magnéticos surgiu como a principal opção para armazenamento de dados em larga escala em data lakes e data warehouses em nuvem.

Os SSDs ainda desempenham um papel significativo nos sistemas OLAP. Alguns bancos de dados OLAP aproveitam o cache SSD para oferecer suporte a consultas de alto desempenho em dados acessados com frequência. À medida que o OLAP de baixa latência se torna mais popular, esperamos que o uso de SSD nesses sistemas siga o exemplo.

Memória de acesso aleatório

Normalmente usamos os termos *memória de acesso aleatório*(RAM) e *memória* intercambiavelmente. A rigor, drives magnéticos e SSDs também servem como memória que armazena dados para posterior recuperação por acesso aleatório, mas a RAM possui várias características específicas:

- É conectado a uma CPU e mapeado no espaço de endereço da CPU.
- Armazena o código que as CPUs executam e os dados que este código processa diretamente.
- Isso é *volátil*, enquanto unidades magnéticas e SSDs são *não volátil*. Embora possam ocasionalmente falhar e corromper ou perder dados, as unidades geralmente retêm os dados quando desligadas. A RAM perde dados em menos de um segundo quando está desenergizada.
- Ele oferece velocidades de transferência significativamente mais altas e tempos de recuperação mais rápidos do que o armazenamento SSD. A memória DDR5 — o mais recente padrão amplamente usado para RAM — oferece latência de recuperação de dados da ordem de 100 ns, cerca de 1.000 vezes mais rápido que o SSD. Uma CPU típica pode suportar largura de banda de 100 GB/s para memória anexada e milhões de IOPS. (As estatísticas variam drasticamente dependendo do número de canais de memória e outros detalhes de configuração.)
- É significativamente mais caro do que o armazenamento SSD, aproximadamente US\$ 10/GB (no momento em que este livro foi escrito).
- É limitado na quantidade de RAM anexada a uma CPU individual e controlador de memória. Isso aumenta ainda mais a complexidade e o custo. Servidores com muita memória normalmente utilizam muitas CPUs interconectadas em uma placa, cada uma com um bloco de RAM anexado.
- Ainda é significativamente mais lento que o cache da CPU, um tipo de memória localizada diretamente na matriz da CPU ou no mesmo pacote. O cache armazena dados acessados com frequência e recentemente para recuperação ultrarrápida durante o processamento. Os projetos de CPU incorporam várias camadas de cache de tamanhos e características de desempenho variados.

Quando falamos de memória do sistema, quase sempre queremos dizer *RAM dinâmica*, uma forma de memória de alta densidade e baixo custo. A RAM dinâmica armazena dados como cargas em capacitores. Esses capacitores vazam com o tempo, então os dados devem ser frequentemente *atualizado* (ler e reescrever) para evitar perda de dados. O controlador de memória de hardware lida com esses detalhes técnicos; os engenheiros de dados simplesmente precisam se preocupar com as características de largura de banda e latência de recuperação. Outras formas de memória, como *RAM estática*, são usados em aplicativos especializados, como caches de CPU.

CPUs atuais praticamente sempre empregam *o arquitetura von Neumann*, com código e dados armazenados juntos no mesmo espaço de memória. No entanto, as CPUs normalmente também suportam a opção de desabilitar a execução de código em páginas específicas de memória para maior segurança. Esta característica lembra *o arquitetura de Harvard*, que separa código e dados.

A RAM é usada em vários sistemas de armazenamento e processamento e pode ser usada para armazenamento em cache, processamento de dados ou índices. Vários bancos de dados tratam a RAM como uma camada de armazenamento primário, permitindo desempenho ultrarrápido de leitura e gravação. Nesses aplicativos, os engenheiros de dados devem sempre ter em mente a volatilidade da RAM. Mesmo que os dados armazenados na memória sejam replicados em um cluster, uma queda de energia que desative vários nós pode causar perda de dados. As arquiteturas destinadas a armazenar dados de forma durável podem usar backups de bateria e despejar automaticamente todos os dados no disco em caso de perda de energia.

Rede e CPU

Por que estamos mencionando a rede e a CPU como ingredientes básicos para armazenar dados? Cada vez mais, os sistemas de armazenamento são distribuídos para melhorar o desempenho, a durabilidade e a disponibilidade. Mencionamos especificamente que os discos magnéticos individuais oferecem desempenho de transferência relativamente baixo, mas um cluster de discos paraleliza as leituras para uma escala de desempenho significativa. Enquanto os padrões de armazenamento, como matrizes redundantes de discos independentes (RAID), são paralelizados em um único servidor, os clusters de armazenamento de objetos em nuvem operam em uma escala muito maior, com discos distribuídos em uma rede e até mesmo em vários centros de dados e zonas de disponibilidade.

zonas de disponibilidadesão uma construção de nuvem padrão que consiste em ambientes de computação com energia, água e outros recursos independentes. O armazenamento multizonal aumenta a disponibilidade e a durabilidade dos dados.

As CPUs lidam com os detalhes das solicitações de serviço, agregando leituras e distribuindo gravações. O armazenamento se torna um aplicativo da Web com uma API, componentes de serviço de back-end e balanceamento de carga. O desempenho do dispositivo de rede e a topologia da rede são fatores-chave na obtenção de alto desempenho.

Os engenheiros de dados precisam entender como a rede afetará os sistemas que eles constroem e usam. Os engenheiros equilibram constantemente a durabilidade e a disponibilidade alcançadas ao distribuir os dados geograficamente versus os benefícios de desempenho e custo de manter o armazenamento em uma pequena área geográfica e próximo aos consumidores ou gravadores de dados. [Apêndice Babrange](#) redes em nuvem e as principais ideias relevantes.

Serialização

Serializaçãoé outro ingrediente de armazenamento bruto e um elemento crítico do design do banco de dados. As decisões sobre a serialização informarão o desempenho das consultas em uma rede, a sobrecarga da CPU, a latência da consulta e muito mais. Projetar um data lake, por

Por exemplo, envolve a escolha de um sistema de armazenamento básico (por exemplo, Amazon S3) e padrões para serialização que equilibram a interoperabilidade com considerações de desempenho.

O que é serialização, exatamente? Os dados armazenados na memória do sistema pelo software geralmente não estão em um formato adequado para armazenamento em disco ou transmissão em uma rede. A serialização é o processo de compactação e compactação de dados em um formato padrão que um leitor será capaz de decodificar. Os formatos de serialização fornecem um padrão de troca de dados. Podemos codificar dados de maneira baseada em linha como um arquivo XML, JSON ou CSV e passá-lo para outro usuário que pode decodificá-lo usando uma biblioteca padrão. Um algoritmo de serialização possui lógica para lidar com tipos, impõe regras na estrutura de dados e permite a troca entre linguagens de programação e CPUs. O algoritmo de serialização também possui regras para lidar com exceções. Por exemplo, objetos Python podem conter referências cíclicas;

O armazenamento de banco de dados de baixo nível também é uma forma de serialização. Bancos de dados relacionais orientados a linha organizam dados como linhas no disco para oferecer suporte a pesquisas rápidas de atualizações no local. Os bancos de dados colulares organizam os dados em arquivos de colunas para otimizar a compactação altamente eficiente e dar suporte a verificações rápidas de grandes volumes de dados. Cada opção de serialização vem com um conjunto de compensações e os engenheiros de dados ajustam essas opções para otimizar o desempenho de acordo com os requisitos.

Fornecemos um catálogo mais detalhado de técnicas e formatos comuns de serialização de dados em [Apêndice A](#). Sugerimos que os engenheiros de dados se familiarizem com práticas e formatos comuns de serialização, especialmente os formatos atuais mais populares (por exemplo, Apache Parquet), serialização híbrida (por exemplo, Apache Hudi) e serialização na memória (por exemplo, Apache Arrow).

Compressão

Compressão é outro componente crítico da engenharia de armazenamento. Em um nível básico, a compactação torna os dados menores, mas os algoritmos de compactação interagem com outros detalhes dos sistemas de armazenamento de maneiras complexas.

A compactação altamente eficiente tem três vantagens principais em sistemas de armazenamento. Primeiro, os dados são menores e, portanto, ocupam menos espaço no disco. Em segundo lugar, a compactação aumenta a velocidade prática de varredura por disco. Com uma taxa de compactação de 10:1, passamos de uma varredura de 200 MB/s por disco magnético para uma taxa efetiva de 2 GB/s por disco.

A terceira vantagem está no desempenho da rede. Dado que uma conexão de rede entre uma instância do Amazon EC2 e S3 fornece 10 gigabits por segundo (Gbps) de largura de banda, uma taxa de compactação de 10:1 aumenta a largura de banda efetiva da rede para 100 Gbps.

A compactação também vem com desvantagens. A compactação e descompactação de dados envolve tempo extra e consumo de recursos para ler ou gravar dados. Empreendemos uma discussão mais detalhada sobre algoritmos de compressão e compensações em [Apêndice A](#).

Cache

Já mencionamos o cache em nossa discussão sobre RAM. A ideia central do cache é armazenar dados acessados com frequência ou recentemente em uma camada de acesso rápido. Quanto mais rápido o cache, maior o custo e menos espaço de armazenamento disponível. Os dados acessados com menos frequência são armazenados em um armazenamento mais barato e mais lento. Os caches são essenciais para o fornecimento, processamento e transformação de dados.

À medida que analisamos os sistemas de armazenamento, é útil colocar cada tipo de armazenamento que utilizamos dentro de um *hierarquia de cache* ([Tabela 6-1](#)). A maioria dos sistemas de dados práticos depende de muitas camadas de cache montadas a partir do armazenamento com características de desempenho variadas. Isso começa dentro das CPUs; os processadores podem implantar até quatro camadas de cache. Descemos na hierarquia para RAM e SSDs. O armazenamento de objetos em nuvem é um nível inferior que oferece suporte à retenção e durabilidade de dados a longo prazo, ao mesmo tempo em que permite o fornecimento de dados e a movimentação dinâmica de dados em pipelines.

Tabela 6-1. Uma hierarquia de cache heurística exibindo tipos de armazenamento com preço aproximado e características de desempenho

Tipo de armazenamento	Latência de busca de dados	Largura de banda	Preço
Cache da CPU	1 nanosegundo	1 TB/s	N / D
BATER	0,1 microsegundos	100 GB/s	\$ 10/GB
SSD	0,1 milissegundos	4 GB/s	US\$ 0,20/GB
disco rígido	4 milissegundos	300 MB/s	US\$ 0,03/GB
Armazenamento de objetos	100 milissegundos	10 GB/s	US\$ 0,02/GB por mês
Armazenamento de arquivo	12 horas	Igual ao armazenamento de objetos quando os dados estiverem disponíveis	US\$ 0,004/GB por mês

^aUm microsegundo é 1.000 nanosegundos e um milissegundo é 1.000 microsegundos.

Podemos pensar no armazenamento de arquivos como um *cache reverso*. O armazenamento de arquivo fornece características de acesso inferiores para custos baixos. O armazenamento de arquivo geralmente é usado para backups de dados e para atender aos requisitos de conformidade de retenção de dados. Em cenários típicos, esses dados serão acessados apenas em caso de emergência (por exemplo, dados em um banco de dados podem ser perdidos e precisam ser recuperados, ou uma empresa pode precisar consultar dados históricos para fins legais).

Sistemas de armazenamento de dados

Esta seção aborda os principais sistemas de armazenamento de dados que você encontrará como engenheiro de dados. Os sistemas de armazenamento existem em um nível de abstração acima dos ingredientes brutos. Por exemplo, os discos magnéticos são um ingrediente de armazenamento bruto, enquanto os principais objetos de armazenamento em nuvem

plataformas e HDFS são sistemas de armazenamento que utilizam discos magnéticos. Existem níveis ainda mais altos de abstração de armazenamento, como data lakes e lakehouses (que abordamos em “[Abstrações de armazenamento de engenharia de dados](#)” na página 215).

Máquina única versus armazenamento distribuído

À medida que o armazenamento de dados e os padrões de acesso se tornam mais complexos e superam a utilidade de um único servidor, a distribuição de dados para mais de um servidor torna-se necessária. Os dados podem ser armazenados em vários servidores, conhecidos como *armazenamento distribuído*. Este é um sistema distribuído cuja finalidade é armazenar dados de forma distribuída ([Figura 6-4](#)).

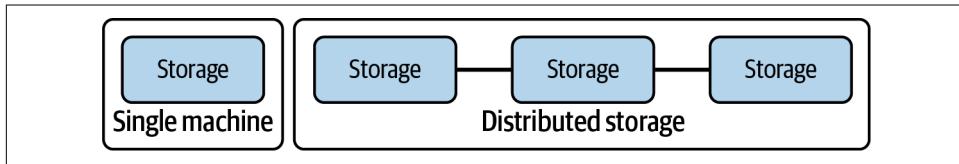


Figura 6-4. Máquina única versus armazenamento distribuído em vários servidores

O armazenamento distribuído coordena as atividades de vários servidores para armazenar, recuperar e processar dados mais rapidamente e em maior escala, ao mesmo tempo em que fornece redundância caso um servidor fique indisponível. O armazenamento distribuído é comum em arquiteturas nas quais você deseja redundância e escalabilidade integradas para grandes quantidades de dados. Por exemplo, armazenamento de objetos, Apache Spark e armazéns de dados em nuvem dependem de arquiteturas de armazenamento distribuído.

Os engenheiros de dados devem sempre estar cientes dos paradigmas de consistência dos sistemas distribuídos, que exploraremos a seguir.

Consistência eventual versus forte

Um desafio com sistemas distribuídos é que seus dados estão espalhados por vários servidores. Como esse sistema mantém os dados consistentes? Infelizmente, os sistemas distribuídos representam um dilema para armazenamento e precisão de consulta. Leva tempo para replicar as alterações nos nós de um sistema; geralmente existe um equilíbrio entre obter dados atuais e obter dados atuais “tipo” em um banco de dados distribuído. Vejamos dois padrões de consistência comuns em sistemas distribuídos: eventual e forte.

Cobrimos a conformidade ACID ao longo deste livro, começando em [capítulo 5](#). Outra sigla é *BASE*, que significa *basicamente disponível, soft-state, consistência eventual*. Pense nisso como o oposto do ACID. BASE é a base da consistência eventual. Vamos explorar brevemente seus componentes:

Basicamente disponível

A consistência não é garantida, mas as leituras e gravações do banco de dados são feitas com base no melhor esforço, o que significa que dados consistentes estão disponíveis na maior parte do tempo.

Soft-state

O estado da transação é difuso e é incerto se a transação foi confirmada ou não.

Consistência eventual

No *algum* ponto, a leitura dos dados retornará valores consistentes.

Se a leitura de dados em um sistema eventualmente consistente não é confiável, por que usá-la? A consistência eventual é uma compensação comum em sistemas distribuídos de grande escala. Se você deseja dimensionar horizontalmente (em vários nós) para processar dados em grandes volumes, a consistência geralmente é o preço a pagar. A consistência eventual permite que você recupere dados rapidamente sem verificar se possui a versão mais recente em todos os nós.

O oposto de consistência eventual é *consistência forte*. Com consistência forte, o banco de dados distribuído garante que as gravações em qualquer nó sejam primeiro distribuídas com um consenso e que todas as leituras no banco de dados retornem valores consistentes. Você usará consistência forte quando puder tolerar uma latência de consulta mais alta e exigir dados corretos toda vez que ler do banco de dados.

Geralmente, os engenheiros de dados tomam decisões sobre consistência em três lugares. Primeiro, a própria tecnologia de banco de dados prepara o terreno para um certo nível de consistência. Em segundo lugar, os parâmetros de configuração do banco de dados terão um impacto na consistência. Em terceiro lugar, os bancos de dados geralmente oferecem suporte a alguma configuração de consistência em um nível de consulta individual. Por exemplo, [DynamoDB Name](#) suporta leituras eventualmente consistentes e leituras fortemente consistentes. As leituras fortemente consistentes são mais lentas e consomem mais recursos, por isso é melhor usá-las com moderação, mas elas estarão disponíveis quando a consistência for necessária.

Você deve entender como seu banco de dados lida com a consistência. Mais uma vez, os engenheiros de dados têm a tarefa de entender profundamente a tecnologia e usá-la para resolver problemas adequadamente. Um engenheiro de dados pode precisar negociar requisitos de consistência com outras partes interessadas técnicas e de negócios. Observe que esse é um problema tecnológico e organizacional; certifique-se de ter reunido os requisitos de suas partes interessadas e de escolher suas tecnologias adequadamente.

Armazenamento de arquivo

Lidamos com arquivos todos os dias, mas a noção de arquivo é um tanto sutil. A *arquivo* é uma entidade de dados com características específicas de leitura, gravação e referência usadas por software e sistemas operacionais. Definimos um arquivo para ter as seguintes características:

comprimento finito

Um arquivo é um fluxo de bytes de tamanho finito.

Acrescentar operações

Podemos anexar bytes ao arquivo até os limites do sistema de armazenamento do host.

Acesso aleatório

Podemos ler de qualquer local no arquivo ou gravar atualizações em qualquer local.

Armazenamento de objetos se comporta de maneira muito semelhante ao armazenamento de arquivos, mas com diferenças importantes. Embora tenhamos definido o cenário para o armazenamento de objetos discutindo primeiro o armazenamento de arquivos, o armazenamento de objetos é indiscutivelmente muito mais importante para o tipo de engenharia de dados que você fará hoje. Iremos referenciar a discussão sobre armazenamento de objetos extensivamente nas próximas páginas.

Os sistemas de armazenamento de arquivos organizam os arquivos em uma árvore de diretórios. A referência de diretório para um arquivo pode ser assim:

/Users/matthewousley/output.txt

Quando essa referência de arquivo é passada para o sistema operacional, ela inicia no diretório raiz /, encontra Usuários, matthewousley, e finalmente output.txt. Trabalhando a partir da esquerda, cada diretório está contido dentro de um diretório pai, até que finalmente chegamos ao arquivo output.txt. Este exemplo usa a semântica do Unix, mas a semântica de referência de arquivo do Windows é semelhante. O sistema de arquivos armazena cada diretório como metadados sobre os arquivos e diretórios que ele contém. Esses metadados consistem no nome de cada entidade, detalhes de permissão relevantes e um ponteiro para a entidade real. Para localizar um arquivo no disco, o sistema operacional examina os metadados em cada nível hierárquico e segue o ponteiro para a próxima entidade do subdiretório até finalmente chegar ao próprio arquivo.

Observe que outras entidades de dados semelhantes a arquivos geralmente não têm necessariamente todas essas propriedades. Por exemplo, *objetos* no armazenamento de objetos suportam apenas a primeira característica, comprimento finito, mas ainda são extremamente úteis. Discutimos isso em "["Armazenamento de objetos"](#) na página 205.

Nos casos em que os paradigmas de armazenamento de arquivos são necessários para um pipeline, tenha cuidado com o estado e tente usar ambientes efêmeros o máximo possível. Mesmo se você precisar processar arquivos em um servidor com um disco conectado, use o armazenamento de objeto para armazenamento intermediário entre as etapas de processamento. Tente reservar o processamento manual de arquivos de baixo nível para etapas únicas de ingestão ou estágios exploratórios de desenvolvimento de pipeline.

Armazenamento em disco local

O tipo mais conhecido de armazenamento de arquivos é um sistema de arquivos gerenciado pelo sistema operacional em uma partição de disco local de SSD ou disco magnético. O New Technology File System (NTFS) e o ext4 são sistemas de arquivos populares no Windows e no Linux, respectivamente. O sistema operacional lida com os detalhes de armazenamento de entidades de diretório, arquivos e metadados. Os sistemas de arquivos são projetados para gravar dados para permitir uma recuperação fácil em caso de perda de energia durante uma gravação, embora quaisquer dados não gravados ainda sejam perdidos.

Os sistemas de arquivos locais geralmente suportam leitura completa após consistência de gravação; a leitura imediatamente após uma gravação retornará os dados gravados. Os sistemas operacionais também empregam várias estratégias de bloqueio para gerenciar tentativas de gravação simultâneas em um arquivo.

Sistemas de arquivos de disco local também podem oferecer suporte a recursos avançados, como registro em diário, instantâneos, redundância, extensão do sistema de arquivos em vários discos, criptografia de disco completo e compactação. Em “[Armazenamento em bloco](#)” na [página 202](#), também discutimos o RAID.

Armazenamento conectado à rede

Armazenamento conectado à rede(NAS) fornecem um sistema de armazenamento de arquivos para clientes em uma rede. NAS é uma solução predominante para servidores; eles geralmente são fornecidos com hardware de interface NAS dedicado integrado. Embora haja penalidades de desempenho para acessar sistema de arquivos em uma rede, também existem vantagens significativas para a virtualização de armazenamento, incluindo redundância e confiabilidade, controle refinado de recursos, pool de armazenamento em vários discos para grandes volumes virtuais e compartilhamento de arquivos em vários máquinas. Os engenheiros devem estar cientes do modelo de consistência fornecido por sua solução NAS, especialmente quando vários clientes potencialmente acessarão os mesmos dados.

Uma alternativa popular ao NAS é uma rede de área de armazenamento (SAN), mas os sistemas SAN fornecem acesso em nível de bloco sem a abstração do sistema de arquivos. Cobrimos sistemas SAN em “[Armazenamento em bloco](#)” na [página 202](#).

Serviços de sistema de arquivos em nuvem

Os serviços de sistema de arquivos em nuvem fornecem um sistema de arquivos totalmente gerenciado para uso com várias VMs e aplicativos em nuvem, incluindo potencialmente clientes fora do ambiente de nuvem. Os sistemas de arquivos em nuvem não devem ser confundidos com armazenamento padrão anexado a VMs — geralmente, armazenamento em bloco com um sistema de arquivos gerenciado pelo sistema operacional da VM. Os sistemas de arquivos em nuvem se comportam como soluções NAS, mas os detalhes de rede, gerenciamento de clusters de disco, falhas e configuração são totalmente controlados pelo fornecedor de nuvem.

Por exemplo, Amazon Elastic File System (EFS) é um exemplo extremamente popular de um serviço de sistema de arquivos em nuvem. O armazenamento é exposto através do [protocolo NFS4](#), que também é usado por sistemas NAS. O EFS fornece dimensionamento automático e preços de pagamento por armazenamento sem a necessidade de reserva de armazenamento avançada. O serviço também oferece/*local*/ consistência de leitura após gravação (ao ler da máquina que executou a gravação). Ele também oferece consistência de abertura após fechamento em todo o sistema de arquivos. Em outras palavras, quando um aplicativo fecha um arquivo, os leitores subsequentes verão as alterações salvas no arquivo fechado.

Armazenamento em bloco

Fundamentalmente, *armazenamento em bloco* é o tipo de armazenamento bruto fornecido por SSDs e discos magnéticos. Na nuvem, o armazenamento de blocos virtualizados é o padrão para VMs. Essas abstrações de armazenamento em bloco permitem um controle preciso do tamanho do armazenamento, escalabilidade e durabilidade dos dados além do oferecido pelos discos brutos.

Em nossa discussão anterior sobre SSDs e discos magnéticos, mencionamos que, com esses dispositivos de acesso aleatório, o sistema operacional pode buscar, ler e gravar quaisquer dados no disco. *Abloquear* é a menor unidade endereçável de dados suportada por um disco. Isso costumava ser 512 bytes de dados utilizáveis em discos mais antigos, mas agora cresceu para 4.096 bytes para a maioria dos discos atuais, tornando as gravações menos granulares, mas reduzindo drasticamente a sobrecarga de gerenciamento de blocos. Os blocos geralmente contêm bits extras para detecção/correção de erros e outros metadados.

Blocos em discos magnéticos são arranjados geometricamente em um prato físico. Dois blocos na mesma trilha podem ser lidos sem mover a cabeça, enquanto a leitura de dois blocos em trilhas separadas requer uma busca. O tempo de busca pode ocorrer entre blocos em um SSD, mas isso é infinitesimal em comparação com o tempo de busca para trilhas de disco magnético.

Aplicativos de armazenamento em bloco

Os sistemas de banco de dados transacionais geralmente acessam discos em um nível de bloco para dispor os dados para desempenho ideal. Para bancos de dados orientados a linhas, isso significava originalmente que as linhas de dados eram gravadas como fluxos contínuos; a situação ficou mais complicada com a chegada dos SSDs e suas melhorias de desempenho de tempo de busca associadas, mas os bancos de dados transacionais ainda contam com o alto desempenho de acesso aleatório oferecido pelo acesso direto a um dispositivo de armazenamento em bloco.

O armazenamento em bloco também continua sendo a opção padrão para discos de inicialização do sistema operacional em VMs na nuvem. O dispositivo de bloco é formatado da mesma forma que seria diretamente em um disco físico, mas o armazenamento geralmente é virtualizado. (Ver “[Armazenamento de blocos virtualizados em nuvem](#)” na página 203.)

ATAQUE

ATAQUE apoia *Matriz redundante de discos independentes*, como notado anteriormente. O RAID controla simultaneamente vários discos para melhorar a durabilidade dos dados, aprimorar o desempenho e combinar a capacidade de várias unidades. Uma matriz pode aparecer para o sistema operacional como um único dispositivo de bloco. Muitos esquemas de codificação e paridade estão disponíveis, dependendo do equilíbrio desejado entre largura de banda efetiva aprimorada e tolerância a falhas mais alta (tolerância para muitas falhas de disco).

Rede da área de armazenamento

Rede da área de armazenamento(SAN) fornecem dispositivos de armazenamento de blocos virtualizados em uma rede, normalmente a partir de um pool de armazenamento. A abstração de SAN pode permitir dimensionamento de armazenamento refinado e aprimorar o desempenho, a disponibilidade e a durabilidade. Você pode encontrar sistemas SAN se estiver trabalhando com sistemas de armazenamento locais; você também pode encontrar uma versão em nuvem do SAN, como na próxima subseção.

Armazenamento de blocos virtualizados em nuvem

*Armazenamento de blocos virtualizados em nuvem*As soluções são semelhantes à SAN, mas liberam os engenheiros de lidar com clusters SAN e detalhes de rede. Veremos o Amazon Elastic Block Store (EBS) como um exemplo padrão; outras nuvens públicas têm ofertas semelhantes. EBS é o armazenamento padrão para máquinas virtuais Amazon EC2; outros provedores de nuvem também tratam o armazenamento de objetos virtualizados como um componente chave de suas ofertas de VM.

A EBS oferece vários níveis de serviço com diferentes características de desempenho. Geralmente, as métricas de desempenho do EBS são dadas em IOPS e taxa de transferência (velocidade de transferência). Os níveis de desempenho mais altos do armazenamento EBS são suportados por discos SSD, enquanto o armazenamento com suporte de disco magnético oferece IOPS mais baixo, mas custa menos por gigabyte.

Os volumes EBS armazenam dados separados do servidor host da instância, mas na mesma zona para oferecer suporte a alto desempenho e baixa latência ([Figura 6-5](#)). Isso permite que os volumes do EBS persistam quando uma instância do EC2 é encerrada, quando um servidor host falha ou até mesmo quando a instância é excluída. O armazenamento EBS é adequado para aplicativos como bancos de dados, onde a durabilidade dos dados é uma alta prioridade. Além disso, o EBS replica todos os dados para pelo menos duas máquinas host separadas, protegendo os dados se um disco falhar.

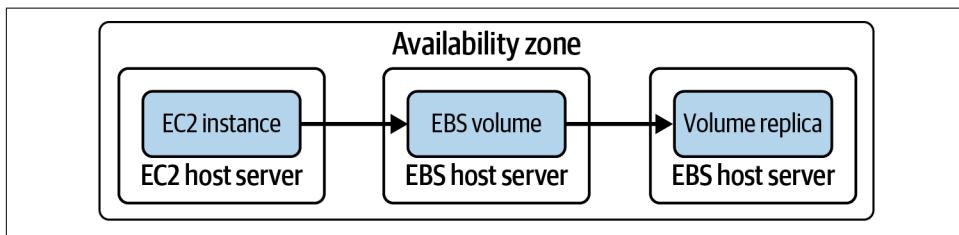


Figura 6-5. Os volumes EBS replicam dados para vários hosts e discos para alta durabilidade e disponibilidade, mas não são resistentes à falha de uma zona de disponibilidade

A virtualização de armazenamento EBS também oferece suporte a vários recursos avançados. Por exemplo, os volumes do EBS permitem snapshots instantâneos de um determinado momento enquanto a unidade é usada. Embora ainda leve algum tempo para que o instantâneo seja replicado para o S3, o EBS pode efetivamente congelar o estado dos blocos de dados quando o instantâneo é obtido, enquanto permite que a máquina cliente continue usando o disco. Além disso, os instantâneos após o backup completo inicial são diferenciais; apenas os blocos alterados são gravados no S3 para minimizar os custos de armazenamento e o tempo de backup.

Os volumes EBS também são altamente escaláveis. No momento da redação deste artigo, algumas classes de volume do EBS podem ser dimensionadas para até 64 TiB, 256.000 IOPS e 4.000 MiB/s.

Volumes de instância local

Os provedores de nuvem também oferecem volumes de armazenamento em bloco fisicamente conectados ao servidor host que executa uma máquina virtual. Esses volumes de armazenamento geralmente têm um custo muito baixo (incluído no preço da VM no caso do armazenamento de instância EC2 da Amazon) e fornecem baixa latência e alto IOPS.

Volumes de armazenamento de instâncias ([Figura 6-6](#)) se comportam essencialmente como um disco fisicamente conectado a um servidor em um data center. Uma diferença importante é que, quando uma VM é desligada ou excluída, o conteúdo do disco anexado localmente é perdido, independentemente de esse evento ter sido causado por uma ação intencional do usuário. Isso garante que uma nova máquina virtual não possa ler o conteúdo do disco pertencente a um cliente diferente.

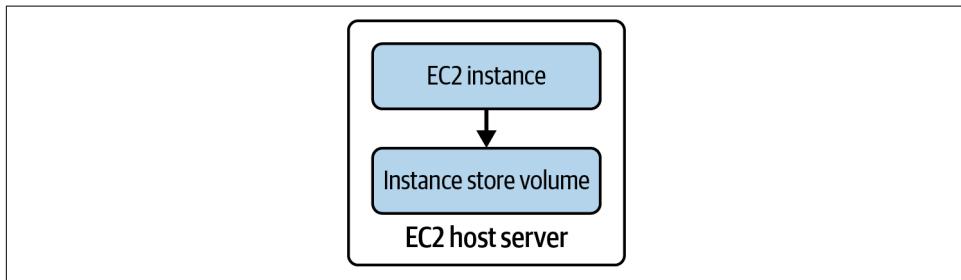


Figura 6-6. Os volumes de armazenamento de instâncias oferecem alto desempenho e baixo custo, mas não protegem os dados em caso de falha de disco ou desligamento da VM

Os discos conectados localmente não oferecem suporte a nenhum dos recursos avançados de virtualização oferecidos pelos serviços de armazenamento virtualizado, como o EBS. O disco anexado localmente não é replicado, portanto, uma falha de disco físico pode perder ou corromper dados, mesmo que a VM do host continue em execução. Além disso, os volumes anexados localmente não oferecem suporte a instantâneos ou outros recursos de backup.

Apesar dessas limitações, os discos conectados localmente são extremamente úteis. Em muitos casos, usamos discos como cache local e, portanto, não precisamos de todos os recursos avançados de virtualização de um serviço como o EBS. Por exemplo, suponha que estejamos executando o AWS EMR em instâncias do EC2. Podemos estar executando um trabalho efêmero que consome dados do S3, armazena-os temporariamente no sistema de arquivos distribuído em execução nas instâncias, processa os dados e grava os resultados de volta no S3. O sistema de arquivos EMR é construído em replicação e redundância e serve como cache em vez de armazenamento permanente. O armazenamento de instância EC2 é uma solução perfeitamente adequada neste caso e pode melhorar o desempenho, pois os dados podem ser lidos e processados localmente sem passar por uma rede (consulte [Figura 6-7](#)).

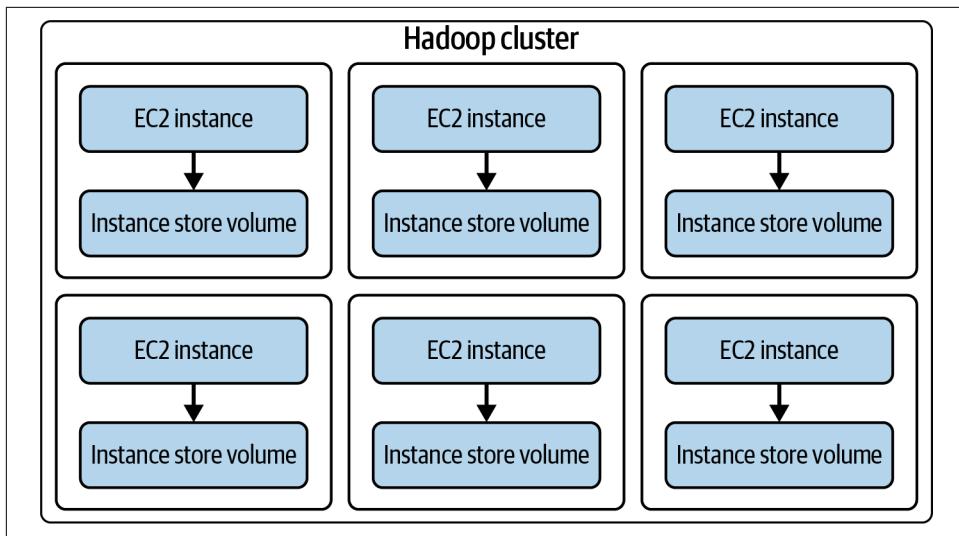


Figura 6-7. Os volumes de armazenamento de instâncias podem ser usados como um cache de processamento em um cluster Hadoop efêmero

Recomendamos que os engenheiros pensem no armazenamento conectado localmente nos piores cenários. Quais são as consequências de uma falha de disco local? De uma VM acidental ou desligamento do cluster? De uma interrupção de nuvem zonal ou regional? Se nenhum desses cenários tiver consequências catastróficas quando os dados em volumes anexados localmente forem perdidos, o armazenamento local pode ser uma opção econômica e de alto desempenho. Além disso, estratégias de mitigação simples (backups de ponto de verificação periódicos para S3) podem evitar a perda de dados.

Armazenamento de objetos

Armazenamento de objetos contém *objetos* de todas as formas e tamanhos (Figura 6-8). O termo *armazenamento de objetos* é meio confuso porque *objetos* têm vários significados em ciência da computação. Nesse contexto, estamos falando de uma construção especializada semelhante a um arquivo. Pode ser qualquer tipo de arquivo - TXT, CSV, JSON, imagens, vídeos ou áudio.

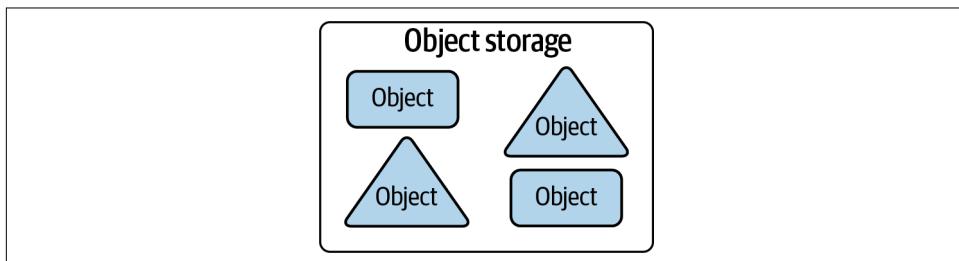


Figura 6-8. O armazenamento de objetos contém objetos imutáveis de todas as formas e tamanhos. Ao contrário dos arquivos em um disco local, os objetos não podem ser modificados no local.

Os armazenamentos de objetos cresceram em importância e popularidade com o surgimento do big data e da nuvem. Amazon S3, Azure Blob Storage e Google Cloud Storage (GCS) são armazenamentos de objetos amplamente usados. Além disso, muitos data warehouses em nuvem (e um número crescente de bancos de dados) utilizam o armazenamento de objetos como sua camada de armazenamento, e os data lakes em nuvem geralmente ficam em armazenamentos de objetos.

Embora muitos sistemas de armazenamento de objetos no local possam ser instalados em clusters de servidores, vamos nos concentrar principalmente em armazenamentos de objetos em nuvem totalmente gerenciados. Do ponto de vista operacional, uma das características mais atraentes do armazenamento de objetos em nuvem é que ele é simples de gerenciar e usar. O armazenamento de objetos foi indiscutivelmente um dos primeiros serviços “sem servidor”; os engenheiros não precisam considerar as características dos clusters ou discos de servidores subjacentes.

Um armazenamento de objeto é um armazenamento de chave-valor para objetos de dados imutáveis. Perdemos muito da flexibilidade de gravação que esperamos com o armazenamento de arquivos em um disco local em um armazenamento de objeto. Os objetos não oferecem suporte a gravações aleatórias ou operações de anexação; em vez disso, eles são gravados uma vez como um fluxo de bytes. Após essa gravação inicial, os objetos se tornam imutáveis. Para alterar dados em um objeto ou anexar dados a ele, devemos reescrever o objeto completo. Os armazenamentos de objetos geralmente oferecem suporte a leituras aleatórias por meio de solicitações de intervalo, mas essas pesquisas podem ter um desempenho muito pior do que leituras aleatórias de dados armazenados em um SSD.

Para um desenvolvedor de software acostumado a alavancar o armazenamento de arquivos de acesso aleatório local, as características dos objetos podem parecer restrições, mas menos é mais; os armazenamentos de objetos não precisam oferecer suporte a bloqueios ou alterar a sincronização, permitindo o armazenamento de dados em clusters de disco massivos. Os armazenamentos de objetos oferecem suporte a gravações e leituras de fluxo paralelo de alto desempenho em muitos discos, e esse paralelismo é oculto para os engenheiros, que podem simplesmente lidar com o fluxo em vez de se comunicar com discos individuais. Em um ambiente de nuvem, a velocidade de gravação aumenta com o número de fluxos sendo gravados até os limites de cota definidos pelo fornecedor. A largura de banda de leitura pode ser dimensionada com o número de solicitações paralelas, o número de máquinas virtuais empregadas para ler dados e o número de núcleos de CPU.

Armazenamentos típicos de objetos em nuvem salvam dados em várias zonas de disponibilidade, reduzindo drasticamente as chances de que o armazenamento fique totalmente off-line ou seja perdido de forma irrecuperável. Essa durabilidade e disponibilidade estão embutidas no custo; os fornecedores de armazenamento em nuvem oferecem outras classes de armazenamento a preços com desconto em troca de durabilidade ou disponibilidade reduzida. Discutiremos isso em “[Classes e camadas de armazenamento](#)” na página 220.

O armazenamento de objetos em nuvem é um ingrediente-chave na separação de computação e armazenamento, permitindo que os engenheiros processem dados com clusters efêmeros e dimensionem esses clusters para cima e para baixo sob demanda. Este é um fator chave para disponibilizar big data para organizações menores que não podem arcar com o custo de possuir hardware para trabalhos de dados que serão executados apenas ocasionalmente. Algumas grandes empresas de tecnologia continuarão a executar o Hadoop permanente

clusters em seu hardware. Ainda assim, a tendência geral é que a maioria das organizações moverá o processamento de dados para a nuvem, usando um armazenamento de objeto como armazenamento essencial e camada de serviço enquanto processa dados em clusters efêmeros.

No armazenamento de objetos, o espaço de armazenamento disponível também é altamente escalável, uma característica ideal para sistemas de big data. O espaço de armazenamento é limitado pelo número de discos que o provedor de armazenamento possui, mas esses provedores lidam com exabytes de dados. Em um ambiente de nuvem, o espaço de armazenamento disponível é virtualmente ilimitado; na prática, o limite primário de espaço de armazenamento para clientes de nuvem pública é o orçamento. Do ponto de vista prático, os engenheiros podem armazenar rapidamente grandes quantidades de dados para projetos sem planejar com meses de antecedência os servidores e discos necessários.

Armazenamentos de objetos para aplicativos de engenharia de dados

Do ponto de vista da engenharia de dados, os armazenamentos de objetos fornecem excelente desempenho para grandes leituras e gravações em lote. Isso corresponde bem ao caso de uso de sistemas OLAP massivos. Um pouco do folclore da engenharia de dados diz que os armazenamentos de objetos não são bons para atualizações, mas isso é apenas parcialmente verdadeiro. Os armazenamentos de objetos são uma opção inferior para cargas de trabalho transacionais com muitas atualizações pequenas a cada segundo; esses casos de uso são muito mais bem atendidos por bancos de dados transacionais ou sistemas de armazenamento em bloco. Os armazenamentos de objetos funcionam bem para uma taxa baixa de operações de atualização, onde cada operação atualiza um grande volume de dados.

Os armazenamentos de objetos são agora o padrão ouro de armazenamento para data lakes. Nos primórdios dos data lakes, escrever uma vez, ler muitos (WORM) era o padrão operacional, mas isso tinha mais a ver com as complexidades do gerenciamento de versões e arquivos de dados do que com as limitações do HDFS e dos armazenamentos de objetos. Desde então, sistemas como o Apache Hudi e o Delta Lake surgiram para gerenciar essa complexidade, e as regulamentações de privacidade, como GDPR e CCPA, tornaram imperativas as capacidades de exclusão e atualização. O gerenciamento de atualizações para armazenamento de objetos é a ideia central por trás do conceito de data lakehouse, que apresentamos em [Capítulo 3](#).

O armazenamento de objetos é um repositório ideal para dados não estruturados em qualquer formato além desses aplicativos de dados estruturados. O armazenamento de objetos pode abrigar quaisquer dados binários sem restrições de tipo ou estrutura e frequentemente desempenha um papel nos pipelines de ML para texto bruto, imagens, vídeo e áudio.

pesquisa de objeto

Como mencionamos, os armazenamentos de objetos são armazenamentos de valor-chave. O que isso significa para os engenheiros? É fundamental entender que, ao contrário dos armazenamentos de arquivos, os armazenamentos de objetos não utilizam uma árvore de diretórios para encontrar objetos. O armazenamento de objetos usa um contêiner lógico de nível superior (um bucket no S3 e GCS) e faz referência a objetos por chave. Um exemplo simples no S3 pode ser assim:

S3://oreilly-data-engineering-book/data-example.json

Nesse caso,S3://oreilly-data-engineering-book/é o nome do balde edados-exemplo.jsoné a chave que aponta para um determinado objeto. Os nomes de bucket do S3 devem ser exclusivos em toda a AWS. As chaves são exclusivas dentro de um bucket. Embora os armazenamentos de objetos em nuvem pareçam oferecer suporte à semântica da árvore de diretórios, não existe uma verdadeira hierarquia de diretórios. Podemos armazenar um objeto com o seguinte caminho completo:

S3://oreilly-data-engineering-book/project-data/11/23/2021/data.txt

Superficialmente, isso se parece com subdiretórios que você pode encontrar em um sistema de pastas de arquivos regular:dados do projeto, 11, 23,e2021.Muitas interfaces de console em nuvem permitem que os usuários visualizem os objetos dentro de um “diretório” e as ferramentas de linha de comando em nuvem geralmente oferecem suporte a comandos no estilo Unix, comolsdentro de um diretório de armazenamento de objetos. No entanto, nos bastidores, o sistema de objetos não percorre uma árvore de diretórios para alcançar o objeto. Em vez disso, ele simplesmente vê uma chave (projeto-data/23/11/2021/data.txt)isso acontece para corresponder à semântica do diretório. Isso pode parecer um pequeno detalhe técnico, mas os engenheiros precisam entender que certas operações no nível do “diretório” são caras em um armazenamento de objetos. Para correraws ls S3://oreilly-data-engineering-book/project-dados/11/o armazenamento de objeto deve filtrar as chaves no prefixo da chavedados do projeto/11.Se o balde contiver milhões de objetos, essa operação poderá levar algum tempo, mesmo que o “subdiretório” hospede apenas alguns objetos.

Consistência e controle de versão de objetos

Conforme mencionado, os armazenamentos de objetos não oferecem suporte a atualizações ou acréscimos no local como regra geral. Escrevemos um novo objeto sob a mesma chave para atualizar um objeto. Quando os engenheiros de dados utilizam atualizações em processos de dados, eles devem estar cientes do modelo de consistência para o armazenamento de objeto que estão usando. Os armazenamentos de objetos podem ser eventualmente consistentes ou fortemente consistentes. Por exemplo, até recentemente, o S3 era *eventualmente consistente*; depois que uma nova versão de um objeto foi gravada sob a mesma chave, o armazenamento de objetos pode, às vezes, retornar a versão antiga do objeto. O*eventual*parte de *consistência eventual*significa que, após o tempo suficiente, o cluster de armazenamento atinge um estado tal que somente a versão mais recente do objeto será retornada. Isso contrasta com o *consistência forte* modelo que esperamos de discos locais conectados a um servidor: a leitura após uma gravação retornará os dados gravados mais recentemente.

Pode ser desejável impor consistência forte em um armazenamento de objeto por vários motivos, e métodos padrão são usados para conseguir isso. Uma abordagem é adicionar um banco de dados fortemente consistente (por exemplo, PostgreSQL) à mistura. Escrever um objeto agora é um processo de duas etapas:

1. Escreva o objeto.
2. Grave os metadados retornados para a versão do objeto no banco de dados fortemente consistente.

Os metadados da versão (um hash de objeto ou um carimbo de data/hora de objeto) podem identificar exclusivamente uma versão de objeto em conjunto com a chave de objeto. Para ler um objeto, um leitor realiza as seguintes etapas:

1. Busque os metadados de objeto mais recentes do banco de dados fortemente consistente.
2. Consulte os metadados do objeto usando a chave do objeto. Leia os dados do objeto se eles corresponderem aos metadados obtidos do banco de dados consistente.
3. Se os metadados do objeto não corresponderem, repita a etapa 2 até que a versão mais recente do objeto seja retornada.

Uma implementação prática tem exceções e casos extremos a serem considerados, como quando o objeto é reescrito durante esse processo de consulta. Essas etapas podem ser gerenciadas por trás de uma API para que um leitor de objeto veja um armazenamento de objeto fortemente consistente ao custo de maior latência para acesso a objeto.

O controle de versão do objeto está intimamente relacionado à consistência do objeto. Quando reescrevemos um objeto sob uma chave existente em um armazenamento de objeto, estamos essencialmente escrevendo um objeto totalmente novo, definindo referências da chave existente para o objeto e excluindo as referências de objeto antigas. A atualização de todas as referências no cluster leva tempo, portanto, o potencial para leituras obsoletas. Eventualmente, o coletor de lixo do cluster de armazenamento desaloca o espaço dedicado aos dados desreferenciados, reciclando a capacidade do disco para uso por novos objetos.

Com o controle de versão do objeto ativado, adicionamos metadados adicionais ao objeto que estipula uma versão. Enquanto a referência de chave padrão é atualizada para apontar para o novo objeto, mantemos outros ponteiros para versões anteriores. Também mantemos uma lista de versões para que os clientes possam obter uma lista de todas as versões de objetos e, em seguida, extrair uma versão específica. Como as versões antigas do objeto ainda são referenciadas, elas não são limpas pelo coletor de lixo.

Se referenciamos um objeto com uma versão, o problema de consistência com alguns sistemas de armazenamento de objetos desaparece: os metadados de chave e versão juntos formam uma referência única a um objeto de dados imutável específico. Sempre obteremos o mesmo objeto de volta quando usarmos este par, desde que não o excluímos. O problema de consistência ainda existe quando um cliente solicita uma versão “padrão” ou “mais recente” de um objeto.

A principal sobrecarga que os engenheiros precisam considerar com o controle de versão do objeto é o custo de armazenamento. Versões históricas de objetos geralmente têm os mesmos custos de armazenamento associados às versões atuais. Os custos da versão do objeto podem ser quase insignificantes ou catastroficamente caros, dependendo de vários fatores. O tamanho dos dados é um problema, assim como a frequência de atualização; mais versões de objetos podem levar a um tamanho de dados significativamente maior. Lembre-se de que estamos falando sobre versão de objeto de força bruta. Os sistemas de armazenamento de objetos geralmente armazenam dados de objetos completos para cada versão, não instantâneos diferenciais.

Os engenheiros também têm a opção de implantar políticas de ciclo de vida de armazenamento. As políticas de ciclo de vida permitem a exclusão automática de versões antigas do objeto quando certas condições são atendidas (por exemplo, quando uma versão do objeto atinge uma certa idade ou existem muitas versões mais recentes). Os fornecedores de nuvem também oferecem vários níveis de dados de arquivamento a preços com grandes descontos, e o processo de arquivamento pode ser gerenciado usando políticas de ciclo de vida.

Classes e níveis de armazenamento

Os fornecedores de nuvem agora oferecem classes de armazenamento que oferecem desconto nos preços de armazenamento de dados em troca de acesso reduzido ou durabilidade reduzida. Nós usamos o termo *acesso reduzido* aqui porque muitos desses níveis de armazenamento ainda tornam os dados altamente disponíveis, mas com altos custos de recuperação em troca de custos de armazenamento reduzidos.

Vejamos alguns exemplos no S3, já que a Amazon é uma referência para os padrões de serviços em nuvem. A classe de armazenamento S3 Standard-Infrequent Access oferece descontos nos custos mensais de armazenamento para aumentar os custos de recuperação de dados. (Ver “[Um breve desvio na economia da nuvem](#)” na [página 125](#) para uma discussão teórica sobre a economia dos níveis de armazenamento em nuvem.) A Amazon também oferece o nível Amazon S3 One Zone-Infrequent Access, replicando apenas para uma única zona. A disponibilidade projetada cai de 99,9% para 99,5% para compensar a possibilidade de uma interrupção zonal. A Amazon ainda reivindica durabilidade de dados extremamente alta, com a ressalva de que os dados serão perdidos se uma zona de disponibilidade for destruída.

Mais abaixo nas camadas de acesso reduzido estão as camadas de arquivamento no S3 Glacier. O S3 Glacier promete uma redução drástica nos custos de armazenamento de longo prazo para custos de acesso muito mais altos. Os usuários têm várias opções de velocidade de recuperação, de minutos a horas, com custos de recuperação mais altos para um acesso mais rápido. Por exemplo, no momento da redação deste artigo, o S3 Glacier Deep Archive oferece descontos ainda maiores nos custos de armazenamento; A Amazon anuncia que os custos de armazenamento começam em \$ 1 por terabyte por mês. Em troca, a restauração de dados leva 12 horas. Além disso, essa classe de armazenamento é projetada para dados que serão armazenados de 7 a 10 anos e acessados apenas uma a duas vezes por ano.

Esteja ciente de como você planeja utilizar o armazenamento de arquivamento, pois é fácil acessar e muitas vezes caro para acessar os dados, especialmente se você precisar deles com mais frequência do que o esperado. Ver [Capítulo 4](#) para uma discussão mais extensa sobre economia de armazenamento de arquivos.

Sistemas de arquivos com armazenamento de objetos

As soluções de sincronização de armazenamento de objetos tornaram-se cada vez mais populares. Ferramentas como s3fs e Amazon S3 File Gateway permitem que os usuários montem um bucket S3 como armazenamento local. Os usuários dessas ferramentas devem estar cientes das características das gravações no sistema de arquivos e como elas irão interagir com as características e preços do armazenamento de objetos. O File Gateway, por exemplo, lida com alterações em arquivos de maneira bastante eficiente, combinando partes de objetos em um novo objeto usando os recursos avançados do S3. No entanto, a gravação transacional de alta velocidade sobrecarregará os recursos de atualização de um objeto

loja. A montagem do armazenamento de objetos como um sistema de arquivos local funciona bem para arquivos que são atualizados com pouca frequência.

Sistemas de armazenamento baseados em cache e memória

Conforme discutido em “[Ingredientes brutos de armazenamento de dados](#)” na página 191, RAM oferece excelente latência e velocidades de transferência. No entanto, a RAM tradicional é extremamente vulnerável à perda de dados porque uma queda de energia de até um segundo pode apagar os dados. Os sistemas de armazenamento baseados em RAM geralmente são focados em aplicativos de cache, apresentando dados para acesso rápido e alta largura de banda. Os dados geralmente devem ser gravados em um meio mais durável para fins de retenção.

Esses sistemas de cache ultrarrápidos são úteis quando os engenheiros de dados precisam fornecer dados com latência de recuperação ultrarrápida.

Exemplo: cache de objetos leves e Memcached

Memcached é um armazenamento de valor-chave projetado para armazenar em cache resultados de consulta de banco de dados, respostas de chamada de API e muito mais. Memcached usa estruturas de dados simples, suportando tipos string ou integer. O Memcached pode fornecer resultados com latência muito baixa e, ao mesmo tempo, aliviar a carga dos sistemas de back-end.

Exemplo: Redis, cache de memória com persistência opcional

Como o Memcached, *Redis* é um armazenamento de chave-valor, mas suporta tipos de dados um pouco mais complexos (como listas ou conjuntos). O Redis também incorpora vários mecanismos de persistência, incluindo captura instantânea e registro no diário. Com uma configuração típica, o Redis grava dados aproximadamente a cada dois segundos. O Redis é, portanto, adequado para aplicativos de desempenho extremamente alto, mas pode tolerar uma pequena quantidade de perda de dados.

O sistema de arquivos distribuídos do Hadoop

No passado recente, “Hadoop” era praticamente sinônimo de “big data”. O Hadoop Distributed File System é baseado em [Sistema de arquivos do Google \(GFS\)](#) e foi inicialmente projetado para processar dados com o [Modelo de programação MapReduce](#). O Hadoop é semelhante ao armazenamento de objetos, mas com uma diferença fundamental: o Hadoop combina computação e armazenamento nos mesmos nós, onde os armazenamentos de objetos geralmente têm suporte limitado para processamento interno.

O Hadoop divide arquivos grandes em *blocos*, blocos de dados com menos de algumas centenas de megabytes de tamanho. O sistema de arquivos é gerenciado pelo *NameNode*, que mantém diretórios, metadados de arquivo e um catálogo detalhado que descreve a localização dos blocos de arquivo no cluster. Em uma configuração típica, cada bloco de dados é replicado para três nós. Isso aumenta a durabilidade e a disponibilidade dos dados. Se um disco ou nó falhar, o fator de replicação para alguns blocos de arquivo cairá abaixo de 3. O *NameNode*

instrua outros nós a replicar esses blocos de arquivo para que eles alcancem novamente o fator de replicação correto. Assim, a probabilidade de perda de dados é muito baixa, salvo uma *falsa correlacionada* (por exemplo, um asteroide atingindo o centro de dados).

O Hadoop não é simplesmente um sistema de armazenamento. O Hadoop combina recursos de computação com nós de armazenamento para permitir o processamento de dados no local. Isso foi originalmente obtido usando o modelo de programação MapReduce, que discutimos em [Capítulo 8](#).

Hadoop está morto. Viva Hadoop!

Muitas vezes vemos alegações de que o Hadoop está morto. Isso é apenas parcialmente verdade. O Hadoop não é mais uma tecnologia quente e de ponta. Muitas ferramentas do ecossistema Hadoop, como o Apache Pig, agora estão em suporte de vida e são usadas principalmente para executar trabalhos herdados. O modelo de programação MapReduce puro caiu no esquecimento. O HDFS continua sendo amplamente utilizado em vários aplicativos e organizações.

O Hadoop ainda aparece em muitas instalações legadas. Muitas organizações que adotaram o Hadoop durante o auge da febre do big data não têm planos imediatos de migrar para tecnologias mais novas. Essa é uma boa opção para empresas que executam clusters Hadoop massivos (mil nós) e têm os recursos para manter sistemas locais com eficiência. As empresas menores podem querer reconsiderar a sobrecarga de custos e as limitações de escala de executar um pequeno cluster Hadoop em vez de migrar para soluções em nuvem.

Além disso, o HDFS é um ingrediente-chave de muitos mecanismos de big data atuais, como o Amazon EMR. Na verdade, o Apache Spark ainda é comumente executado em clusters HDFS. Discutimos isso com mais detalhes em "[Separação da computação do armazenamento](#)" na página 220.

Armazenamento de streaming

Os dados de streaming têm requisitos de armazenamento diferentes dos dados sem streaming. No caso de filas de mensagens, os dados armazenados são temporais e espera-se que desapareçam após um determinado período. No entanto, estruturas de streaming escalonáveis e distribuídas, como o Apache Kafka, agora permitem a retenção de dados de streaming de duração extremamente longa. O Kafka oferece suporte à retenção de dados indefinida, enviando mensagens antigas e acessadas com pouca frequência para o armazenamento de objetos. Concorrentes do Kafka (incluindo Amazon Kinesis, Apache Pulsar e Google Cloud Pub/Sub) também suportam longa retenção de dados.

Intimamente relacionada à retenção de dados nesses sistemas está a noção de replay. *Repetir* permite que um sistema de streaming retorne uma série de dados históricos armazenados. Replay é o mecanismo de recuperação de dados padrão para sistemas de armazenamento de streaming. A reprodução pode ser usada para executar consultas em lote em um intervalo de tempo ou para reprocessar dados em um pipeline de streaming. [Capítulo 7](#) cobre o replay com mais profundidade.

Outros mecanismos de armazenamento surgiram para aplicativos de análise em tempo real. Em certo sentido, os bancos de dados transacionais surgiram como os primeiros mecanismos de consulta em tempo real; os dados se tornam visíveis para consultas assim que são gravados. No entanto, esses bancos de dados têm

limitações bem conhecidas de dimensionamento e bloqueio, especialmente para consultas analíticas executadas em grandes volumes de dados. Embora as versões escaláveis de bancos de dados transacionais orientados por linha tenham superado algumas dessas limitações, elas ainda não são realmente otimizadas para análise em escala.

Índices, particionamento e clustering

Os índices fornecem um mapa da tabela para campos específicos e permitem uma pesquisa extremamente rápida de registros individuais. Sem índices, um banco de dados precisaria varrer uma tabela inteira para encontrar os registros que satisfizessem uma *ONDE* *doença*.

Na maioria dos RDBMSs, os índices são usados para chaves primárias de tabela (permitindo identificação única de linhas) e chaves estrangeiras (permitindo junções com outras tabelas). Os índices também podem ser aplicados a outras colunas para atender às necessidades de aplicativos específicos. Usando índices, um RDBMS pode pesquisar e atualizar milhares de linhas por segundo.

Não abordamos os registros de bancos de dados transacionais em profundidade neste livro; numerosos recursos técnicos estão disponíveis sobre este tópico. Em vez disso, estamos interessados na evolução dos índices em sistemas de armazenamento orientados a análises e alguns novos desenvolvimentos em índices para casos de uso de análises.

A evolução das linhas para as colunas

Um data warehouse inicial era normalmente criado no mesmo tipo de RDBMS usado para aplicativos transacionais. A crescente popularidade dos sistemas MPP significou uma mudança em direção ao processamento paralelo para melhorias significativas no desempenho da varredura em grandes quantidades de dados para fins analíticos. No entanto, esses MPPs orientados a linhas ainda usavam índices para oferecer suporte a junções e verificação de condição.

Em “[Ingredientes brutos de armazenamento de dados](#)” na página 191, discutimos a serialização colunar. *Serialização colunar* permite que um banco de dados verifique apenas as colunas necessárias para uma consulta específica, às vezes reduzindo drasticamente a quantidade de dados lidos do disco. Além disso, organizar os dados por coluna agrupa valores semelhantes próximos uns dos outros, gerando taxas de compactação altas com sobrecarga mínima de compactação. Isso permite que os dados sejam verificados mais rapidamente do disco e em uma rede.

Os bancos de dados colunares funcionam mal para casos de uso transacional, ou seja, quando tentamos procurar um grande número de linhas individuais de forma assíncrona. No entanto, eles funcionam muito bem quando grandes quantidades de dados devem ser digitalizadas, por exemplo, para transformações de dados complexos, agregações, cálculos estatísticos ou avaliação de condições complexas em grandes conjuntos de dados.

No passado, os bancos de dados colunares apresentavam desempenho ruim em junções, portanto, o conselho para os engenheiros de dados era desnormalizar os dados, usando esquemas amplos, matrizes e dados aninhados sempre que possível. O desempenho de junção para bancos de dados colunares melhorou drasticamente nos últimos anos, portanto, embora ainda possa haver vantagens de desempenho em

desnormalização, isso não é mais uma necessidade. Você aprenderá mais sobre normalização e desnormalização em [Capítulo 8](#).

De índices a partições e clustering

Embora os bancos de dados colunares permitam velocidades de digitalização rápidas, ainda é útil reduzir a quantidade de dados digitalizados o máximo possível. Além de verificar apenas os dados em colunas relevantes para uma consulta, podemos particionar uma tabela em várias subtabelas dividindo-a em um campo. É bastante comum em casos de uso de análise e ciência de dados verificar um intervalo de tempo, portanto, o particionamento baseado em data e hora é extremamente comum. Bancos de dados colunares geralmente suportam uma variedade de outros esquemas de partição também.

Clusters permitem uma organização mais refinada dos dados dentro das partições. Um esquema de agrupamento aplicado em um banco de dados colunar classifica os dados por um ou alguns campos, colocando valores semelhantes. Isso melhora o desempenho para filtrar, classificar e unir esses valores.

Exemplo: microparticionamento em floco de neve

Mencionamos Floco de Neve [microparticionamento](#) porque é um bom exemplo de desenvolvimentos recentes e evolução nas abordagens de armazenamento colunar. *Micro partições* são conjuntos de linhas entre 50 e 500 megabytes em tamanho não compactado. Snowflake usa uma abordagem algorítmica que tenta agrupar linhas semelhantes. Isso contrasta com a abordagem ingênua tradicional de particionamento em um único campo designado, como uma data. O Snowflake procura especificamente valores que são repetidos em um campo em muitas linhas. Isso permite agressividade *de poda* de consultas baseadas em predicados. Por exemplo, um `ONDE` cláusula pode estipular o seguinte:

ONDE `Data de criação='2022-01-02'`

Nessa consulta, o Snowflake exclui quaisquer micropartições que não incluem essa data, removendo efetivamente esses dados. O Snowflake também permite a sobreposição de micropartições, potencialmente particionando em vários campos mostrando repetições significativas.

A remoção eficiente é facilitada pelo banco de dados de metadados do Snowflake, que armazena uma descrição de cada micropartição, incluindo o número de linhas e intervalos de valores para campos. Em cada estágio de consulta, o Snowflake analisa as micropartições para determinar quais precisam ser verificadas. Floco de neve usa o termo *armazenamento colunar híbrido*,² referindo-se parcialmente ao fato de que suas tabelas são divididas em pequenos grupos de linhas, embora o armazenamento seja fundamentalmente colunar. O banco de dados de metadados desempenha um papel semelhante a um índice em um banco de dados relacional tradicional.

2 Benoit Dageville, "The Snowflake Elastic Data Warehouse," *SIGMOD '16: Anais do International 2016 Conferência sobre Gestão de Dados* (junho de 2016): 215–226, <https://oreil.ly/Tc1su>.

Abstrações de armazenamento de engenharia de dados

Abstrações de armazenamento de engenharia de dados são organização de dados e padrões de consulta que estão no centro do ciclo de vida da engenharia de dados e são construídos sobre os sistemas de armazenamento de dados discutidos anteriormente (consulte [Figura 6-3](#)). Introduzimos muitas dessas abstrações em [Capítulo 3](#), e vamos revisitá-las aqui.

Os principais tipos de abstrações com os quais nos preocuparemos são aqueles que suportam casos de uso de ciência de dados, análises e relatórios. Isso inclui data warehouse, data lake, data lakehouse, plataformas de dados e catálogos de dados. Não cobriremos os sistemas de origem, pois eles são discutidos em [capítulo 5](#).

A abstração de armazenamento necessária como engenheiro de dados se resume a algumas considerações importantes:

Finalidade e caso de uso

Você deve primeiro identificar o propósito de armazenar os dados. Para que isso é usado?

Atualizar padrões

A abstração é otimizada para atualizações em massa, inserções de streaming ou upserts?

Custo

Quais são os custos financeiros diretos e indiretos? A hora de valorizar? Os custos de oportunidade?

Armazenamento e computação separados

A tendência é separar armazenamento e computação, mas a maioria dos sistemas híbrida separação e colocation. Cobrimos isso em "[Separação da computação do armazenamento](#)" na [página 220](#) uma vez que afeta a finalidade, a velocidade e o custo.

Você deve saber que a popularidade de separar o armazenamento da computação significa que as linhas entre os bancos de dados OLAP e os data lakes estão cada vez mais tênues. Os principais data warehouses e data lakes na nuvem estão em rota de colisão. No futuro, as diferenças entre esses dois podem ser apenas no nome, pois podem ser funcional e tecnicamente muito semelhantes sob o capô.

O Armazém de Dados

Os data warehouses são uma arquitetura de dados OLAP padrão. Conforme discutido em [Capítulo 3](#), O termo *armazém de dados* refere-se a plataformas de tecnologia (por exemplo, Google BigQuery e Teradata), uma arquitetura para centralização de dados e um padrão organizacional dentro de uma empresa. Em termos de tendências de armazenamento, evoluímos da construção de data warehouses em bancos de dados transacionais convencionais, sistemas MPP baseados em linha (por exemplo, Teradata e IBM Netezza) e sistemas MPP colunares (por exemplo, Vertica e Teradata Columnar) para data warehouses em nuvem e plataformas de dados. (Veja nossa discussão sobre armazenamento de dados em [Capítulo 3](#) para obter mais detalhes sobre sistemas MPP.)

Na prática, os data warehouses em nuvem costumam ser usados para organizar dados em um data lake, uma área de armazenamento para grandes quantidades de dados brutos não processados, conforme originalmente concebido por James Dixon.³ Os data warehouses em nuvem podem lidar com grandes quantidades de texto bruto e documentos JSON complexos. A limitação é que os data warehouses em nuvem não podem lidar com dados verdadeiramente não estruturados, como imagens, vídeo ou áudio, ao contrário de um verdadeiro data lake. Os data warehouses em nuvem podem ser acoplados ao armazenamento de objetos para fornecer uma solução completa de data lake.

o lago de dados

O *lago de dados* foi originalmente concebido como um armazenamento massivo onde os dados eram retidos em formato bruto e não processado. Inicialmente, os data lakes foram construídos principalmente em sistemas Hadoop, onde o armazenamento barato permitia a retenção de grandes quantidades de dados sem a sobrecarga de custos de um sistema MPP proprietário.

Os últimos cinco anos viram dois grandes desenvolvimentos na evolução do armazenamento de data lake. Primeiro, uma grande migração para *separação de computação e armazenamento* ocorreu. Na prática, isso significa uma mudança do Hadoop para o armazenamento de objetos em nuvem para retenção de dados a longo prazo. Em segundo lugar, os engenheiros de dados descobriram que grande parte da funcionalidade oferecida pelos sistemas MPP (gerenciamento de esquema; capacidade de atualização, mesclagem e exclusão) e inicialmente descartada na corrida para os data lakes era, de fato, extremamente útil. Isso levou à noção de data lakehouse.

A Casa do Lago de Dados

O *data lakehouse* é uma arquitetura que combina aspectos do data warehouse e do data lake. Como geralmente é concebido, a casa do lago armazena dados no armazenamento de objetos como um lago. No entanto, o lakehouse adiciona a esse arranjo recursos projetados para simplificar o gerenciamento de dados e criar uma experiência de engenharia semelhante a um data warehouse. Isso significa suporte robusto a tabelas e esquemas e recursos para gerenciar atualizações e exclusões incrementais. Lakehouses normalmente também suportam histórico de tabela e reversão; isso é feito retendo versões antigas de arquivos e metadados.

Um sistema lakehouse é uma camada de metadados e gerenciamento de arquivos implantada com ferramentas de gerenciamento e transformação de dados. A Databricks promoveu fortemente o conceito de lakehouse com o Delta Lake, um sistema de gerenciamento de armazenamento de código aberto.

Seria negligente não apontar que a arquitetura do data lakehouse é semelhante à arquitetura usada por várias plataformas de dados comerciais, incluindo BigQuery e Snowflake. Esses sistemas armazenam dados em armazenamento de objetos e fornecem

³ James Dixon, "Data Lakes Revisited," *Blogue de James Dixon*, 25 de setembro de 2014, <https://oreil.ly/FH25v>.

gerenciamento automatizado de metadados, histórico de tabelas e recursos de atualização/exclusão. As complexidades do gerenciamento de arquivos e armazenamento subjacentes são totalmente ocultas do usuário.

A principal vantagem do data lakehouse sobre as ferramentas proprietárias é a interoperabilidade. É muito mais fácil trocar dados entre ferramentas quando armazenados em um formato de arquivo aberto. A reserialização de dados de um formato de banco de dados proprietário incorre em sobrecarga de processamento, tempo e custo. Em uma arquitetura de data lakehouse, várias ferramentas podem se conectar à camada de metadados e ler dados diretamente do armazenamento de objetos.

É importante ressaltar que grande parte dos dados em um data lakehouse podem não ter uma estrutura de tabela imposta. Podemos impor recursos de data warehouse onde precisamos deles em um lakehouse, deixando outros dados em um formato bruto ou mesmo não estruturado.

A tecnologia de data lakehouse está evoluindo rapidamente. Uma variedade de novos concorrentes para Delta Lake surgiram, incluindo Apache Hudi e Apache Iceberg. Ver [Apêndice A](#) para mais detalhes.

Plataformas de dados

Cada vez mais, os fornecedores estão estilizando seus produtos como *plataformas de dados*. Esses fornecedores criaram seus ecossistemas de ferramentas interoperáveis com forte integração na camada principal de armazenamento de dados. Ao avaliar as plataformas, os engenheiros devem garantir que as ferramentas oferecidas atendam às suas necessidades. As ferramentas não fornecidas diretamente na plataforma ainda podem interoperar, com sobrecarga de dados extra para intercâmbio de dados. As plataformas também enfatizam a estreita integração com o armazenamento de objetos para casos de uso não estruturados, conforme mencionado em nossa discussão sobre data warehouses em nuvem.

Neste ponto, a noção de plataforma de dados francamente ainda não foi totalmente desenvolvida. No entanto, a corrida continua para criar um jardim murado de ferramentas de dados, simplificando o trabalho de engenharia de dados e gerando um aprisionamento significativo de fornecedores.

Arquitetura de armazenamento de fluxo para lote

A arquitetura de armazenamento stream-to-batch tem muitas semelhanças com a arquitetura Lambda, embora alguns possam questionar os detalhes técnicos. Essencialmente, os dados que fluem por um tópico no sistema de armazenamento de streaming são gravados para vários consumidores.

Alguns desses consumidores podem ser sistemas de processamento em tempo real que geram estatísticas no fluxo. Além disso, um consumidor de armazenamento em lote grava dados para retenção de longo prazo e consultas em lote. O consumidor de lote pode ser o AWS Kinesis Firehose, que pode gerar objetos S3 com base em gatilhos configuráveis (por exemplo, hora e tamanho do lote). Sistemas como o BigQuery ingerem dados de streaming em um buffer de streaming. Esse buffer de streaming é reserializado automaticamente no armazenamento de objeto colunar. O mecanismo de consulta oferece suporte à consulta contínua do buffer de streaming e dos dados do objeto para fornecer aos usuários uma visão atual e quase em tempo real da tabela.

Grandes ideias e tendências em armazenamento

Nesta seção, discutiremos algumas grandes ideias sobre armazenamento — considerações importantes que você precisa ter em mente ao desenvolver sua arquitetura de armazenamento. Muitas dessas considerações fazem parte de tendências maiores. Por exemplo, os catálogos de dados se encaixam na tendência de engenharia e gerenciamento de dados “empresariais”. A separação da computação do armazenamento é agora um fato consumado em sistemas de dados em nuvem. E o compartilhamento de dados é uma consideração cada vez mais importante à medida que as empresas adotam a tecnologia de dados.

Catálogo de Dados

A catálogo de dados é um armazenamento de metadados centralizado para todos os dados em uma organização. Estritamente falando, um catálogo de dados não é uma abstração de armazenamento de dados de nível superior, mas se integra a vários sistemas e abstrações. Os catálogos de dados geralmente funcionam em fontes de dados operacionais e analíticos, integram a linhagem de dados e a apresentação de relacionamentos de dados e permitem a edição de descrições de dados pelo usuário.

Os catálogos de dados costumam ser usados para fornecer um local central onde as pessoas podem visualizar seus dados, consultas e armazenamento de dados. Como engenheiro de dados, você provavelmente será responsável por configurar e manter as várias integrações de dados do pipeline de dados e sistemas de armazenamento que se integrarão ao catálogo de dados e à integridade do próprio catálogo de dados.

Integração de aplicativos de catálogo

Idealmente, os aplicativos de dados são projetados para integrar-se às APIs de catálogo para lidar diretamente com seus metadados e atualizações. Como os catálogos são mais amplamente utilizados em uma organização, fica mais fácil abordar esse ideal.

Escaneamento automatizado

Na prática, os sistemas de catalogação geralmente precisam contar com uma camada de varredura automatizada que coleta metadados de vários sistemas, como data lakes, data warehouses e bancos de dados operacionais. Catálogos de dados podem coletar metadados existentes e também podem usar ferramentas de varredura para inferir metadados, como relacionamentos de chave ou a presença de dados confidenciais.

Portal de dados e camada social

Catálogos de dados também normalmente fornecem uma camada de acesso humano por meio de uma interface da Web, onde os usuários podem pesquisar dados e visualizar relacionamentos de dados. Os catálogos de dados podem ser aprimorados com uma camada social que oferece a funcionalidade Wiki. Isso permite que os usuários forneçam informações sobre seus conjuntos de dados, solicitem informações de outros usuários e publiquem atualizações assim que estiverem disponíveis.

Casos de uso do catálogo de dados

Os catálogos de dados têm casos de uso organizacionais e técnicos. Os catálogos de dados tornam os metadados facilmente disponíveis para os sistemas. Por exemplo, um catálogo de dados é um ingrediente-chave do data lakehouse, permitindo a descoberta de tabelas para consultas.

Organizacionalmente, os catálogos de dados permitem que usuários de negócios, analistas, cientistas de dados e engenheiros pesquisem dados para responder a perguntas. Os catálogos de dados simplificam as comunicações e a colaboração entre organizações.

Compartilhamento de dados

Compartilhamento de dados permite que organizações e indivíduos compartilhem dados específicos e permissões cuidadosamente definidas com entidades específicas. O compartilhamento de dados permite que os cientistas de dados compartilhem dados de um sandbox com seus colaboradores dentro de uma organização. Em todas as organizações, o compartilhamento de dados facilita a colaboração entre empresas parceiras. Por exemplo, uma empresa de tecnologia de anúncios pode compartilhar dados de publicidade com seus clientes.

Um ambiente multilocatário em nuvem torna a colaboração interorganizacional muito mais fácil. No entanto, também apresenta novos desafios de segurança. As organizações devem controlar cuidadosamente as políticas que regem quem pode compartilhar dados com quem para evitar exposição acidental ou exfiltração deliberada.

O compartilhamento de dados é um recurso central de muitas plataformas de dados em nuvem. Ver [capítulo 5](#) para uma discussão mais extensa sobre compartilhamento de dados.

Esquema

Qual é a forma esperada dos dados? Qual é o formato do arquivo? É estruturado, semiestruturado ou não estruturado? Quais tipos de dados são esperados? Como os dados se encaixam em uma hierarquia maior? Ele está conectado a outros dados por meio de chaves compartilhadas ou outros relacionamentos?

Observe que o esquema não precisa ser *relacional*. Em vez disso, os dados se tornam mais úteis quando temos o máximo de informações sobre sua estrutura e organização. Para imagens armazenadas em um data lake, essas informações de esquema podem explicar o formato da imagem, a resolução e a forma como as imagens se encaixam em uma hierarquia maior.

O esquema pode funcionar como uma espécie de pedra de Roseta, instruções que nos dizem como ler os dados. Existem dois padrões de esquema principais: esquema na gravação e esquema na leitura. *Esquema na gravação* é essencialmente o padrão tradicional de data warehouse: uma tabela tem um esquema integrado; qualquer gravação na tabela deve estar em conformidade. Para dar suporte ao esquema na gravação, um data lake deve integrar um metastore de esquema.

Com *esquema na leitura*, o esquema é criado dinamicamente quando os dados são gravados e um leitor deve determinar o esquema ao ler os dados. Idealmente, o esquema na leitura é implementado usando formatos de arquivo que implementam informações de esquema incorporadas, como

como Parquet ou JSON. Arquivos CSV são notórios por inconsistência de esquema e não são recomendados nessa configuração.

A principal vantagem do esquema na gravação é que ele impõe padrões de dados, tornando os dados mais fáceis de consumir e utilizar no futuro. O esquema na leitura enfatiza a flexibilidade, permitindo que praticamente qualquer dado seja gravado. Isso vem com o custo de maior dificuldade em consumir dados no futuro.

Separação da Computação do Armazenamento

Uma ideia-chave que revisitamos ao longo deste livro é a separação entre computação e armazenamento. Isso surgiu como um padrão de consulta e acesso a dados padrão na era da nuvem atual. Os data lakes, como discutimos, armazenam dados em armazenamentos de objetos e aumentam a capacidade de computação temporária para lê-los e processá-los. A maioria dos produtos OLAP totalmente gerenciados agora depende do armazenamento de objetos nos bastidores. Para entender as motivações para separar computação e armazenamento, devemos primeiro olhar para a colocação de computação e armazenamento.

Colocation de computação e armazenamento

A colocação de computação e armazenamento tem sido um método padrão para melhorar o desempenho do banco de dados. Para bancos de dados transacionais, a colocação de dados permite leituras de disco rápidas e de baixa latência e alta largura de banda. Mesmo quando virtualizamos o armazenamento (por exemplo, usando o Amazon EBS), os dados ficam relativamente próximos à máquina host.

A mesma ideia básica se aplica a sistemas de consulta analítica executados em um cluster de máquinas. Por exemplo, com HDFS e MapReduce, a abordagem padrão é localizar blocos de dados que precisam ser verificados no cluster e, em seguida, enviar *mapa*empregos para esses blocos. A varredura e o processamento de dados para a etapa do mapa são estritamente locais. *O reduzir*A etapa envolve o embaralhamento de dados no cluster, mas manter as etapas do mapa local preserva efetivamente mais largura de banda para embaralhamento, proporcionando melhor desempenho geral; as etapas do mapa que filtram fortemente também reduzem drasticamente a quantidade de dados a serem embaralhados.

Separação de computação e armazenamento

Se a colocação de computação e armazenamento oferece alto desempenho, por que mudar para a separação de computação e armazenamento? Existem várias motivações.

Efemeridade e escalabilidade.Na nuvem, vimos uma mudança drástica em direção à efemeridade. Em geral, é mais barato comprar e hospedar um servidor do que alugá-lo de um provedor de nuvem, *desde que você esteja executando 24 horas por dia sem parar por anos a fio*. Na prática, as cargas de trabalho variam drasticamente e eficiências significativas são obtidas com um modelo de pagamento conforme o uso se os servidores puderem ser dimensionados para cima e para baixo. Isso é verdade para servidores da Web no varejo on-line e também para tarefas de lote de big data que podem ser executadas apenas periodicamente.

Os recursos de computação efêmeros permitem que os engenheiros criem clusters massivos para concluir os trabalhos no prazo e exclam os clusters quando esses trabalhos são concluídos. Os benefícios de desempenho de operar temporariamente em escala ultra alta podem superar as limitações de largura de banda do armazenamento de objetos.

Durabilidade e disponibilidade de dados. Os armazenamentos de objetos em nuvem reduzem significativamente o risco de perda de dados e geralmente fornecem tempo de atividade (disponibilidade) extremamente alto. Por exemplo, o S3 armazena dados em várias zonas; se um desastre natural destruir uma zona, os dados das zonas restantes ainda estarão disponíveis. Ter várias zonas disponíveis também reduz as chances de uma interrupção de dados. Se os recursos de uma zona ficarem inativos, os engenheiros podem ativar os mesmos recursos em uma zona diferente.

O potencial para uma configuração incorreta que destrói dados no armazenamento de objetos ainda é um tanto assustador, mas mitigações simples de implantar estão disponíveis. A cópia de dados para várias regiões de nuvem reduz esse risco, pois as alterações de configuração geralmente são implantadas em apenas uma região por vez. A replicação de dados para vários provedores de armazenamento pode reduzir ainda mais o risco.

Separação híbrida e colocation

As realidades práticas de separar a computação do armazenamento são mais complicadas do que sugerimos. Na prática, hibridizamos constantemente colocation e separação para obter os benefícios de ambas as abordagens. Essa hibridização é normalmente feita de duas maneiras: cache multicamada e armazenamento de objeto híbrido.

Com *cache multicamadas*, utilizamos o armazenamento de objetos para retenção e acesso de dados de longo prazo, mas aumentamos o armazenamento local para ser usado durante consultas e vários estágios de pipelines de dados. Tanto o Google quanto a Amazon oferecem versões de armazenamento de objetos híbridos (armazenamento de objetos totalmente integrado à computação).

Vejamos exemplos de como alguns mecanismos de processamento populares hibridizam a separação e a colocação de armazenamento e computação.

Exemplo: AWS EMR com S3 e HDFS. Serviços de big data, como o Amazon EMR, criam clusters HDFS temporários para processar dados. Os engenheiros têm a opção de referenciar S3 e HDFS como um sistema de arquivos. Um padrão comum é colocar o HDFS em unidades SSD, extrair do S3 e salvar os dados das etapas intermediárias de processamento no HDFS local. Fazer isso pode obter ganhos de desempenho significativos em relação ao processamento diretamente do S3. Os resultados completos são gravados de volta no S3 assim que o cluster conclui suas etapas e o cluster e o HDFS são excluídos. Outros consumidores leem os dados de saída diretamente do S3.

Exemplo: Apache Spark. Na prática, o Spark geralmente executa tarefas no HDFS ou em algum outro sistema de arquivos distribuído efêmero para oferecer suporte ao armazenamento de dados de alto desempenho entre as etapas de processamento. Além disso, o Spark depende muito do armazenamento de dados na memória para melhorar o processamento. O problema de possuir a infraestrutura para executar o Spark

é que a RAM dinâmica (DRAM) é extremamente cara; ao separar computação e armazenamento na nuvem, podemos alugar grandes quantidades de memória e liberá-la quando o trabalho for concluído.

Exemplo: Druida Apache. O Apache Druid depende fortemente de SSDs para obter alto desempenho. Como os SSDs são significativamente mais caros do que os discos magnéticos, o Druid mantém apenas uma cópia dos dados em seu cluster, reduzindo os custos de armazenamento “ao vivo” por um fator de três.

Claro, manter a durabilidade dos dados ainda é crítico, então o Druid usa um armazenamento de objeto como sua camada de durabilidade. Quando os dados são ingeridos, eles são processados, serializados em colunas compactadas e gravados em SSDs de cluster e armazenamento de objetos. No caso de falha do nó ou corrupção de dados do cluster, os dados podem ser recuperados automaticamente para novos nós. Além disso, o cluster pode ser desligado e totalmente recuperado do armazenamento SSD.

Exemplo: armazenamento de objetos híbridos. O sistema de armazenamento de arquivos Colossus do Google oferece suporte ao controle refinado da localização do bloco de dados, embora essa funcionalidade não seja exposta diretamente ao público. O BigQuery usa esse recurso para colocar tabelas de clientes em um único local, permitindo largura de banda ultra alta para consultas nesse local.⁴ Nós nos referimos a isso como *armazenamento de objetos híbridos* porque combina as abstrações limpas do armazenamento de objetos com algumas vantagens de colocar computação e armazenamento. A Amazon também oferece alguma noção de armazenamento de objetos híbridos por meio do S3 Select, um recurso que permite aos usuários filtrar dados S3 diretamente em clusters S3 antes que os dados sejam retornados pela rede.

Especulamos que as nuvens públicas adotarão o armazenamento de objetos híbridos mais amplamente para melhorar o desempenho de suas ofertas e fazer uso mais eficiente dos recursos de rede disponíveis. Alguns podem já estar fazendo isso sem divulgar isso publicamente.

O conceito de armazenamento de objetos híbridos ressalta que ainda pode haver vantagens em ter acesso de baixo nível ao hardware em vez de depender da nuvem pública de outra pessoa. Os serviços de nuvem pública não expõem detalhes de baixo nível de hardware e sistemas (por exemplo, localizações de blocos de dados para o Colossus), mas esses detalhes podem ser extremamente úteis na otimização e aprimoramento do desempenho. Veja nossa discussão sobre a economia da nuvem em [Capítulo 4](#).

Embora agora estejamos vendo uma migração em massa de dados para nuvens públicas, acreditamos que muitos fornecedores de serviços de dados em hiperescala que atualmente são executados em nuvens públicas fornecidas por outros fornecedores podem construir seus data centers no futuro, embora com profunda integração de rede em nuvens públicas.

⁴ Valliappa Lakshmanan e Jordan Tigani, *Google BigQuery: o guia definitivo* (Sebastopol, CA: O'Reilly, 2019), 16–17, 188, <https://oreil.ly/5aXxu>.

Clonagem de cópia zero

Sistemas baseados em nuvem baseados em suporte de armazenamento de objetos *clonagem de cópia zero*. Isso normalmente significa que uma nova cópia virtual de um objeto é criada (por exemplo, uma nova tabela) sem necessariamente copiar fisicamente os dados subjacentes. Normalmente, novos ponteiros são criados para os arquivos de dados brutos e alterações futuras nessas tabelas não serão registradas na tabela antiga. Para aqueles familiarizados com o funcionamento interno de linguagens orientadas a objetos, como Python, esse tipo de cópia “superficial” é familiar em outros contextos.

A clonagem de cópia zero é um recurso atraente, mas os engenheiros devem entender seus pontos fortes e limitações. Por exemplo, clonar um objeto em um ambiente de data lake e, em seguida, excluir os arquivos no objeto original também pode eliminar o novo objeto.

Para sistemas baseados em armazenamento de objetos totalmente gerenciados (por exemplo, Snowflake e BigQuery), os engenheiros precisam estar extremamente familiarizados com os limites exatos da cópia superficial. Os engenheiros têm mais acesso ao armazenamento de objetos subjacentes em sistemas de data lake, como o Databricks — uma bênção e uma maldição. Os engenheiros de dados devem ter muito cuidado antes de excluir qualquer arquivo bruto no armazenamento de objeto subjacente. Databricks e outras tecnologias de gerenciamento de data lake às vezes também suportam uma noção de *cópia profunda*, por meio do qual todos os objetos de dados subjacentes são copiados. Este é um processo mais caro, mas também mais robusto no caso de arquivos serem perdidos ou excluídos involuntariamente.

Ciclo de vida de armazenamento de dados e retenção de dados

Armazenar dados não é tão simples quanto salvá-los no armazenamento de objetos ou em disco e esquecê-los. Você precisa pensar no ciclo de vida do armazenamento de dados e na retenção de dados. Ao pensar na frequência de acesso e nos casos de uso, pergunte: “Qual é a importância dos dados para os usuários downstream e com que frequência eles precisam acessá-los?” Este é o ciclo de vida do armazenamento de dados. Outra pergunta que você deve fazer é: “Por quanto tempo devo manter esses dados?” Você precisa reter os dados indefinidamente ou pode descartá-los após um determinado período de tempo? Isso é retenção de dados. Vamos mergulhar em cada um deles.

Dados quentes, mornos e frios

Você sabia que os dados têm uma temperatura? Dependendo da frequência com que os dados são acessados, podemos agrupar a maneira como eles são armazenados em três categorias de persistência: quente, morno e frio. Os padrões de acesso à consulta diferem para cada conjunto de dados ([Figura 6-9](#)). Normalmente, os dados mais recentes são consultados com mais frequência do que os dados mais antigos. Vejamos os dados quentes, frios e quentes nessa ordem.

	Hot storage	Warm storage	Cold storage
Access	Very frequent	Infrequent	Infrequent
Storage cost	High	Medium	Cheap
Retrieval cost	Cheap	Medium	High

Figura 6-9. Custos de dados quentes, mornos e frios associados à frequência de acesso

Dados quentes. *dados quentes* tem requisitos de acesso instantâneo ou frequente. O armazenamento subjacente para dados dinâmicos é adequado para acesso e leituras rápidas, como SSD ou memória. Devido ao tipo de hardware envolvido com dados dinâmicos, armazenar dados dinâmicos geralmente é a forma de armazenamento mais cara. Exemplos de casos de uso para dados quentes incluem a recuperação de recomendações de produtos e resultados de páginas de produtos. O custo de armazenamento de dados quentes é o mais alto desses três níveis de armazenamento, mas a recuperação costuma ser barata.

O cache de resultados de consulta é outro exemplo de dados dinâmicos. Quando uma consulta é executada, alguns mecanismos de consulta mantêm os resultados da consulta no cache. Por um tempo limitado, quando a mesma consulta é executada, em vez de executar novamente a mesma consulta no armazenamento, o cache de resultados da consulta fornece os resultados armazenados em cache. Isso permite tempos de resposta de consulta muito mais rápidos em vez de emitir repetidamente a mesma consulta de forma redundante. Nos próximos capítulos, abordaremos os caches de resultados de consulta com mais detalhes.

Dados quentes. *dados quentes* é acessado semi-regularmente, digamos, uma vez por mês. Nenhuma regra rígida e rápida indica a frequência com que os dados mornos são acessados, mas é menos do que dados quentes e mais do que dados frios. Os principais provedores de nuvem oferecem camadas de armazenamento de objetos que acomodam dados quentes. Por exemplo, o S3 oferece um nível de acesso infrequente e o Google Cloud tem um nível de armazenamento semelhante chamado Nearline. Os fornecedores fornecem seus modelos de frequência de acesso recomendada e os engenheiros também podem fazer sua modelagem e monitoramento de custos. O armazenamento de dados quentes é mais barato do que os dados quentes, com custos de recuperação um pouco mais caros.

Dados frios. No outro extremo, *dados frios* são dados acessados com pouca frequência. O hardware usado para arquivar dados frios geralmente é barato e durável, como HDD, armazenamento em fita e sistemas de arquivamento baseados em nuvem. Os dados frios destinam-se principalmente ao arquivamento de longo prazo, quando há pouca ou nenhuma intenção de acessar os dados. Embora o armazenamento de dados frios seja barato, a recuperação de dados frios costuma ser cara.

Considerações sobre a camada de armazenamento. Ao considerar a camada de armazenamento para seus dados, considere os custos de cada camada. Se você armazenar todos os seus dados em armazenamento quente, todos os dados poderão ser acessados rapidamente. Mas isso tem um preço tremendo! Por outro lado, se você armazenar todos os

dados em armazenamento frio para economizar custos, você certamente reduzirá seus custos de armazenamento, mas à custa de tempos de recuperação prolongados e altos custos de recuperação se precisar acessar dados. O preço do armazenamento diminui de um armazenamento mais rápido/de melhor desempenho para um armazenamento mais baixo.

O armazenamento a frio é popular para arquivar dados. Historicamente, o armazenamento a frio envolvia backups físicos e, muitas vezes, o envio desses dados para terceiros que os arquivavam em um cofre literal. O armazenamento a frio é cada vez mais popular na nuvem. Todo fornecedor de nuvem oferece uma solução de dados frios e você deve pesar o custo de enviar dados para o armazenamento frio em relação ao custo e ao tempo para recuperá-los.

Os engenheiros de dados precisam levar em conta o transbordamento do armazenamento quente para quente/frio. A memória é cara e finita. Por exemplo, se dados dinâmicos forem armazenados na memória, eles poderão ser transferidos para o disco quando houver muitos dados novos para armazenar e não houver memória suficiente. Alguns bancos de dados podem mover dados acessados com pouca frequência para camadas mornas ou frias, descarregando os dados no disco rígido ou no armazenamento de objetos. O último é cada vez mais comum devido ao custo-benefício do armazenamento de objetos. Se você estiver na nuvem e usando serviços gerenciados, o transbordamento de disco ocorrerá automaticamente.

Se você estiver usando armazenamento de objetos baseado em nuvem, crie políticas de ciclo de vida automatizadas para seus dados. Isso reduzirá drasticamente seus custos de armazenamento. Por exemplo, se seus dados precisarem ser acessados apenas uma vez por mês, mova os dados para um nível de armazenamento de acesso não frequente. Se seus dados tiverem 180 dias e não forem acessados para consultas atuais, mova-os para uma camada de armazenamento de arquivamento. Em ambos os casos, você pode automatizar a migração de dados do armazenamento regular de objetos e economizar dinheiro. Dito isso, considere os custos de recuperação, tanto em tempo quanto em dinheiro, usando níveis de armazenamento pouco frequentes ou de estilo de arquivamento. Os tempos e custos de acesso e recuperação podem variar dependendo do provedor de nuvem. Alguns provedores de nuvem simplificam e barateiam a migração de dados para o armazenamento de arquivo, mas é caro e lento recuperar seus dados.

Retenção de dados

Nos primórdios do “big data”, havia uma tendência de errar acumulando todos os dados possíveis, independentemente de sua utilidade. A expectativa era: “podemos precisar desses dados no futuro”. Esse acúmulo de dados inevitavelmente se tornou pesado e sujo, dando origem a pântanos de dados e repressões regulatórias sobre a retenção de dados, entre outras consequências e pesadelos. Atualmente, os engenheiros de dados precisam considerar a retenção de dados: quais dados você *precisa* manter e como *longo* você deve mantê-lo? Aqui estão algumas coisas para se pensar sobre a retenção de dados.

Valor. Os dados são um ativo, portanto, você deve saber o valor dos dados que está armazenando. Obviamente, o valor é subjetivo e depende de quanto vale para seu caso de uso imediato e sua organização mais ampla. Esses dados são impossíveis de recriar ou podem ser facilmente recriados consultando sistemas upstream? Qual é o impacto para os usuários downstream se esses dados estiverem disponíveis ou não?

Tempo.O valor para usuários downstream também depende da idade dos dados. Novos dados são normalmente mais valiosos e acessados com frequência do que dados mais antigos. As limitações técnicas podem determinar por quanto tempo você pode armazenar dados em determinados níveis de armazenamento. Por exemplo, se você armazenar dados quentes em cache ou memória, provavelmente precisará definir um tempo de vida (TTL), para poder expirar os dados após um determinado ponto ou mantê-los em armazenamento quente ou frio. Caso contrário, seu armazenamento dinâmico ficará cheio e as consultas aos dados dinâmicos sofrerão atrasos no desempenho.

Conformidade.Certos regulamentos (por exemplo, HIPAA e Payment Card Industry, ou PCI) podem exigir que você mantenha os dados por um determinado período. Nessas situações, os dados simplesmente precisam estar acessíveis mediante solicitação, mesmo que a probabilidade de uma solicitação de acesso seja baixa. Outros regulamentos podem exigir que você mantenha os dados apenas por um período limitado de tempo, e você precisará ter a capacidade de excluir informações específicas a tempo e dentro das diretrizes de conformidade. Você precisará de um processo de armazenamento e arquivamento de dados — junto com a capacidade de pesquisar os dados — que atenda aos requisitos de retenção do regulamento específico que você precisa cumprir. Claro, você vai querer equilibrar a conformidade com o custo.

Custo.Os dados são um ativo que (espero) tem um ROI. Do lado do custo do ROI, uma despesa óbvia de armazenamento está associada aos dados. Considere a linha do tempo na qual você precisa reter os dados. Dada a nossa discussão sobre dados quentes, mornos e frios, implemente práticas automáticas de gerenciamento do ciclo de vida dos dados e move os dados para armazenamento frio se você não precisar dos dados após a data de retenção exigida. Ou exclua os dados se realmente não forem necessários.

Armazenamento de locatário único versus armazenamento de vários locatários

Em [Capítulo 3](#), cobrimos as compensações entre a arquitetura de locatário único e multilocatário. Para recapitular, *com monolocatário* a arquitetura, cada grupo de inquilinos (por exemplo, usuários individuais, grupos de usuários, contas ou clientes) obtém seu próprio conjunto dedicado de recursos, como rede, computação e armazenamento. *A Multi inquilino* a arquitetura inverte isso e compartilha esses recursos entre grupos de usuários. Ambas as arquiteturas são amplamente utilizadas. Esta seção analisa as implicações do armazenamento de locatário único e multilocatário.

Adotar o armazenamento de locatário único significa que cada locatário obtém seu armazenamento dedicado. No exemplo em [Figura 6-10](#), cada locatário obtém um banco de dados. Nenhum dado é compartilhado entre esses bancos de dados e o armazenamento é totalmente isolado. Um exemplo de uso do armazenamento de locatário único é que os dados de cada cliente devem ser armazenados isoladamente e não podem ser combinados com os dados de nenhum outro cliente. Nesse caso, cada cliente obtém seu próprio banco de dados.

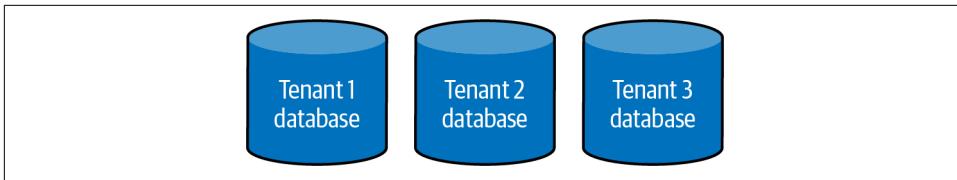


Figura 6-10. No armazenamento de inquilino único, cada inquilino obtém seu próprio banco de dados

Armazenamento de dados separado implica em esquemas separados e independentes, estruturas de balde e tudo relacionado ao armazenamento. Isso significa que você tem a liberdade de projetar o ambiente de armazenamento de cada locatário para ser uniforme ou deixá-los evoluir como quiserem. A variação do esquema entre os clientes pode ser uma vantagem e uma complicaçāo; como sempre, considere as compensações. Se o esquema de cada inquilino não for uniforme em todos os inquilinos, isso terá consequências importantes se você precisar consultar várias tabelas de inquilinos para criar uma exibição unificada de todos os dados do inquilino.

O armazenamento multilocatário permite o armazenamento de vários locatários em um único banco de dados. Por exemplo, em vez do cenário de locatário único em que os clientes obtêm seu próprio banco de dados, vários clientes podem residir nos mesmos esquemas de banco de dados ou tabelas em um banco de dados multilocatário. Armazenar dados de vários locatários significa que os dados de cada locatário são armazenados no mesmo local (Figura 6-11).

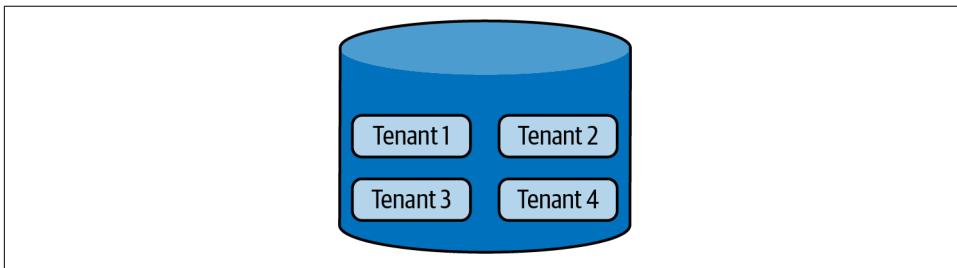


Figura 6-11. Neste armazenamento multitenant, quatro inquilinos ocupam o mesmo banco de dados

Você precisa estar ciente de consultar o armazenamento único e multilocatário, que abordamos com mais detalhes em [Capítulo 8](#).

Com quem você trabalhará

O armazenamento está no centro da infraestrutura de engenharia de dados. Você interagirá com os proprietários de sua infraestrutura de TI — normalmente, DevOps, segurança e arquitetos de nuvem. Definir domínios de responsabilidade entre a engenharia de dados e outras equipes é fundamental. Os engenheiros de dados têm autoridade para implantar sua infraestrutura em uma conta da AWS ou outra equipe deve lidar com essas alterações? Trabalhe com outras equipes para definir processos simplificados para que as equipes possam trabalhar juntas com eficiência e rapidez.

A divisão de responsabilidades pelo armazenamento de dados dependerá significativamente da maturidade da organização envolvida. O engenheiro de dados provavelmente gerenciará os sistemas de armazenamento e o fluxo de trabalho se a empresa estiver no início de sua maturidade de dados. Se a empresa estiver mais atrasada em sua maturidade de dados, o engenheiro de dados provavelmente gerenciará uma seção do sistema de armazenamento. Esse engenheiro de dados provavelmente também interagirá com engenheiros em ambos os lados do armazenamento — ingestão e transformação.

O engenheiro de dados precisa garantir que os sistemas de armazenamento usados pelos usuários downstream estejam disponíveis com segurança, contenham dados de alta qualidade, tenham ampla capacidade de armazenamento e funcionem quando as consultas e transformações forem executadas.

Subcorrentes

As subcorrentes para armazenamento são significativas porque o armazenamento é um hub crítico para todos os estágios do ciclo de vida da engenharia de dados. Ao contrário de outras subcorrentes para as quais os dados podem estar em movimento (ingestão) ou consultados e transformados, as subcorrentes para armazenamento diferem porque o armazenamento é tão onipresente.

Segurança

Embora os engenheiros geralmente vejam a segurança como um impedimento para seu trabalho, eles devem abraçar a ideia de que a segurança é um facilitador essencial. Segurança robusta em repouso e em movimento com controle de acesso de dados refinado permite que os dados sejam compartilhados e consumidos de forma mais ampla dentro de uma empresa. O valor dos dados aumenta significativamente quando isso é possível.

Como sempre, exercite o princípio do menor privilégio. Não dê acesso total ao banco de dados a ninguém, a menos que seja necessário. Isso significa que a maioria dos engenheiros de dados não precisa de acesso total ao banco de dados na prática. Além disso, preste atenção aos controles de acesso em nível de coluna, linha e célula em seu banco de dados. Forneça aos usuários apenas as informações de que precisam e nada mais.

Gestão de dados

O gerenciamento de dados é crítico à medida que lemos e gravamos dados com sistemas de armazenamento.

Catálogos de dados e gerenciamento de metadados

Os dados são aprimorados por metadados robustos. A catalogação capacita cientistas de dados, analistas e engenheiros de ML, permitindo a descoberta de dados. A linhagem de dados acelera o tempo para rastrear problemas de dados e permite que os consumidores localizem fontes brutas upstream. Ao desenvolver seus sistemas de armazenamento, invista em seus metadados. A integração de um dicionário de dados com essas outras ferramentas permite que os usuários compartilhem e registrem o conhecimento institucional de forma robusta.

O gerenciamento de metadados também melhora significativamente a governança de dados. Além de simplesmente habilitar a catalogação e linhagem passiva de dados, considere a implementação de análises nesses sistemas para obter uma visão clara e ativa do que está acontecendo com seus dados.

Controle de versão de dados no armazenamento de objetos

Os principais sistemas de armazenamento de objetos em nuvem permitem o controle de versão de dados. O controle de versão de dados pode ajudar na recuperação de erros quando os processos falham e os dados são corrompidos. O controle de versão também é benéfico para rastrear o histórico de conjuntos de dados usados para construir modelos. Assim como o controle de versão de código permite que os desenvolvedores rastreiem confirmações que causam bugs, o controle de versão de dados pode ajudar os engenheiros de ML a rastrear alterações que levam à degradação do desempenho do modelo.

Privacidade

O GDPR e outras regulamentações de privacidade afetaram significativamente o design do sistema de armazenamento. Quaisquer dados com implicações de privacidade têm um ciclo de vida que os engenheiros de dados devem gerenciar. Os engenheiros de dados devem estar preparados para responder às solicitações de exclusão de dados e removê-los seletivamente, conforme necessário. Além disso, os engenheiros podem acomodar privacidade e segurança por meio de anonimização e mascaramento.

DataOps

O DataOps não é ortogonal ao gerenciamento de dados e existe uma área significativa de sobreposição. A DataOps preocupa-se com o monitoramento operacional tradicional dos sistemas de armazenamento e com o monitoramento dos próprios dados, indissociáveis dos metadados e da qualidade.

Monitoramento de sistemas

Os engenheiros de dados devem monitorar o armazenamento de várias maneiras. Isso inclui monitorar os componentes de armazenamento da infraestrutura, onde eles existem, mas também monitorar o armazenamento de objetos e outros sistemas “sem servidor”. Os engenheiros de dados devem assumir a liderança em FinOps (gerenciamento de custos), monitoramento de segurança e monitoramento de acesso.

Observação e monitoramento de dados

Embora os sistemas de metadados descritos sejam críticos, uma boa engenharia deve considerar a natureza entrópica dos dados, buscando ativamente entender suas características e observando as principais mudanças. Os engenheiros podem monitorar estatísticas de dados, aplicar métodos de detecção de anomalias ou regras simples e testar e validar ativamente as inconsistências lógicas.

Arquitetura de dados

Capítulo 3 abrange os fundamentos da arquitetura de dados, já que o armazenamento é o ponto crítico do ciclo de vida da engenharia de dados.

Considere as seguintes dicas de arquitetura de dados. Projete para confiabilidade e durabilidade necessárias. Entenda os sistemas de origem upstream e como esses dados, uma vez ingeridos, serão armazenados e acessados. Entenda os tipos de modelos de dados e consultas que ocorrerão a jusante.

Se for esperado que os dados cresçam, você pode negociar o armazenamento com seu provedor de nuvem? Adote uma abordagem ativa para FinOps e trate-a como parte central das conversas sobre arquitetura. Não optimize prematuramente, mas prepare-se para escalar se existirem oportunidades de negócios ao operar em grandes volumes de dados.

Aposte em sistemas totalmente gerenciados e entenda os SLAs do provedor. Os sistemas totalmente gerenciados geralmente são muito mais robustos e escaláveis do que os sistemas que você precisa cuidar.

Orquestração

A orquestração está altamente emaranhada com o armazenamento. O armazenamento permite que os dados fluam por pipelines e a orquestração é a bomba. A orquestração também ajuda os engenheiros a lidar com a complexidade dos sistemas de dados, possivelmente combinando muitos sistemas de armazenamento e mecanismos de consulta.

Engenharia de software

Podemos pensar em engenharia de software no contexto de armazenamento de duas maneiras. Primeiro, o código que você escreve deve funcionar bem com seu sistema de armazenamento. Certifique-se de que o código que você escreve armazena os dados corretamente e não causa acidentalmente dados, vazamentos de memória ou problemas de desempenho. Em segundo lugar, defina sua infraestrutura de armazenamento como código e use recursos de computação efêmeros na hora de processar seus dados. Como o armazenamento é cada vez mais distinto da computação, você pode girar recursos automaticamente para cima e para baixo enquanto mantém seus dados no armazenamento de objetos. Isso mantém sua infraestrutura limpa e evita o acoplamento de suas camadas de armazenamento e consulta.

Conclusão

O armazenamento está em toda parte e é a base de muitos estágios do ciclo de vida da engenharia de dados. Neste capítulo, você aprendeu sobre ingredientes brutos, tipos, abstrações e grandes ideias sobre sistemas de armazenamento. Obtenha um conhecimento profundo do funcionamento interno e das limitações dos sistemas de armazenamento que você usará. Conheça os tipos de dados, atividades e cargas de trabalho apropriadas para seu armazenamento.

Recursos adicionais

- Página da Wikipédia “DBMS Orientado a Colunas”
- “O Projeto e Implementação de Sistemas de Banco de Dados Orientados a Colunas Modernos”por Daniel Abadi et al.
- *Projetando aplicativos com uso intensivo de dados*por Martin Kleppmann (O'Reilly)
- “Mergulhando no Lago Delta: Aplicação e Evolução do Esquema”por Burak Yavuz et al.
- “Dados quentes x dados frios: por que é importante”por Afzaal Ahmad Zeeshan
- IDCs“A criação e a replicação de dados crescerão a uma taxa mais rápida do que a capacidade de armazenamento instalada, de acordo com as previsões globais de DataSphere e StorageSphere da IDC” comunicado à imprensa
- “Rowise vs. Banco de dados colunar? Teoria e na Prática”por Mangat Rai Modi
- “Antipadrões da solução floco de neve: o provável cientista de dados”por John Aven
- “O que é um banco de dados vetorial?”por Bryan Turriff
- “O que é armazenamento de objetos? Uma definição e visão geral”por Alex Chan
- “O que, quando, por que e como das cargas incrementais”por Tim Mitchell

Ingestão

Você aprendeu sobre os vários sistemas de origem que provavelmente encontrará como engenheiro de dados e sobre as formas de armazenar dados. Vamos agora voltar nossa atenção para os padrões e escolhas que se aplicam à ingestão de dados de vários sistemas de origem. Neste capítulo, discutimos a ingestão de dados (consulte [Figura 7-1](#)), as principais considerações de engenharia para a fase de ingestão, os principais padrões para ingestão de lote e streaming, tecnologias que você encontrará, com quem trabalhará ao desenvolver seu pipeline de ingestão de dados e como as subcorrentes se apresentam na fase de ingestão.

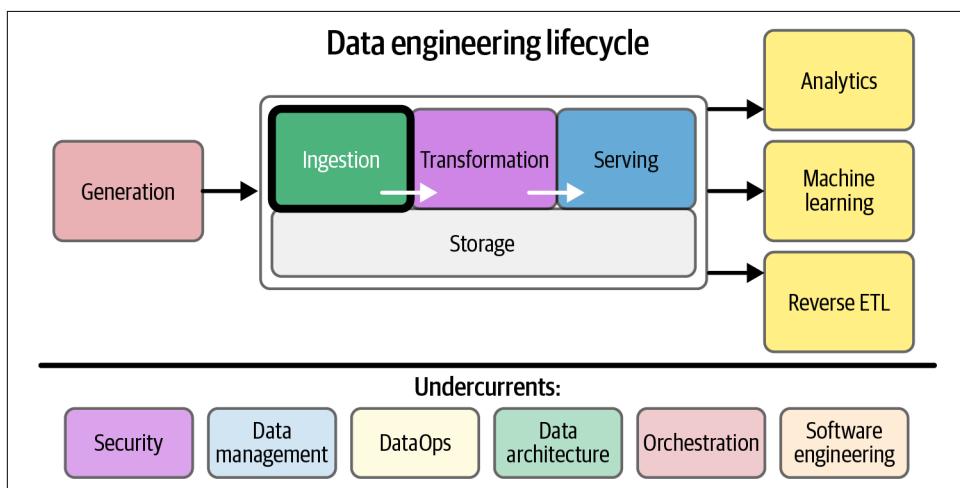


Figura 7-1. Para começar a processar os dados, devemos ingeri-los

O que é ingestão de dados?

ingestão de dados é o processo de mover dados de um lugar para outro. A ingestão de dados implica a movimentação de dados dos sistemas de origem para o armazenamento no ciclo de vida da engenharia de dados, com a ingestão como uma etapa intermediária (Figura 7-2).



Figura 7-2. Os dados do sistema 1 são ingeridos no sistema 2

Vale a pena contrastar rapidamente a ingestão de dados com a integração de dados. Enquanto *ingestão de dados* é o movimento de dados do ponto A para B, *integração de dados* combina dados de fontes diferentes em um novo conjunto de dados. Por exemplo, você pode usar a integração de dados para combinar dados de um sistema de CRM, dados de análise de publicidade e análise da web para criar um perfil de usuário, que é salvo em seu data warehouse. Além disso, usando ETL reverso, você pode enviar esse perfil de usuário recém-criado *voltar* ao seu CRM para que os vendedores possam usar os dados para priorizar leads. Descrevemos a integração de dados mais detalhadamente em [Capítulo 8](#), onde discutimos transformações de dados; ETL reverso é coberto em [Capítulo 9](#).

Ressaltamos também que a ingestão de dados é diferente da *ingestão interna* dentro de um sistema. Os dados armazenados em um banco de dados são copiados de uma tabela para outra ou os dados em um fluxo são temporariamente armazenados em cache. Consideraremos isso outra parte do processo geral de transformação de dados abordado em [Capítulo 8](#).

Pipelines de dados definidos

Os pipelines de dados começam nos sistemas de origem, mas a ingestão é o estágio em que os engenheiros de dados começam a projetar ativamente as atividades do pipeline de dados. No espaço da engenharia de dados, ocorre muita cerimônia em torno dos padrões de movimento e processamento de dados, com padrões estabelecidos como ETL, padrões mais recentes como ELT e novos nomes para práticas estabelecidas há muito tempo (ETL reverso) e compartilhamento de dados.

Todos esses conceitos são englobados na ideia de um *pipeline de dados*. É essencial entender os detalhes desses vários padrões e saber que um pipeline de dados moderno inclui todos eles. À medida que o mundo se afasta de uma abordagem monolítica tradicional com restrições rígidas na movimentação de dados e se aproxima de um ecossistema aberto de serviços em nuvem que são montados como peças de LEGO para realizar produtos, os engenheiros de dados priorizam o uso das ferramentas certas para alcançar o resultado desejado em vez de aderir a uma filosofia estreita de movimentação de dados.

Em geral, aqui está nossa definição de pipeline de dados:

Um pipeline de dados é a combinação de arquitetura, sistemas e processos que movem os dados pelos estágios do ciclo de vida da engenharia de dados.

Nossa definição é deliberadamente fluida - e intencionalmente vaga - para permitir que os engenheiros de dados insiram tudo o que precisam para realizar a tarefa em mãos. Um pipeline de dados pode ser um sistema ETL tradicional, no qual os dados são ingeridos de um sistema transacional local, passados por um processador monolítico e gravados em um data warehouse. Ou pode ser um pipeline de dados baseado em nuvem que extraí dados de 100 fontes, combina-os em 20 tabelas amplas, treina cinco outros modelos de ML, os implanta na produção e monitora o desempenho contínuo. Um pipeline de dados deve ser flexível o suficiente para atender a qualquer necessidade ao longo do ciclo de vida da engenharia de dados.

Vamos manter essa noção de pipelines de dados em mente enquanto avançamos neste capítulo.

Principais considerações de engenharia para a fase de ingestão

Ao se preparar para arquitetar ou construir um sistema de ingestão, aqui estão algumas considerações e perguntas principais a serem feitas relacionadas à ingestão de dados:

- Qual é o caso de uso dos dados que estou ingerindo?
- Posso reutilizar esses dados e evitar a ingestão de várias versões do mesmo conjunto de dados?
- **Para onde vão os dados? Qual é o destino?**
- Com que frequência os dados devem ser atualizados a partir da fonte?
- Qual é o volume de dados esperado?
- Em que formato estão os dados? O armazenamento downstream e a transformação podem aceitar esse formato?
- Os dados de origem estão em boas condições para uso imediato a jusante? Ou seja, os dados são de boa qualidade? Qual pós-processamento é necessário para atendê-lo? Quais são os riscos de qualidade de dados (por exemplo, o tráfego de bots para um site pode contaminar os dados)?
- Os dados requerem processamento em andamento para ingestão downstream se os dados forem de uma fonte de streaming?

Essas perguntas reduzem a ingestão de lote e streaming e se aplicam à arquitetura subjacente que você criará, construirá e manterá. Independentemente da frequência com que os dados são ingeridos, você deve considerar estes fatores ao projetar sua arquitetura de ingestão:

- Delimitado versus ilimitado
- Frequência
- Síncrono versus assíncrono

- Serialização e desserialização
- Rendimento e escalabilidade
- Confiabilidade e durabilidade
- Carga útil
- Push versus pull versus padrões de votação

Vejamos cada um deles.

Dados limitados versus dados ilimitados

Como você deve se lembrar de [Capítulo 3](#), os dados vêm em duas formas: limitados e ilimitados ([Figura 7-3](#)). *dados ilimitados* são os dados como existem na realidade, como os eventos acontecem, esporadicamente ou continuamente, em andamento e fluindo. *dados limitados* é uma maneira conveniente de agrupar dados em algum tipo de limite, como o tempo.

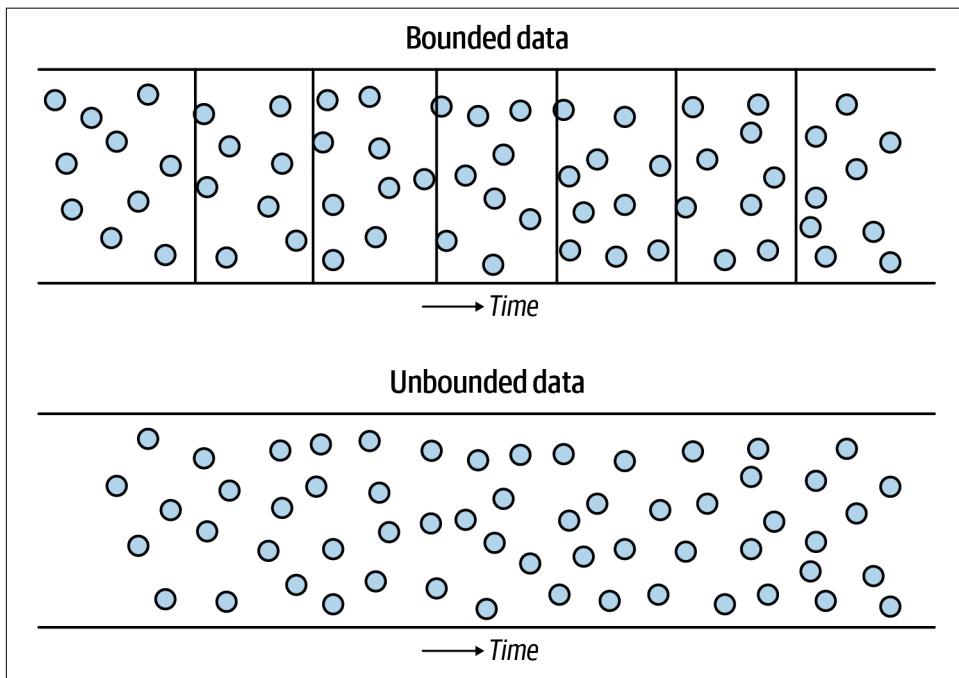


Figura 7-3. Dados limitados versus dados ilimitados

Adotemos este mantra: *Todos os dados são ilimitados até que sejam limitados*. Como muitos mantras, este não é preciso 100% das vezes. A lista de compras que escrevi esta tarde contém dados limitados. Escrevi como um fluxo de consciência (dados ilimitados) em um pedaço de papel, onde os pensamentos agora existem como uma lista de coisas (dados limitados) que preciso comprar no supermercado. No entanto, a ideia é correta para

propósitos práticos para a grande maioria dos dados com os quais você lidará em um contexto de negócios. Por exemplo, um varejista on-line processará as transações dos clientes 24 horas por dia até que o negócio falhe, a economia pare ou o sol exploda.

Os processos de negócios há muito impõem limites artificiais aos dados cortando lotes discretos. Tenha em mente o verdadeiro caráter ilimitado de seus dados; os sistemas de ingestão de streaming são simplesmente uma ferramenta para preservar a natureza ilimitada dos dados, para que as etapas subsequentes do ciclo de vida também possam processá-los continuamente.

Frequência

Uma das decisões críticas que os engenheiros de dados devem tomar ao projetar processos de ingestão de dados é a frequência de ingestão de dados. Os processos de ingestão podem ser em lote, microlote ou em tempo real.

As frequências de ingestão variam dramaticamente de lenta a rápida ([Figura 7-4](#)). No final lento, uma empresa pode enviar seus dados fiscais para uma empresa de contabilidade uma vez por ano. No lado mais rápido, um sistema CDC pode recuperar novas atualizações de log de um banco de dados de origem uma vez por minuto. Ainda mais rápido, um sistema pode ingerir continuamente eventos de sensores de IoT e processá-los em segundos. As frequências de ingestão de dados geralmente são misturadas em uma empresa, dependendo do caso de uso e das tecnologias.

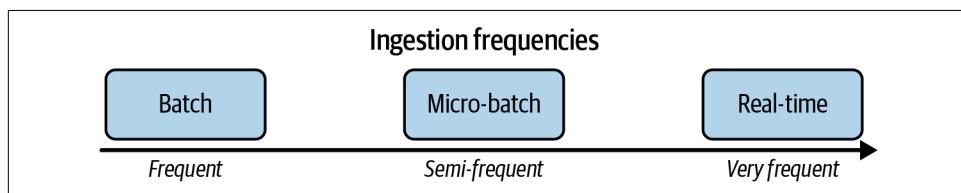


Figura 7-4. O lote de espectro para frequências de ingestão em tempo real

Observamos que os padrões de ingestão em “tempo real” estão se tornando cada vez mais comuns. Colocamos “tempo real” entre aspas porque nenhum sistema de ingestão é genuinamente em tempo real. Qualquer banco de dados, fila ou pipeline tem latência inherente ao entregar dados a um sistema de destino. É mais correto falar de *quase em tempo real*, mas costumamos usar *tempo real*/por brevidade. O padrão quase em tempo real geralmente elimina uma frequência de atualização explícita; os eventos são processados no pipeline um a um à medida que chegam ou em microlotes (ou seja, lotes em intervalos de tempo concisos). Para este livro, usaremos *tempo real*/*transmissão intercambiável*.

Mesmo com um processo de ingestão de dados de streaming, o downstream do processamento em lote é relativamente padrão. No momento da redação deste artigo, os modelos de ML são normalmente treinados em lotes, embora o treinamento on-line contínuo esteja se tornando mais comum. Raramente os engenheiros de dados têm a opção de criar um pipeline quase em tempo real sem componentes de lote. Em vez disso, eles escolhem onde os limites do lote ocorrerão - ou seja, o

os dados do ciclo de vida da engenharia de dados serão divididos em lotes. Depois que os dados chegam a um processo em lote, a frequência do lote se torna um gargalo para todo o processamento posterior.

Além disso, os sistemas de streaming são os mais adequados para muitos tipos de fonte de dados. Em aplicações de IoT, o padrão típico é que cada sensor grave eventos ou medições em sistemas de streaming conforme eles acontecem. Embora esses dados possam ser gravados diretamente em um banco de dados, uma plataforma de ingestão de streaming, como Amazon Kinesis ou Apache Kafka, é mais adequada para o aplicativo. Os aplicativos de software podem adotar padrões semelhantes gravando eventos em uma fila de mensagens à medida que ocorrem, em vez de esperar que um processo de extração extraia eventos e informações de estado de um banco de dados de back-end. Esse padrão funciona excepcionalmente bem para arquiteturas orientadas a eventos que já trocam mensagens por meio de filas. E, novamente, as arquiteturas de streaming geralmente coexistem com o processamento em lote.

Ingestão síncrona versus assíncrona

Comingestão síncrona, a origem, a ingestão e o destino têm dependências complexas e estão fortemente acoplados. Como você pode ver em Figura 7-5, cada estágio do ciclo de vida da engenharia de dados tem processos A, B e C diretamente dependentes um do outro. Se o processo A falhar, os processos B e C não poderão iniciar; se o processo B falhar, o processo C não inicia. Esse tipo de fluxo de trabalho síncrono é comum em sistemas ETL mais antigos, nos quais os dados extraídos de um sistema de origem devem ser transformados antes de serem carregados em um data warehouse. Os processos downstream da ingestão não podem ser iniciados até que todos os dados no lote tenham sido ingeridos. Se o processo de ingestão ou transformação falhar, todo o processo deverá ser executado novamente.

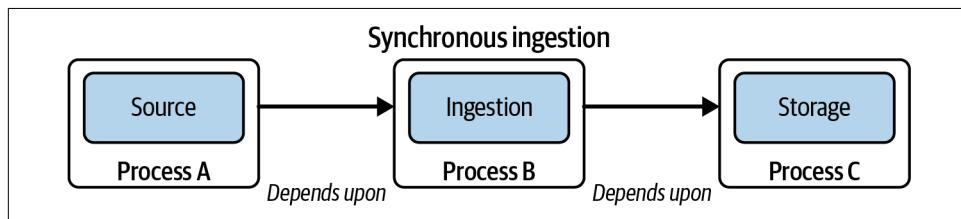


Figura 7-5. Um processo de ingestão síncrona é executado como etapas de lote discretas

Aqui está um pequeno estudo de caso de como não para projetar seus pipelines de dados. Em uma empresa, o próprio processo de transformação era uma série de dezenas de fluxos de trabalho síncronos fortemente acoplados, com todo o processo levando mais de 24 horas para ser concluído. Se qualquer etapa desse pipeline de transformação falhasse, todo o processo de transformação precisava ser reiniciado desde o início! Nesse caso, vimos processo após processo falhar e, devido a mensagens de erro inexistentes ou enigmáticas, consertar o pipeline foi um jogo de pescaria que levou mais de uma semana para diagnosticar e corrigir. Entretanto, o negócio não tinha relatórios atualizados durante esse tempo. As pessoas não estavam felizes.

Com *ingestão assíncrona*, as dependências agora podem operar no nível de eventos individuais, da mesma forma que fariam em um back-end de software construído a partir de microsserviços ([Figura 7-6](#)). Eventos individuais ficam disponíveis no armazenamento assim que são ingeridos individualmente. Veja o exemplo de um aplicativo web na AWS que emite eventos no Amazon Kinesis Data Streams (aqui atuando como um buffer). O stream é lido pelo Apache Beam, que analisa e enriquece os eventos e os encaminha para um segundo stream do Kinesis; O Kinesis Data Firehose acumula eventos e grava objetos no Amazon S3.

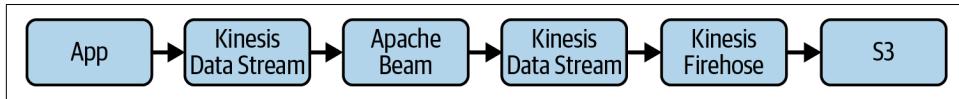


Figura 7-6. Processamento assíncrono de um fluxo de eventos na AWS

A grande ideia é que, em vez de depender do processamento assíncrono, em que um processo em lote é executado para cada estágio à medida que o lote de entrada é fechado e certas condições de tempo são atendidas, cada estágio do pipeline assíncrono pode processar itens de dados conforme eles se tornam disponíveis em paralelo no Aglomerado de feixes. A taxa de processamento depende dos recursos disponíveis. O Kinesis Data Stream atua como o amortecedor, moderando a carga para que os picos de taxa de evento não sobrecarreguem o processamento downstream. Os eventos passarão pelo pipeline rapidamente quando a taxa de eventos for baixa e qualquer acúmulo tiver sido eliminado. Observe que poderíamos modificar o cenário e usar um Streaming de dados do Kinesis para armazenamento, eventualmente extraíndo eventos para o S3 antes que eles expirem fora do stream.

Serialização e desserialização

Mover dados da origem para o destino envolve serialização e desserialização. Como lembrete, *serialização* significa codificar os dados de uma fonte e preparar estruturas de dados para transmissão e estágios intermediários de armazenamento.

Ao ingerir dados, certifique-se de que seu destino possa desserializar os dados recebidos. Vimos dados ingeridos de uma fonte, mas depois inertes e inutilizáveis no destino porque os dados não podem ser desserializados adequadamente. Veja a discussão mais extensa sobre serialização em [Apêndice A](#).

Taxa de transferência e escalabilidade

Em teoria, sua ingestão nunca deve ser um gargalo. Na prática, gargalos de ingestão são bastante comuns. A taxa de transferência de dados e a escalabilidade do sistema tornam-se críticas à medida que seus volumes de dados crescem e os requisitos mudam. Projete seus sistemas para dimensionar e encolher para corresponder com flexibilidade à taxa de transferência de dados desejada.

De onde você está ingerindo dados importa muito. Se você estiver recebendo dados à medida que são gerados, o sistema upstream terá algum problema que possa afetar seus pipelines de ingestão downstream? Por exemplo, suponha que um banco de dados de origem fique inativo. Quando

voltar a ficar on-line e tentar preencher os carregamentos de dados expirados, sua ingestão será capaz de acompanhar esse fluxo repentino de dados acumulados?

Outra coisa a considerar é sua capacidade de lidar com a ingestão de dados em rajada. A geração de dados raramente acontece em uma taxa constante e geralmente tem fluxos e refluxos. O buffer integrado é necessário para coletar eventos durante picos de taxa para evitar que os dados sejam perdidos. O buffer reduz o tempo enquanto o sistema é dimensionado e permite que os sistemas de armazenamento acomodem rajadas mesmo em um sistema escalável dinamicamente.

Sempre que possível, use serviços gerenciados que lidam com o dimensionamento da taxa de transferência para você. Embora você possa realizar essas tarefas manualmente adicionando mais servidores, shards ou workers, muitas vezes isso não é um trabalho de valor agregado e há uma boa chance de você perder alguma coisa. Muito desse trabalho pesado agora é automatizado. Não reinvente a roda de ingestão de dados se não for necessário.

Confiabilidade e Durabilidade

Confiabilidade e durabilidade são vitais nos estágios de ingestão de pipelines de dados. *Confiabilidade* implica alto tempo de atividade e failover adequado para sistemas de ingestão. *Durabilidade* envolve garantir que os dados não sejam perdidos ou corrompidos.

Algumas fontes de dados (por exemplo, dispositivos IoT e caches) podem não reter dados se não forem ingeridos corretamente. Uma vez perdido, ele se foi para sempre. Nesse sentido, o *confiabilidade* dos sistemas de ingestão leva diretamente ao *durabilidade* de dados gerados. Se os dados forem ingeridos, os processos downstream podem, teoricamente, ser executados com atraso se forem interrompidos temporariamente.

Nosso conselho é avaliar os riscos e criar um nível adequado de redundância e autocorreção com base no impacto e no custo da perda de dados. Confiabilidade e durabilidade têm custos diretos e indiretos. Por exemplo, seu processo de ingestão continuará se uma zona da AWS ficar inativa? Que tal uma região inteira? E a rede elétrica ou a internet? Claro, nada é de graça.

Quanto isso vai te custar? Você pode construir um sistema altamente redundante e ter uma equipe de plantão 24 horas por dia para lidar com interrupções. Isso também significa que seus custos de nuvem e mão-de-obra se tornam proibitivos (custos diretos) e o trabalho contínuo cobra um preço significativo de sua equipe (custos indiretos). Não há uma única resposta correta e você precisa avaliar os custos e benefícios de suas decisões de confiabilidade e durabilidade.

Não presuma que você pode construir um sistema que ingerirá dados de forma confiável e duradoura em todos os cenários possíveis. Mesmo o orçamento quase infinito do governo federal dos Estados Unidos não pode garantir isso. Em muitos cenários extremos, a ingestão de dados realmente não importa. Haverá pouco para ingerir se a internet cair, mesmo se você construir vários centros de dados isolados em bunkers subterrâneos com energia independente. Avalie continuamente as compensações e os custos de confiabilidade e durabilidade.

Carga útil

A *carga útil* é o conjunto de dados que você está ingerindo e tem características como tipo, forma, tamanho, esquema e tipos de dados e metadados. Vejamos algumas dessas características para entender por que isso é importante.

Tipo

O *tipo* é a quantidade de dados que você manipula afetando diretamente a forma como eles são tratados a jusante no ciclo de vida da engenharia de dados. Tipo consiste em tipo e formato. Os dados têm um tipo—tabular, imagem, vídeo, texto, etc. O tipo influencia diretamente o formato dos dados ou a forma como são expressos em bytes, nomes e extensões de arquivo. Por exemplo, um tipo tabular de dados pode estar em formatos como CSV ou Parquet, com cada um desses formatos tendo diferentes padrões de bytes para serialização e desserialização. Outro tipo de dado é uma imagem, que tem formato JPG ou PNG e é inherentemente não estruturada.

Forma

Cada carga tem um *formato* que descreve suas dimensões. A forma dos dados é crítica em todo o ciclo de vida da engenharia de dados. Por exemplo, as dimensões de pixel e vermelho, verde e azul (RGB) de uma imagem são necessárias para treinar modelos de aprendizado profundo. Como outro exemplo, se você estiver tentando importar um arquivo CSV para uma tabela de banco de dados e seu CSV tiver mais colunas do que a tabela de banco de dados, provavelmente ocorrerá um erro durante o processo de importação. Aqui estão alguns exemplos das formas de vários tipos de dados:

Tabular

O número de linhas e colunas no conjunto de dados, geralmente expresso como M linhas e N colunas

JSON semiestruturado

Os pares chave-valor e profundidade de aninhamento ocorrem com subelementos

Texto não estruturado

Número de palavras, caracteres ou bytes no corpo do texto

Imagens

A largura, altura e profundidade de cor RGB (por exemplo, 8 bits por pixel)

Áudio não compactado

Número de canais (por exemplo, dois para estéreo), profundidade da amostra (por exemplo, 16 bits por amostra), taxa de amostragem (por exemplo, 48 kHz) e duração (por exemplo, 10.003 segundos)

Tamanho

O *tamanho* dos dados descreve o número de bytes de uma carga útil. Uma carga útil pode variar em tamanho de bytes únicos a terabytes e maiores. Para reduzir o tamanho de uma carga útil,

pode ser compactado em vários formatos, como ZIP e TAR (consulte a discussão sobre compactação em [Apêndice A](#)).

Uma carga massiva também pode ser dividida em partes, o que efetivamente reduz o tamanho da carga em subseções menores. Ao carregar um arquivo enorme em um armazenamento de objetos em nuvem ou data warehouse, essa é uma prática comum, pois os pequenos arquivos individuais são mais fáceis de transmitir em uma rede (especialmente se estiverem compactados). Os arquivos fragmentados menores são enviados ao seu destino e, em seguida, remontados após a chegada de todos os dados.

Esquema e tipos de dados

Muitos payloads de dados têm um esquema, como dados tabulares e semiestruturados. Conforme mencionado anteriormente neste livro, um esquema descreve os campos e os tipos de dados dentro desses campos. Outros dados, como texto não estruturado, imagens e áudio, não terão um esquema explícito ou tipos de dados. No entanto, eles podem vir com descrições técnicas de arquivos sobre forma, dados e formato de arquivo, codificação, tamanho, etc.

Embora você possa se conectar a bancos de dados de várias maneiras (como exportação de arquivo, CDC, JDBC/ODBC), a conexão é fácil. O grande desafio da engenharia é entender o esquema subjacente. Os aplicativos organizam os dados de várias maneiras, e os engenheiros precisam estar intimamente familiarizados com a organização dos dados e com os padrões de atualização relevantes para entendê-los. O problema foi um tanto exacerbado pela popularidade do mapeamento objeto-relacional (ORM), que gera automaticamente esquemas baseados na estrutura do objeto em linguagens como Java ou Python. As estruturas naturais em uma linguagem orientada a objetos geralmente mapeiam para algo confuso em um banco de dados operacional. Os engenheiros de dados podem precisar se familiarizar com a estrutura de classe do código do aplicativo.

Schema não é apenas para bancos de dados. Conforme discutimos, as APIs apresentam suas complicações de esquema. Muitas APIs de fornecedores têm métodos de relatórios amigáveis que preparam dados para análises. Em outros casos, os engenheiros não têm tanta sorte. A API é um wrapper fino em torno dos sistemas subjacentes, exigindo que os engenheiros entendam os componentes internos do aplicativo para usar os dados.

Grande parte do trabalho associado à ingestão de esquemas de origem ocorre no estágio de transformação do ciclo de vida da engenharia de dados, que discutimos em [Capítulo 8](#). Colocamos essa discussão aqui porque os engenheiros de dados precisam começar a estudar os esquemas de origem assim que planejam ingerir dados de uma nova origem.

A comunicação é crítica para entender os dados de origem, e os engenheiros também têm a oportunidade de reverter o fluxo de comunicação e ajudar os engenheiros de software a melhorar os dados onde eles são produzidos. Mais adiante neste capítulo, retornaremos a esse tópico em “[Com quem você trabalhará](#)” na [página 262](#).

Detectar e manipular alterações de esquema upstream e downstream. Mudanças no esquema freqüentemente ocorrem em sistemas de origem e geralmente estão fora do controle dos engenheiros de dados. Exemplos de alterações de esquema incluem o seguinte:

- Adicionando uma nova coluna
- Alterar um tipo de coluna
- Criando uma nova tabela
- Renomear uma coluna

Está se tornando cada vez mais comum que as ferramentas de ingestão automatizem a detecção de alterações de esquema e até mesmo atualizem automaticamente as tabelas de destino. Em última análise, isso é uma espécie de bênção mista. As alterações de esquema ainda podem interromper os pipelines downstream de preparo e ingestão.

Os engenheiros ainda devem implementar estratégias para responder às mudanças automaticamente e alertar sobre mudanças que não podem ser acomodadas automaticamente. A automação é excelente, mas os analistas e cientistas de dados que dependem desses dados devem ser informados sobre as mudanças de esquema que violam as suposições existentes. Mesmo que a automação possa acomodar uma mudança, o novo esquema pode afetar adversamente o desempenho de relatórios e modelos. A comunicação entre aqueles que fazem alterações no esquema e os afetados por essas alterações é tão importante quanto uma automação confiável que verifica as alterações no esquema.

Registros de esquema. No streaming de dados, cada mensagem tem um esquema, e esses esquemas podem evoluir entre produtores e consumidores. *A registro de esquema* é um repositório de metadados usado para manter a integridade do esquema e do tipo de dados em face de esquemas em constante mudança. Os registros de esquema também podem rastrear versões e histórico de esquema. Ele descreve o modelo de dados para mensagens, permitindo serialização e desserialização consistentes entre produtores e consumidores. Os registros de esquema são usados na maioria das principais ferramentas de dados e nuvens.

Metadados

Além das características aparentes que acabamos de abordar, uma carga geralmente contém metadados, que discutimos pela primeira vez em [Capítulo 2](#). Metadados são dados sobre dados. Os metadados podem ser tão críticos quanto os próprios dados. Uma das limitações significativas da abordagem inicial do data lake — ou pântano de dados, que poderia se tornar um site de superfundo de dados — foi a total falta de atenção aos metadados. Sem uma descrição detalhada dos dados, pode ser de pouco valor. Já discutimos alguns tipos de metadados (por exemplo, esquema) e os abordaremos várias vezes ao longo deste capítulo.

Padrões Push versus Pull versus Poll

Mencionamos push versus pull quando introduzimos o ciclo de vida da engenharia de dados em [Capítulo 2](#). A empurrarestratégia ([Figura 7-7](#)) envolve um sistema de origem enviando dados para um destino, enquanto uma puxarestratégia ([Figura 7-8](#)) envolve um alvo lendo dados diretamente de uma fonte. Como mencionamos naquela discussão, as linhas entre essas estratégias são tênues.

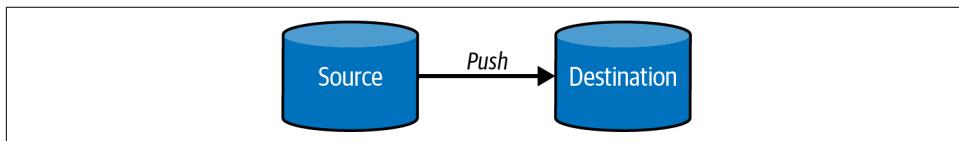


Figura 7-7. Empurrando dados da origem para o destino

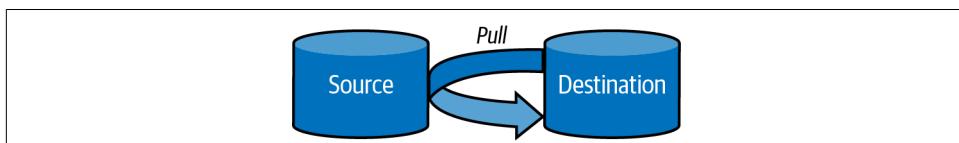


Figura 7-8. Um destino puxando dados de uma fonte

Outro padrão relacionado ao puxar é [sondagem](#) para dados ([Figura 7-9](#)). A sondagem envolve a verificação periódica de uma fonte de dados quanto a quaisquer alterações. Quando as alterações são detectadas, o destino extrai os dados como faria em uma situação de pull regular.

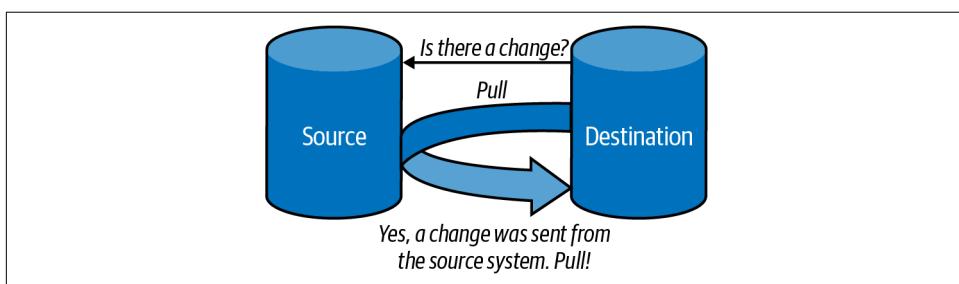


Figura 7-9. Sondagem de mudanças em um sistema de origem

Considerações de ingestão de lote

A ingestão em lote, que envolve o processamento de dados em massa, geralmente é uma maneira conveniente de ingerir dados. Isso significa que os dados são ingeridos obtendo um subconjunto de dados de um sistema de origem, com base em um intervalo de tempo ou no tamanho dos dados acumulados ([Figura 7-10](#)).

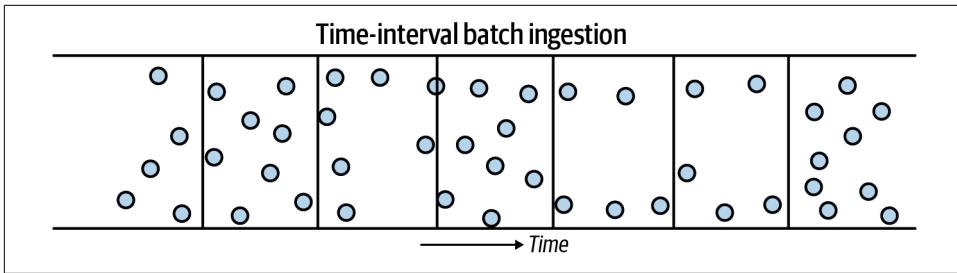


Figura 7-10. Ingestão em lote de intervalo de tempo

Ingestão em lote de intervalo de tempo é difundido em ETL de negócios tradicional para armazenamento de dados. Esse padrão geralmente é usado para processar dados uma vez por dia, durante a noite fora do horário comercial, para fornecer relatórios diários, mas outras frequências também podem ser usadas.

Ingestão em lote baseada em tamanho (Figura 7-11) é bastante comum quando os dados são movidos de um sistema baseado em streaming para o armazenamento de objetos; em última análise, você deve cortar os dados em blocos discretos para processamento futuro em um data lake. Alguns sistemas de ingestão baseados em tamanho podem dividir dados em objetos com base em vários critérios, como o tamanho em bytes do número total de eventos.

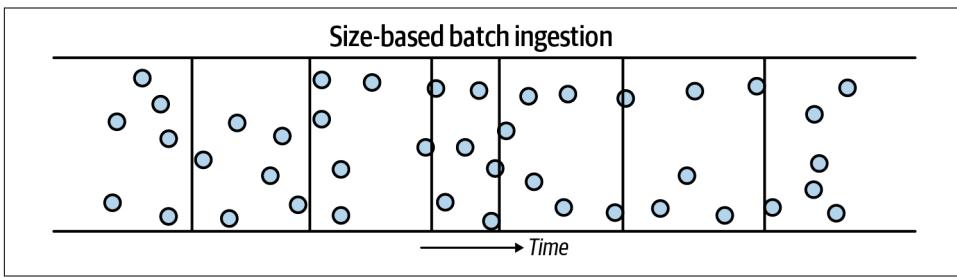


Figura 7-11. Ingestão em lote baseada em tamanho

Alguns padrões de ingestão de lote comumente usados, que discutimos nesta seção, incluem o seguinte:

- Instantâneo ou extração diferencial
- Exportação e ingestão com base em arquivo
- ETL versus ELT
- Inserções, atualizações e tamanho do lote
- Migração de dados

Instantâneo ou Extração Diferencial

Os engenheiros de dados devem escolher se desejam capturar instantâneos completos de um sistema de origem ou diferencial (às vezes chamado *incremental*) atualizações. Com *instantâneos completos*, os engenheiros capturam todo o estado atual do sistema de origem em cada leitura de atualização. Com o *atualização diferencial* padrão, os engenheiros podem extrair apenas as atualizações e alterações desde a última leitura do sistema de origem. Embora as atualizações diferenciais sejam ideais para minimizar o tráfego de rede e o uso de armazenamento de destino, as leituras completas de instantâneos permanecem extremamente comuns devido à sua simplicidade.

Exportação e processamento com base em arquivo

Os dados são frequentemente movidos entre bancos de dados e sistemas usando arquivos. Os dados são serializados em arquivos em um formato trocável e esses arquivos são fornecidos a um sistema de ingestão. Consideraremos a exportação baseada em arquivo *uma baseado em push* padrão de ingestão. Isso ocorre porque o trabalho de exportação e preparação de dados é feito no lado do sistema de origem.

A ingestão baseada em arquivo tem várias vantagens potenciais sobre uma abordagem de conexão direta com o banco de dados. Muitas vezes, é indesejável permitir o acesso direto aos sistemas de back-end por motivos de segurança. Com a ingestão baseada em arquivo, os processos de exportação são executados no lado da fonte de dados, dando aos engenheiros do sistema de origem controle total sobre quais dados são exportados e como os dados são pré-processados. Depois que os arquivos são concluídos, eles podem ser fornecidos ao sistema de destino de várias maneiras. Métodos comuns de troca de arquivos são armazenamento de objetos, protocolo de transferência segura de arquivos (SFTP), intercâmbio eletrônico de dados (EDI) ou cópia segura (SCP).

ETL versus ELT

[Capítulo 3](#) introduziu ETL e ELT, ambos padrões extremamente comuns de ingestão, armazenamento e transformação que você encontrará em cargas de trabalho em lote. A seguir estão breves definições das partes de extração e carregamento de ETL e ELT:

Extraír

Isso significa obter dados de um sistema de origem. Enquanto *extraír* parece implicar *puxar* dados, também pode ser baseado em *push*. A extração também pode exigir a leitura de metadados e alterações de esquema.

Carregar

Depois que os dados são extraídos, eles podem ser transformados (ETL) antes de serem carregados em um destino de armazenamento ou simplesmente carregados no armazenamento para transformação futura. Ao carregar dados, você deve estar atento ao tipo de sistema que está carregando, ao esquema dos dados e ao impacto do carregamento no desempenho.

Cobrimos ETL e ELT com mais detalhes em [Capítulo 8](#).

Inserções, atualizações e tamanho do lote

Os sistemas orientados a lote geralmente têm um desempenho ruim quando os usuários tentam executar muitas operações em pequenos lotes, em vez de um número menor de operações grandes. Por exemplo, embora seja comum inserir uma linha por vez em um banco de dados transacional, esse é um padrão ruim para muitos bancos de dados colunares, pois força a criação de muitos arquivos pequenos e abaixo do ideal e força o sistema a executar um alto número de *criar objeto* operações. A execução de muitas pequenas operações de atualização no local é um problema ainda maior porque faz com que o banco de dados verifique cada arquivo de coluna existente para executar a atualização.

Entenda os padrões de atualização apropriados para o banco de dados ou armazenamento de dados com o qual você está trabalhando. Além disso, entenda que certas tecnologias são desenvolvidas especificamente para altas taxas de inserção. Por exemplo, Apache Druid e Apache Pinot podem lidar com altas taxas de inserção. O SingleStore pode gerenciar cargas de trabalho híbridas que combinam características OLAP e OLTP. O BigQuery tem um desempenho ruim em uma alta taxa de inserções de linha única de SQL simples, mas extremamente bom se os dados forem alimentados por meio de seu buffer de fluxo. Conheça os limites e características de suas ferramentas.

Migração de dados

A migração de dados para um novo banco de dados ou ambiente geralmente não é trivial e os dados precisam ser movidos em massa. Às vezes, isso significa mover tamanhos de dados de centenas de terabytes ou muito maiores, geralmente envolvendo a migração de tabelas específicas e a movimentação de bancos de dados e sistemas inteiros.

As migrações de dados provavelmente não são uma ocorrência regular para um engenheiro de dados, mas você deve estar familiarizado com elas. Como costuma ser o caso da ingestão de dados, o gerenciamento de esquema é uma consideração crucial. Suponha que você esteja migrando dados de um sistema de banco de dados para outro (digamos, SQL Server para Snowflake). Não importa o quanto os dois bancos de dados se assemelham, quase sempre existem diferenças sutis na maneira como eles lidam com o esquema. Felizmente, geralmente é fácil testar a ingestão de uma amostra de dados e encontrar problemas de esquema antes de realizar uma migração completa da tabela.

A maioria dos sistemas de dados tem melhor desempenho quando os dados são movidos em massa, em vez de linhas ou eventos individuais. O armazenamento de arquivos ou objetos geralmente é um excelente estágio intermediário para a transferência de dados. Além disso, um dos maiores desafios da migração do banco de dados não é a movimentação dos dados em si, mas a movimentação das conexões do pipeline de dados do sistema antigo para o novo.

Esteja ciente de que muitas ferramentas estão disponíveis para automatizar vários tipos de migrações de dados. Especialmente para migrações grandes e complexas, sugerimos examinar essas opções antes de fazer isso manualmente ou escrever sua própria solução de migração.

Considerações sobre a ingestão de mensagens e streams

A ingestão de dados de eventos é comum. Esta seção aborda questões que você deve considerar ao ingerir eventos, com base nos tópicos abordados nos capítulos 5 e 6.

Evolução do Esquema

A evolução do esquema é comum ao lidar com dados de eventos; os campos podem ser adicionados ou removidos, ou os tipos de valor podem mudar (digamos, uma string para um número inteiro). A evolução do esquema pode ter impactos não intencionais em seus pipelines e destinos de dados. Por exemplo, um dispositivo IoT obtém uma atualização de firmware que adiciona um novo campo ao evento que ele transmite, ou uma API de terceiros introduz alterações em sua carga útil de eventos ou inúmeros outros cenários. Tudo isso afeta potencialmente seus recursos downstream.

Para aliviar os problemas relacionados à evolução do esquema, aqui estão algumas sugestões. Primeiro, se sua estrutura de processamento de eventos tiver um registro de esquema (discutido anteriormente neste capítulo), use-o para controlar a versão de suas alterações de esquema. Em seguida, uma fila de mensagens mortas (descrita em “[Tratamento de erros e filas de devoluções](#)” na página 259) pode ajudá-lo a investigar problemas com eventos que não são tratados adequadamente. Por fim, a rota de baixa fidelidade (e a mais eficaz) é comunicar-se regularmente com as partes interessadas upstream sobre possíveis alterações de esquema e abordar proativamente as alterações de esquema com as equipes que introduzem essas alterações, em vez de reagir ao recebimento de alterações de interrupção.

Dados de Chegada Atrasada

Embora você provavelmente prefira que todos os dados de eventos cheguem a tempo, os dados de eventos podem chegar atrasados. Um grupo de eventos pode ocorrer no mesmo período de tempo (tempos de eventos semelhantes), mas alguns podem chegar mais tarde do que outros (tempos de ingestão atrasados) devido a várias circunstâncias.

Por exemplo, um dispositivo IoT pode atrasar o envio de uma mensagem devido a problemas de latência da Internet. Isso é comum ao ingerir dados. Você deve estar ciente dos dados que chegam tarde e do impacto nos sistemas e usos posteriores. Suponha que você assuma que o tempo de ingestão ou processo é o mesmo que o tempo do evento. Você pode obter alguns resultados estranhos se seus relatórios ou análises dependerem de um retrato preciso de quando os eventos ocorrem. Para lidar com dados atrasados, você precisa definir um horário limite para quando os dados atrasados não serão mais processados.

Pedidos e entrega múltipla

As plataformas de streaming geralmente são construídas a partir de sistemas distribuídos, o que pode causar algumas complicações. Especificamente, as mensagens podem ser entregues fora de ordem e mais de uma vez (entrega pelo menos uma vez). Veja a discussão sobre plataformas de transmissão de eventos em [capítulo 5](#) para mais detalhes.

Repetir

Repetir permite que os leitores solicitem uma variedade de mensagens do histórico, permitindo que você retroceda seu histórico de eventos até um ponto específico no tempo. A reprodução é um recurso importante em muitas plataformas de ingestão de streaming e é particularmente útil quando você precisa reingrerir e reprocessar dados para um intervalo de tempo específico. Por exemplo, o RabbitMQ normalmente exclui as mensagens depois que todos os assinantes as consomem. Kafka, Kinesis e Pub/Sub oferecem suporte à retenção e reprodução de eventos.

Tempo de Viver

Por quanto tempo você preservará o registro do seu evento? Um parâmetro chave é *tempo máximo de retenção de mensagem*, também conhecido como *tempo de Viver*(TTL). TTL geralmente é uma configuração que você definirá por quanto tempo deseja que os eventos vivam antes de serem reconhecidos e ingeridos. Qualquer evento não reconhecido que não seja ingerido depois que seu TTL expira desaparece automaticamente. Isso é útil para reduzir a contrapressão e o volume desnecessário de eventos em seu pipeline de ingestão de eventos.

Encontre o equilíbrio certo do impacto TTL em nosso pipeline de dados. Um TTL extremamente curto (milissegundos ou segundos) pode fazer com que a maioria das mensagens desapareça antes do processamento. Um TTL muito longo (várias semanas ou meses) criará um acúmulo de muitas mensagens não processadas, resultando em longos tempos de espera.

Vejamos como algumas plataformas populares lidam com o TTL no momento em que este livro foi escrito. O Google Cloud Pub/Sub oferece suporte a períodos de retenção de até 7 dias. A retenção do Amazon Kinesis Data Streams pode ser ativada em até 365 dias. O Kafka pode ser configurado para retenção indefinida, limitada pelo espaço em disco disponível. (O Kafka também oferece suporte à opção de gravar mensagens mais antigas no armazenamento de objetos em nuvem, desbloqueando espaço de armazenamento e retenção virtualmente ilimitados.)

Tamanho da mensagem

O tamanho da mensagem é um problema facilmente ignorado: você deve garantir que a estrutura de streaming em questão possa lidar com o tamanho máximo esperado da mensagem. O Amazon Kinesis oferece suporte a um tamanho máximo de mensagem de 1 MB. O padrão do Kafka é esse tamanho máximo, mas pode ser configurado para um máximo de 20 MB ou mais. (A configurabilidade pode variar em plataformas de serviço gerenciado.)

Tratamento de erros e filas de mensagens mortas

Às vezes, os eventos não são ingeridos com sucesso. Talvez um evento seja enviado para um tópico inexistente ou fila de mensagens, o tamanho da mensagem pode ser muito grande ou o evento expirou além de seu TTL. Eventos que não podem ser ingeridos precisam ser reencaminhados e armazenados em um local separado chamado de *fila de mensagens mortas*.

Uma fila de mensagens mortas separa eventos problemáticos de eventos que podem ser aceitos pelo consumidor (Figura 7-12). Se os eventos não forem reencaminhados para uma fila de devoluções, esses eventos errôneos correm o risco de bloquear a ingestão de outras mensagens. Os engenheiros de dados podem usar uma fila de mensagens mortas para diagnosticar por que ocorrem erros de ingestão de eventos e resolver problemas de pipeline de dados e podem reprocessar algumas mensagens na fila após corrigir a causa subjacente dos erros.

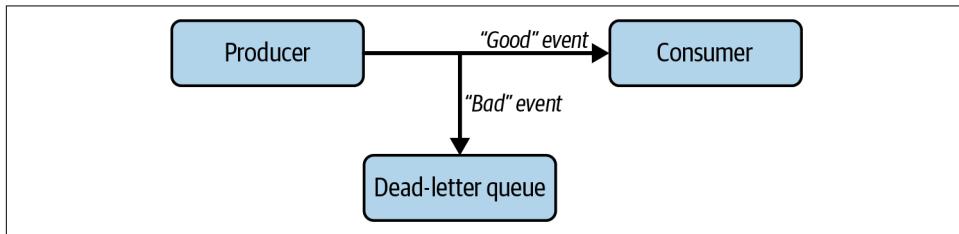


Figura 7-12. Eventos “bons” são passados para o consumidor, enquanto eventos “ruins” são armazenados em uma fila de mensagens mortas

Puxe e empurre o consumidor

Um consumidor que se inscreve em um tópico pode obter eventos de duas maneiras: push e pull. Vejamos como algumas tecnologias de streaming extraem e enviam dados. Kafka e Kinesis suportam apenas assinaturas pull. Os assinantes leem as mensagens de um tópico e confirmam quando elas foram processadas. Além de assinaturas pull, Pub/Sub e RabbitMQ oferecem suporte a assinaturas push, permitindo que esses serviços gravem mensagens para um ouvinte.

As assinaturas pull são a escolha padrão para a maioria dos aplicativos de engenharia de dados, mas você pode querer considerar os recursos push para aplicativos especializados. Observe que os sistemas de ingestão de mensagens pull-only ainda podem enviar por push se você adicionar uma camada extra para lidar com isso.

Localização

Muitas vezes, é desejável integrar o streaming em vários locais para redundância aprimorada e consumir dados próximos de onde são gerados. Como regra geral, quanto mais próximo estiver sua ingestão de onde os dados se originam, melhor será sua largura de banda e latência. No entanto, você precisa equilibrar isso com os custos de movimentação de dados entre regiões para executar análises em um conjunto de dados combinado. Como sempre, os custos de saída de dados podem aumentar rapidamente. Faça uma avaliação cuidadosa dos trade-offs ao construir sua arquitetura.

Formas de ingerir dados

Agora que descrevemos alguns dos padrões significativos subjacentes à ingestão de lote e streaming, vamos nos concentrar nas maneiras de ingerir dados. Embora vamos citar

algumas formas comuns, tenha em mente que o universo de práticas e tecnologias de ingestão de dados é vasto e cresce a cada dia.

Coneção direta com o banco de dados

Os dados podem ser extraídos de bancos de dados para ingestão por consulta e leitura em uma conexão de rede. Mais comumente, essa conexão é feita usando ODBC ou JDBC.

O ODBC usa um driver hospedado por um cliente que acessa o banco de dados para traduzir comandos emitidos para a API ODBC padrão em comandos emitidos para o banco de dados. O banco de dados retorna os resultados da consulta pela rede, onde o motorista os recebe e os converte de volta em um formulário padrão para ser lido pelo cliente. Para ingestão, o aplicativo que utiliza o driver ODBC é uma ferramenta de ingestão. A ferramenta de ingestão pode extrair dados por meio de várias consultas pequenas ou de uma única consulta grande.

O JDBC é conceitualmente muito semelhante ao ODBC. Um driver Java se conecta a um banco de dados remoto e serve como uma camada de tradução entre a API JDBC padrão e a interface de rede nativa do banco de dados de destino. Pode parecer estranho ter uma API de banco de dados dedicada a uma única linguagem de programação, mas existem fortes motivações para isso. A Java Virtual Machine (JVM) é padrão, portátil em arquiteturas de hardware e sistemas operacionais e fornece o desempenho de código compilado por meio de um compilador just-in-time (JIT). A JVM é uma VM de compilação extremamente popular para executar código de maneira portável.

JDBC fornece extraordinária portabilidade de driver de banco de dados. Os drivers ODBC são fornecidos como binários nativos de sistema operacional e arquitetura; os fornecedores de banco de dados devem manter versões para cada arquitetura/versão de sistema operacional que desejam oferecer suporte. Por outro lado, os fornecedores podem fornecer um único driver JDBC compatível com qualquer linguagem JVM (por exemplo, Java, Scala, Clojure ou Kotlin) e estrutura de dados JVM (ou seja, Spark). O JDBC tornou-se tão popular que também é usado como uma interface para linguagens não-JVM, como Python; o ecossistema Python fornece ferramentas de tradução que permitem que o código Python se comunique com um driver JDBC em execução em uma JVM local.

JDBC e ODBC são usados extensivamente para ingestão de dados de bancos de dados relacionais, retornando ao conceito geral de conexões diretas de banco de dados. Vários aprimoramentos são usados para acelerar a ingestão de dados. Muitas estruturas de dados podem paralelizar várias conexões simultâneas e consultas de partição para extrair dados em paralelo. Por outro lado, nada é de graça; usar conexões paralelas também aumenta a carga no banco de dados de origem.

JDBC e ODBC foram por muito tempo os padrões de ouro para ingestão de dados de bancos de dados, mas esses padrões de conexão estão começando a mostrar sua idade para muitos aplicativos de engenharia de dados. Esses padrões de conexão lutam com dados aninhados e enviam dados como linhas. Isso significa que os dados aninhados nativos devem ser recodificados como dados de string para

ser enviados pela rede, e as colunas dos bancos de dados colunares devem ser reserializadas como linhas.

Conforme discutido em “[Exportação e processamento com base em arquivo](#)” na página 256, muitos bancos de dados agora suportam exportação de arquivo nativo que ignora JDBC/ODBC e exporta dados diretamente em formatos como Parquet, ORC e Avro. Como alternativa, muitos data warehouses em nuvem fornecem APIs REST diretas.

As conexões JDBC geralmente devem ser integradas com outras tecnologias de ingestão. Por exemplo, geralmente usamos um processo de leitura para conectar a um banco de dados com JDBC, gravar os dados extraídos em vários objetos e, em seguida, orquestrar a ingestão em um sistema downstream (consulte [Figura 7-13](#)). O processo de leitura pode ser executado em uma instância de nuvem totalmente efêmera ou em um sistema de orquestração.

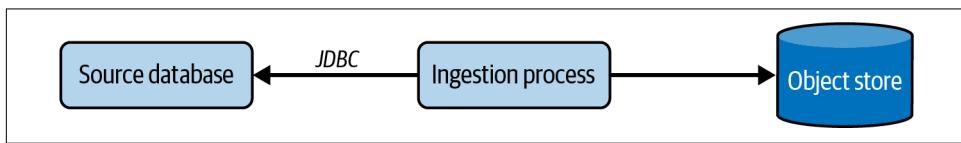


Figura 7-13. Um processo de ingestão lê de um banco de dados de origem usando JDBC e, em seguida, grava objetos no armazenamento de objetos. Um banco de dados de destino (não mostrado) pode ser acionado para ingerir os dados com uma chamada de API de um sistema de orquestração.

Alterar captura de dados

Alterar captura de dados(CDC), introduzido em [Capítulo 2](#), é o processo de ingestão de alterações de um sistema de banco de dados de origem. Por exemplo, podemos ter um sistema PostgreSQL de origem que suporta um aplicativo e periodicamente ou continuamente ingere mudanças na tabela para análises.

Observe que nossa discussão aqui não é de forma alguma exaustiva. Apresentamos a você padrões comuns, mas sugerimos que você leia a documentação em um banco de dados específico para lidar com os detalhes das estratégias de CDC.

CDC orientado a lote

Se a tabela do banco de dados em questão tiver uma `updated_at` contendo a última vez que um registro foi gravado ou atualizado, podemos consultar a tabela para encontrar todas as linhas atualizadas desde um horário especificado. Definimos o carimbo de data/hora do filtro com base na última vez que capturamos as linhas alteradas das tabelas. Esse processo nos permite extrair alterações e atualizar diferencialmente uma tabela de destino.

Essa forma de CDC orientado a lote tem uma limitação importante: embora possamos determinar facilmente quais linhas foram alteradas desde um ponto no tempo, não obtemos necessariamente todas as alterações aplicadas a essas linhas. Considere o exemplo de execução de CDC em lote em uma tabela de conta bancária a cada 24 horas. Esta tabela operacional mostra a atual

saldo da conta para cada conta. Quando o dinheiro entra e sai das contas, o aplicativo bancário executa uma transação para atualizar o saldo.

Quando executamos uma consulta para retornar todas as linhas da tabela de contas que foram alteradas nas últimas 24 horas, veremos os registros de cada conta que registrou uma transação. Suponha que um determinado cliente tenha sacado dinheiro cinco vezes usando um cartão de débito nas últimas 24 horas. Nossa consulta retornará apenas o último saldo da conta registrado no período de 24 horas; outros registros durante o período não aparecerão. Esse problema pode ser atenuado utilizando um esquema somente de inserção, em que cada transação da conta é registrada como um novo registro na tabela (consulte “[Somente inserção](#)” na [página 162](#)).

CDC Contínuo

CDC Contínuo captura todo o histórico da tabela e pode suportar ingestão de dados quase em tempo real, seja para replicação de banco de dados em tempo real ou para alimentar análises de streaming em tempo real. Em vez de executar consultas periódicas para obter um lote de alterações na tabela, o CDC contínuo trata cada gravação no banco de dados como um evento.

Podemos capturar um fluxo de eventos para CDC contínuo de algumas maneiras. Uma das abordagens mais comuns com um banco de dados transacional como o PostgreSQL é *CDC baseado em log*. O log binário do banco de dados registra todas as alterações no banco de dados sequencialmente (consulte “[Logs do banco de dados](#)” na [página 161](#)). Uma ferramenta CDC pode ler esse log e enviar os eventos para um destino, como a plataforma de streaming Apache Kafka Debezium.

Alguns bancos de dados oferecem suporte a um paradigma de CDC gerenciado e simplificado. Por exemplo, muitos bancos de dados hospedados na nuvem podem ser configurados para acionar diretamente uma função sem servidor ou gravar em um fluxo de eventos sempre que uma alteração ocorre no banco de dados. Isso libera completamente os engenheiros de se preocuparem com os detalhes de como os eventos são capturados no banco de dados e encaminhados.

CDC e replicação de banco de dados

O CDC pode ser usado para replicar entre bancos de dados: os eventos são armazenados em buffer em um fluxo *assincronamente* gravados em um segundo banco de dados. No entanto, muitos bancos de dados oferecem suporte nativo a uma versão de replicação fortemente acoplada (replicação síncrona) que mantém a réplica totalmente sincronizada com o banco de dados principal. A replicação síncrona normalmente requer que o banco de dados primário e a réplica sejam do mesmo tipo (por exemplo, PostgreSQL para PostgreSQL). A vantagem da replicação síncrona é que o banco de dados secundário pode descarregar o trabalho do banco de dados primário agindo como uma réplica de leitura; consultas de leitura podem ser redirecionadas para a réplica. A consulta retornará os mesmos resultados que seriam retornados do banco de dados primário.

As réplicas de leitura geralmente são usadas em padrões de ingestão de dados em lote para permitir que grandes varreduras sejam executadas sem sobrecarregar o banco de dados de produção principal. Além disso, um aplicativo pode ser configurado para fazer failover para a réplica se o banco de dados principal se tornar

indisponível. Nenhum dado será perdido no failover porque a réplica está totalmente sincronizada com o banco de dados principal.

A vantagem da replicação CDC assíncrona é um padrão de arquitetura fracamente acoplado. Embora a réplica possa ser ligeiramente atrasada em relação ao banco de dados primário, isso geralmente não é um problema para aplicativos analíticos, e os eventos agora podem ser direcionados para uma variedade de destinos; podemos executar a replicação do CDC ao mesmo tempo em que direcionamos os eventos para o armazenamento de objetos e um processador de análise de streaming.

Considerações do CDC

Como qualquer coisa em tecnologia, o CDC não é gratuito. O CDC consome vários recursos do banco de dados, como memória, largura de banda do disco, armazenamento, tempo de CPU e largura de banda da rede. Os engenheiros devem trabalhar com equipes de produção e executar testes antes de ativar o CDC em sistemas de produção para evitar problemas operacionais. Considerações semelhantes se aplicam à replicação síncrona.

Para CDC em lote, esteja ciente de que executar qualquer consulta em lote grande em um sistema de produção transacional pode causar carga excessiva. Execute essas consultas apenas fora do horário comercial ou use uma réplica de leitura para evitar sobrecarregar o banco de dados principal.

APIs

A maior parte da engenharia de software é apenas encanamento.

— Karl Hughes¹

Como mencionamos em [capítulo 5](#), as APIs são uma fonte de dados que continua a crescer em importância e popularidade. Uma organização típica pode ter centenas de fontes de dados externas, como plataformas SaaS ou empresas parceiras. A dura realidade é que não existe um padrão adequado para troca de dados por APIs. Os engenheiros de dados podem gastar uma quantidade significativa de tempo lendo a documentação, comunicando-se com proprietários de dados externos e escrevendo e mantendo o código de conexão da API.

Três tendências estão mudando lentamente esta situação. Primeiro, muitos fornecedores fornecem bibliotecas de cliente de API para várias linguagens de programação que removem grande parte da complexidade do acesso à API.

Em segundo lugar, várias plataformas de conectores de dados estão disponíveis agora como SaaS, código aberto ou código aberto gerenciado. Essas plataformas fornecem conectividade de dados pronta para uso para muitas fontes de dados; eles oferecem estruturas para escrever conectores personalizados para fontes de dados não suportadas. Ver ["Conectores de dados gerenciados"](#) na página 266.

¹ Karl Hughes, "The Bulk of Software Engineering Is Just Plumbing," Karl Hughes website, 8 de julho de 2018, <https://oreil.ly/uIuqJ>.

A terceira tendência é o surgimento do compartilhamento de dados (discutido em [capítulo 5](#)) — ou seja, a capacidade de trocar dados por meio de uma plataforma padrão, como BigQuery, Snowflake, Redshift ou S3. Depois que os dados chegam a uma dessas plataformas, é fácil armazená-los, processá-los ou movê-los para outro lugar. O compartilhamento de dados teve um grande e rápido impacto no espaço da engenharia de dados.

Não reinvente a roda quando o compartilhamento de dados não for uma opção e o acesso direto à API for necessário. Embora um serviço gerenciado possa parecer uma opção cara, considere o valor de seu tempo e o custo de oportunidade de criar conectores de API quando você poderia gastar seu tempo em um trabalho de maior valor.

Além disso, muitos serviços gerenciados agora oferecem suporte à criação de conectores de API personalizados. Isso pode fornecer especificações técnicas de API em um formato padrão ou escrever código de conector que é executado em uma estrutura de função sem servidor (por exemplo, AWS Lambda) enquanto permite que o serviço gerenciado lide com os detalhes de agendamento e sincronização. Mais uma vez, esses serviços podem economizar muito tempo para os engenheiros, tanto no desenvolvimento quanto na manutenção contínua.

Reserve seu trabalho de conexão personalizada para APIs que não são bem suportadas pelas estruturas existentes; você descobrirá que ainda há muitos deles para trabalhar. A manipulação de conexões de API personalizadas tem dois aspectos principais: desenvolvimento de software e operações. Seguir as melhores práticas de desenvolvimento de software; você deve usar controle de versão, entrega contínua e teste automatizado. Além de seguir as práticas recomendadas de DevOps, considere uma estrutura de orquestração, que pode simplificar drasticamente a carga operacional de ingestão de dados.

Filas de mensagens e plataformas de streaming de eventos

Filas de mensagens e plataformas de streaming de eventos são formas comuns de ingerir dados em tempo real de aplicativos móveis e da Web, sensores de IoT e dispositivos inteligentes. À medida que os dados em tempo real se tornam mais onipresentes, você frequentemente se encontrará introduzindo ou adaptando maneiras de lidar com dados em tempo real em seus fluxos de trabalho de ingestão. Como tal, é essencial saber como ingerir dados em tempo real. A ingestão popular de dados em tempo real inclui filas de mensagens ou plataformas de streaming de eventos, que abordamos em [Capítulo 5](#). Embora ambos sejam sistemas de origem, eles também atuam como formas de ingerir dados. Em ambos os casos, você consome eventos do editor que assina.

Lembre-se das diferenças entre mensagens e fluxos. *Amensagem* é tratado no nível do evento individual e deve ser transitório. Depois que uma mensagem é consumida, ela é reconhecida e removida da fila. Por outro lado, *um fluxo* ingere eventos em um log ordenado. O log persiste pelo tempo que você desejar, permitindo que os eventos sejam consultados em vários intervalos, agregados e combinados com outros fluxos para criar novas transformações publicadas para consumidores downstream. Em [Figura 7-14](#), temos dois produtores (produtores 1 e 2) enviando eventos para dois consumidores

(consumidores 1 e 2). Esses eventos são combinados em um novo conjunto de dados e enviados a um produtor para consumo downstream.

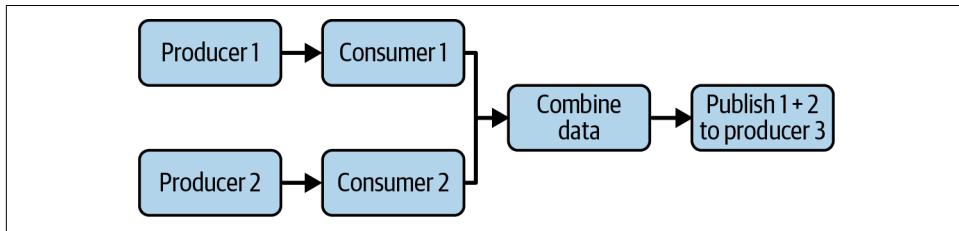


Figura 7-14. Dois conjuntos de dados são produzidos e consumidos (produtores 1 e 2) e então combinados, com os dados combinados publicados para um novo produtor (produtor 3)

O último ponto é uma diferença essencial entre ingestão em lote e streaming. Enquanto o lote geralmente envolve fluxos de trabalho estáticos (ingerir dados, armazená-los, transformá-los e servi-los), mensagens e fluxos são fluidos. A ingestão pode ser não linear, com os dados sendo publicados, consumidos, republicados e retomados. Ao projetar seus fluxos de trabalho de ingestão em tempo real, lembre-se de como os dados fluirão.

Outra consideração é a taxa de transferência de seus pipelines de dados em tempo real. Mensagens e eventos devem fluir com a menor latência possível, o que significa que você deve fornecer largura de banda e taxa de transferência de partição (ou estilhaço) adequadas. Forneça recursos suficientes de memória, disco e CPU para processamento de eventos e, se você estiver gerenciando seus pipelines em tempo real, incorpore dimensionamento automático para lidar com picos e economizar dinheiro à medida que a carga diminui. Por esses motivos, gerenciar sua plataforma de streaming pode acarretar uma sobrecarga significativa. Considere serviços gerenciados para seus pipelines de ingestão em tempo real e concentre sua atenção em maneiras de obter valor de seus dados em tempo real.

Conectores de dados gerenciados

Atualmente, se você está pensando em escrever um conector de ingestão de dados para um banco de dados ou API, pergunte-se: isso já foi criado? Além disso, existe algum serviço que gerencie os detalhes essenciais dessa conexão para mim? [“APIs” na página 254](#) menciona a popularidade das plataformas e estruturas do conector de dados gerenciados. Essas ferramentas visam fornecer um conjunto padrão de conectores disponíveis fora da caixa para poupar engenheiros de dados que constroem encanamentos complicados para se conectar a uma fonte específica. Em vez de criar e gerenciar um conector de dados, você terceiriza esse serviço para terceiros.

Geralmente, as opções no espaço permitem que os usuários definam um destino e uma fonte, ingerem de várias maneiras (por exemplo, CDC, replicação, truncar e recarregar), definir permissões e credenciais, configurar uma frequência de atualização e iniciar a sincronização de dados. O fornecedor ou nuvem nos bastidores gerencia e monitora totalmente as sincronizações de dados. Se a sincronização de dados falhar, você receberá um alerta com informações registradas sobre a causa do erro.

Sugerimos usar plataformas de conectores gerenciados em vez de criar e gerenciar seus conectores. Fornecedores e projetos de OSS normalmente têm centenas de opções de conectores pré-construídos e podem facilmente criar conectores personalizados. Atualmente, a criação e o gerenciamento de conectores de dados são um trabalho pesado indiferenciado e devem ser terceirizados sempre que possível.

Movendo dados com armazenamento de objetos

O armazenamento de objetos é um sistema multilocatário em nuvens públicas e oferece suporte ao armazenamento de grandes quantidades de dados. Isso torna o armazenamento de objetos ideal para mover dados para dentro e fora de data lakes, entre equipes e transferir dados entre organizações. Você pode até mesmo fornecer acesso de curto prazo a um objeto com um URL assinado, dando permissão temporária ao usuário.

Em nossa opinião, o armazenamento de objetos é a maneira mais ideal e segura de lidar com a troca de arquivos. O armazenamento em nuvem pública implementa os padrões de segurança mais recentes, possui um histórico robusto de escalabilidade e confiabilidade, aceita arquivos de tipos e tamanhos arbitrários e fornece movimentação de dados de alto desempenho. Discutimos o armazenamento de objetos muito mais extensivamente em [Capítulo 6](#).

EDI

Outra realidade prática para engenheiros de dados é *intercâmbio eletrônico de dados*(EDI). O termo é vago o suficiente para se referir a qualquer método de movimentação de dados. Geralmente se refere a meios um tanto arcaicos de troca de arquivos, como por e-mail ou pen drive. Os engenheiros de dados descobrirão que algumas fontes de dados não oferecem suporte a meios mais modernos de transporte de dados, geralmente devido a sistemas de TI arcaicos ou limitações de processos humanos.

Os engenheiros podem pelo menos aprimorar o EDI por meio da automação. Por exemplo, eles podem configurar um servidor de e-mail baseado em nuvem que salva arquivos no armazenamento de objetos da empresa assim que são recebidos. Isso pode acionar processos de orquestração para ingerir e processar dados. Isso é muito mais robusto do que um funcionário baixar o arquivo anexado e carregá-lo manualmente em um sistema interno, o que ainda vemos com frequência.

Bancos de dados e exportação de arquivos

Os engenheiros devem estar cientes de como os sistemas de banco de dados de origem lidam com a exportação de arquivos. A exportação envolve grandes varreduras de dados que carregam significativamente o banco de dados para muitos sistemas transacionais. Os engenheiros do sistema de origem devem avaliar quando essas verificações podem ser executadas sem afetar o desempenho do aplicativo e podem optar por uma estratégia para atenuar a carga. As consultas de exportação podem ser divididas em exportações menores consultando intervalos de chave ou uma partição por vez. Como alternativa, uma réplica de leitura pode reduzir a carga. As réplicas de leitura são especialmente apropriadas se as exportações ocorrerem muitas vezes ao dia e coincidirem com uma alta carga do sistema de origem.

Os principais data warehouses em nuvem são altamente otimizados para exportação direta de arquivos. Por exemplo, Snowflake, BigQuery, Redshift e outros oferecem suporte à exportação direta para armazenamento de objetos em vários formatos.

Problemas práticos com formatos de arquivo comuns

Os engenheiros também devem estar cientes dos formatos de arquivo a serem exportados. O CSV ainda é onipresente e altamente propenso a erros no momento em que este livro foi escrito. Ou seja, o delimitador padrão do CSV também é um dos caracteres mais familiares do idioma inglês - a vírgula! Mas fica pior.

O CSV não é de forma alguma um formato uniforme. Os engenheiros devem estipular o delimitador, os caracteres de aspas e o escape para lidar adequadamente com a exportação de dados de string. O CSV também não codifica informações de esquema nativamente ou oferece suporte direto a estruturas aninhadas. A codificação do arquivo CSV e as informações do esquema devem ser configuradas no sistema de destino para garantir a ingestão apropriada. A detecção automática é um recurso de conveniência fornecido em muitos ambientes de nuvem, mas é inadequado para ingestão de produção. Como prática recomendada, os engenheiros devem registrar a codificação CSV e os detalhes do esquema nos metadados do arquivo.

Formatos de exportação mais robustos e expressivos incluem [Parquet](#), [Avro](#), [Seta](#), e [ORC](#) ou [JSON](#). Esses formatos codificam informações de esquema nativamente e manipulam dados de cadeia de caracteres arbitrários sem nenhuma intervenção específica. Muitos deles também lidam com estruturas de dados aninhadas nativamente para que os campos JSON sejam armazenados usando estruturas aninhadas internas em vez de strings simples. Para bancos de dados colunares, os formatos colunares (Parquet, Arrow, ORC) permitem uma exportação de dados mais eficiente porque as colunas podem ser transcodificadas diretamente entre os formatos. Esses formatos também são geralmente mais otimizados para mecanismos de consulta. O formato de arquivo Arrow foi projetado para mapear dados diretamente na memória do mecanismo de processamento, proporcionando alto desempenho em ambientes de data lake.

A desvantagem desses formatos mais recentes é que muitos deles não são suportados nativamente pelos sistemas de origem. Os engenheiros de dados geralmente são forçados a trabalhar com dados CSV e, em seguida, criar tratamento robusto de exceções e detecção de erros para garantir a qualidade dos dados na ingestão. Ver [Apêndice A](#) para uma discussão mais extensa sobre formatos de arquivo.

Concha

O [conchá](#) é uma interface pela qual você pode executar comandos para ingerir dados. O shell pode ser usado para fluxos de trabalho de script para praticamente qualquer ferramenta de software, e o script de shell ainda é usado extensivamente em processos de ingestão. Um shell script pode ler dados de um banco de dados, reserializá-los em um formato de arquivo diferente, carregá-los no armazenamento de objetos e acionar um processo de ingestão em um banco de dados de destino. Embora o armazenamento de dados em uma única instância ou servidor não seja altamente escalável, muitas de nossas fontes de dados não são particularmente grandes e essas abordagens funcionam muito bem.

Além disso, os fornecedores de nuvem geralmente fornecem ferramentas robustas baseadas em CLI. É possível executar processos de ingestão complexos simplesmente emitindo comandos para o AWS CLI. À medida que os processos de ingestão se tornam mais complicados e o SLA se torna mais rigoroso, os engenheiros devem considerar a mudança para um sistema de orquestração adequado.

SSH

SSH é uma estratégia de ingestão, mas um protocolo usado com outras estratégias de ingestão. Usamos o SSH de algumas maneiras. Primeiro, o SSH pode ser usado para transferência de arquivos com SCP, conforme mencionado anteriormente. Em segundo lugar, os túneis SSH são usados para permitir conexões seguras e isoladas com bancos de dados.

Os bancos de dados de aplicativos nunca devem ser expostos diretamente na Internet. Em vez disso, os engenheiros podem configurar um bastion host, ou seja, uma instância de host intermediário que pode se conectar ao banco de dados em questão. Esta máquina host está exposta na Internet, embora bloqueada para acesso mínimo apenas de endereços IP especificados a portas especificadas. Para se conectar ao banco de dados, uma máquina remota primeiro abre uma conexão de túnel SSH para o bastion host e, em seguida, conecta-se da máquina host ao banco de dados.

SFTP e SCP

Acessar e enviar dados de FTP seguro (SFTP) e cópia segura (SCP) são técnicas com as quais você deve estar familiarizado, mesmo que os engenheiros de dados normalmente não as usem regularmente (TI ou segurança/secOps cuidarão disso).

Os engenheiros se assustam com a menção de SFTP (ocasionalmente, até ouvimos casos de FTP sendo usado na produção). Independentemente disso, o SFTP ainda é uma realidade prática para muitas empresas. Eles trabalham com empresas parceiras que consomem ou fornecem dados usando SFTP e não estão dispostos a confiar em outros padrões. Para evitar vazamentos de dados, a análise de segurança é fundamental nessas situações.

SCP é um protocolo de troca de arquivos executado em uma conexão SSH. O SCP pode ser uma opção segura de transferência de arquivos se for configurado corretamente. Novamente, adicionar controle de acesso à rede adicional (defesa em profundidade) para aumentar a segurança do SCP é altamente recomendado.

Webhooks

Webhooks, como discutimos em [capítulo 5](#), são muitas vezes referidos como *APIs reversas*. Para uma API de dados REST típica, o provedor de dados fornece especificações de API aos engenheiros que eles usam para escrever seu código de ingestão de dados. O código faz requisições e recebe dados em respostas.

Com um webhook ([Figura 7-15](#)), o provedor de dados define uma especificação de solicitação de API, mas o provedor de dados faz *chamadas de API* ao invés de recebê-los; é responsabilidade do consumidor de dados fornecer um endpoint de API para o provedor chamar. O

o consumidor é responsável por ingerir cada solicitação e lidar com a agregação, armazenamento e processamento de dados.

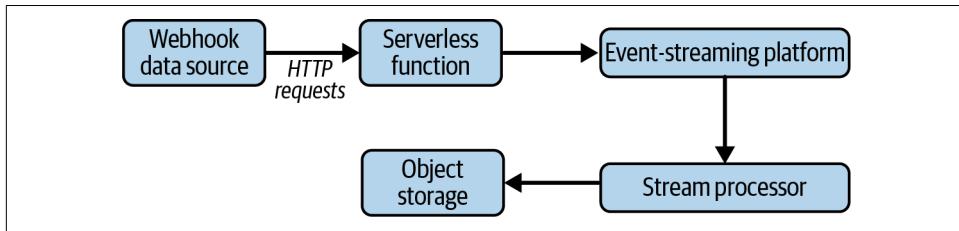


Figura 7-15. Uma arquitetura básica de ingestão de webhook criada a partir de serviços em nuvem

As arquiteturas de ingestão de dados baseadas em webhook podem ser frágeis, difíceis de manter e inefficientes. Usando ferramentas apropriadas disponíveis no mercado, os engenheiros de dados podem criar arquiteturas de webhook mais robustas com menores custos de manutenção e infraestrutura. Por exemplo, um padrão de webhook na AWS pode usar uma estrutura de função sem servidor (Lambda) para receber eventos de entrada, uma plataforma de streaming de eventos gerenciada para armazenar e armazenar mensagens em buffer (Kinesis), uma estrutura de processamento de fluxo para lidar com análises em tempo real (Flink) e um armazenamento de objeto para armazenamento de longo prazo (S3).

Você notará que essa arquitetura faz muito mais do que simplesmente ingerir os dados. Isso ressalta o envolvimento da ingestão com os outros estágios do ciclo de vida da engenharia de dados; muitas vezes é impossível definir sua arquitetura de ingestão sem tomar decisões sobre armazenamento e processamento.

Interface web

As interfaces da Web para acesso a dados continuam sendo uma realidade prática para engenheiros de dados. Frequentemente nos deparamos com situações em que nem todos os dados e funcionalidades em uma plataforma SaaS são expostos por meio de interfaces automatizadas, como APIs e descarte de arquivos. Em vez disso, alguém deve acessar manualmente uma interface da Web, gerar um relatório e baixar um arquivo para uma máquina local. Isso tem desvantagens óbvias, como as pessoas que se esquecem de executar o relatório ou que seus laptops morram. Sempre que possível, escolha ferramentas e fluxos de trabalho que permitam acesso automatizado aos dados.

Raspagem da web

Raspagem da web extrai automaticamente dados de páginas da web, geralmente combinando os vários elementos HTML da página da web. Você pode coletar sites de comércio eletrônico para extrair informações de preços de produtos ou coletar vários sites de notícias para seu agregador de notícias. A raspagem da Web é amplamente difundida e você pode encontrá-la como engenheiro de dados. É também uma área obscura onde as linhas éticas e legais são indistintas.

Aqui estão alguns conselhos de alto nível a serem observados antes de realizar qualquer projeto de web scraping. Primeiro, pergunte a si mesmo se você deve fazer scraping na web ou se os dados estão disponíveis de terceiros. Se sua decisão for raspar a web, seja um bom cidadão. Não crie inadvertidamente um ataque de negação de serviço (DoS) e não bloquee seu endereço IP. Compreenda a quantidade de tráfego que você gera e controle suas atividades de rastreamento na web de forma adequada. Só porque você pode ativar milhares de funções Lambda simultâneas para raspar não significa que você deva; a raspagem excessiva da web pode levar à desativação de sua conta da AWS.

Em segundo lugar, esteja ciente das implicações legais de suas atividades. Novamente, gerar ataques DoS pode acarretar consequências legais. Ações que violam os termos de serviço podem causar dores de cabeça para seu empregador ou para você pessoalmente.

Em terceiro lugar, as páginas da web mudam constantemente sua estrutura de elementos HTML, tornando complicado manter seu web scraper atualizado. Pergunte a si mesmo: a dor de cabeça de manter esses sistemas vale o esforço?

A raspagem da Web tem implicações interessantes para o estágio de processamento do ciclo de vida da engenharia de dados; os engenheiros devem pensar em vários fatores no início de um projeto de webscraping. O que você pretende fazer com os dados? Você está apenas puxando os campos obrigatórios do HTML raspado usando o código Python e, em seguida, gravando esses valores em um banco de dados? Você pretende manter o código HTML completo dos sites raspados e processar esses dados usando uma estrutura como o Spark? Essas decisões podem levar a arquiteturas muito diferentes a jusante da ingestão.

Dispositivos de transferência para migração de dados

Para grandes quantidades de dados (100 TB ou mais), a transferência de dados diretamente pela Internet pode ser um processo lento e caro. Nessa escala, a maneira mais rápida e eficiente de mover dados não é por fio, mas por caminhão. Os fornecedores de nuvem oferecem a capacidade de enviar seus dados por meio de uma “caixa de discos rígidos” física. Basta solicitar um dispositivo de armazenamento, chamado de *aparelho de transferência*, carregue seus dados de seus servidores e, em seguida, envie-os de volta ao fornecedor da nuvem, que fará o upload de seus dados.

A sugestão é considerar o uso de um dispositivo de transferência se o tamanho dos seus dados estiver em torno de 100 TB. No extremo, a AWS ainda oferece [Moto de neve](#), um dispositivo de transferência enviado a você em um semirreboque! O Snowmobile destina-se a levantar e deslocar todo um data center, no qual os tamanhos de dados estão na casa dos petabytes ou maiores.

Dispositivos de transferência são úteis para criar configurações de nuvem híbrida ou multicloud. Por exemplo, o dispositivo de transferência de dados da Amazon (AWS Snowball) suporta importação e exportação. Para migrar para uma segunda nuvem, os usuários podem exportar seus dados para um dispositivo Snowball e depois importá-los para um segundo dispositivo de transferência para mover os dados para o GCP ou Azure. Isso pode parecer estranho, mas mesmo quando é viável enviar dados

a internet entre nuvens, as taxas de saída de dados tornam essa proposta cara. Dispositivos de transferência física são uma alternativa mais barata quando os volumes de dados são significativos.

Lembre-se de que os dispositivos de transferência e os serviços de migração de dados são eventos únicos de ingestão de dados e não são sugeridos para cargas de trabalho contínuas. Suponha que você tenha cargas de trabalho que requerem movimentação constante de dados em um cenário híbrido ou multicloud. Nesse caso, seus tamanhos de dados provavelmente estão agrupando ou transmitindo tamanhos de dados muito menores continuamente.

Compartilhamento de dados

Compartilhamento de dados está crescendo como uma opção popular para consumir dados (consulte os Capítulos 5 e 6). Os provedores de dados oferecerão conjuntos de dados para assinantes de terceiros, gratuitamente ou a um custo. Esses conjuntos de dados geralmente são compartilhados de maneira somente leitura, o que significa que você pode integrar esses conjuntos de dados com seus próprios dados (e outros conjuntos de dados de terceiros), mas você não possui o conjunto de dados compartilhado. No sentido estrito, isso não é ingestão, onde você obtém a posse física do conjunto de dados. Se o provedor de dados decidir remover seu acesso a um conjunto de dados, você não terá mais acesso a ele.

Muitas plataformas de nuvem oferecem compartilhamento de dados, permitindo que você compartilhe seus dados e consuma dados de vários provedores. Algumas dessas plataformas também fornecem mercados de dados onde empresas e organizações podem oferecer seus dados para venda.

Com quem você trabalhará

A ingestão de dados fica em vários limites organizacionais. Ao desenvolver e gerenciar pipelines de ingestão de dados, os engenheiros de dados trabalharão com pessoas e sistemas upstream (produtores de dados) e downstream (consumidores de dados).

Partes interessadas a montante

Muitas vezes existe uma desconexão significativa entre os responsáveis por gerar dados — normalmente, engenheiros de software — e os engenheiros de dados que prepararão esses dados para análise e ciência de dados. Engenheiros de software e engenheiros de dados geralmente trabalham em silos organizacionais separados; se eles pensam em engenheiros de dados, eles normalmente os veem simplesmente como consumidores downstream da exaustão de dados de seu aplicativo, não como partes interessadas.

Vemos este estado atual de coisas como um problema e uma oportunidade significativa. Os engenheiros de dados podem melhorar a qualidade de seus dados convidando engenheiros de software para serem partes interessadas nos resultados da engenharia de dados. A grande maioria dos engenheiros de software está bem ciente do valor da análise e da ciência de dados, mas não necessariamente tem incentivos alinhados para contribuir diretamente com os esforços de engenharia de dados.

Simplesmente melhorar a comunicação é um primeiro passo significativo. Freqüentemente, os engenheiros de software já identificaram dados potencialmente valiosos para consumo downstream. A abertura de um canal de comunicação incentiva os engenheiros de software a preparar os dados para os consumidores e comunicar sobre as alterações de dados para evitar regressões no pipeline.

Além da comunicação, os engenheiros de dados podem destacar as contribuições dos engenheiros de software aos membros da equipe, executivos e, principalmente, aos gerentes de produto. Envolver os gerentes de produto no resultado e tratar os dados downstream processados como parte de um produto os encoraja a alocar o escasso desenvolvimento de software para a colaboração com engenheiros de dados. Idealmente, os engenheiros de software podem trabalhar parcialmente como extensões da equipe de engenharia de dados; isso permite que eles colaborem em vários projetos, como a criação de uma arquitetura orientada a eventos para permitir análises em tempo real.

Partes interessadas a jusante

Quem é o cliente final para ingestão de dados? Os engenheiros de dados se concentram em profissionais de dados e líderes de tecnologia, como cientistas de dados, analistas e diretores técnicos. Eles também fariam bem em lembrar seu círculo mais amplo de partes interessadas nos negócios, como diretores de marketing, vice-presidentes da cadeia de suprimentos e CEOs.

Muitas vezes, vemos engenheiros de dados perseguindo projetos sofisticados (por exemplo, barramentos de streaming em tempo real ou sistemas de dados complexos), enquanto os gerentes de marketing digital ao lado baixam relatórios do Google Ads manualmente. Veja a engenharia de dados como um negócio e reconheça quem são seus clientes. Frequentemente, a automação básica dos processos de ingestão tem um valor significativo, especialmente para departamentos como marketing, que controlam orçamentos massivos e estão no centro da receita da empresa. O trabalho básico de ingestão pode parecer tedioso, mas agregar valor a essas partes centrais da empresa abrirá mais orçamento e mais oportunidades empolgantes de engenharia de dados de longo prazo.

Os engenheiros de dados também podem convidar mais executivos a participar desse processo colaborativo. Por uma boa razão, a cultura orientada por dados está bastante na moda nos círculos de liderança empresarial. Ainda assim, cabe aos engenheiros de dados e outros profissionais de dados fornecer aos executivos orientação sobre a melhor estrutura para um negócio orientado a dados. Isso significa comunicar o valor de diminuir as barreiras entre produtores de dados e engenheiros de dados, ao mesmo tempo em que apoia os executivos na quebra de silos e na criação de incentivos para levar a uma cultura orientada por dados mais unificada.

Outra vez, *comunicação* é a palavra de ordem. A comunicação honesta desde o início e frequentemente com as partes interessadas ajudará muito a garantir que a ingestão de dados agregue valor.

Subcorrentes

Praticamente todas as subcorrentes tocam a fase de ingestão, mas vamos enfatizar as mais salientes aqui.

Segurança

A movimentação de dados introduz vulnerabilidades de segurança porque você precisa transferir dados entre locais. A última coisa que você deseja é capturar ou comprometer os dados durante a movimentação.

Considere onde os dados residem e para onde estão indo. Os dados que precisam ser movidos dentro de seu VPC devem usar endpoints seguros e nunca deixar os limites do VPC. Use uma VPN ou uma conexão privada dedicada se precisar enviar dados entre a nuvem e uma rede local. Isso pode custar dinheiro, mas a segurança é um bom investimento. Se seus dados trafegarem pela Internet pública, certifique-se de que a transmissão seja criptografada. É sempre uma boa prática criptografar dados na rede.

Gestão de dados

Naturalmente, o gerenciamento de dados começa na ingestão de dados. Este é o ponto de partida para catalogação de linhagem e dados; a partir deste ponto, os engenheiros de dados precisam pensar sobre mudanças de esquema, ética, privacidade e conformidade.

Alterações de esquema

Alterações de esquema (como adicionar, alterar ou remover colunas em uma tabela de banco de dados) permanecem, de nossa perspectiva, uma questão não resolvida no gerenciamento de dados. A abordagem tradicional é um processo cuidadoso de revisão de comando e controle. Trabalhando com clientes em grandes empresas, recebemos orçamentos de seis meses para a adição de um único campo. Este é um impedimento inaceitável para a agilidade.

No extremo oposto do espectro, qualquer alteração de esquema na origem aciona tabelas de destino para serem recriadas com o novo esquema. Isso resolve problemas de esquema no estágio de ingestão, mas ainda pode interromper pipelines downstream e sistemas de armazenamento de destino.

Uma solução possível, sobre a qual nós, os autores, meditamos por um tempo, é uma abordagem pioneira do controle de versão do Git. Quando Linus Torvalds estava desenvolvendo o Git, muitas de suas escolhas foram inspiradas pelas limitações do Concurrent Versions System (CVS). O CVS é completamente centralizado; ele suporta apenas uma versão oficial atual do código, armazenada em um servidor de projeto central. Para tornar o Git um sistema verdadeiramente distribuído, Torvalds usou a noção de uma árvore; cada desenvolvedor pode manter sua ramificação processada do código e, em seguida, mesclar para ou de outras ramificações.

Há alguns anos, tal abordagem aos dados era impensável. Os sistemas MPP locais são normalmente operados perto da capacidade máxima de armazenamento. No entanto, o armazenamento é barato em ambientes de big data e data warehouse em nuvem. Pode-se facilmente manter várias versões de uma tabela com diferentes esquemas e até mesmo diferentes transformações upstream. As equipes podem dar suporte a várias versões de “desenvolvimento” de uma tabela usando ferramentas de orquestração como o Airflow; mudanças de esquema, upstream

transformação e alterações de código podem aparecer nas tabelas de desenvolvimento antes das alterações oficiais no *principal* mesa.

Ética de dados, privacidade e conformidade

Os clientes geralmente pedem nosso conselho sobre como criptografar dados confidenciais em bancos de dados, o que geralmente nos leva a fazer uma pergunta fundamental: você precisa dos dados confidenciais que está tentando criptografar? Acontece que essa questão geralmente é negligenciada ao criar requisitos e resolver problemas.

Os engenheiros de dados devem sempre se treinar para fazer essa pergunta ao configurar pipelines de ingestão. Eles inevitavelmente encontrarão dados confidenciais; a tendência natural é ingeri-lo e encaminhá-lo para a próxima etapa do pipeline. Mas se esses dados não são necessários, por que coletá-los? Por que não simplesmente descartar campos confidenciais antes que os dados sejam armazenados? Os dados não podem vazar se nunca forem coletados.

Onde é realmente necessário acompanhar identidades confidenciais, é prática comum aplicar tokenização para anonimizar identidades em treinamento e análise de modelo. Mas os engenheiros devem observar onde essa tokenização é usada. Se possível, faça hash dos dados no momento da ingestão.

Os engenheiros de dados não podem evitar trabalhar com dados altamente confidenciais em alguns casos. Alguns sistemas analíticos devem apresentar informações confidenciais identificáveis. Os engenheiros devem agir de acordo com os mais altos padrões éticos sempre que lidarem com dados confidenciais. Além disso, eles podem implementar uma variedade de práticas para reduzir o manuseio direto de dados confidenciais. Mire o máximo possível para *produção sem toque* onde dados sensíveis estão envolvidos. Isso significa que os engenheiros desenvolvem e testam o código em dados simulados ou limpos em ambientes de desenvolvimento e preparação, mas implantações de código automatizadas na produção.

A produção sem contato é um ideal pelo qual os engenheiros devem se esforçar, mas inevitavelmente surgem situações que não podem ser totalmente resolvidas em ambientes de desenvolvimento e preparação. Alguns bugs podem não ser reproduzíveis sem olhar para os dados ao vivo que estão acionando uma regressão. Para esses casos, implemente um processo de vidro quebrado: exija que pelo menos duas pessoas aprovem o acesso a dados confidenciais no ambiente de produção. Esse acesso deve ser restrito a um problema específico e ter uma data de expiração.

Nosso último conselho sobre dados confidenciais: tenha cuidado com soluções tecnológicas ingênuas para problemas humanos. Tanto a criptografia quanto a tokenização são frequentemente tratadas como balas mágicas de privacidade. A maioria dos sistemas de armazenamento baseados em nuvem e quase todos os bancos de dados criptografam dados em repouso e em movimento por padrão. Geralmente, não vemos problemas de criptografia, mas problemas de acesso a dados. A solução é aplicar uma camada extra de criptografia a um único campo ou controlar o acesso a esse campo? Afinal, ainda é preciso gerenciar rigidamente o acesso à chave de criptografia. Existem casos de uso legítimos para criptografia de campo único, mas cuidado com a criptografia ritualística.

Na frente de tokenização, use o bom senso e avalie os cenários de acesso a dados. Se alguém tivesse o e-mail de um de seus clientes, poderia facilmente fazer o hash do e-mail e encontrar o cliente em seus dados? Fazer hash de dados sem pensar e sem salgar e outras estratégias pode não proteger a privacidade tão bem quanto você pensa.

DataOps

Pipelines de dados confiáveis são a base do ciclo de vida da engenharia de dados. Quando eles falham, todas as dependências downstream chegam a um ponto insuportável. Armazéns de dados e data lakes não são reabastecidos com dados atualizados, e os cientistas e analistas de dados não podem fazer seu trabalho com eficiência; o negócio é forçado a voar às cegas.

Garantir que seus pipelines de dados sejam monitorados adequadamente é uma etapa crucial em direção à confiabilidade e à resposta eficaz a incidentes. Se há um estágio no ciclo de vida da engenharia de dados em que o monitoramento é crítico, é o estágio de ingestão.

Monitoramento fraco ou inexistente significa que os dutos podem ou não estar funcionando. Voltando à nossa discussão anterior sobre o tempo, certifique-se de acompanhar os vários aspectos do tempo - criação de eventos, ingestão, processo e tempos de processamento. Seus pipelines de dados devem processar dados de forma previsível em lotes ou fluxos. Vimos inúmeros exemplos de relatórios e modelos de ML gerados a partir de dados obsoletos. Em um caso extremo, uma falha no pipeline de ingestão não foi detectada por seis meses. (Alguém pode questionar a utilidade concreta dos dados neste caso, mas isso é outro assunto).

O que você deve monitorar? Tempo de atividade, latência e volumes de dados processados são bons lugares para começar. Se um trabalho de ingestão falhar, como você responderá? Em geral, inclua o monitoramento em seus pipelines desde o início, em vez de aguardar a implantação.

O monitoramento é fundamental, assim como o conhecimento do comportamento dos sistemas upstream dos quais você depende e como eles geram dados. Você deve estar ciente do número de eventos gerados por intervalo de tempo com o qual está preocupado (eventos/minuto, eventos/segundo e assim por diante) e o tamanho médio de cada evento. Seu pipeline de dados deve lidar com a frequência e o tamanho dos eventos que você está ingerindo.

Isso também se aplica a serviços de terceiros. No caso desses serviços, o que você ganhou em termos de eficiência operacional enxuta (redução do número de funcionários) é substituído por sistemas dos quais você depende, estando fora de seu controle. Se você estiver usando um serviço de terceiros (nuvem, serviço de integração de dados etc.), como será alertado se houver uma interrupção? Qual é o seu plano de resposta se um serviço do qual você depende repentinamente ficar offline?

Infelizmente, não existe um plano de resposta universal para falhas de terceiros. Se você puder fazer failover para outros servidores, preferencialmente em outra zona ou região, definitivamente configure isso.

Se seus processos de ingestão de dados são criados internamente, você tem o teste adequado e a automação de implantação para garantir que o código funcione na produção? E se o código apresentar erros ou falhar, você pode revertê-lo para uma versão funcional?

Testes de qualidade de dados

Muitas vezes nos referimos aos dados como um assassino silencioso. Se dados válidos e de qualidade são a base do sucesso nos negócios de hoje, usar dados ruins para tomar decisões é muito pior do que não ter dados. Dados incorretos causaram danos incalculáveis às empresas; esses desastres de dados às vezes são chamados *datastrophes*.²

Os dados são entrópicos; muitas vezes muda de maneiras inesperadas sem aviso prévio. Uma das diferenças inerentes entre DevOps e DataOps é que esperamos regressões de software apenas quando implementamos alterações, enquanto os dados geralmente apresentam regressões de forma independente devido a eventos fora de nosso controle.

Os engenheiros de DevOps geralmente são capazes de detectar problemas usando condições binárias. A taxa de falha de solicitação ultrapassou um determinado limite? E a latência de resposta? No espaço de dados, as regressões geralmente se manifestam como distorções estatísticas sutis. Uma mudança nas estatísticas de termos de pesquisa é resultado do comportamento do cliente? De um aumento no tráfego de bots que escapou da rede? De uma ferramenta de teste de site implantada em alguma outra parte da empresa?

Assim como as falhas do sistema no DevOps, algumas regressões de dados são imediatamente visíveis. Por exemplo, no início dos anos 2000, o Google fornecia termos de pesquisa para sites quando os usuários chegavam da pesquisa. Em 2011, o Google começou a reter essas informações em alguns casos para proteger melhor a privacidade do usuário. Os analistas rapidamente viram “não fornecido” borbulhando no topo de seus relatórios.³

As regressões de dados verdadeiramente perigosas são silenciosas e podem vir de dentro ou de fora de uma empresa. Os desenvolvedores de aplicativos podem alterar o significado dos campos do banco de dados sem se comunicar adequadamente com as equipes de dados. Alterações nos dados de fontes de terceiros podem passar despercebidas. Na melhor das hipóteses, os relatórios quebram de maneiras óbvias. Freqüentemente, as métricas de negócios são distorcidas sem o conhecimento dos tomadores de decisão.

Sempre que possível, trabalhe com engenheiros de software para corrigir problemas de qualidade de dados na fonte. É surpreendente quantos problemas de qualidade de dados podem ser tratados respeitando as melhores práticas básicas de engenharia de software, como logs para capturar o histórico de alterações de dados, verificações (nulos etc.) .

As ferramentas tradicionais de teste de dados geralmente são construídas em lógica binária simples. Os nulos estão aparecendo em um campo não anulável? Itens novos e inesperados estão aparecendo em uma coluna categórica? O teste de dados estatísticos é um novo domínio, mas que provavelmente crescerá dramaticamente nos próximos cinco anos.

2 Andy Petrella, “Datastrophes,” *Média*, 1º de março de 2021, <https://oreil.ly/h6FRW>.

3 Danny Sullivan, “Dark Google: um ano desde que os termos de pesquisa foram ‘não fornecidos’”, *MarTech*, 19 de outubro de 2012, <https://oreil.ly/Fp8ta>.

Orquestração

A ingestão geralmente fica no início de um gráfico de dados grande e complexo; como a ingestão é o primeiro estágio do ciclo de vida da engenharia de dados, os dados ingeridos fluirão para muito mais etapas de processamento de dados e os dados de muitas fontes se misturarão de maneiras complexas. Como enfatizamos ao longo deste livro, a orquestração é um processo crucial para coordenar essas etapas.

As organizações em um estágio inicial de maturidade de dados podem optar por implantar processos de ingestão como tarefas cron agendadas simples. No entanto, é crucial reconhecer que essa abordagem é frágil e pode diminuir a velocidade de implantação e desenvolvimento da engenharia de dados.

À medida que a complexidade do pipeline de dados aumenta, uma verdadeira orquestração é necessária. Por verdadeira orquestração, queremos dizer um sistema capaz de agendar gráficos de tarefas completas em vez de tarefas individuais. Uma orquestração pode iniciar cada tarefa de ingestão no horário agendado apropriado. O processamento downstream e as etapas de transformação começam quando as tarefas de ingestão são concluídas. Mais adiante, as etapas de processamento levam a etapas de processamento adicionais.

Engenharia de software

O estágio de ingestão do ciclo de vida da engenharia de dados é de engenharia intensiva. Esse estágio fica no limite do domínio da engenharia de dados e geralmente faz interface com sistemas externos, onde os engenheiros de software e de dados precisam criar uma variedade de encanamentos personalizados.

Nos bastidores, a ingestão é incrivelmente complicada, geralmente com equipes operando estruturas de código aberto como Kafka ou Pulsar, ou algumas das maiores empresas de tecnologia executando suas próprias soluções de ingestão bifurcadas ou desenvolvidas internamente. Conforme discutido neste capítulo, os conectores de dados gerenciados simplificaram o processo de ingestão, como Fivetran, Matillion e Airbyte. Os engenheiros de dados devem aproveitar as melhores ferramentas disponíveis - principalmente ferramentas e serviços gerenciados que fazem muito trabalho pesado para você - e desenvolver alta competência em desenvolvimento de software em áreas importantes. Vale a pena usar processos adequados de controle de versão e revisão de código e implementar testes apropriados, mesmo para qualquer código relacionado à ingestão.

Ao escrever software, seu código precisa ser desacoplado. Evite escrever sistemas monolíticos com dependências rígidas nos sistemas de origem ou destino.

Conclusão

Em seu trabalho como engenheiro de dados, a ingestão provavelmente consumirá uma parte significativa de sua energia e esforço. No fundo, a ingestão é um encanamento, conectando tubos a outros tubos, garantindo que os dados fluam de forma consistente e segura para seu destino. Às vezes,

as minúcias da ingestão podem parecer tediosas, mas os aplicativos de dados empolgantes (por exemplo, análise e ML) não podem acontecer sem isso.

Como enfatizamos, também estamos no meio de uma mudança radical, passando de lotes para pipelines de dados de streaming. Esta é uma oportunidade para os engenheiros de dados descobrirem aplicativos interessantes para transmissão de dados, comunicá-los aos negócios e implantar novas tecnologias empolgantes.

Recursos adicionais

- Airbytes[Página da web “Conexões e modos de sincronização”](#)
- Capítulo 6, “Batch é um caso especial de streaming”, em *Introdução ao Apache Flink* por Ellen Friedman e Kostas Tzoumas (O'Reilly)
- “O modelo de fluxo de dados: uma abordagem prática para balancear exatidão, latência e custo em processamento de dados em grande escala, ilimitado e fora de ordem” por Tyler Akidau et al.
- do Google Cloud[Página da Web “Streaming Pipelines”](#)
- da Microsoft[Documentação “Janela de Instantâneo \(Azure Stream Analytics\)”](#)

Consultas, modelagem e transformação

Até este ponto, os estágios do ciclo de vida da engenharia de dados foram principalmente sobre a passagem de dados de um lugar para outro ou seu armazenamento. Neste capítulo, você aprenderá como tornar os dados úteis. Compreendendo consultas, modelagem e transformações (consulte [Figura 8-1](#)), você terá as ferramentas para transformar ingredientes de dados brutos em algo consumível pelas partes interessadas downstream.

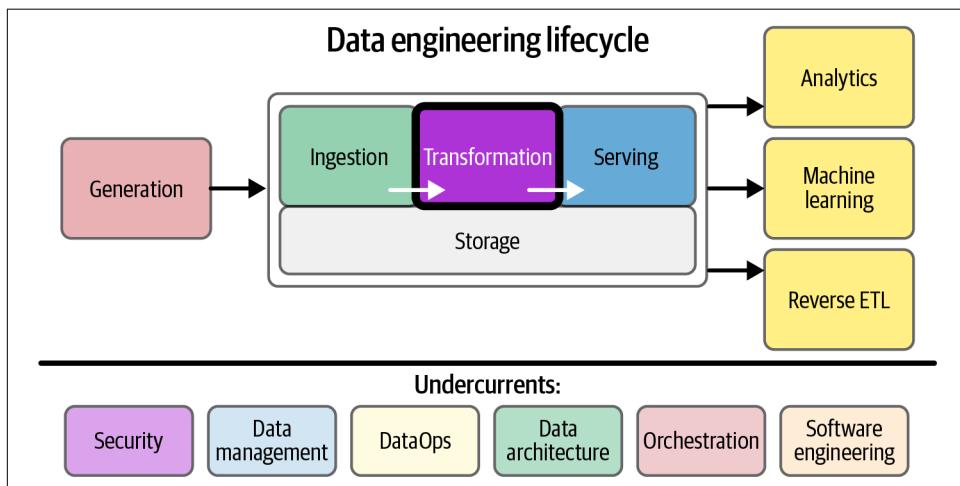


Figura 8-1. As transformações nos permitem criar valor a partir dos dados

Primeiro, discutiremos as consultas e os padrões significativos subjacentes a elas. Em segundo lugar, veremos os principais padrões de modelagem de dados que você pode usar para introduzir a lógica de negócios em seus dados. Em seguida, abordaremos as transformações, que pegam a lógica de seus modelos de dados e os resultados de consultas e os tornam úteis para

consumo a jusante. Por fim, abordaremos com quem você trabalhará e as tendências subjacentes relacionadas a este capítulo.

Uma variedade de técnicas pode ser usada para consultar, modelar e transformar dados em bancos de dados SQL e NoSQL. Esta seção se concentra nas consultas feitas a um sistema OLAP, como um data warehouse ou um data lake. Embora existam muitas linguagens para consulta, por uma questão de conveniência e familiaridade, ao longo da maior parte deste capítulo, vamos nos concentrar fortemente no SQL, a linguagem de consulta mais popular e universal. A maioria dos conceitos de bancos de dados OLAP e SQL serão traduzidos para outros tipos de bancos de dados e linguagens de consulta. Este capítulo pressupõe que você tenha um conhecimento da linguagem SQL e conceitos relacionados, como chaves primárias e estrangeiras. Se essas ideias não lhe são familiares, há inúmeros recursos disponíveis para ajudá-lo a começar.

Uma nota sobre os termos usados neste capítulo. Por conveniência, usaremos o termo *base de dados* como um atalho para um mecanismo de consulta e o armazenamento que ele está consultando; pode ser um data warehouse na nuvem ou dados de consulta do Apache Spark armazenados no S3. Assumimos que o banco de dados tem um mecanismo de armazenamento que organiza os dados sob o capô. Isso se estende a consultas baseadas em arquivo (carregando um arquivo CSV em um notebook Python) e consultas em formatos de arquivo como Parquet.

Além disso, observe que este capítulo se concentra principalmente na consulta, padrões de modelagem e transformações relacionadas a dados estruturados e semiestruturados, que os engenheiros de dados usam com frequência. Muitas das práticas discutidas também podem ser aplicadas ao trabalho com dados não estruturados, como imagens, vídeo e texto bruto.

Antes de entrarmos na modelagem e transformação de dados, vamos ver as consultas — o que são, como funcionam, considerações para melhorar o desempenho da consulta e consultas sobre dados de streaming.

Consultas

As consultas são uma parte fundamental da engenharia de dados, ciência de dados e análise. Antes de aprender sobre os padrões e tecnologias subjacentes para transformações, você precisa entender o que são consultas, como elas funcionam em vários dados e técnicas para melhorar o desempenho da consulta.

Esta seção se preocupa principalmente com consultas em dados tabulares e semiestruturados. Como engenheiro de dados, você consultará e transformará esses tipos de dados com mais frequência. Antes de entrarmos em tópicos mais complicados sobre consultas, modelagem de dados e transformações, vamos começar respondendo a uma pergunta bastante simples: o que é uma consulta?

O que é uma consulta?

Frequentemente encontramos pessoas que sabem como escrever SQL, mas não estão familiarizadas com o funcionamento de uma consulta nos bastidores. Parte desse material introdutório sobre consultas será familiar para engenheiros de dados experientes; sinta-se à vontade para avançar se isso se aplicar a você.

A consulta permite recuperar e agir sobre os dados. Relembre nossa conversa em [capítulo 5](#) sobre o CRUD. Quando uma consulta recupera dados, ela está emitindo uma solicitação para ler um padrão de registros. Isto é o *R*(leia) no CRUD. Você pode emitir uma consulta que obtém todos os registros de uma tabela foo, como SELEÇÃO * DE foo. Ou você pode aplicar um predicado (condição lógica) para filtrar seus dados recuperando apenas registros onde o ou ia é 1, usando a consulta SQL SELECT * FROM foo WHERE id=1.

Muitos bancos de dados permitem criar, atualizar e excluir dados. Estes são os *RUMINAÇÕES* em CRUD; sua consulta criará, modificará ou destruirá os registros existentes. Vamos revisar alguns outros acrônimos comuns que você encontrará ao trabalhar com linguagens de consulta.

Linguagem de definição de dados

Em um alto nível, primeiro você precisa criar os objetos de banco de dados antes de adicionar dados. Você vai usar *linguagem de definição de dados*(DDL) comandos para executar operações em objetos de banco de dados, como o próprio banco de dados, esquemas, tabelas ou usuários; DDL define o estado dos objetos em seu banco de dados.

Os engenheiros de dados usam expressões SQL DDL comuns: CRIAR, DEIXAR, e ATUALIZAR. Por exemplo, você pode criar um banco de dados usando a expressão DDL Barra CRIAR BANCO DE DADOS. Depois disso, você também pode criar novas tabelas (CREATE tabela bar_table) ou deletar uma mesa (DROP tabela bar_table).

Linguagem de manipulação de dados

Depois de usar DDL para definir objetos de banco de dados, você precisa adicionar e alterar dados dentro desses objetos, que é o objetivo principal da *linguagem de manipulação de dados*(DML). Alguns comandos DML comuns que você usará como engenheiro de dados são os seguintes:

SELEÇÃO
INSERIR
ATUALIZAR
EXCLUIR
CÓPIA DE
MERGE

Por exemplo, você pode INSERIR novos registros em uma tabela de banco de dados, ATUALIZAR existentes, e SELECIONAR registros específicos.

Linguagem de controle de dados

Você provavelmente deseja limitar o acesso a objetos de banco de dados e controlar minuciosamente quem tem acesso ao que. *Linguagem de controle de dados*(DCL) permite controlar o acesso aos objetos do banco de dados ou aos dados usando comandos SQL como CONCEDER, NEGAR, e REVOGAR.

Vamos percorrer um breve exemplo usando comandos DCL. Uma nova cientista de dados chamada Sarah se junta à sua empresa e ela precisa de acesso somente leitura a um banco de dados chamado *data_science_db*. Você concede a Sarah acesso a esse banco de dados usando o seguinte comando DCL:

```
GRANT SELECT ON data_science_db PARA user_name Sarah;
```

É um mercado de trabalho quente, e Sarah trabalhou na empresa por apenas alguns meses antes de ser contratada por uma grande empresa de tecnologia. Adeus, Sara! Sendo um engenheiro de dados voltado para a segurança, você remove a capacidade de Sarah de ler do banco de dados:

```
REVOGAR SELECCIONAR ATIVADO data_science_db PARA user_name Sarah;
```

Solicitações e problemas de controle de acesso são comuns, e entender o DCL ajudará você a resolver problemas se você ou um membro da equipe não puder acessar os dados de que precisa, além de impedir o acesso a dados de que não precisa.

Linguagem de controle de transação

Como o próprio nome sugere, *linguagem de controle de transação*(TCL) suporta comandos que controlam os detalhes das transações. Com TCL, podemos definir pontos de verificação de confirmação, condições quando as ações serão revertidas e muito mais. Dois comandos TCL comuns incluem COMPROMETER-SE e REVERSÃO.

A vida de uma consulta

Como funciona uma consulta e o que acontece quando uma consulta é executada? Vamos abordar os fundamentos de alto nível da execução de consultas (Figura 8-2), usando um exemplo de uma consulta SQL típica em execução em um banco de dados.

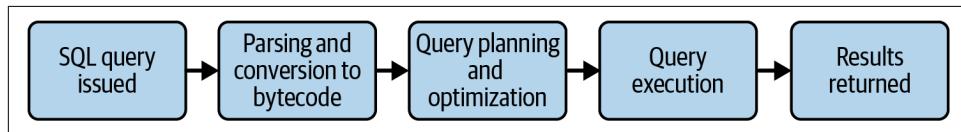


Figura 8-2. A vida de uma consulta SQL em um banco de dados

Embora a execução de uma consulta possa parecer simples - escrever código, executá-lo e obter resultados - muita coisa está acontecendo nos bastidores. Quando você executa uma consulta SQL, aqui está um resumo do que acontece:

1. O mecanismo de banco de dados compila o SQL, analisando o código para verificar a semântica adequada e garantindo que os objetos de banco de dados referenciados existam e que o usuário atual tenha o acesso apropriado a esses objetos.
2. O código SQL é convertido em bytecode. Este bytecode expressa as etapas que devem ser executadas no mecanismo de banco de dados em um formato eficiente e legível por máquina.
3. O otimizador de consulta do banco de dados analisa o bytecode para determinar como executar a consulta, reordenando e refatorando as etapas para usar os recursos disponíveis da maneira mais eficiente possível.
4. A consulta é executada e os resultados são produzidos.

O otimizador de consultas

As consultas podem ter tempos de execução extremamente diferentes, dependendo de como são executadas. O trabalho de um otimizador de consulta é otimizar o desempenho da consulta e minimizar os custos dividindo a consulta em etapas apropriadas em uma ordem eficiente. O otimizador avaliará junções, índices, tamanho da varredura de dados e outros fatores. O otimizador de consulta tenta executar a consulta da maneira menos dispendiosa.

Os otimizadores de consulta são fundamentais para o desempenho da sua consulta. Cada banco de dados é diferente e executa consultas de maneiras que são óbvia e sutilmente diferentes umas das outras. Você não trabalhará diretamente com um otimizador de consulta, mas entender algumas de suas funcionalidades o ajudará a escrever consultas com melhor desempenho. Você precisará saber como analisar o desempenho de uma consulta, usando coisas como um plano de explicação ou análise de consulta, descrito na seção a seguir.

Melhorando o desempenho da consulta

Na engenharia de dados, você inevitavelmente encontrará consultas com baixo desempenho. Saber identificar e corrigir essas consultas é inestimável. Não lute contra seu banco de dados. Aprenda a trabalhar com seus pontos fortes e aumentar suas fraquezas. Esta seção mostra várias maneiras de melhorar o desempenho da consulta.

Otimize sua estratégia e esquema de junção

Um único conjunto de dados (como uma tabela ou arquivo) raramente é útil sozinho; criamos valor combinando-o com outros conjuntos de dados. *Junta-se* é um dos meios mais comuns de combinar conjuntos de dados e criar novos. Presumimos que você esteja familiarizado com os tipos significativos de junções (por exemplo, interno, externo, esquerdo, cruzado) e os tipos de relacionamentos de junção (por exemplo, um para um, um para muitos, muitos para um e muitos para muitos).

As junções são críticas na engenharia de dados e são bem suportadas e têm bom desempenho em muitos bancos de dados. Mesmo os bancos de dados colunares, que no passado tinham uma reputação de baixo desempenho de junção, agora geralmente oferecem excelente desempenho.

Uma técnica comum para melhorar o desempenho da consulta é *juntar-se previamente* dados. Se você achar que as consultas analíticas estão juntando os mesmos dados repetidamente, geralmente faz sentido juntar os dados com antecedência e fazer com que as consultas sejam lidas a partir da versão pré-junta dos dados para que você não repita o trabalho computacional intensivo. Isso pode significar alterar o esquema e relaxar as condições de normalização para ampliar as tabelas e utilizar estruturas de dados mais recentes (como arrays ou structs) para substituir relacionamentos de entidades frequentemente associados. Outra estratégia é manter um esquema mais normalizado, mas unir tabelas para os casos de uso de análise e ciência de dados mais comuns. Podemos simplesmente criar tabelas pré-unidas e treinar usuários para utilizá-las ou ingressar em visualizações materializadas (consulte “[Visualizações materializadas, federação e virtualização de consulta](#)” na página 333).

Em seguida, considere os detalhes e a complexidade de suas condições de junção. A lógica de junção complexa pode consumir recursos computacionais significativos. Podemos melhorar o desempenho de junções complexas de algumas maneiras.

Muitos bancos de dados orientados a linhas permitem que você indexe um resultado calculado a partir de uma linha. Por exemplo, o PostgreSQL permite que você crie um índice em um campo de string convertido para letras minúsculas; quando o otimizador encontra uma consulta onde `omais baixo()` função aparece dentro de um predicado, ela pode aplicar o índice. Você também pode criar uma nova coluna derivada para ingressar, embora precise treinar os usuários para ingressar nessa coluna.

Explosão de linha

Um problema obscuro, mas frustrante, é [explosão de linha](#). Isso ocorre quando temos um grande número de correspondências muitos-para-muitos, devido à repetição nas chaves de junção ou como consequência da lógica de junção. Suponha que a chave de junção na tabela A tenha o valoresse repetido cinco vezes e a chave de junção na tabela B contém esse mesmo valor repetido 10 vezes. Isso leva a uma junção cruzada dessas linhas: cadaesselinha da tabela A emparelhada com cadaesselinha da tabela B. Isso cria $5 \times 10 = 50$ linhas na saída. Agora suponha que muitas outras repetições estejam na chave de junção. A explosão de linhas geralmente gera linhas suficientes para consumir uma quantidade massiva de recursos do banco de dados ou até mesmo causar uma falha na consulta.

Também é essencial saber como seu otimizador de consulta lida com junções. Alguns bancos de dados podem reordenar junções e predicados, enquanto outros não. Uma explosão de linha em um estágio inicial de consulta pode fazer com que a consulta falhe, mesmo que um predicado posterior remova corretamente muitas das repetições na saída. A reordenação de predicados pode reduzir significativamente os recursos computacionais exigidos por uma consulta.

Por fim, use expressões de tabela comuns (CTEs) em vez de subconsultas aninhadas ou tabelas temporárias. Os CTEs permitem que os usuários componham consultas complexas de maneira legível, ajudando você a entender o fluxo de sua consulta. A importância da legibilidade para consultas complexas não pode ser subestimada.

Em muitos casos, os CTEs também oferecem melhor desempenho do que um script que cria tabelas intermediárias; se você tiver que criar tabelas intermediárias, considere a criação de tabelas temporárias. Se você quiser saber mais sobre CTEs, uma rápida pesquisa na Web fornecerá muitas informações úteis.

Use o plano de explicação e entenda o desempenho da sua consulta

Como você aprendeu na seção anterior, o otimizador de consulta do banco de dados influencia a execução de uma consulta. O plano de explicação do otimizador de consulta mostrará como o otimizador de consulta determinou sua consulta ideal de menor custo, os objetos de banco de dados usados (tabelas, índices, cache etc.) e várias estatísticas de desempenho e consumo de recursos em cada estágio de consulta. Alguns bancos de dados fornecem uma representação visual dos estágios da consulta. Em contraste, outros disponibilizam o plano de explicação via SQL com o EXPLICAR comando, que exibe a sequência de etapas que o banco de dados executará para executar a consulta.

além de usar EXPLICAR para entender como sua consulta será executada, você deve monitorar o desempenho de sua consulta, visualizando métricas sobre o consumo de recursos do banco de dados. A seguir estão algumas áreas para monitorar:

- Uso de recursos importantes, como disco, memória e rede.
- Tempo de carregamento de dados versus tempo de processamento.
- Tempo de execução da consulta, número de registros, tamanho dos dados digitalizados e quantidade de dados embaralhados.
- Consultas concorrentes que podem causar contenção de recursos em seu banco de dados.
- Número de conexões simultâneas usadas versus conexões disponíveis. Conexões simultâneas com excesso de assinaturas podem ter efeitos negativos em seus usuários, que podem não conseguir se conectar ao banco de dados.

Evite verificações completas de tabelas

Todas as consultas verificam os dados, mas nem todas as verificações são criadas iguais. Como regra geral, você deve consultar apenas os dados de que precisa. Quando você corre SELECT * sem predicados, você está examinando toda a tabela e recuperando cada linha e coluna. Isso é muito ineficiente em termos de desempenho e caro, especialmente se você estiver usando um banco de dados pré-pago que cobra por bytes verificados ou recursos de computação utilizados durante a execução de uma consulta.

Sempre que possível, use *podapara* reduzir a quantidade de dados verificados em uma consulta. Bancos de dados colunares e orientados a linhas requerem diferentes estratégias de remoção. Em um banco de dados orientado a colunas, você deve selecionar apenas as colunas necessárias. A maioria dos bancos de dados OLAP orientados a colunas também fornece ferramentas adicionais para otimizar suas tabelas para melhor desempenho de consulta. Por exemplo, se você tiver uma tabela muito grande (vários terabytes de tamanho ou mais), o Snowflake e o BigQuery oferecem a opção de definir uma chave de cluster em uma tabela, que ordena os dados da tabela de forma a permitir que as consultas acessem com mais eficiência porções de conjuntos de dados muito grandes. O BigQuery também permite particionar uma tabela em segmentos menores, permitindo consultar apenas partições específicas em vez de toda a tabela. (Esteja ciente de que clustering inadequado e estratégias de distribuição de chaves podem degradar o desempenho.)

Em bancos de dados orientados a linhas, a remoção geralmente gira em torno de índices de tabela, que você aprendeu em [Capítulo 6](#). A estratégia geral é criar índices de tabela que melhorem o desempenho para suas consultas mais sensíveis ao desempenho, sem sobrecarregar a tabela com tantos índices que prejudiquem o desempenho.

Saiba como seu banco de dados lida com os commits

um banco de dados *comrometer-se* é uma alteração em um banco de dados, como criar, atualizar ou excluir um registro, tabela ou outros objetos de banco de dados. Muitos bancos de dados suportam *transações*— ou seja, uma noção de cometer várias operações simultaneamente de forma a manter um estado consistente. Observe que o termo *transação* está um pouco sobrecarregado; ver [capítulo 5](#). O objetivo de uma transação é manter um estado consistente de um banco de dados enquanto ele está ativo e em caso de falha. As transações também manipulam o isolamento quando vários eventos simultâneos podem estar lendo, gravando e excluindo dos mesmos objetos de banco de dados. Sem transações, os usuários obteriam informações potencialmente conflitantes ao consultar um banco de dados.

Você deve estar intimamente familiarizado com a forma como seu banco de dados lida com confirmações e transações e determinar a consistência esperada dos resultados da consulta. Seu banco de dados lida com gravações e atualizações de maneira compatível com ACID? Sem conformidade com ACID, sua consulta pode retornar resultados inesperados. Isso pode resultar de uma leitura suja, que acontece quando uma linha é lida e uma transação não confirmada alterou a linha. As leituras sujas são um comportamento esperado do seu banco de dados? Se sim, como você lida com isso? Além disso, esteja ciente de que durante as transações de atualização e exclusão, alguns bancos de dados criam novos arquivos para representar o novo estado do banco de dados e retêm os arquivos antigos para referências de ponto de verificação de falha. Nesses bancos de dados, a execução de um grande número de pequenas confirmações pode causar confusão e consumir um espaço de armazenamento significativo que pode precisar ser aspirado periodicamente.

Vamos considerar brevemente três bancos de dados para entender o impacto dos commits (observe que esses exemplos são atuais no momento em que este livro foi escrito). Primeiro, suponha que estamos olhando para um PostgreSQL RDBMS e aplicando transações ACID. Cada transação consiste em

um pacote de operações que falharão ou serão bem-sucedidas como um grupo. Também podemos executar consultas analíticas em várias linhas; essas consultas apresentarão uma imagem consistente do banco de dados em um determinado momento.

A desvantagem da abordagem do PostgreSQL é que ela requer *bloqueio de linha* (bloqueio de leituras e gravações em determinadas linhas), o que pode degradar o desempenho de várias maneiras. O PostgreSQL não é otimizado para grandes varreduras ou grandes quantidades de dados apropriados para aplicativos analíticos de larga escala.

Em seguida, considere o Google BigQuery. Ele utiliza um modelo point-in-time full table commit. Quando uma consulta de leitura é emitida, o BigQuery faz a leitura do instantâneo confirmado mais recente da tabela. Quer a consulta seja executada por um segundo ou duas horas, ela lerá apenas a partir desse instantâneo e não verá nenhuma alteração subsequente. O BigQuery não bloqueia a tabela enquanto eu a leio. Em vez disso, as operações de gravação subsequentes criariam novos commits e novos instantâneos enquanto a consulta continua a ser executada no instantâneo em que foi iniciada.

Para evitar o estado inconsistente, o BigQuery permite apenas uma operação de gravação por vez. Nesse sentido, o BigQuery não fornece nenhuma simultaneidade de gravação. (No sentido de que pode gravar grandes quantidades de dados em paralelo *dentro de uma única consulta de gravação*, é altamente concorrente.) Se mais de um cliente tentar gravar simultaneamente, as consultas de gravação serão enfileiradas na ordem de chegada. O modelo de confirmação do BigQuery é semelhante aos modelos de confirmação usados por Snowflake, Spark e outros.

Por último, vamos considerar o MongoDB. Nos referimos ao MongoDB como um *banco de dados de consistência variável*. Os engenheiros têm várias opções de consistência configuráveis, tanto para o banco de dados quanto no nível de consultas individuais. O MongoDB é celebrado por sua extraordinária escalabilidade e simultaneidade de gravação, mas é um tanto notório por problemas que surgem quando os engenheiros abusam dele.¹

Por exemplo, em determinados modos, o MongoDB suporta desempenho de gravação ultra-alto. No entanto, isso tem um custo: o banco de dados descartará as gravações sem cerimônia e silenciosamente se ficar sobrecarregado com o tráfego. Isso é perfeitamente adequado para aplicativos que podem perder alguns dados, por exemplo, aplicativos de IoT em que simplesmente queremos muitas medições, mas não nos preocupamos em capturar todas as medições. Não é uma ótima opção para aplicativos que precisam capturar dados e estatísticas exatas.

Nada disso é para dizer que esses são bancos de dados ruins. Eles são todos bancos de dados fantásticos quando são escolhidos para aplicativos apropriados e configurados corretamente. O mesmo vale para praticamente qualquer tecnologia de banco de dados.

¹ Veja, por exemplo, Emin Gün Sirer, "NoSQL Meets Bitcoin and Brings Down Two Exchanges: The Story of Flexcoin e Poloniex", *Hacking*, distribuído, 6 de abril de 2014, <https://oreil.ly/RM3QX>.

As empresas não contratam engenheiros simplesmente para hackear códigos isoladamente. Para serem dignos de seu título, os engenheiros devem desenvolver uma compreensão profunda dos problemas que devem solucionar e das ferramentas tecnológicas. Isso se aplica a modelos de compromisso e consistência e a todos os outros aspectos do desempenho da tecnologia. Escolhas e configurações apropriadas de tecnologia podem, em última análise, diferenciar um sucesso extraordinário de um fracasso maciço. Referir-se [Capítulo 6](#) para uma discussão mais profunda sobre consistência.

Registros mortos a vácuo

Como acabamos de discutir, as transações incorrem na sobrecarga de criar novos registros durante certas operações, como atualizações, exclusões e operações de índice, enquanto retêm os registros antigos como ponteiros para o último estado do banco de dados. Como esses registros antigos se acumulam no sistema de arquivos do banco de dados, eles eventualmente não precisam mais ser referenciados. Você deve remover esses registros mortos em um processo chamado *aspirar*.

Você pode aspirar uma única tabela, várias tabelas ou todas as tabelas em um banco de dados. Não importa como você escolha aspirar, excluir registros de banco de dados inativos é importante por alguns motivos. Em primeiro lugar, ele libera espaço para novos registros, levando a uma tabela menos volumosa e a consultas mais rápidas. Em segundo lugar, registros novos e relevantes significam que os planos de consulta são mais precisos; registros desatualizados podem levar o otimizador de consulta a gerar planos abaixo do ideal e imprecisos. Por fim, a aspiração limpa os índices ruins, permitindo um melhor desempenho do índice.

As operações de vácuo são tratadas de forma diferente, dependendo do tipo de banco de dados. Por exemplo, em bancos de dados apoiados por armazenamento de objetos (BigQuery, Snowflake, Databricks), a única desvantagem da retenção de dados antigos é que ela usa espaço de armazenamento, potencialmente custando dinheiro, dependendo do modelo de preços de armazenamento para o banco de dados. No Snowflake, os usuários não podem aspirar diretamente. Em vez disso, eles controlam um intervalo de “viagem no tempo” que determina por quanto tempo os instantâneos da tabela são retidos antes de serem automaticamente aspirados. O BigQuery utiliza uma janela de histórico fixa de sete dias. Databricks geralmente retém dados indefinidamente até que sejam aspirados manualmente; a aspiração é importante para controlar os custos diretos de armazenamento do S3.

O Amazon Redshift lida com seus discos de cluster em várias configurações,² e a aspiração podem afetar o desempenho e o armazenamento disponível. VÁCUO é executado automaticamente nos bastidores, mas às vezes os usuários podem querer executá-lo manualmente para fins de ajuste.

A limpeza torna-se ainda mais crítica para bancos de dados relacionais, como PostgreSQL e MySQL. Um grande número de operações transacionais pode causar um rápido acúmulo de registros mortos, e os engenheiros que trabalham nesses sistemas precisam se familiarizar com os detalhes e o impacto da aspiração.

² algumas configurações de redshift em vez disso, confie no armazenamento de objetos.

Aproveite os resultados da consulta em cache

Digamos que você tenha uma consulta intensiva que costuma executar em um banco de dados que cobra pela quantidade de dados consultados. Cada vez que uma consulta é executada, isso custa dinheiro. Em vez de reexecutar a mesma consulta no banco de dados repetidamente e incorrer em cobranças massivas, não seria bom se os resultados da consulta fossem armazenados e disponíveis para recuperação instantânea? Felizmente, muitos bancos de dados OLAP em nuvem armazenam os resultados da consulta.

Quando uma consulta é executada inicialmente, ela recupera dados de várias fontes, filtra e junta-os e gera um resultado. Essa consulta inicial - uma consulta fria - é semelhante à noção de dados frios que exploramos em [Capítulo 6](#). Para fins de argumentação, digamos que essa consulta levou 40 segundos para ser executada. Supondo que seu banco de dados armazene em cache os resultados da consulta, executar novamente a mesma consulta pode retornar resultados em 1 segundo ou menos. Os resultados foram armazenados em cache e a consulta não precisou ser executada a frio. Sempre que possível, aproveite os resultados do cache de consulta para reduzir a pressão em seu banco de dados e fornecer uma melhor experiência do usuário para consultas executadas com frequência. Observe também que *visualizações materializadas* fornecem outra forma de cache de consulta (consulte "[Visualizações materializadas, federação e virtualização de consulta](#)" na [página 333](#)).

Consultas sobre dados de streaming

Os dados de streaming estão constantemente em movimento. Como você pode imaginar, consultar dados de streaming é diferente de dados em lote. Para aproveitar ao máximo um fluxo de dados, devemos adaptar padrões de consulta que refletem sua natureza em tempo real. Por exemplo, sistemas como Kafka e Pulsar facilitam a consulta de fontes de dados de streaming. Vejamos algumas maneiras comuns de fazer isso.

Padrões de consulta básicos em streams

Lembre-se do CDC contínuo, discutido em [Capítulo 7](#). O CDC, nessa forma, essencialmente configura um banco de dados analítico como um seguidor rápido de um banco de dados de produção. Um dos padrões de consulta de streaming mais antigos envolve simplesmente consultar o banco de dados analítico, recuperando resultados estatísticos e agregações com um pequeno atraso em relação ao banco de dados de produção.

A abordagem do seguidor rápido. Como isso é um padrão de consulta de streaming? Não poderíamos conseguir a mesma coisa simplesmente executando nossas consultas no banco de dados de produção? Em princípio, sim; na prática, não. Os bancos de dados de produção geralmente não são equipados para lidar com cargas de trabalho de produção e executar simultaneamente grandes varreduras analíticas em quantidades significativas de dados. A execução de tais consultas pode retardar o aplicativo de produção ou até mesmo fazer com que ele trave.³ O padrão básico de consulta do CDC nos permite fornecer análises em tempo real com um impacto mínimo no sistema de produção.

³ Os autores estão cientes de um incidente envolvendo um novo analista em uma grande rede de supermercados SELEÇÃO * em um banco de dados de produção e desativando um banco de dados de inventário crítico por três dias.

O padrão fast-follower pode utilizar um banco de dados transacional convencional como seguidor, mas há vantagens significativas em usar um sistema adequado orientado a OLAP ([Figura 8-3](#)). Tanto o Druid quanto o BigQuery combinam um buffer de streaming com armazenamento colunar de longo prazo em uma configuração um tanto semelhante à arquitetura Lambda (consulte [Capítulo 3](#)). Isso funciona muito bem para calcular estatísticas de rastreamento em vastos dados históricos com atualizações quase em tempo real.

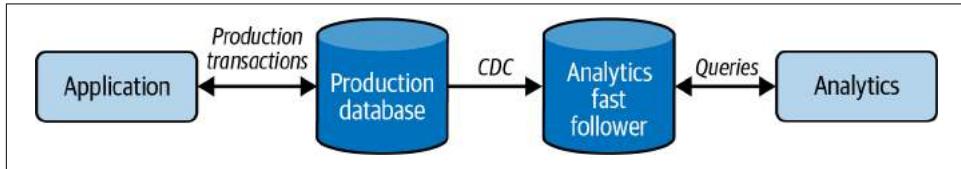


Figura 8-3. CDC com um banco de dados analítico de acompanhamento rápido

A abordagem CDC fast-follower tem limitações críticas. Ele não repensa fundamentalmente os padrões de consulta em lote. Você ainda está correndo `SELECIONE` consultas no estado atual da tabela e perdendo a oportunidade de acionar eventos dinamicamente fora das alterações no fluxo.

A arquitetura Kappa. Em seguida, lembre-se da arquitetura Kappa que discutimos em [Capítulo 3](#). A ideia principal dessa arquitetura é manipular todos os dados como eventos e armazenar esses eventos como um fluxo em vez de uma tabela ([Figura 8-4](#)). Quando os bancos de dados de aplicativos de produção são a origem, a arquitetura Kappa armazena eventos do CDC. Os fluxos de eventos também podem fluir diretamente de um back-end de aplicativo, de um enxame de dispositivos IoT ou de qualquer sistema que gere eventos e possa enviá-los por uma rede. Em vez de simplesmente tratar um sistema de armazenamento de streaming como um buffer, a arquitetura Kappa retém os eventos no armazenamento durante um período de retenção mais prolongado, e os dados podem ser consultados diretamente desse armazenamento. O período de retenção pode ser bastante longo (meses ou anos). Observe que isso é muito mais longo do que o período de retenção usado em sistemas puramente orientados para tempo real, geralmente uma semana no máximo.

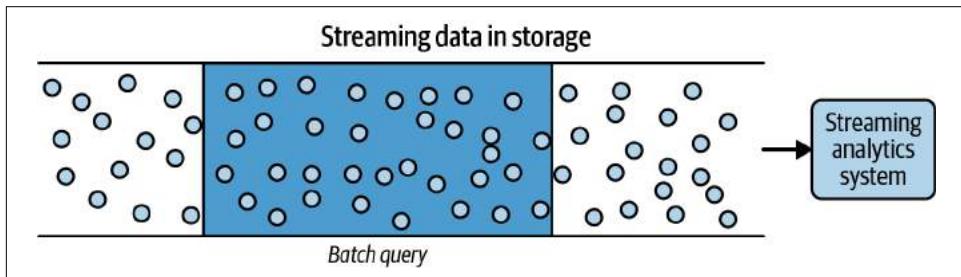


Figura 8-4. A arquitetura Kappa é construída em torno de sistemas de armazenamento e ingestão de streaming

A “grande ideia” na arquitetura Kappa é tratar o armazenamento de streaming como uma camada de transporte em tempo real e um banco de dados para recuperar e consultar dados históricos. Isto acontece

seja por meio dos recursos de consulta direta do sistema de armazenamento de streaming ou com a ajuda de ferramentas externas. Por exemplo, Kafka KSQL suporta agregação, cálculos estatísticos e até mesmo sessão. Se os requisitos de consulta forem mais complexos ou os dados precisarem ser combinados com outras fontes de dados, uma ferramenta externa, como Spark, lê um intervalo de tempo de dados do Kafka e calcula os resultados da consulta. O sistema de armazenamento de streaming também pode alimentar outros aplicativos ou um processador de stream como Flink ou Beam.

Janelas, gatilhos, estatísticas emitidas e dados atrasados

Uma limitação fundamental das consultas em lote tradicionais é que esse paradigma geralmente trata o mecanismo de consulta como um observador externo. Um ator externo aos dados faz com que a consulta seja executada — talvez um cron job de hora em hora ou um gerente de produto abrindo um painel.

Os sistemas de streaming mais amplamente usados, por outro lado, suportam a noção de cálculos acionados diretamente dos próprios dados. Eles podem emitir estatísticas médias e medianas sempre que um determinado número de registros é coletado no buffer ou gerar um resumo quando uma sessão do usuário é encerrada.

O Windows é um recurso essencial para consultas e processamento de streaming. As janelas são pequenos lotes processados com base em gatilhos dinâmicos. As janelas são geradas dinamicamente ao longo do tempo de algumas maneiras. Vejamos alguns tipos comuns de janelas: sessão, tempo fixo e deslizante. Também veremos as marcas d'água.

Janela da sessão. A *janela da sessão* agrupa eventos que ocorrem próximos e filtra períodos de inatividade quando nenhum evento ocorre. Podemos dizer que uma sessão de usuário é qualquer intervalo de tempo sem intervalo de inatividade de cinco minutos ou mais. Nossa sistema em lote coleta dados por uma chave de ID de usuário, ordena eventos, determina as lacunas e limites de sessão e calcula estatísticas para cada sessão. Os engenheiros de dados geralmente fazem sessões de dados retrospectivamente, aplicando condições de tempo à atividade do usuário em aplicativos da Web e de desktop.

Em uma sessão de streaming, esse processo pode acontecer dinamicamente. Observe que as janelas de sessão são por chave; no exemplo anterior, cada usuário obtém seu próprio conjunto de janelas. O sistema acumula dados por usuário. Se ocorrer um intervalo de cinco minutos sem atividade, o sistema fecha a janela, envia seus cálculos e libera os dados. Caso cheguem novos eventos para uso, o sistema inicia uma nova janela de sessão.

As janelas de sessão também podem prever dados atrasados. Permitindo que os dados cheguem com até cinco minutos de atraso para contabilizar as condições da rede e a latência do sistema, o sistema abrirá a janela se um evento de chegada tardia indicar atividade menos de cinco minutos após o último evento. Teremos mais a dizer sobre dados atrasados ao longo deste capítulo. **Figura 8-5** mostra três janelas de sessão, cada uma separada por cinco minutos de inatividade.

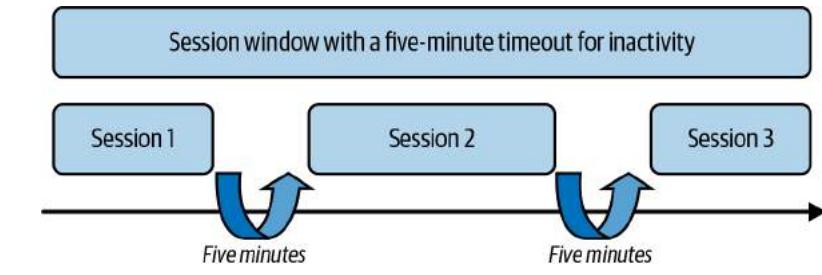


Figura 8-5. Janela de sessão com um tempo limite de cinco minutos para inatividade

Tornar a sessão dinâmica e quase em tempo real muda fundamentalmente sua utilidade. Com a sessão retrospectiva, podemos automatizar ações específicas um dia ou uma hora após o encerramento da sessão do usuário (por exemplo, um e-mail de acompanhamento com um cupom para um produto visualizado pelo usuário). Com a sessão dinâmica, o usuário pode receber um alerta em um aplicativo móvel que é imediatamente útil com base em sua atividade nos últimos 15 minutos.

Janelas de tempo fixo. *Ahora marcada* (também conhecido como *caindo*) apresenta períodos de tempo fixos que são executados em uma programação fixa e processa todos os dados desde que a janela anterior foi fechada. Por exemplo, podemos fechar uma janela a cada 20 segundos e processar todos os dados que chegam da janela anterior para fornecer uma estatística média e mediana ([Figura 8-6](#)). As estatísticas seriam emitidas assim que pudessem ser calculadas após o fechamento da janela.

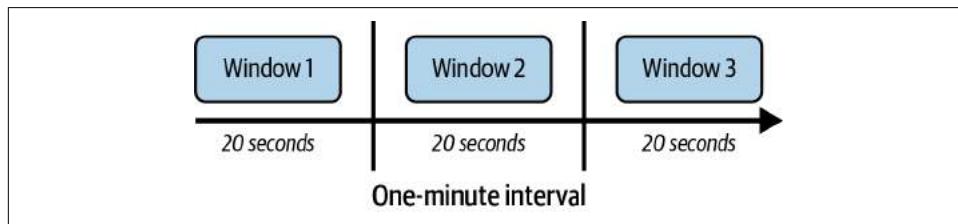


Figura 8-6. Janela em queda/fixa

Isso é semelhante ao processamento tradicional de ETL em lote, no qual podemos executar um trabalho de atualização de dados todos os dias ou a cada hora. O sistema de streaming nos permite gerar janelas com mais frequência e entregar resultados com menor latência. Como enfatizaremos repetidamente, o batch é um caso especial de streaming.

Janelas deslizantes. Os eventos em uma janela deslizante são agrupados em janelas de duração fixa, onde janelas separadas podem se sobrepor. Por exemplo, poderíamos gerar uma nova janela de 60 segundos a cada 30 segundos ([Figura 8-7](#)). Assim como fizemos antes, podemos emitir estatísticas médias e medianas.

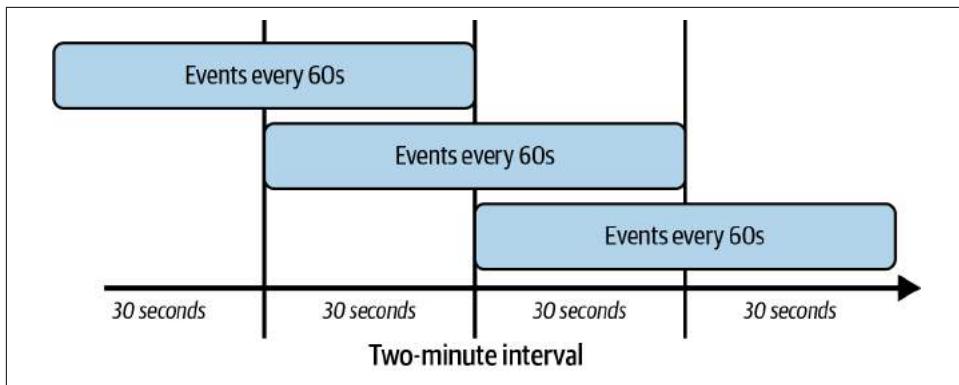


Figura 8-7. Janelas deslizantes

O deslizamento pode variar. Por exemplo, podemos pensar na janela como realmente deslizando continuamente, mas emitindo estatísticas apenas quando certas condições (gatilhos) são atendidas. Suponha que usamos uma janela deslizante contínua de 30 segundos, mas calculamos uma estatística apenas quando um usuário clicou em um banner específico. Isso levaria a uma taxa extremamente alta de saída quando muitos usuários clicarem no banner e nenhum cálculo durante uma pausa.

Marcas d'água. Cobrimos vários tipos de janelas e seus usos. Conforme discutido em [Capítulo 7](#), os dados às vezes são ingeridos fora da ordem em que foram originados. A *marca d'água* (Figura 8-8) é um limite usado por uma janela para determinar se os dados em uma janela estão dentro do intervalo de tempo estabelecido ou se são considerados atrasados. Se chegarem dados novos na janela, mas mais antigos do que o carimbo de data/hora da marca d'água, eles serão considerados dados atrasados.

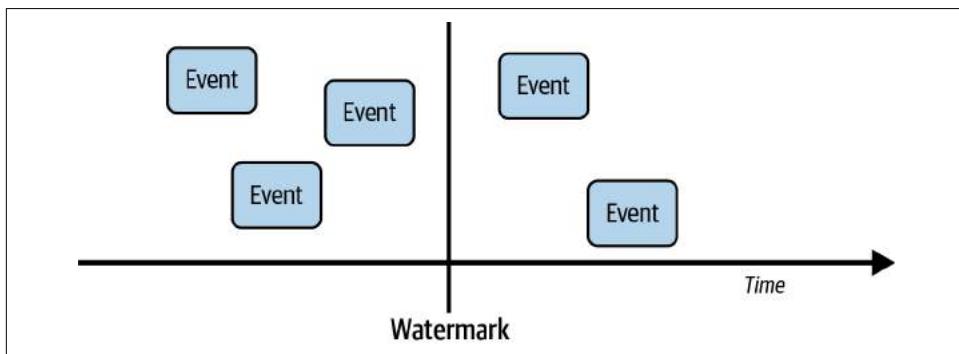


Figura 8-8. Marca d'água agindo como um limite para dados atrasados

Combinando streams com outros dados

Como mencionamos antes, muitas vezes derivamos valor dos dados combinando-os com outros dados. Os dados de streaming não são diferentes. Por exemplo, vários fluxos podem ser combinados ou um fluxo pode ser combinado com dados históricos em lote.

Junções de tabelas convencionais. Algumas tabelas podem ser alimentadas por streams (Figura 8-9). A abordagem mais básica para esse problema é simplesmente unir essas duas tabelas em um banco de dados. Um fluxo pode alimentar uma ou ambas as tabelas.

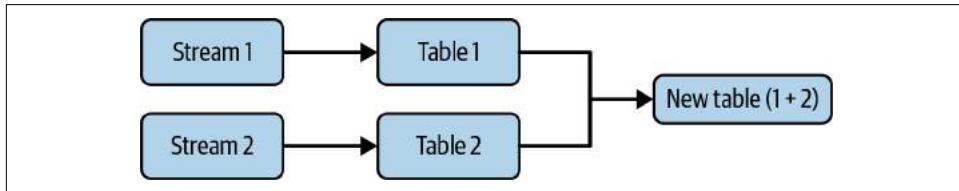


Figura 8-9. Unindo duas tabelas alimentadas por streams

Enriquecimento. *Enriquecimento* significa que juntamos um fluxo a outros dados (Figura 8-10). Normalmente, isso é feito para fornecer dados aprimorados em outro fluxo. Por exemplo, suponha que um varejista on-line receba um fluxo de eventos de um parceiro de negócios contendo produtos e IDs de usuário. O varejista deseja aprimorar esses eventos com detalhes do produto e informações demográficas dos usuários. O varejista alimenta esses eventos para uma função sem servidor que procura o produto e o usuário em um banco de dados na memória (digamos, um cache), adiciona as informações necessárias ao evento e envia os eventos aprimorados para outro fluxo.

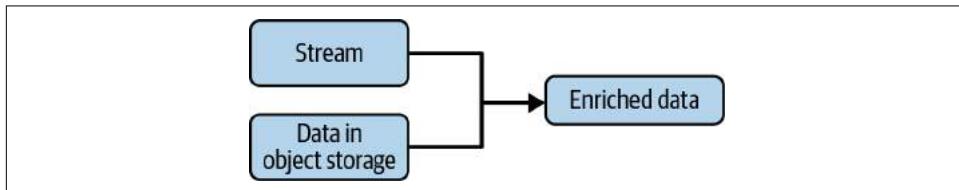


Figura 8-10. Neste exemplo, um fluxo é enriquecido com dados residentes no armazenamento de objeto, resultando em um novo conjunto de dados enriquecido

Na prática, a fonte de enriquecimento pode se originar em quase qualquer lugar — uma tabela em um data warehouse na nuvem ou RDBMS ou um arquivo no armazenamento de objetos. É simplesmente uma questão de ler a fonte e armazenar os dados de enriquecimento necessários em um local apropriado para recuperação pelo fluxo.

Associação de fluxo a fluxo. Cada vez mais, os sistemas de streaming suportam junção direta de stream a stream. Suponha que um varejista on-line deseja unir seus dados de eventos da web com dados de streaming de uma plataforma de anúncios. A empresa pode alimentar ambos os fluxos no Spark,

mas uma variedade de complicações surgem. Por exemplo, os fluxos podem ter latências significativamente diferentes para chegada no ponto em que a junção é tratada no sistema de fluxo. A plataforma de anúncios pode fornecer seus dados com um atraso de cinco minutos. Além disso, certos eventos podem ser significativamente atrasados, por exemplo, um evento de fechamento de sessão para um usuário ou um evento que ocorre no telefone offline e aparece no stream somente depois que o usuário está de volta ao alcance da rede móvel.

Como tal, as arquiteturas típicas de junção de streaming dependem de buffers de streaming. O intervalo de retenção do buffer é configurável; um intervalo de retenção mais longo requer mais armazenamento e outros recursos. Os eventos são associados aos dados no buffer e eventualmente são removidos após o intervalo de retenção ter passado (Figura 8-11).⁴

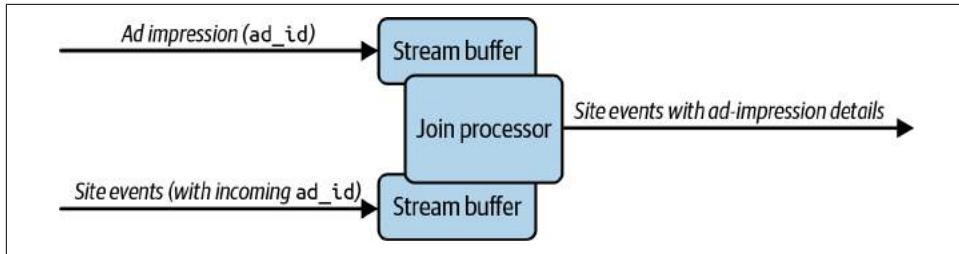


Figura 8-11. Uma arquitetura para unir fluxos armazena em buffer cada fluxo e junta eventos se eventos relacionados forem encontrados durante o intervalo de retenção do buffer

Agora que abordamos como as consultas funcionam para dados em lote e streaming, vamos discutir como tornar seus dados úteis ao modelá-los.

Modelagem de dados

A modelagem de dados é algo que vemos negligenciado com frequência perturbadora. Muitas vezes vemos as equipes de dados começarem a construir sistemas de dados sem um plano de jogo para organizar seus dados de uma forma que seja útil para os negócios. Isto é um erro. Arquiteturas de dados bem construídas devem refletir os objetivos e a lógica de negócios da organização que depende desses dados. A modelagem de dados envolve a escolha deliberada de uma estrutura coerente para os dados e é uma etapa crítica para tornar os dados úteis para os negócios.

A modelagem de dados tem sido uma prática há décadas de uma forma ou de outra. Por exemplo, vários tipos de técnicas de normalização (discutidas em "Normalização" na página 290) têm sido usados para modelar dados desde os primórdios dos RDBMSs; as técnicas de modelagem de armazenamento de dados existem desde pelo menos o início da década de 1990 e, sem dúvida, há mais tempo. Como costumam acontecer os pêndulos em tecnologia, a modelagem de dados tornou-se um tanto fora de moda no início e meados da década de 2010. A ascensão do data lake 1.0, NoSQL e big

⁴Figura 8-11 e o exemplo que ele descreve são significativamente baseados em "Apresentando Stream—Stream Joins em Apache Spark 2.3" por Tathagata Das e Joseph Torres (Databricks Blogue de engenharia, 13 de março de 2018).

os sistemas de dados permitiram que os engenheiros contornassem a modelagem de dados tradicional, às vezes para ganhos legítimos de desempenho. Outras vezes, a falta de modelagem de dados rigorosa criava pântanos de dados, juntamente com muitos dados redundantes, incompatíveis ou simplesmente errados.

Hoje em dia, o pêndulo parece estar voltando para a modelagem de dados. A crescente popularidade do gerenciamento de dados (em particular, governança e qualidade de dados) está aumentando a necessidade de uma lógica de negócios coerente. A ascensão meteórica da proeminência dos dados nas empresas cria um crescente reconhecimento de que a modelagem é crítica para a obtenção de valor nos níveis mais altos da pirâmide da hierarquia de necessidades da ciência de dados. Dito isso, acreditamos que novos paradigmas são necessários para realmente abraçar as necessidades de streaming de dados e ML. Nesta seção, examinamos as técnicas atuais de modelagem de dados convencionais e refletimos brevemente sobre o futuro da modelagem de dados.

O que é um modelo de dados?

Amo~~odelo~~ de dados representa a forma como os dados se relacionam com o mundo real. Ele reflete como os dados devem ser estruturados e padronizados para melhor refletir os processos, definições, fluxos de trabalho e lógica de sua organização. Um bom modelo de dados captura como a comunicação e o trabalho fluem naturalmente em sua organização. Em contraste, um modelo de dados ruim (ou inexistente) é aleatório, confuso e incoerente.

Alguns profissionais de dados consideram a modelagem de dados tediosa e reservada para “grandes empresas”. Como a maioria das boas práticas de higiene, como passar o fio dental nos dentes e ter uma boa noite de sono, a modelagem de dados é reconhecida como uma boa coisa a se fazer, mas muitas vezes é ignorada na prática. Idealmente, toda organização deve modelar seus dados apenas para garantir que a lógica e as regras de negócios sejam traduzidas na camada de dados.

Ao modelar dados, é fundamental se concentrar em traduzir o modelo em resultados de negócios. Um bom modelo de dados deve estar correlacionado com decisões de negócios impactantes. Por exemplo, um *cliente* pode significar coisas diferentes para diferentes departamentos de uma empresa. Alguém que comprou de você nos últimos 30 dias é um cliente? E se eles não compraram de você nos últimos seis meses ou um ano? Definir e modelar cuidadosamente esses dados do cliente pode ter um grande impacto nos relatórios posteriores sobre o comportamento do cliente ou na criação de modelos de rotatividade de clientes, nos quais o tempo desde a última compra é uma variável crítica.



Um bom modelo de dados contém definições consistentes. Na prática, as definições costumam ser confusas em toda a empresa. Você consegue pensar em conceitos ou termos em sua empresa que possam significar coisas diferentes para pessoas diferentes?

Nossa discussão se concentra principalmente na modelagem de dados em lote, já que é onde surgiram a maioria das técnicas de modelagem de dados. Também veremos algumas abordagens para modelar dados de streaming e considerações gerais para modelagem.

Modelos de dados conceituais, lógicos e físicos

Ao modelar dados, a ideia é passar dos conceitos abstratos de modelagem para a implementação concreta. Ao longo deste continuum ([Figura 8-12](#)), três principais modelos de dados são conceitual, lógico e físico. Esses modelos formam a base para as várias técnicas de modelagem que descrevemos neste capítulo:

Conceptual

Contém regras e lógica de negócios e descreve os dados do sistema, como esquemas, tabelas e campos (nomes e tipos). Ao criar um modelo conceitual, muitas vezes é útil visualizá-lo em um diagrama entidade-relacionamento (ER), que é uma ferramenta padrão para visualizar as relações entre várias entidades em seus dados (pedidos, clientes, produtos, etc.). Por exemplo, um diagrama ER pode codificar as conexões entre ID do cliente, nome do cliente, endereço do cliente e pedidos do cliente. A visualização de relacionamentos entre entidades é altamente recomendada para projetar um modelo de dados conceitual coerente.

Lógico

Detalha como o modelo conceitual será implementado na prática, adicionando significativamente mais detalhes. Por exemplo, adicionariíamos informações sobre os tipos de ID do cliente, nomes de clientes e endereços personalizados. Além disso, mapeamos chaves primárias e estrangeiras.

Físico

Define como o modelo lógico será implementado em um sistema de banco de dados. Adicionariíamos bancos de dados, esquemas e tabelas específicos ao nosso modelo lógico, incluindo detalhes de configuração.

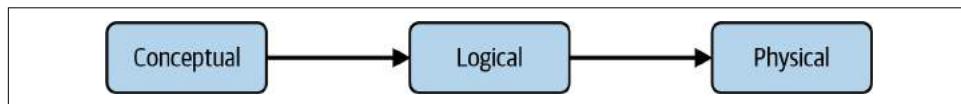


Figura 8-12. O continuum de modelos de dados: conceitual, lógico e físico

A modelagem de dados bem-sucedida envolve as partes interessadas do negócio no início do processo. Os engenheiros precisam obter definições e objetivos de negócios para os dados. A modelagem de dados deve ser um esporte de contato total, cujo objetivo é fornecer aos negócios dados de qualidade para insights açãoáveis e automação inteligente. Esta é uma prática da qual todos devem participar continuamente.

Outra consideração importante para a modelagem de dados é a resolução na qual os dados são armazenados e consultados. A granulação geralmente está no nível de uma chave primária em uma tabela, como ID do cliente, ID do pedido e ID do produto; geralmente é acompanhado por uma data ou carimbo de data/hora para maior fidelidade.

Por exemplo, suponha que uma empresa acabou de começar a implantar relatórios de BI. A empresa é pequena o suficiente para que a mesma pessoa preencha as funções de engenheiro de dados e analista. Uma solicitação chega para um relatório que resume os pedidos diários dos clientes. Especificamente, o relatório deve listar todos os clientes que fizeram pedidos, o número de pedidos feitos naquele dia e o valor total gasto.

Este relatório é inherentemente granular. Não contém detalhes sobre os gastos por pedido ou os itens de cada pedido. É tentador para o engenheiro/analista de dados ingerir dados do banco de dados de ordens de produção e reduzi-los a uma tabela de relatórios com apenas os dados agregados básicos necessários para o relatório. No entanto, isso implicaria recomeçar quando chegasse uma solicitação para um relatório com agregação de dados mais refinada.

Como o engenheiro de dados é bastante experiente, ele decide criar tabelas com dados detalhados sobre pedidos de clientes, incluindo cada pedido, item, custo de item, IDs de item etc. Essencialmente, suas tabelas contêm todos os detalhes sobre pedidos de clientes. A granulação dos dados está no nível do pedido do cliente. Esses dados do pedido do cliente podem ser analisados como estão ou agregados para estatísticas resumidas sobre a atividade do pedido do cliente.

Em geral, você deve se esforçar para modelar seus dados no menor nível de granulação possível. A partir daqui, é fácil agregar esse conjunto de dados altamente granular. O inverso não é verdadeiro e geralmente é impossível restaurar detalhes que foram agregados.

Normalização

Normalização é uma prática de modelagem de dados de banco de dados que reforça o controle estrito sobre os relacionamentos de tabelas e colunas em um banco de dados. O objetivo da normalização é remover a redundância de dados dentro de um banco de dados e garantir a integridade referencial. Basicamente, é *não se repita*(DRY) aplicado aos dados em um banco de dados.⁵

A normalização é normalmente aplicada a bancos de dados relacionais contendo tabelas com linhas e colunas (usamos os termos *coluna* e *campo* alternadamente nesta seção). Foi introduzido pela primeira vez pelo pioneiro do banco de dados relacional Edgar Codd no início dos anos 1970.

⁵ Para mais detalhes sobre o princípio DRY, consulte *O programador pragmático* por David Thomas e Andrew Hunt (Addison-Wesley Professional, 2019).

Codd delineou quatro objetivos principais da normalização:⁶

- Para liberar a coleção de relações de dependências indesejáveis de inserção, atualização e exclusão
- Reduzir a necessidade de reestruturar a coleta de relações, à medida que novos tipos de dados são introduzidos e, assim, aumentar a vida útil dos programas aplicativos
- Para tornar o modelo relacional mais informativo para os usuários
- Tornar a coleção de relações neutra para as estatísticas de consulta, onde essas estatísticas podem mudar com o passar do tempo

Codd introduziu a ideia de *formas normais*. As formas normais são sequenciais, com cada forma incorporando as condições das formas anteriores. Descrevemos as três primeiras formas normais de Codd aqui:

Desnormalizado

Sem normalização. Dados aninhados e redundantes são permitidos.

Primeira forma normal (1NF)

Cada coluna é única e tem um único valor. A tabela tem uma chave primária única.

Segunda forma normal (2FN)

Os requisitos de 1NF, mais dependências parciais são removidos.

Terceira forma normal (3FN)

Os requisitos da 2NF, mais cada tabela contém apenas campos relevantes relacionados à sua chave primária e não possui dependências transitivas.

Vale a pena gastar um momento para descompactar alguns termos que acabamos de jogar para você. A *chave primária única* é um único campo ou um conjunto de vários campos que determina exclusivamente as linhas na tabela. Cada valor de chave ocorre no máximo uma vez; caso contrário, um valor seria mapeado para várias linhas na tabela. Assim, todos os outros valores em uma linha dependem (podem ser determinados a partir) da chave. A *dependência parcial* ocorre quando um subconjunto de campos em uma chave composta pode ser usado para determinar uma coluna não-chave da tabela. A *dependência transitiva* ocorre quando um campo não-chave depende de outro campo não-chave.

Vejamos os estágios de normalização - de desnormalizado a 3NF - usando um exemplo de comércio eletrônico de pedidos de clientes ([Tabela 8-1](#)). Daremos explicações concretas de cada um dos conceitos introduzidos no parágrafo anterior.

6 EF Codd, "Further Normalization of the Data Base Relational Model," IBM Research Laboratory (1971),
<https://oreil.ly/Muajm>.

Tabela 8-1. Detalhes do pedido

Código do pedido	Itens de ordem	Identificação do Cliente	Nome do cliente	Data do pedido
100	<pre>[{ "sku":1, "preço":50, "quantidade":1, "nome:"Thingamajig"}, { "sku":2, "preço":25, "quantidade":2, "nome:"Objeto ou pessoa desconhecido" }]</pre>	5	joe reis	2022-03-01

Primeiro, isso desnormalizou a tabela de pedidos. A tabela contém cinco campos. A chave primária é o ID do pedido. Observe que o campo 'Itens de ordem' contém um objeto aninhado com dois SKUs juntos com seu preço, quantidade e nome.

Para converter esses dados para 1NF, vamos mover os itens de ordem para quatro campos (Tabela 8-2). Agora temos uma tabela de pedidos na qual os campos não contêm repetições ou dados aninhados.

Tabela 8-2. Detalhes do pedido sem repetições ou dados aninhados

Código do pedido	SKU	Preço	Quantidade	Nome do Produto	Identificação do Cliente	Nome do cliente	Data do pedido
100	1	50	1	Thingamajig	5	joe reis	2022-03-01
100	2	25	2	Objeto ou pessoa desconhecido	5	joe reis	2022-03-01

O problema é que agora não temos uma chave primária única. Ou seja, 100 ocorre no campo 'Código do pedido' em duas linhas diferentes. Para entender melhor a situação, vejamos uma amostra maior de nossa tabela (Tabela 8-3).

Tabela 8-3. Detalhes do pedido com uma amostra maior

Código do pedido	SKU	Preço	Quantidade	Nome do Produto	Identificação do Cliente	Nome do cliente	Data do pedido
100	1	50	1	Thingamajig	5	joe reis	2022-03-01
100	2	25	2	Objeto ou pessoa desconhecido	5	joe reis	2022-03-01
101	3	75	1	Whozeewhatzit	7	Matt Housley	2022-03-01
102	1	50	1	Thingamajig	7	Matt Housley	2022-03-01

Para criar uma chave primária (composta) exclusiva, vamos numerar as linhas em cada ordem adicionando uma coluna chamada LineItemNúmero (Tabela 8-4).

Tabela 8-4. Detalhes do pedido com LineItemNumber coluna

Ordem EUA	Item da linha Número	SKU	Preço	Quantidade	produtos Nome	Cliente EUA	Cliente Nome	Data do pedido
100	1	1	50	1	Thingama gabarito	5	joe reis	2022-03-01
100	2	2	25	2	Whatchama chame-o	5	joe reis	2022-03-01
101	1	3	75	1	Whozee o que é isso	7	Matt Housley	2022-03-01
102	1	1	50	1	Thingama gabarito	7	Matt Housley	2022-03-01

A chave composta (OrderID, LineItemNumber) agora é uma chave primária exclusiva.

Para chegar ao 2NF, precisamos garantir que não existam dependências parciais. A dependência parcial é uma coluna não-chave totalmente determinada por um subconjunto das colunas na chave primária (composta) exclusiva; dependências parciais podem ocorrer somente quando a chave primária é composta. No nosso caso, as últimas três colunas são determinadas pelo número do pedido. Para corrigir esse problema, vamos dividir Detalhes do pedido em duas tabelas: Pedidos e OrderLineItem (Tabelas 8-5 e 8-6).

Tabela 8-5. Pedidos

Código do pedido	Identificação do Cliente	Nome do cliente	Data do pedido
100	5	joe reis	2022-03-01
101	7	Matt Housley	2022-03-01
102	7	Matt Housley	2022-03-01

Tabela 8-6. OrderLineItem

Código do pedido	LineItemNumber	SKU	Preço	Quantidade	Nome do Produto
100	1	1	50	1	Thingamajig
100	2	2	25	2	Objeto ou pessoa desconhecido
101	1	3	75	1	Whozeewhatzit
102	1	1	50	1	Thingamajig

A chave composta (OrderID, LineItemNumber) é uma chave primária única para Item de linha do pedido, enquanto Código do pedido é uma chave primária para Pedidos.

Notar que SKU determina Nome do Produto em OrderLineItem. Aquilo é, SKU depende da chave composta, e Nome do Produto depende de SKU. Esta é uma dependência transitiva. vamos quebrar OrderLineItem em OrderLineItem e Skus (Tabelas 8-7 e 8-8).

Tabela 8-7.OrderLineItem

Código do pedido	LineItemNumber	SKU	Preço	Quantidade
100	1	1	50	1
100	2	2	25	2
101	1	3	75	1
102	1	1	50	1

Tabela 8-8.Skus

SKU	Nome do Produto
1	Thingamajig
2	Objeto ou pessoa desconhecido
3	Whozeewhatzit

Agora, ambasOrderLineItem e Skus estão na 3FN. Notar quePedidos não satisfaz 3NF. Quais dependências transitivas estão presentes? Como você consertaria isso?

Existem formas normais adicionais (até 6NF no sistema Boyce-Codd), mas são muito menos comuns que as três primeiras. Um banco de dados geralmente é considerado normalizado se estiver na terceira forma normal, e essa é a convenção que usamos neste livro.

O grau de normalização que você deve aplicar aos seus dados depende do seu caso de uso. Não existe uma solução única para todos, especialmente em bancos de dados onde alguma desnormalização apresenta vantagens de desempenho. Embora a desnormalização possa parecer um antipadrão, é comum em muitos sistemas OLAP que armazenam dados semiestruturados. Estude as convenções de normalização e as melhores práticas de banco de dados para escolher uma estratégia apropriada.

Técnicas para modelagem de dados analíticos em lote

Ao descrever a modelagem de dados para data lakes ou data warehouses, você deve presumir que os dados brutos assumem muitas formas (por exemplo, estruturados e semiestruturados), mas a saída é um modelo de dados estruturados de linhas e colunas. No entanto, várias abordagens para modelagem de dados podem ser usadas nesses ambientes. As grandes abordagens que você provavelmente encontrará são Kimball, Inmon e Data Vault.

Na prática, algumas dessas técnicas podem ser combinadas. Por exemplo, vemos algumas equipes de dados começarem com o Data Vault e, em seguida, adicionarem um esquema estrela Kimball ao lado dele. Também veremos modelos de dados amplos e desnormalizados e outras técnicas de modelagem de dados em lote que você deve ter em seu arsenal. Conforme discutimos cada uma dessas técnicas, usaremos o exemplo de modelagem de transações que ocorrem em um sistema de pedidos de comércio eletrônico.



Nossa cobertura das três primeiras abordagens - Inmon, Kimball e Data Vault - é superficial e dificilmente faz justiça à sua respectiva complexidade e nuances. No final de cada seção, listamos os livros canônicos de seus criadores. Para um engenheiro de dados, esses livros são leituras obrigatórias e recomendamos que você os leia, apenas para entender como e por que a modelagem de dados é fundamental para dados analíticos em lote.

Inmon

O pai do data warehouse, Bill Inmon, criou sua abordagem para modelagem de dados em 1989. Antes do data warehouse, a análise costumava ocorrer diretamente no próprio sistema de origem, com a consequência óbvia de sobrecarregar os bancos de dados transacionais de produção com consultas. O objetivo do data warehouse era separar o sistema de origem do sistema analítico.

Inmon define um data warehouse da seguinte maneira:⁷

Um data warehouse é uma coleção de dados orientada por assunto, integrada, não volátil e variável no tempo para dar suporte às decisões da administração. O data warehouse contém dados corporativos granulares. Os dados no data warehouse podem ser usados para muitos propósitos diferentes, incluindo sentar e esperar por requisitos futuros que são desconhecidos hoje.

As quatro partes críticas de um data warehouse podem ser descritas da seguinte forma:

Orientado para o assunto

O data warehouse concentra-se em uma área de assunto específica, como vendas ou marketing.

Integrado

Os dados de fontes diferentes são consolidados e normalizados.

Não volátil

Os dados permanecem inalterados depois que são armazenados em um data warehouse.

Tempo variável

Vários intervalos de tempo podem ser consultados.

Vamos examinar cada uma dessas partes para entender sua influência em um modelo de dados Inmon. Primeiro, o modelo lógico deve se concentrar em uma área específica. Por exemplo, se o *orientação do assunto* é "vendas", então o modelo lógico contém todos os detalhes relacionados a vendas — chaves de negócios, relacionamentos, atributos, etc. Em seguida, esses detalhes são *integrados* em um modelo de dados consolidado e altamente normalizado. Finalmente, os dados são armazenados inalterados em um *não volátil tempo variável* maneira, o que significa que você pode (teoricamente) consultar os dados originais pelo tempo que o histórico de armazenamento permitir. O data warehouse Inmon deve

⁷ HW Inmon, *Construindo o Data Warehouse* (Hoboken: Wiley, 2005).

aderir estritamente a todas essas quatro partes críticas em apoio às decisões da administração. Este é um ponto sutil, mas posiciona o data warehouse para análise, não OLTP.

Aqui está outra característica chave do data warehouse da Inmon:⁸

A segunda característica saliente do data warehouse é que ele é integrado. De todos os aspectos de um data warehouse, a integração é o mais importante. Os dados são alimentados a partir de várias fontes distintas no data warehouse. À medida que os dados são alimentados, eles são convertidos, reformatados, resequenciados, resumidos, etc. O resultado é que os dados – uma vez que residem no data warehouse – têm uma única imagem corporativa física.

Com o data warehouse da Inmon, os dados são integrados de toda a organização em um modelo de ER granular e altamente normalizado, com ênfase implacável em ETL. Devido à natureza orientada para o assunto do data warehouse, o data warehouse Inmon consiste em bancos de dados de origem chave e sistemas de informações usados em uma organização. Os dados dos principais sistemas de origem de negócios são ingeridos e integrados em um data warehouse altamente normalizado (3NF) que geralmente se assemelha à estrutura de normalização do próprio sistema de origem; os dados são trazidos de forma incremental, começando com as áreas de negócios de maior prioridade. O requisito de normalização estrita garante a menor duplicação de dados possível, o que leva a menos erros analíticos downstream porque os dados não divergirão ou sofrerão redundâncias. O data warehouse representa uma “única fonte de verdade,” que suporta os requisitos gerais de informação do negócio. Os dados são apresentados para relatórios downstream e análises por meio de data marts específicos de negócios e departamentos, que também podem ser desnormalizados.

Vejamos como um data warehouse Inmon é usado para comércio eletrônico ([Figura 8-13](#)). Os sistemas de origem de negócios são pedidos, estoque e marketing. Os dados desses sistemas de origem são ETLED para o data warehouse e armazenados em 3NF. Idealmente, o data warehouse abrange de forma holística as informações do negócio. Para fornecer dados para solicitações de informações específicas do departamento, os processos ETL pegam dados do data warehouse, transformam os dados e os colocam em data marts downstream para serem visualizados em relatórios.

Uma opção popular para modelar dados em um data mart é um esquema em estrela (discutido na seção a seguir sobre Kimball), embora qualquer modelo de dados que forneça informações facilmente acessíveis também seja adequado. No exemplo anterior, vendas, marketing e compras têm seu próprio esquema em estrela, alimentado a montante pelos dados granulares no data warehouse. Isso permite que cada departamento tenha sua própria estrutura de dados, exclusiva e otimizada para suas necessidades específicas.

A Inmon continua a inovar no espaço de data warehouse, atualmente focando em ETL textual no data warehouse. Ele também é um escritor e pensador prolífico, escrevendo sobre

8 Inmon, *Construindo o Data Warehouse*.

60 livros e inúmeros artigos. Para ler mais sobre o data warehouse de Inmon, consulte seus livros listados em “[Recursos adicionais](#)” na página 335.

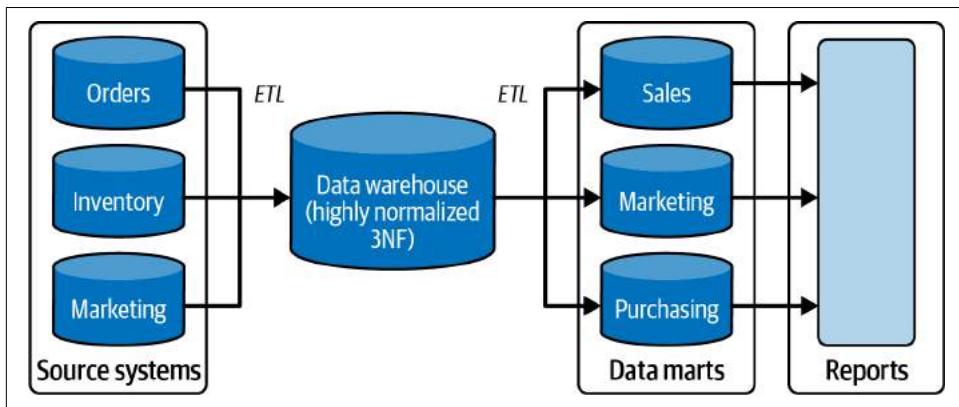


Figura 8-13. Um data warehouse de comércio eletrônico

Kimball

Se houver espectros para modelagem de dados, Kimball está no extremo oposto de Inmon. Criada por Ralph Kimball no início da década de 1990, essa abordagem de modelagem de dados se concentra menos na normalização e, em alguns casos, na aceitação da desnормalização. Como Inmon diz sobre a diferença entre o data warehouse e o data mart, “Um data mart nunca é um substituto para um data warehouse”.⁹

Enquanto o Inmon integra dados de toda a empresa no data warehouse e fornece análises específicas do departamento por meio de data marts, o modelo Kimball é de baixo para cima, incentivando você a modelar e fornecer análises de departamento ou negócios no próprio data warehouse (Inmon argumenta que isso abordagem distorce a definição de um data warehouse). A abordagem de Kimball efetivamente torna o data mart o próprio data warehouse. Isso pode permitir iteração e modelagem mais rápidas do que o Inmon, com a compensação de uma possível integração de dados mais flexível, redundância de dados e duplicação.

Na abordagem de Kimball, os dados são modelados com dois tipos gerais de tabelas: fatos e dimensões. Você pode pensar em um *tabela de fatos* como uma tabela de números, e *tabelas de dimensões* como dados qualitativos referenciando um fato. As tabelas de dimensões envolvem uma única tabela de fatos em um relacionamento chamado *esquema em estrela* (Figura 8-14).¹⁰ Vamos ver fatos, dimensões e esquemas em estrela.

9 Inmon, *Construindo o Data Warehouse*.

10 Embora dimensões e fatos sejam frequentemente associados a Kimball, eles foram usados pela primeira vez na General Mills e Dartmouth University na década de 1960 e teve adoção inicial na Nielsen e IRI, entre outras empresas.

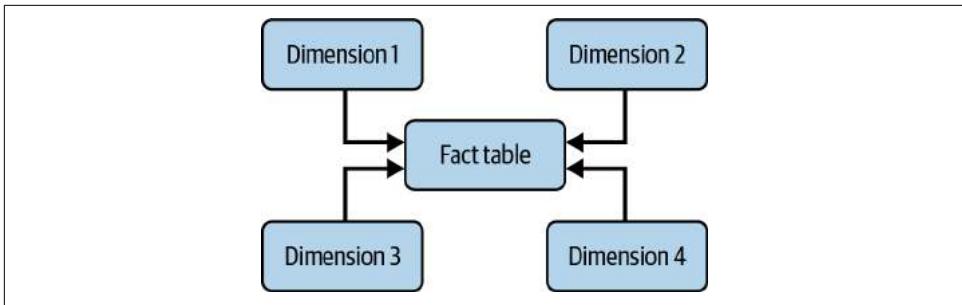


Figura 8-14. Um esquema de estrela de Kimball, com fatos e dimensões

Tabelas de fatos. O primeiro tipo de tabela em um esquema em estrela é a tabela de fatos, que contém *factual*, dados quantitativos e relacionados a eventos. Os dados em uma tabela de fatos são imutáveis porque os fatos se relacionam a eventos. Portanto, as tabelas de fatos não mudam e são apenas anexadas. As tabelas de fatos são geralmente estreitas e longas, o que significa que não têm muitas colunas, mas muitas linhas que representam eventos. As tabelas de fatos devem ter a menor granulação possível.

As consultas em um esquema em estrela começam com a tabela de fatos. Cada linha de uma tabela de fatos deve representar a granulação dos dados. Evite agregar ou derivar dados dentro de uma tabela de fatos. Se precisar executar agregações ou derivações, faça-o em uma consulta downstream, tabela de data mart ou visualização. Finalmente, as tabelas de fatos não fazem referência a outras tabelas de fatos; eles fazem referência apenas a dimensões.

Vejamos um exemplo de uma tabela de fatos elementar ([Tabela 8-9](#)). Uma pergunta comum em sua empresa pode ser: "Mostre-me as vendas brutas, por pedido de cada cliente, por data". Novamente, os fatos devem ter a menor granulação possível - neste caso, o orderID da venda, cliente, data e valor bruto da venda. Observe que os tipos de dados na tabela de fatos são todos números (inteiros e flutuantes); não há cordas. Também, neste exemplo, Chave do cliente 7 tem dois pedidos no mesmo dia, refletindo o grão da mesa. Em vez disso, a tabela de fatos possui chaves que fazem referência a tabelas de dimensão contendo seus respectivos atributos, como as informações de cliente e data. O valor bruto de vendas representa a venda total para as vendas e evento.

Tabela 8-9. Uma tabela de fatos

Código do pedido	CustomerKey	DateKey	Valor BrutoVendas
100	5	20220301	100,00
101	7	20220301	75,00
102	7	20220301	50,00

Tabelas de dimensões. O segundo tipo principal de tabela em um modelo de dados Kimball é chamado de *dimensão*. As tabelas de dimensões fornecem os dados de referência, atributos e contexto relacional para os eventos armazenados nas tabelas de fatos. As tabelas de dimensão são menores que o fato

mesas e assumem uma forma oposta, geralmente larga e curta. Quando associadas a uma tabela de fatos, as dimensões podem descrever o quê, onde e quando dos eventos. As dimensões são desnormalizadas, com possibilidade de dados duplicados. Isso está correto no modelo de dados Kimball. Vejamos as duas dimensões mencionadas no exemplo anterior da tabela de fatos.

Em um modelo de dados Kimball, as datas são normalmente armazenadas em uma dimensão de data, permitindo que você faça referência à chave de data (DateKey) entre a tabela de dimensão de fato e data. Com a tabela de dimensão de data, você pode responder facilmente a perguntas como: "Quais são minhas vendas totais no primeiro trimestre de 2022?" ou "Quantos clientes compram a mais na terça do que na quarta?" Observe que temos cinco campos além da chave de data ([Tabela 8-10](#)). A beleza de uma dimensão de data é que você pode adicionar quantos novos campos fizer sentido para analisar seus dados.

Tabela 8-10. Uma tabela de dimensão de data

DateKey	Data-ISO	Ano	Trimestre	Mês	Dia da semana
20220301	2022-03-01	2022	1	3	Terça-feira
20220302	2022-03-02	2022	1	3	Quarta-feira
20220303	2022-03-03	2022	1	3	Quinta-feira

Tabela 8-11 também faz referência a outra dimensão - a dimensão do cliente - pelo CustomerKey campo. A dimensão do cliente contém vários campos que descrevem o cliente: nome e sobrenome, código postal e alguns campos de data de aparência peculiar. Vejamos esses campos de data, pois eles ilustram outro conceito no modelo de dados de Kimball: uma dimensão Tipo 2 que muda lentamente, que descreveremos com mais detalhes a seguir.

Tabela 8-11. Uma tabela de dimensão do cliente Tipo 2

CustomerKey	Primeiro nome	Sobrenome	CEP	EFF_StartDate	EFF_EndDate
5	Joe	reis	84108	2019-01-04	9999-01-01
7	Matt	Housley	84101	2020-05-04	19-09-2021
7	Matt	Housley	84123	19-09-2021	9999-01-01
11	Lana	Bela	90210	2022-02-04	9999-01-01

Por exemplo, dê uma olhada na CustomerKey 5, com o EFF_StartDate (EFF_StartDate significando *data de início efetiva*) de 2019-01-04 e um EFF_EndDate de 9999-01-01. Isso significa que o registro do cliente de Joe Reis foi criado na tabela de dimensão do cliente em 2019-01-04 e tem uma data final de 9999-01-01. Interessante. O que significa esta data de término? Isso significa que o registro do cliente está ativo e não foi alterado.

Agora vamos ver o registro do cliente de Matt Housley (ClienteChave = 7). Observe as duas entradas para a data de início de Housley: 2020-05-04 e 19-09-2021. Parece que Housley mudou seu CEP em 19/09/2021, resultando em uma alteração em seu registro de cliente.

Quando os dados forem consultados para os registros de clientes mais recentes, você consultará onde a data final é igual a 9999-01-01.

Uma dimensão de alteração lenta (SCD) é necessária para rastrear as alterações nas dimensões. O exemplo anterior é um SCD Tipo 2: um novo registro é inserido quando um registro existente é alterado. Embora os SCDS possam subir até sete níveis, vejamos os três mais comuns:

Tipo 1

Substitua os registros de dimensão existentes. Isso é super simples e significa que você não tem acesso aos registros de dimensão histórica excluídos.

Tipo 2

Mantenha um histórico completo dos registros de dimensão. Quando um registro é alterado, esse registro específico é sinalizado como alterado e um novo registro de dimensão é criado para refletir o status atual dos atributos. Em nosso exemplo, Housley mudou para um novo CEP, o que acionou seu registro inicial para refletir uma data de término efetiva, e um novo registro foi criado para mostrar seu novo CEP.

Tipo 3

Um SCD Tipo 3 é semelhante a um SCD Tipo 2, mas em vez de criar uma nova linha, uma alteração em um SCD Tipo 3 cria um novo campo. Usando o exemplo anterior, vamos ver como isso se parece com um SCD Tipo 3 nas tabelas a seguir.

Em [Tabela 8-12](#), Housley mora no CEP 84101. Quando Housley muda para um novo código postal, o SCD Tipo 3 cria dois novos campos, um para seu novo código postal e a data da alteração ([Tabela 8-13](#)). O campo de código postal original também foi renomeado para refletir que este é o registro mais antigo.

Tabela 8-12. Tipo 3 mudando lentamente a dimensão

CustomerKey	Primeiro nome	Sobrenome	CEP
7	Matt	Housley	84101

Tabela 8-13. Tabela de dimensões do cliente tipo 3

CustomerKey	Primeiro nome	Sobrenome	CEP original	CEP atual	Data atual
7	Matt	Housley	84101	84123	19-09-2021

Dos tipos de SCDS descritos, o Tipo 1 é o comportamento padrão da maioria dos data warehouses, e o Tipo 2 é aquele que vemos mais comumente usado na prática. Há muito o que saber sobre dimensões e sugerimos usar esta seção como ponto de partida para se familiarizar com o funcionamento e uso das dimensões.

Esquema estrela. Agora que você tem uma compreensão básica dos fatos e dimensões, é hora de integrá-los em um esquema em estrela. O *esquema em estrela* representa o modelo de dados do negócio. Ao contrário de abordagens altamente normalizadas para modelagem de dados, o esquema em estrela é uma tabela de fatos cercada pelas dimensões necessárias. Isso resulta em menos junções do que outros modelos de dados, o que acelera o desempenho da consulta. Outra vantagem de um esquema em estrela é indiscutivelmente mais fácil para os usuários corporativos entenderem e usarem.

Observe que o esquema em estrela não deve refletir um relatório específico, embora você possa modelar um relatório em um data mart downstream ou diretamente em sua ferramenta de BI. O esquema em estrela deve capturar os fatos e atributos de sua *lógica de negócio* e ser suficientemente flexível para responder às respectivas perguntas críticas.

Como um esquema em estrela tem uma tabela de fatos, às vezes você terá vários esquemas em estrela que tratam de diferentes fatos do negócio. Você deve se esforçar para reduzir o número de dimensões sempre que possível, pois esses dados de referência podem ser potencialmente reutilizados entre diferentes tabelas de fatos. Uma dimensão que é reutilizada em vários esquemas em estrela, compartilhando assim os mesmos campos, é chamada de *dimensão conformada*. Uma dimensão conformada permite combinar várias tabelas de fatos em vários esquemas em estrela. Lembre-se de que dados redundantes são aceitáveis com o método Kimball, mas evite replicar as mesmas tabelas de dimensão para evitar definições de negócios e integridade de dados flutuantes.

O modelo de dados Kimball e o esquema em estrela têm muitas nuances. Você deve estar ciente de que este modo é apropriado apenas para dados em lote e não para dados de streaming. Como o modelo de dados de Kimball é popular, há uma boa chance de você topar com ele.

Cofre de dados

Enquanto Kimball e Inmon se concentram na estrutura da lógica de negócios no data warehouse, o *Cofre de dados* oferece uma abordagem diferente para a modelagem de dados.¹¹ Criada na década de 1990 por Dan Linstedt, a metodologia Data Vault separa os aspectos estruturais dos dados de um sistema de origem de seus atributos. Em vez de representar a lógica de negócios em fatos, dimensões ou tabelas altamente normalizadas, um Cofre de Dados simplesmente carrega dados de sistemas de origem diretamente em um punhado de tabelas criadas especificamente de maneira apenas para inserção. Ao contrário das outras abordagens de modelagem de dados sobre as quais você aprendeu, não há noção de dados bons, ruins ou conformados em um Data Vault.

Atualmente, os dados se movem rapidamente e os modelos de dados precisam ser ágeis, flexíveis e escaláveis; a metodologia Data Vault visa atender a essa necessidade. O objetivo dessa metodologia é manter os dados o mais próximo possível do negócio, mesmo enquanto os dados do negócio evoluem.

¹¹ O Data Vault tem duas versões, 1.0 e 2.0. Esta seção se concentra no Data Vault 2.0, mas vamos chamá-lo *Dados Cofre* por uma questão de brevidade.

Um modelo de Data Vault consiste em três tipos principais de tabelas: hubs, links e satélites ([Figura 8-15](#)). Resumindo, um *hub* armazena chaves de negócios, um *link* mantém relacionamentos entre as chaves de negócios e um *satélite* representa os atributos e o contexto de uma chave comercial. Um usuário consultará um hub, que será vinculado a uma tabela satélite contendo os atributos relevantes da consulta. Vamos explorar hubs, links e satélites com mais detalhes.



Figura 8-15. Tabelas do Data Vault: hubs, links e satélites conectados entre si

Hubs. As consultas geralmente envolvem a pesquisa por uma chave de negócios, como um ID de cliente ou um ID de pedido de nosso exemplo de comércio eletrônico. Um hub é a entidade central de um Data Vault que retém um registro de todas as chaves de negócios exclusivas carregadas no Data Vault.

Um hub sempre contém os seguintes campos padrão:

Chave de hash

A chave primária usada para juntar dados entre sistemas. Este é um campo de hash calculado (MD5 ou similar).

Data de carregamento

A data em que os dados foram carregados no hub.

Fonte de registro

A fonte da qual o registro exclusivo foi obtido.

Chave(s) de negócios

A chave usada para identificar um registro exclusivo.

É importante observar que um hub é somente de inserção e os dados não são alterados em um hub. Depois que os dados são carregados em um hub, eles são permanentes.

Ao projetar um hub, identificar a chave de negócios é fundamental. Pergunte a si mesmo: Qual é o *elemento comercial identificável*?¹² Dito de outra forma, como os usuários geralmente procuram dados? Idealmente, isso é descoberto quando você constrói o modelo de dados conceitual de sua organização e antes de começar a construir seu Data Vault.

Usando nosso cenário de comércio eletrônico, vejamos um exemplo de hub para produtos. Primeiro, vamos ver o design físico de um hub de produtos ([Tabela 8-14](#)).

12 Kent Graziano, "Data Vault 2.0 Modeling Basics," Vertabelo, 20 de outubro de 2015, <https://oreil.ly/iuW1U>.

Tabela 8-14. Um projeto físico para um hub de produtos

HubProduto
ProductHashKey
CarregarData
RecordSource
ID do produto

Na prática, o hub do produto fica assim quando preenchido com dados (Tabela 8-15). Neste exemplo, três produtos diferentes são carregados em um hub a partir de um sistema ERP em duas datas distintas.

Tabela 8-15. Um hub de produto preenchido com dados

ProductHashKey	CarregarData	RecordSource	ID do produto
4041fd80ab...	2020-01-02	ERP	1
de8435530d...	2021-03-09	ERP	2
cf27369bd8...	2021-03-09	ERP	3

Já que estamos nisso, vamos criar outro hub para pedidos (Tabela 8-16) usando o mesmo esquema que Hub Produto, e preencha-o com alguns dados de pedido de amostra.

Tabela 8-16. Um hub de pedidos preenchido com dados

OrderHashKey	CarregarData	RecordSource	Código do pedido
f899139df5...	2022-03-01	Local na rede Internet	100
38b3eff8ba...	2022-03-01	Local na rede Internet	101
ec8956637a...	2022-03-01	Local na rede Internet	102

Ligações. A tabela de links rastreia os relacionamentos das chaves de negócios entre os hubs. As tabelas de link conectam os hubs, idealmente com a granulação mais baixa possível. Como as tabelas de link conectam dados de vários hubs, elas são muitas para muitas. Os relacionamentos do modelo do Data Vault são diretos e manipulados por meio de alterações nos links. Isso fornece excelente flexibilidade no caso inevitável de alteração dos dados subjacentes. Você simplesmente cria um novo link que liga os conceitos de negócios (ou hubs) para representar o novo relacionamento. É isso! Agora vamos ver maneiras de visualizar dados contextualmente usando satélites.

De volta ao nosso exemplo de comércio eletrônico, gostaríamos de associar pedidos a produtos. Vamos ver como pode ser uma tabela de links para pedidos e produtos (Tabela 8-17).

Tabela 8-17. Uma tabela de links para produtos e pedidos

LinkPedidoProduto
OrderProductHashKey
CarregarData

LinkPedidoProduto

RecordSource

ProductHashKey

OrderHashKey

Quando o LinkPedidoProduto tabela está preenchida, aqui está o que parece ([Tabela 8-18](#)). Observe que estamos usando a fonte de registro do pedido neste exemplo.

Tabela 8-18. Uma tabela de links conectando pedidos e produtos

OrderProductHashKey	CarregarData	RecordSource	ProductHashKey	OrderHashKey
ff64ec193d...	2022-03-01	Local na rede Internet	4041fd80ab...	f899139df5...
ff64ec193d...	2022-03-01	Local na rede Internet	de8435530d...	f899139df5...
e232628c25...	2022-03-01	Local na rede Internet	cf27369bd8...	38b3eff8ba...
26166a5871...	2022-03-01	Local na rede Internet	4041fd80ab...	ec8956637a...

Satélites. Descrevemos relacionamentos entre hubs e links que envolvem chaves, datas de carregamento e fontes de registro. Como você tem uma noção do que essas relações significam? *Satélites* são atributos descritivos que dão significado e contexto aos hubs. Os satélites podem se conectar a hubs ou links. Os únicos campos obrigatórios em um satélite são uma chave primária que consiste na chave comercial do hub pai e uma data de carregamento. Além disso, um satélite pode conter tantos atributos que façam sentido.

Vejamos um exemplo de satélite para o produto seixo ([Tabela 8-19](#)). Neste exemplo, o Produto Satélite tabela contém informações adicionais sobre o produto, como nome e preço do produto.

*Tabela 8-19. Produto Satélite***Produto Satélite**

ProductHashKey

CarregarData

RecordSource

Nome do Produto

Preço

E aqui está o Produto Satélite tabela com alguns dados de amostra ([Tabela 8-20](#)).

Tabela 8-20. Uma tabela satélite de produtos com dados de amostra

ProductHashKey	CarregarData	RecordSource	Nome do Produto	Preço
4041fd80ab...	2020-01-02	ERP	Thingamajig	50
de8435530d...	2021-03-09	ERP	Objeto ou pessoa desconhecido	25
cf27369bd8...	2021-03-09	ERP	Whozeewhatzit	75

Vamos juntar tudo isso e unir as tabelas de hub, produto e link em um Data Vault (Figura 8-16).

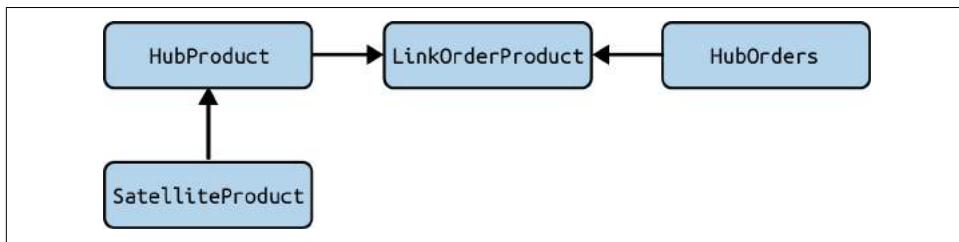


Figura 8-16. O Cofre de Dados para pedidos e produtos

Existem outros tipos de tabelas do Cofre de Dados, incluindo tabelas point-in-time (PIT) e de ponte. Não os abordamos aqui, mas os mencionamos porque o Data Vault é bastante abrangente. Nossa objetivo é simplesmente fornecer uma visão geral do poder do Data Vault.

Ao contrário de outras técnicas de modelagem de dados que discutimos, em um Data Vault, a lógica de negócios é criada e interpretada quando os dados dessas tabelas são consultados. Esteja ciente de que o modelo Data Vault pode ser usado com outras técnicas de modelagem de dados. Não é incomum que um Data Vault seja a zona de aterrissagem para dados analíticos, após o que é modelado separadamente em um data warehouse, geralmente usando um esquema em estrela. O modelo Data Vault também pode ser adaptado para NoSQL e fontes de dados de streaming. O Data Vault é um tópico enorme, e esta seção destina-se simplesmente a torná-lo ciente de sua existência.

Tabelas largas desnormalizadas

As abordagens rígidas de modelagem que descrevemos, especialmente Kimball e Inmon, foram desenvolvidas quando os data warehouses eram caros, locais e fortemente limitados por recursos com computação e armazenamento fortemente acoplados. Embora a modelagem de dados em lote tenha sido tradicionalmente associada a essas abordagens rígidas, abordagens mais relaxadas estão se tornando mais comuns.

Há razões para isso. Primeiro, a popularidade da nuvem significa que o armazenamento é muito barato. É mais barato armazenar dados do que agonizar sobre a maneira ideal de representar os dados no armazenamento. Em segundo lugar, a popularidade dos dados aninhados (JSON e similares) significa que os esquemas são flexíveis em sistemas analíticos e de origem.

Você tem a opção de modelar rigidamente seus dados conforme descrevemos ou pode optar por lançar todos os seus dados em uma única tabela ampla. *Amesa larga* é exatamente o que parece: uma coleção altamente desnormalizada e muito ampla de muitos campos, normalmente criada em um banco de dados colunar. Um campo pode ser um valor único ou conter dados aninhados. Os dados são organizados junto com uma ou várias chaves; essas chaves estão intimamente ligadas ao grande volume de dados.

Uma tabela ampla pode potencialmente ter milhares de colunas, enquanto menos de 100 são típicas em bancos de dados relacionais. Tabelas largas são geralmente esparsas; a grande maioria das entradas em um determinado campo pode ser nula. Isso é extremamente caro em um banco de dados relacional tradicional porque o banco de dados aloca uma quantidade fixa de espaço para cada entrada de campo; nulos praticamente não ocupam espaço em um banco de dados colunar. Um esquema amplo em um banco de dados relacional retarda drasticamente a leitura porque cada linha deve alocar todo o espaço especificado pelo esquema amplo e o banco de dados deve ler o conteúdo de cada linha em sua totalidade. Por outro lado, um banco de dados colunar lê apenas as colunas selecionadas em uma consulta e a leitura de nulos é essencialmente gratuita.

Tabelas amplas geralmente surgem por meio da evolução do esquema; os engenheiros adicionam campos gradualmente ao longo do tempo. A evolução do esquema em um banco de dados relacional é um processo lento e com muitos recursos. Em um banco de dados colunar, adicionar um campo inicialmente é apenas uma alteração nos metadados. À medida que os dados são gravados no novo campo, novos arquivos são adicionados à coluna.

As consultas analíticas em tabelas amplas geralmente são executadas mais rapidamente do que as consultas equivalentes em dados altamente normalizados que exigem muitas uniões. A remoção de junções pode ter um grande impacto no desempenho da verificação. A tabela ampla simplesmente contém todos os dados que você juntaria em uma abordagem de modelagem mais rigorosa. Fatos e dimensões são representados na mesma tabela. A falta de rigor do modelo de dados também significa que não há muito pensamento envolvido. Carregue seus dados em uma tabela ampla e comece a consultá-los. Especialmente com os esquemas nos sistemas de origem se tornando mais adaptáveis e flexíveis, esses dados geralmente resultam de transações de alto volume, o que significa que há muitos dados. Armazenar isso como dados aninhados em seu armazenamento analítico traz muitos benefícios.

Jogar todos os seus dados em uma única tabela pode parecer uma heresia para um modelador de dados hardcore, e vimos muitas críticas. Quais são algumas dessas críticas? A maior crítica é que, ao misturar seus dados, você perde a lógica de negócios em suas análises. Outra desvantagem é o desempenho de atualizações para coisas como um elemento em uma matriz, o que pode ser muito doloroso.

Vejamos um exemplo de uma tabela ampla ([Tabela 8-21](#)), usando a tabela desnormalizada original de nosso exemplo de normalização anterior. Esta tabela pode ter muito mais colunas - centenas ou mais! - e incluímos apenas algumas colunas para brevidade e facilidade de compreensão. Como você pode ver, esta tabela combina vários tipos de dados, representados ao longo de um grão de pedidos de um cliente em uma data.

Sugerimos o uso de uma tabela ampla quando você não se preocupa com a modelagem de dados ou quando possui muitos dados que precisam de mais flexibilidade do que o rigor da modelagem de dados tradicional oferece. Tabelas amplas também se prestam a dados de streaming, que discutiremos a seguir. À medida que os dados se movem em direção a esquemas de movimento rápido e streaming primeiro, esperamos ver uma nova onda de modelagem de dados, talvez algo na linha de “normalização relaxada”.

Tabela 8-21. Um exemplo de dados desnormalizados

Código do pedido	Itens de ordem	Identificação do Cliente	Cliente Nome	Data do pedido	Site	Site Região
100	<pre>[{ "sku":1, "preço":50, "quantidade":1, "nome:": "Thingamajig" }, { "sku":2, "preço":25, "quantidade":2, "nome:": "Objeto ou pessoa desconhecido" }]</pre>	5	joe reis	01/03/2022	abc.com	EUA

E se você não modelar seus dados?

Você também tem a opção de não modelando seus dados. Nesse caso, basta consultar as fontes de dados diretamente. Esse padrão é frequentemente usado, especialmente quando as empresas estão apenas começando e desejam obter insights rápidos ou compartilhar análises com seus usuários. Embora permita obter respostas para várias perguntas, você deve considerar o seguinte:

- Se eu não modelar meus dados, como sei que os resultados de minhas consultas são consistentes?
- Tenho definições adequadas de lógica de negócios no sistema de origem e minha consulta produzirá respostas verdadeiras?
- Que carga de consulta estou colocando em meus sistemas de origem e como isso afeta os usuários desses sistemas?

Em algum momento, você provavelmente gravitará em direção a um paradigma de modelo de dados em lote mais rígido e uma arquitetura de dados dedicada que não depende dos sistemas de origem para o trabalho pesado.

Modelagem de dados de streaming

Embora muitas técnicas de modelagem de dados estejam bem estabelecidas para lotes, esse não é o caso para dados de streaming. Devido à natureza ilimitada e contínua dos dados de streaming, traduzir técnicas de lote como Kimball para um paradigma de streaming é complicado, se não impossível. Por exemplo, dado um fluxo de dados, como você atualizaria continuamente uma dimensão Tipo 2 que muda lentamente sem deixar seu data warehouse de joelhos?

O mundo está evoluindo do lote para o streaming e do local para a nuvem. As restrições dos métodos de lote mais antigos não se aplicam mais. Dito isso, permanecem grandes questões sobre como modelar dados para equilibrar a necessidade de lógica de negócios com mudanças fluidas de esquema, dados em movimento rápido e autoatendimento. Qual é o equivalente em streaming das abordagens de modelo de dados em lote anteriores? Não há (ainda) uma abordagem de consenso na modelagem de dados de streaming. Conversamos com muitos especialistas em sistemas de dados de streaming, muitos dos quais nos disseram que a modelagem tradicional de dados orientada a lotes não se aplica ao streaming. Alguns sugeriram o Data Vault como uma opção para modelagem de dados de streaming.

Como você deve se lembrar, existem dois tipos principais de fluxos: fluxos de eventos e CDC. Na maioria das vezes, a forma dos dados nesses fluxos é semiestruturada, como JSON. O desafio com a modelagem de dados de streaming é que o esquema da carga útil pode mudar por capricho. Por exemplo, suponha que você tenha um dispositivo IoT que atualizou recentemente seu firmware e introduziu um novo campo. Nesse caso, é possível que seu data warehouse de destino downstream ou pipeline de processamento não esteja ciente dessa alteração e quebre. Isso não é ótimo. Como outro exemplo, um sistema CDC pode reformular um campo como um tipo diferente - digamos, uma string em vez de um formato de data e hora da Organização Internacional para Padronização (ISO). Novamente, como o destino lida com essa mudança aparentemente aleatória?

Os especialistas em dados de streaming com os quais conversamos sugerem que você antecipe as mudanças nos dados de origem e mantenha um esquema flexível. Isso significa que não há modelo de dados rígido no banco de dados analítico. Em vez disso, assuma que os sistemas de origem estão fornecendo os dados corretos com a definição e a lógica de negócios corretas, como existem hoje. E como o armazenamento é barato, armazene o streaming recente e os dados históricos salvos de forma que possam ser consultados juntos. Otimize para análises abrangentes em um conjunto de dados com um esquema flexível. Além disso, em vez de reagir a relatórios, por que não criar uma automação que responda a anomalias e alterações nos dados de streaming?

O mundo da modelagem de dados está mudando e acreditamos que uma mudança radical ocorrerá em breve nos paradigmas do modelo de dados. Essas novas abordagens provavelmente incorporarão métricas e camadas semânticas, pipelines de dados e fluxos de trabalho analíticos tradicionais em uma camada de streaming que fica diretamente sobre o sistema de origem. Como os dados estão sendo gerados em tempo real, a noção de separar artificialmente os sistemas de origem e análise em dois baldes distintos pode não fazer tanto sentido quanto quando os dados se movem de forma mais lenta e previsível. O tempo vai dizer...

Temos mais a dizer sobre o futuro do streaming de dados em [Capítulo 11](#).

Transformações

O resultado líquido da transformação de dados é a capacidade de unificar e integrar dados. Depois que os dados são transformados, eles podem ser vistos como uma única entidade. Mas sem transformar os dados, você não pode ter uma visão unificada dos dados em toda a organização.

— Bill Inmon¹³

Agora que abordamos consultas e modelagem de dados, você deve estar se perguntando: se posso modelar dados, consultá-los e obter resultados, por que preciso pensar em transformações? As transformações manipulam, aprimoram e salvam dados para uso downstream, aumentando seu valor de maneira escalável, confiável e econômica.

Imagine executar uma consulta sempre que quiser visualizar os resultados de um determinado conjunto de dados. Você executaria a mesma consulta dezenas ou centenas de vezes por dia. Imagine que essa consulta envolva análise, limpeza, junção, união e agregação em 20 conjuntos de dados. Para agravar ainda mais a dor, a consulta leva 30 minutos para ser executada, consome recursos significativos e incorre em cobranças substanciais de nuvem em várias repetições. Você e seus stakeholders provavelmente ficariam loucos. Felizmente, você pode *salve os resultados da sua consulta* em vez disso, ou pelo menos execute as partes de computação mais intensiva apenas uma vez, para que as consultas subsequentes sejam simplificadas.

Uma transformação difere de uma consulta. A consulta recupera os dados de várias fontes com base na filtragem e na lógica de junção. A transformação persiste os resultados para consumo por transformações ou consultas adicionais. Esses resultados podem ser armazenados de forma efêmera ou permanente.

Além da persistência, um segundo aspecto que diferencia transformações de consultas é a complexidade. Você provavelmente criará pipelines complexos que combinam dados de várias fontes e reutilizam resultados intermediários para várias saídas finais. Esses pipelines complexos podem normalizar, modelar, agregar ou caracterizar dados. Embora você possa criar fluxos de dados complexos em consultas únicas usando expressões de tabela comuns, scripts ou DAGs, isso rapidamente se torna pesado, inconsistente e intratável. Insira as transformações.

As transformações dependem criticamente de uma das principais correntes ocultas deste livro: a orquestração. A orquestração combina muitas operações discretas, como transformações intermediárias, que armazenam dados temporária ou permanentemente para consumo por transformações ou serviços downstream. Cada vez mais, os pipelines de transformação abrangem não apenas várias tabelas e conjuntos de dados, mas também vários sistemas.

13 Bill Inmon, "Evitando a Horrível Tarefa de Integração de Dados," LinkedIn Pulse, 24 de março de 2022, <https://oreil.ly/yLb71>.

Transformações em Lote

Transformações em lote é executado em blocos discretos de dados, em contraste com as transformações de streaming, em que os dados são processados continuamente à medida que chegam. As transformações em lote podem ser executadas em um cronograma fixo (por exemplo, diariamente, a cada hora ou a cada 15 minutos) para dar suporte a relatórios, análises e modelos de ML contínuos. Nesta seção, você aprenderá vários padrões e tecnologias de transformação em lote.

Junções distribuídas

A ideia básica por trás das junções distribuídas é que precisamos quebrar uma *junção lógica* (a junção definida pela lógica de consulta) em muito menor *junção de nó* que são executados em servidores individuais no cluster. Os padrões básicos de junção distribuída se aplicam se um estiver em MapReduce (discutido em “[MapReduce](#)” na página 322), BigQuery, Snowflake ou Spark, embora os detalhes do armazenamento intermediário entre as etapas de processamento variem (no disco ou na memória). Na melhor das hipóteses, os dados em um lado da junção são pequenos o suficiente para caber em um único nó (*junção de transmissão*). Frequentemente, um recurso mais intensivo *junção aleatória de hash* é necessário.

Associação de transmissão. A *junção de transmissão* é geralmente assimétrico, com uma grande tabela distribuída entre os nós e uma pequena tabela que pode caber facilmente em um único nó ([Figura 8-17](#)). O mecanismo de consulta “transmite” a tabela pequena (tabela A) para todos os nós, onde ela se une às partes da tabela grande (tabela B). As junções de transmissão são muito menos intensivas em computação do que as junções de hash aleatórias.

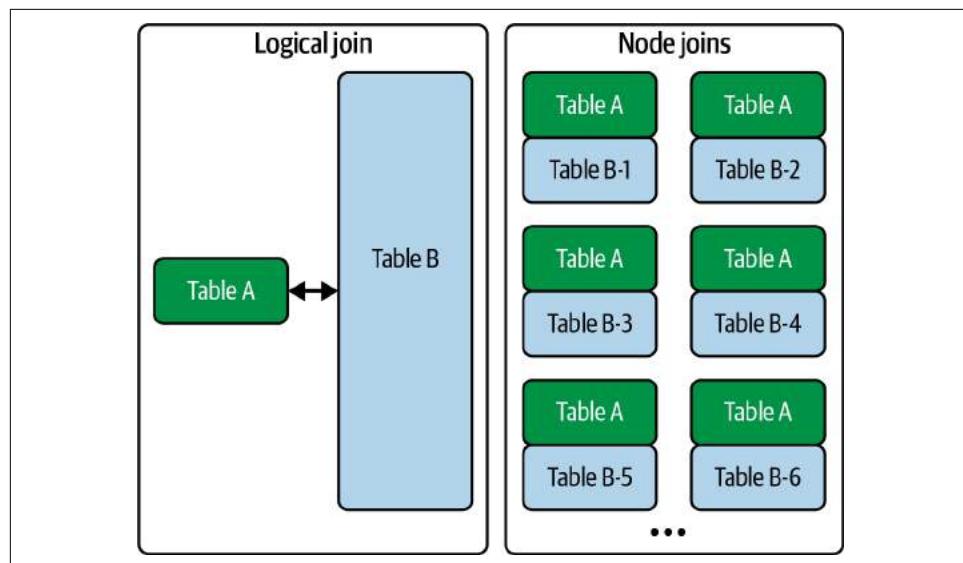


Figura 8-17. Em uma junção de transmissão, o mecanismo de consulta envia a tabela A para todos os nós no cluster para serem unidos com as várias partes da tabela B

Na prática, a tabela A geralmente é uma tabela maior filtrada que o mecanismo de consulta coleta e transmite. Uma das principais prioridades nos otimizadores de consulta é a reordenação de junção. Com a aplicação antecipada de filtros e o movimento de pequenas tabelas para a esquerda (para junções à esquerda), muitas vezes é possível reduzir drasticamente a quantidade de dados processados em cada junção. A pré-filtragem de dados para criar junções de transmissão sempre que possível pode melhorar drasticamente o desempenho e reduzir o consumo de recursos.

Junção de hash aleatória. Se nenhuma das tabelas for pequena o suficiente para caber em um único nó, o mecanismo de consulta usará uma *junção aleatória de hash*. Em [Figura 8-18](#), os mesmos nós são representados acima e abaixo da linha pontilhada. A área acima da linha pontilhada representa o particionamento inicial das tabelas A e B nos nós. Em geral, esse particionamento não terá relação com a chave de junção. Um esquema de hash é usado para reparticionar dados por chave de junção.

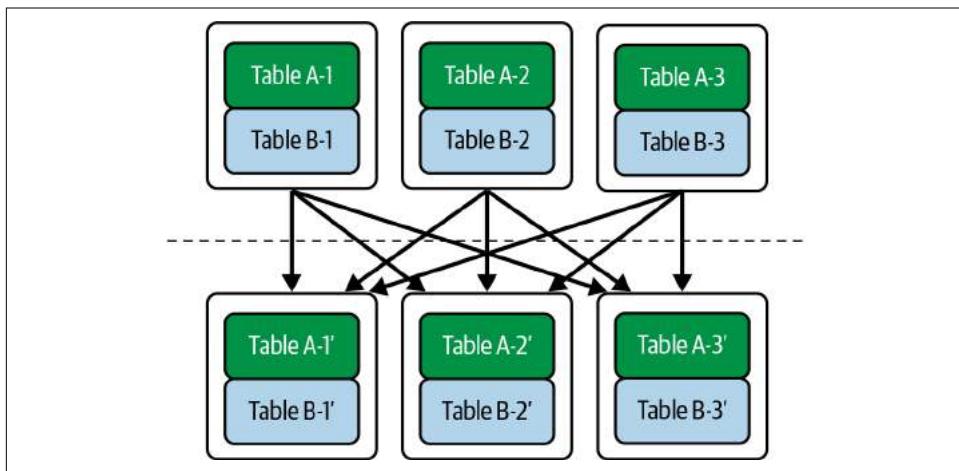


Figura 8-18. Junção aleatória de hash

Neste exemplo, o esquema de hash partitionará a chave de junção em três partes, com cada parte atribuída a um nó. Os dados são então reembalhados para o nó apropriado e as novas partições para as tabelas A e B em cada nó são unidas. As junções de hash aleatórias geralmente consomem mais recursos do que as junções de transmissão.

ETL, ELT e pipelines de dados

Como discutimos em [Capítulo 3](#), um padrão de transformação amplamente difundido que data dos primeiros dias dos bancos de dados relacionais é um ETL em lote. O ETL tradicional depende de um sistema de transformação externo para obter, transformar e limpar dados enquanto os prepara para um esquema de destino, como um data mart ou um esquema em estrela de Kimball. Os dados transformados seriam carregados em um sistema de destino, como um data warehouse, onde a análise de negócios poderia ser executada.

O próprio padrão ETL foi conduzido pelas limitações dos sistemas de origem e destino. A fase de extração tendia a ser um grande gargalo, com as restrições do RDBMS de origem limitando a taxa na qual os dados podiam ser extraídos. E a transformação foi realizada em um sistema dedicado porque o sistema de destino tinha recursos extremamente limitados em armazenamento e capacidade de CPU.

Uma evolução agora popular do ETL é o ELT. Como os sistemas de armazenamento de dados aumentaram em desempenho e capacidade de armazenamento, tornou-se comum simplesmente extrair dados brutos de um sistema de origem, importá-los para um armazenamento de dados com transformação mínima e, em seguida, limpá-los e transformá-los diretamente no sistema de armazenamento. (Veja nossa discussão sobre data warehouses em [Capítulo 3](#) para uma discussão mais detalhada sobre a diferença entre ETL e ELT.)

Uma segunda noção ligeiramente diferente de ELT foi popularizada com o surgimento dos data lakes. Nesta versão, os dados não são transformados no momento em que são carregados. De fato, grandes quantidades de dados podem ser carregadas sem nenhuma preparação e nenhum plano. A suposição é que a etapa de transformação acontecerá em algum momento futuro indeterminado. A ingestão de dados sem um plano é uma ótima receita para um pântano de dados. Como diz Inmon:¹⁴

Sempre fui um fã de ETL devido ao fato de que o ETL força você a transformar os dados antes de colocá-los em um formulário onde você possa trabalhar com eles. Mas algumas organizações querem simplesmente pegar os dados, colocá-los em um banco de dados e depois fazer a transformação... Já vi muitos casos em que a organização diz, oh, vamos apenas colocar os dados e transformá-los mais tarde. E adivinha? Seis meses depois, esses dados nunca foram tocados.

Também vimos que a linha entre ETL e ELT pode ficar um tanto confusa em um ambiente de data lakehouse. Com o armazenamento de objetos como camada base, não fica mais claro o que está no banco de dados e o que está fora do banco de dados. A ambigüidade é ainda mais exacerbada com o surgimento da federação de dados, virtualização e tabelas ao vivo. (Discutiremos esses tópicos mais adiante nesta seção.)

Cada vez mais, sentimos que os termos *ETL* e *ELT* deve ser aplicado apenas no nível micro (dentro de pipelines de transformação individuais) e não no nível macro (para descrever um padrão de transformação para toda a organização). As organizações não precisam mais padronizar em ETL ou ELT, mas podem, em vez disso, se concentrar na aplicação da técnica adequada caso a caso, à medida que criam pipelines de dados.

SQL e ferramentas de transformação baseadas em código

Nesta conjuntura, a distinção entre sistemas de transformação baseados em SQL e não baseados em SQL parece um tanto sintética. Desde a introdução do Hive no Hadoop

14 Alex Woodie, "Lakehouses Prevent Data Swamps, Bill Inmon Says," Datanami, 1º de junho de 2021, <https://oreil.ly/XMwWc>.

plataforma, o SQL tornou-se um cidadão de primeira classe no ecossistema de big data. Por exemplo, o Spark SQL foi um recurso inicial do Apache Spark. Estruturas de streaming prioritário, como Kafka, Flink e Beam, também oferecem suporte a SQL, com vários recursos e funcionalidades.

É mais apropriado pensar em ferramentas somente SQL versus aquelas que oferecem suporte a paradigmas de programação de uso geral mais poderosos. As ferramentas de transformação somente SQL abrangem uma ampla variedade de opções proprietárias e de software livre.

O SQL é declarativo... mas ainda pode criar fluxos de trabalho de dados complexos. Frequentemente ouvimos disperda porque “não é processual”. Isso é tecnicamente correto. SQL é uma linguagem declarativa: em vez de codificar um procedimento de processamento de dados, os escritores SQL estipulam as características de seus dados finais em linguagem teórica de conjuntos; o compilador e o otimizador SQL determinam as etapas necessárias para colocar os dados nesse estado.

Às vezes, as pessoas insinuam que, como o SQL não é processual, ele não pode criar pipelines complexos. Isto é falso. O SQL pode ser usado efetivamente para criar DAGs complexos usando expressões de tabela comuns, scripts SQL ou uma ferramenta de orquestração.

Para ser claro, o SQL tem limites, mas frequentemente vemos engenheiros fazendo coisas em Python e Spark que poderiam ser feitas com mais facilidade e eficiência em SQL. Para ter uma ideia melhor das vantagens e desvantagens de que estamos falando, vejamos alguns exemplos de Spark e SQL.

Exemplo: quando evitar SQL para transformações em lote no Spark. Quando você está determinando Se deseja usar o código nativo Spark ou PySpark em vez do Spark SQL ou outro mecanismo SQL, pergunte-se o seguinte:

1. Quão difícil é codificar a transformação em SQL?
2. Quão legível e sustentável será o código SQL resultante?
3. Parte do código de transformação deve ser enviado para uma biblioteca personalizada para reutilização futura em toda a organização?

Em relação à questão 1, muitas transformações codificadas no Spark podem ser realizadas em instruções SQL bastante simples. Por outro lado, se a transformação não for realizável em SQL, ou se for extremamente difícil de implementar, o Spark nativo é uma opção melhor. Por exemplo, podemos implementar a derivação de palavras em SQL colocando sufixos de palavras em uma tabela, juntando-se a essa tabela, usando uma função de análise para encontrar sufixos em palavras e, em seguida, reduzindo a palavra ao seu radical usando uma função de substring. No entanto, isso soa como um processo extremamente complexo com vários casos extremos a serem considerados. Uma linguagem de programação procedural mais poderosa se encaixa melhor aqui.

A questão 2 está intimamente relacionada. A consulta de derivação de palavras não será legível nem sustentável.

Em relação à questão 3, uma das maiores limitações do SQL é que ele não inclui uma noção natural de bibliotecas ou código reutilizável. Uma exceção é que alguns mecanismos SQL permitem manter funções definidas pelo usuário (UDFs) como objetos dentro de um banco de dados.¹⁵ No entanto, eles não estão comprometidos com um repositório Git sem um sistema CI/CD externo para gerenciar a implantação. Além disso, o SQL não tem uma boa noção de reutilização para componentes de consulta mais complexos. Obviamente, bibliotecas reutilizáveis são fáceis de criar no Spark e no PySpark.

Acrescentaremos que é possível reciclar o SQL de duas maneiras. Primeiro, podemos facilmente reutilizar o resultado de uma consulta SQL confirmado em uma tabela ou criando uma exibição. Geralmente, esse processo é melhor tratado em uma ferramenta de orquestração, como o Airflow, para que as consultas downstream possam ser iniciadas assim que a consulta de origem for concluída. Em segundo lugar, a Data Build Tool (dbt) facilita a reutilização de instruções SQL e oferece uma linguagem de modelo que facilita a personalização.

Exemplo: Otimizando o Spark e outras estruturas de processamento. Acólitos de faíscas frequentemente com-claro que o SQL não lhes dá controle sobre o processamento de dados. O mecanismo SQL pega suas instruções, as otimiza e as compila em suas etapas de processamento. (Na prática, a otimização pode acontecer antes ou depois da compilação, ou ambos.)

Esta é uma reclamação justa, mas existe um corolário. Com o Spark e outras estruturas de processamento de código pesado, o criador do código torna-se responsável por grande parte da otimização que é tratada automaticamente em um mecanismo baseado em SQL. A API do Spark é poderosa e complexa, o que significa que não é tão fácil identificar candidatos para reordenação, combinação ou decomposição. Ao adotar o Spark, as equipes de engenharia de dados precisam se envolver ativamente com os problemas de otimização do Spark, especialmente para trabalhos caros e de longa duração. Isso significa criar experiência em otimização na equipe e ensinar engenheiros individuais a otimizar.

Algumas coisas de nível superior a serem lembradas ao codificar no Spark nativo:

1. Filtre cedo e frequentemente.
2. Confie fortemente na API principal do Spark e aprenda a entender a maneira nativa do Spark de fazer as coisas. Tente contar com bibliotecas públicas bem mantidas se a API nativa do Spark não for compatível com seu caso de uso. Um bom código Spark é substancialmente declarativo.
3. Tenha cuidado com UDFs.
4. Considere misturar SQL.

A recomendação 1 também se aplica à otimização do SQL, com a diferença de que o Spark pode não ser capaz de reordenar algo que o SQL trataria para você

¹⁵ Lembramos que você deve usar UDFs com responsabilidade. As UDFs SQL costumam ter um desempenho razoavelmente bom. Nós vimos JavaScript As UDFs aumentam o tempo de consulta de alguns minutos para várias horas.

automaticamente. O Spark é uma estrutura de processamento de big data, mas quanto menos dados você tiver para processar, menos pesado em recursos e mais eficiente será seu código.

Se você estiver escrevendo um código personalizado extremamente complexo, faça uma pausa e determine se há uma maneira mais nativa de fazer o que você está tentando realizar. Aprenda a entender o Spark idiomático lendo exemplos públicos e trabalhando com tutoriais. Existe algo na API do Spark que pode realizar o que você está tentando fazer? Existe uma biblioteca pública bem conservada e otimizada que possa ajudar?

A terceira recomendação é crucial para o PySpark. Em geral, PySpark é um wrapper de API para Scala Spark. Seu código envia o trabalho para o código Scala nativo em execução na JVM chamando a API. A execução de UDFs do Python força os dados a serem passados para o Python, onde o processamento é menos eficiente. Se você estiver usando UDFs do Python, procure uma maneira mais nativa do Spark de realizar o que está fazendo. Volte para a recomendação: existe uma maneira de realizar sua tarefa usando a API principal ou uma biblioteca bem mantida? Se você precisar usar UDFs, considere reescrevê-los em Scala ou Java para melhorar o desempenho.

Quanto à recomendação 4, o uso do SQL nos permite aproveitar o otimizador Spark Catalyst, que pode ser capaz de extrair mais desempenho do que conseguimos com o código Spark nativo. O SQL geralmente é mais fácil de escrever e manter para operações simples. Combinar o Spark nativo e o SQL nos permite obter o melhor dos dois mundos: funcionalidade poderosa de uso geral combinada com simplicidade, quando aplicável.

Muitos dos conselhos de otimização nesta seção são bastante genéricos e também se aplicariam ao Apache Beam, por exemplo. O ponto principal é que as APIs de processamento de dados programáveis exigem um pouco mais de refinamento de otimização do que o SQL, que talvez seja menos poderoso e mais fácil de usar.

Atualizar padrões

Como as transformações persistem dados, frequentemente atualizaremos os dados persistentes no local. A atualização de dados é um grande ponto problemático para as equipes de engenharia de dados, especialmente durante a transição entre as tecnologias de engenharia de dados. Estamos discutindo DML em SQL, que apresentamos anteriormente no capítulo.

Mencionamos várias vezes ao longo do livro que o conceito original de data lake não levava em conta a atualização dos dados. Isso agora parece sem sentido por vários motivos. A atualização de dados tem sido uma parte fundamental da manipulação dos resultados da transformação de dados, mesmo que a comunidade de big data a tenha descartado. É bobagem reexecutar quantidades significativas de trabalho porque não temos recursos de atualização. Assim, o conceito de data lakehouse agora inclui atualizações. Além disso, o GDPR e outros padrões de exclusão de dados agora exigem que organizações excluam dados de maneira direcionada, mesmo em conjuntos de dados brutos.

Vamos considerar vários padrões básicos de atualização.

Truncar e recarregar. *Truncar* é um padrão de atualização que não atualiza nada. Ele simplesmente limpa os dados antigos. Em um padrão de atualização de truncar e recarregar, uma tabela é limpa de dados e as transformações são executadas novamente e carregadas nessa tabela, gerando efetivamente uma nova versão da tabela.

Insira apenas. *Inserir apenas* insere novos registros sem alterar ou excluir registros antigos. Padrões somente de inserção podem ser usados para manter uma visão atual dos dados, por exemplo, se novas versões de registros forem inseridas sem excluir registros antigos. Uma consulta ou exibição pode apresentar o estado atual dos dados localizando o registro mais recente por chave primária. Observe que os bancos de dados colunares normalmente não impõem chaves primárias. A chave primária seria uma construção usada pelos engenheiros para manter uma noção do estado atual da tabela. A desvantagem dessa abordagem é que pode ser extremamente custoso computacionalmente encontrar o registro mais recente no momento da consulta. Como alternativa, podemos usar uma visualização materializada (abordada posteriormente no capítulo), uma tabela somente de inserção que mantém todos os registros e uma tabela de destino truncada e recarregada que contém o estado atual para servir os dados.



Ao inserir dados em um banco de dados OLAP orientado a coluna, o problema comum é que os engenheiros que fazem a transição de sistemas orientados a linha tentam usar inserções de linha única. Esse antipadrão coloca uma carga enorme no sistema. Também faz com que os dados sejam gravados em muitos arquivos separados; isso é extremamente ineficiente para leituras subsequentes e os dados devem ser agrupados novamente posteriormente. Em vez disso, recomendamos o carregamento de dados em um microlote periódico ou em lote.

Mencionaremos uma exceção ao conselho de não inserir com frequência: a arquitetura Lambda aprimorada usada pelo BigQuery e Apache Druid, que hibridiza um buffer de streaming com armazenamento colunar. Exclusões e atualizações in-loco ainda podem ser caras, como discutiremos a seguir.

Excluir. A exclusão é crítica quando um sistema de origem exclui dados e atende a alterações regulatórias recentes. Em sistemas colunares e data lakes, as exclusões são mais caras que as inserções.

Ao excluir dados, considere se você precisa fazer uma exclusão definitiva ou temporária. A exclusão forçada remove permanentemente um registro de um banco de dados, enquanto uma exclusão reversível marca o registro como "excluído". As exclusões definitivas são úteis quando você precisa remover dados por motivos de desempenho (digamos, uma tabela é muito grande) ou se houver um motivo legal ou de conformidade para fazê-lo. As exclusões temporárias podem ser usadas quando você não deseja excluir um registro permanentemente, mas também deseja filtrá-lo dos resultados da consulta.

Uma terceira abordagem para exclusões está intimamente relacionada às exclusões reversíveis: *inserir exclusão* insere um novo registro com um `DELETE` flag sem modificar a versão anterior do registro. Isso nos permite seguir um padrão somente de inserção, mas ainda contabilizar exclusões. Apenas observe

que nossa consulta para obter o estado mais recente da tabela fica um pouco mais complicada. Agora devemos desduplicar, encontrar a versão mais recente de cada registro por chave e não mostrar nenhum registro cuja versão mais recente mostre excluído.

Upser/merge.Desses padrões de atualização, os padrões upsert e merge são os que consistentemente causam mais problemas para as equipes de engenharia de dados, especialmente para pessoas que estão fazendo a transição de data warehouses baseados em linhas para sistemas de nuvem baseados em colunas.

Upserting pega um conjunto de registros de origem e procura correspondências em uma tabela de destino usando uma chave primária ou outra condição lógica. (Novamente, é responsabilidade da equipe de engenharia de dados gerenciar essa chave primária executando as consultas apropriadas. A maioria dos sistemas colunares não impõe exclusividade.) Quando ocorre uma correspondência de chave, o registro de destino é atualizado (substituído pelo novo registro). Quando não houver correspondência, o banco de dados insere o novo registro. O padrão de mesclagem adiciona a isso a capacidade de excluir registros.

Então qual é o problema? O padrão upsert/merge foi originalmente projetado para bancos de dados baseados em linhas. Em bancos de dados baseados em linhas, as atualizações são um processo natural: o banco de dados procura o registro em questão e o altera no local.

Por outro lado, os sistemas baseados em arquivos não oferecem suporte a atualizações de arquivos no local. Todos esses sistemas utilizam copy on write (COW). Se um registro em um arquivo for alterado ou excluído, todo o arquivo deverá ser reescrito com as novas alterações.

Isso é parte do motivo pelo qual os primeiros usuários de big data e data lakes rejeitaram as atualizações: o gerenciamento de arquivos e atualizações parecia muito complicado. Então, eles simplesmente usaram um padrão somente de inserção e presumiram que os consumidores de dados determinariam o estado atual dos dados no momento da consulta ou nas transformações de downstream. Na realidade, os bancos de dados colunares, como o Vertica, há muito suportam atualizações no local, ocultando a complexidade do COW dos usuários. Eles examinam arquivos, alteram os registros relevantes, gravam novos arquivos e alteram os ponteiros de arquivo para a tabela. Os principais data warehouses em nuvem colunar oferecem suporte a atualizações e fusões, embora os engenheiros devam investigar o suporte de atualização se considerarem a adoção de uma tecnologia exótica.

Existem algumas coisas importantes para entender aqui. Embora os sistemas de dados colunares distribuídos suportem comandos de atualização nativos, as mesclagens têm um custo: o impacto no desempenho da atualização ou exclusão de um único registro pode ser bastante alto. Por outro lado, as fusões podem ser extremamente eficientes para grandes conjuntos de atualizações e podem até superar os bancos de dados transacionais.

Além disso, é importante entender que COW raramente envolve reescrever toda a tabela. Dependendo do sistema de banco de dados em questão, o COW pode operar em várias resoluções (partição, cluster, bloco). Para realizar atualizações de alto desempenho, concentre-se em

desenvolvendo uma estratégia apropriada de particionamento e agrupamento com base em suas necessidades e nas entranhas do banco de dados em questão.

Assim como nas inserções, tenha cuidado com a frequência de atualização ou mesclagem. Vimos muitas equipes de engenharia fazerem a transição entre os sistemas de banco de dados e tentarem executar mesclagens quase em tempo real a partir do CDC, exatamente como faziam em seu sistema antigo. Simplesmente não funciona. Não importa quão bom seja o seu sistema CDC, essa abordagem deixará a maioria dos data warehouses colunares de joelhos. Vimos sistemas ficarem semanas atrasados nas atualizações, onde uma abordagem que simplesmente se mesclasse a cada hora faria muito mais sentido.

Podemos usar várias abordagens para aproximar os bancos de dados colunares do tempo real. Por exemplo, o BigQuery nos permite inserir novos registros em uma tabela e, em seguida, oferece suporte a exibições materializadas especializadas que apresentam uma exibição de tabela desduplicada eficiente e quase em tempo real. O Druid usa armazenamento de dois níveis e SSDs para oferecer suporte a consultas ultrarrápidas em tempo real.

Atualizações de esquema

Os dados têm entropia e podem mudar sem o seu controle ou consentimento. As fontes de dados externas podem alterar seu esquema ou as equipes de desenvolvimento de aplicativos podem adicionar novos campos ao esquema. Uma vantagem dos sistemas colunares sobre os sistemas baseados em linhas é que, embora seja mais difícil atualizar os dados, atualizar o esquema é mais fácil. As colunas geralmente podem ser adicionadas, excluídas e renomeadas.

Apesar dessas melhorias tecnológicas, o gerenciamento prático do esquema organizacional é mais desafiador. Algumas atualizações de esquema serão automatizadas? (Essa é a abordagem que o Fivetran usa ao replicar de fontes.) Por mais conveniente que pareça, há um risco de que as transformações downstream sejam interrompidas.

Existe um processo simples de solicitação de atualização de esquema? Suponha que uma equipe de ciência de dados queira adicionar uma coluna de uma fonte que não foi ingerida anteriormente. Como será o processo de revisão? Os processos downstream serão interrompidos? (Existem consultas que executam SELECIONE *em vez de usar a seleção de coluna explícita? Geralmente, essa é uma prática ruim em bancos de dados colunares.) Quanto tempo levará para implementar a alteração? É possível criar uma bifurcação de tabela, ou seja, uma nova versão de tabela específica para este projeto?

Uma nova opção interessante surgiu para dados semiestruturados. Tomando emprestado uma ideia dos armazenamentos de documentos, muitos data warehouses em nuvem agora oferecem suporte a tipos de dados que codificam dados JSON arbitrários. Uma abordagem armazena JSON bruto em um campo enquanto armazena dados acessados com frequência em campos adjacentes nivelados. Isso ocupa espaço de armazenamento adicional, mas permite a conveniência de dados simplificados, com a flexibilidade de dados semiestruturados para usuários avançados. Os dados acessados com frequência no campo JSON podem ser adicionados diretamente ao esquema ao longo do tempo.

Essa abordagem funciona muito bem quando os engenheiros de dados devem ingerir dados de um armazenamento de documentos de aplicativo com um esquema que muda com frequência. Semi-estruturado

os dados disponíveis como um cidadão de primeira classe em data warehouses são extremamente flexíveis e abrem novas oportunidades para analistas e cientistas de dados, pois os dados não estão mais restritos a linhas e colunas.

Disputa de dados

Disputa de dados pega dados confusos e malformados e os transforma em dados limpos e úteis. Isso geralmente é um processo de transformação em lote.

A disputa de dados tem sido uma grande fonte de dor e segurança no trabalho para engenheiros de dados. Por exemplo, suponha que os desenvolvedores recebam dados EDI (consulte [Capítulo 7](#)) de uma empresa parceira em relação a transações e faturas, possivelmente uma mistura de dados estruturados e texto. O processo típico de disputar esses dados envolve primeiro tentar ingeri-los. Freqüentemente, os dados são tão malformados que envolve um bom pré-processamento de texto. Os desenvolvedores podem optar por ingerir os dados como uma única tabela de campo de texto — uma linha inteira ingerida como um único campo. Os desenvolvedores então começam a escrever consultas para analisar e separar os dados. Com o tempo, eles descobrem anomalias de dados e casos extremos. Eventualmente, eles obterão os dados em uma forma aproximada. Só então o processo de transformação a jusante pode começar.

As ferramentas de organização de dados visam simplificar partes significativas desse processo. Essas ferramentas muitas vezes desencorajam os engenheiros de dados porque afirmam não ter código, o que soa pouco sofisticado. Preferimos pensar em ferramentas de manipulação de dados como ambientes de desenvolvimento integrado (IDEs) para dados malformados. Na prática, os engenheiros de dados gastam muito tempo analisando dados desagradáveis; as ferramentas de automação permitem que os engenheiros de dados gastem tempo em tarefas mais interessantes. As ferramentas de Wrangling também podem permitir que os engenheiros entreguem alguns trabalhos de análise e ingestão aos analistas.

As ferramentas gráficas de transformação de dados normalmente apresentam uma amostra de dados em uma interface visual, com tipos inferidos, estatísticas incluindo distribuições, dados anômalos, outliers e nulos. Os usuários podem adicionar etapas de processamento para corrigir problemas de dados. Uma etapa pode fornecer instruções para lidar com dados digitados incorretamente, dividir um campo de texto em várias partes ou unir-se a uma tabela de pesquisa.

Os usuários podem executar as etapas em um conjunto de dados completo quando o trabalho completo estiver pronto. O trabalho normalmente é enviado para um sistema de processamento de dados escalonável, como o Spark, para grandes conjuntos de dados. Após a execução do trabalho, ele retornará erros e exceções não tratadas. O usuário pode refinar ainda mais a receita para lidar com esses valores discrepantes.

É altamente recomendável que engenheiros aspirantes e experientes experimentem ferramentas de disputa; os principais provedores de nuvem vendem sua versão de ferramentas de manipulação de dados e muitas opções de terceiros estão disponíveis. Os engenheiros de dados podem descobrir que essas ferramentas simplificam significativamente certas partes de seus trabalhos. Organizacionalmente, as equipes de engenharia de dados podem querer considerar o treinamento de especialistas em disputa de dados se eles frequentemente ingerem de fontes de dados novas e confusas.

Exemplo: transformação de dados no Spark

Vejamos um exemplo prático e concreto de transformação de dados. Suponha que construímos um pipeline que ingere dados de três fontes de API no formato JSON. Essa etapa inicial de ingestão é tratada no Airflow. Cada fonte de dados obtém seu prefixo (filepath) em um bucket do S3.

O Airflow aciona um trabalho do Spark chamando uma API. Este trabalho do Spark ingere cada uma das três fontes em um dataframe, convertendo os dados em um formato relacional, com aninhamento em determinadas colunas. O trabalho do Spark combina as três fontes em uma única tabela e, em seguida, filtra os resultados com uma instrução SQL. Os resultados são finalmente gravados em uma tabela Delta Lake formatada em parquet armazenada no S3.

Na prática, o Spark cria um DAG de etapas com base no código que escrevemos para ingerir, unir e gravar os dados. A ingestão básica de dados ocorre na memória do cluster, embora uma das origens de dados seja grande o suficiente para ser despejada no disco durante o processo de ingestão. (Esses dados são gravados no armazenamento do cluster; eles serão recarregados na memória para as etapas de processamento subsequentes.)

A junção requer uma operação aleatória. Uma chave é usada para redistribuir dados no cluster; mais uma vez, ocorre um vazamento no disco conforme os dados são gravados em cada nó. A transformação SQL filtra as linhas na memória e descarta as linhas não utilizadas. Por fim, o Spark converte os dados no formato Parquet, os compacta e os grava de volta no S3. O Airflow chama periodicamente o Spark para ver se o trabalho foi concluído. Depois de confirmar que o trabalho foi concluído, ele marca o Airflow DAG completo como concluído. (Observe que temos duas construções de DAG aqui, um Airflow DAG e um DAG específico para o trabalho do Spark.)

Lógica de negócios e dados derivados

Um dos casos de uso mais comuns para transformação é renderizar a lógica de negócios. Colocamos essa discussão em transformações em lote porque é onde esse tipo de transformação ocorre com mais frequência, mas observe que também pode ocorrer em um pipeline de streaming.

Suponha que uma empresa use vários cálculos de lucro internos especializados. Uma versão pode olhar para os lucros antes dos custos de marketing, e outra pode olhar para o lucro depois de subtrair os custos de marketing. Mesmo que pareça ser um exercício de contabilidade simples, cada uma dessas métricas é altamente complexa de processar.

O lucro antes dos custos de marketing pode precisar contabilizar pedidos fraudulentos; determinar uma estimativa de lucro razoável para o dia útil anterior implica estimar qual porcentagem de receita e lucro será perdida para pedidos cancelados nos próximos dias, conforme a equipe de fraude investiga pedidos suspeitos. Existe algum sinalizador especial no banco de dados que indique um pedido com alta probabilidade de fraude ou que tenha

foi cancelado automaticamente? A empresa assume que uma determinada porcentagem de pedidos será cancelada devido a fraude antes mesmo que o processo de avaliação de risco de fraude seja concluído para pedidos específicos?

Para lucros após custos de marketing, devemos contabilizar todas as complexidades da métrica anterior, mais os custos de marketing atribuídos ao pedido específico. A empresa tem um modelo de atribuição ingênuo - por exemplo, custos de marketing atribuídos a itens ponderados pelo preço? Os custos de marketing também podem ser atribuídos por departamento ou categoria de item ou, nas organizações mais sofisticadas, por item individual com base nos cliques de anúncios do usuário.

A transformação da lógica de negócios que gera essa versão diferenciada do lucro deve integrar todas as sutilezas da atribuição — ou seja, um modelo que vincule pedidos a anúncios específicos e custos de publicidade. Os dados de atribuição são armazenados nas entradas dos scripts ETL ou são extraídos de uma tabela gerada automaticamente a partir dos dados da plataforma de anúncios?

Este tipo de dados de relatório é um exemplo por excelência de *dados derivados*—dados calculados a partir de outros dados armazenados em um sistema de dados. Os críticos de dados derivados apontarão que é um desafio para o ETL manter a consistência nas métricas derivadas.¹⁶ Por exemplo, se a empresa atualizar seu modelo de atribuição, essa alteração pode precisar ser mesclada em muitos scripts ETL para geração de relatórios. (Os scripts ETL são notórios por quebrar o princípio DRY.) A atualização desses scripts ETL é um processo manual e trabalhoso, envolvendo conhecimento de domínio em lógica de processamento e alterações anteriores. Os scripts atualizados também devem ser validados quanto à consistência e precisão.

De nossa perspectiva, essas são críticas legítimas, mas não necessariamente muito construtivas, porque a alternativa aos dados derivados nesse caso é igualmente desagradável. Os analistas precisarão executar suas consultas de relatórios se os dados de lucro não estiverem armazenados no data warehouse, incluindo a lógica de lucro. A atualização de scripts ETL complexos para representar com precisão as alterações na lógica de negócios é uma tarefa trabalhosa e esmagadora, mas fazer com que os analistas atualizem suas consultas de relatórios de forma consistente é quase impossível.

Uma alternativa interessante é colocar a lógica de negócios em um *camada de métricas*,¹⁷ mas ainda aproveitar o data warehouse ou outra ferramenta para fazer o trabalho pesado computacional. Uma camada de métricas codifica a lógica de negócios e permite que analistas e usuários de painéis criem análises complexas a partir de uma biblioteca de métricas definidas. A camada de métricas gera consultas a partir das métricas e as envia para o banco de dados. Discutimos camadas semânticas e métricas com mais detalhes em [Capítulo 9](#).

16 Michael Blaha, "Tenha cuidado com dados derivados", Datavarsity, 5 de dezembro de 2016, <https://oreil.ly/garol>.

17 Benn Stancil, "A peça que faltava na moderna pilha de dados," *benn.substack*, 22 de abril de 2021, <https://oreil.ly/GYf3Z>.

MapReduce

Nenhuma discussão sobre transformação em lote pode ser concluída sem tocar em Map-Reduce. Isso não ocorre porque o MapReduce é amplamente usado por engenheiros de dados atualmente. MapReduce foi o padrão de transformação de dados em lote definidor da era do big data, ainda influencia muitos sistemas distribuídos que os engenheiros de dados usam hoje e é útil para os engenheiros de dados entenderem em um nível básico. MapReduce foi introduzido pelo Google em uma continuação de seu artigo sobre GFS. Foi inicialmente o padrão de processamento de fato do Hadoop, a tecnologia analógica de código aberto do GFS que introduzimos em [Capítulo 6](#).

Um trabalho MapReduce simples consiste em uma coleção de tarefas de mapa que leem blocos de dados individuais espalhados pelos nós, seguidos por um embaralhamento que redistribui os dados resultantes no cluster e uma etapa de redução que agrupa dados em cada nó. Por exemplo, suponha que queremos executar a seguinte consulta SQL:

```
SELEÇÃO A CONTAGEM(*),ID do
usuário DEuser_events
GRUPO POR ID do usuário;
```

Os dados da tabela são distribuídos entre nós em blocos de dados; o trabalho MapReduce gera uma tarefa de mapa por bloco. Cada tarefa de mapa essencialmente executa a consulta em um único bloco — ou seja, gera uma contagem para cada ID de usuário que aparece no bloco. Enquanto um bloco pode conter centenas de megabytes, a tabela completa pode ter petabytes de tamanho. No entanto, a parte do mapa do trabalho é um exemplo quase perfeito de paralelismo embarracoso; a taxa de varredura de dados em todo o cluster basicamente escala linearmente com o número de nós.

Em seguida, precisamos agregar (reduzir) para coletar resultados do cluster completo. Não estamos reunindo resultados em um único nó; em vez disso, redistribuímos os resultados por chave para que cada chave termine em um e apenas um nó. Esta é a etapa de embaralhamento, que geralmente é executada usando um algoritmo de hash nas chaves. Uma vez que os resultados do mapa foram embaralhados, somamos os resultados para cada chave. Os pares chave/contagem podem ser gravados no disco local no nó em que são calculados. Coletamos os resultados armazenados nos nós para visualizar os resultados completos da consulta.

Tarefas de MapReduce do mundo real podem ser muito mais complexas do que as que descrevemos aqui. Uma consulta complexa que filtra com um ONDEA cláusula une três tabelas e aplica uma função de janela que consistiria em vários estágios de mapa e redução.

Após MapReduce

O modelo MapReduce original do Google é extremamente poderoso, mas agora é visto como excessivamente rígido. Ele utiliza várias tarefas efêmeras de curta duração que leem e gravam no disco. Em particular, nenhum estado intermediário é preservado na memória; todos os dados são transferidos entre as tarefas, armazenando-os em disco ou enviando-os pela rede. Esse

simplifica o gerenciamento de estado e fluxo de trabalho e minimiza o consumo de memória, mas também pode direcionar a utilização de largura de banda de disco alto e aumentar o tempo de processamento.

O paradigma MapReduce foi construído em torno da ideia de que a capacidade e a largura de banda do disco magnético eram tão baratas que fazia sentido simplesmente jogar uma grande quantidade de disco nos dados para realizar consultas ultrarrápidas. Isso funcionou até certo ponto; Map- Reduz os registros de processamento de dados configurados repetidamente durante os primeiros dias do Hadoop.

No entanto, vivemos em um mundo pós-MapReduce por algum tempo. O processamento pós-MapReduce não descarta verdadeiramente o MapReduce; ele ainda inclui os elementos map, shuffle e reduce, mas relaxa as restrições de MapReduce para permitir o armazenamento em cache na memória.¹⁸ Lembre-se de que a RAM é muito mais rápida que o SSD e os HDDs em velocidade de transferência e tempo de busca. A persistência de até mesmo uma pequena quantidade de dados criteriosamente escolhidos na memória pode acelerar dramaticamente tarefas específicas de processamento de dados e esmagar totalmente o desempenho do MapReduce.

Por exemplo, Spark, BigQuery e várias outras estruturas de processamento de dados foram projetadas com base no processamento na memória. Essas estruturas tratam os dados como um conjunto distribuído que reside na memória. Se os dados ultrapassarem a memória disponível, isso causará um *derrame para o disco*. O disco é tratado como uma camada de armazenamento de dados de segunda classe para processamento, embora ainda seja altamente valioso.

A nuvem é um dos impulsionadores para a adoção mais ampla do cache de memória; é muito mais eficaz alugar memória durante um trabalho de processamento específico do que possuí-la 24 horas por dia. Os avanços no aproveitamento da memória para transformações continuarão a gerar ganhos no futuro previsível.

Visualizações materializadas, federação e virtualização de consultas

Nesta seção, veremos várias técnicas que virtualizam os resultados da consulta, apresentando-os como objetos semelhantes a tabelas. Essas técnicas podem se tornar parte de um pipeline de transformação ou ficar logo antes do consumo de dados do usuário final.

Visualizações

Primeiro, vamos revisar as exibições para definir o cenário para as exibições materializadas. A visualizaré um objeto de banco de dados que podemos selecionar como qualquer outra tabela. Na prática, uma visão é apenas uma consulta que referencia outras tabelas. Quando selecionamos em uma visualização, esse banco de dados cria uma nova consulta que combina a subconsulta da visualização com a nossa consulta. O otimizador de consulta otimiza e executa a consulta completa.

¹⁸ "Qual é a diferença entre o Apache Spark e o Hadoop MapReduce?", Knowledge Powerhouse YouTube vídeo, 20 de maio de 2017, <https://oreil.ly/WN0eX>.

As visualizações desempenham várias funções em um banco de dados. Primeiro, as exibições podem servir a uma função de segurança. Por exemplo, as exibições podem selecionar apenas colunas específicas e filtrar linhas, fornecendo assim acesso restrito aos dados. Várias exibições podem ser criadas para funções de trabalho, dependendo do acesso aos dados do usuário.

Em segundo lugar, uma exibição pode ser usada para fornecer uma imagem desduplicada atual dos dados. Se estivermos usando um padrão somente de inserção, uma exibição pode ser usada para retornar uma versão desduplicada de uma tabela mostrando apenas a versão mais recente de cada registro.

Em terceiro lugar, as exibições podem ser usadas para apresentar padrões comuns de acesso a dados. Suponha que os analistas de marketing devam executar com frequência uma consulta que une cinco tabelas. Poderíamos criar uma visão que unisse essas cinco tabelas em uma tabela ampla. Os analistas podem então escrever consultas que filtram e agregam sobre esta visualização.

Visualizações materializadas

Mencionamos visualizações materializadas em nossa discussão anterior sobre cache de consulta. Uma desvantagem potencial das visões (não materializadas) é que elas não fazem nenhum pré-computação. No exemplo de uma visualização que une cinco tabelas, essa junção deve ser executada toda vez que um analista de marketing executa uma consulta nessa visualização, e a junção pode ser extremamente cara.

Uma visualização materializada faz alguns ou todos os cálculos de visualização com antecedência. Em nosso exemplo, uma visualização materializada pode salvar os resultados de cinco junções de tabelas toda vez que ocorrer uma alteração nas tabelas de origem. Então, quando um usuário faz referência à exibição, ele está consultando os dados pré-unidos. Uma visualização materializada é uma etapa de transformação de fato, mas o banco de dados gerencia a execução por conveniência.

As exibições materializadas também podem servir a uma função significativa de otimização de consulta, dependendo do banco de dados, mesmo para consultas que não as referenciam diretamente. Muitos otimizadores de consulta podem identificar consultas que “se parecem” com uma visualização materializada. Um analista pode executar uma consulta que usa um filtro que aparece em uma visualização materializada. O otimizador reescreverá a consulta para selecionar a partir dos resultados pré-computados.

Visualizações materializadas que podem ser compostas

Em geral, visualizações materializadas não permitem composição — ou seja, uma visualização materializada não pode selecionar outra visualização materializada. No entanto, vimos recentemente o surgimento de ferramentas que oferecem suporte a esse recurso. Por exemplo, Databricks introduziu a noção de *mesas ao vivo*. Cada tabela é atualizada à medida que os dados chegam das fontes. Os dados fluem para as tabelas subsequentes de forma assíncrona.

Consultas federadas

Consultas federadas são um recurso de banco de dados que permite que um banco de dados OLAP selecione uma fonte de dados externa, como armazenamento de objeto ou RDBMS. Por exemplo, digamos

você precisa combinar dados no armazenamento de objetos e várias tabelas nos bancos de dados MySQL e PostgreSQL. Seu data warehouse pode emitir uma consulta federada para essas fontes e retornar os resultados combinados ([Figura 8-19](#)).

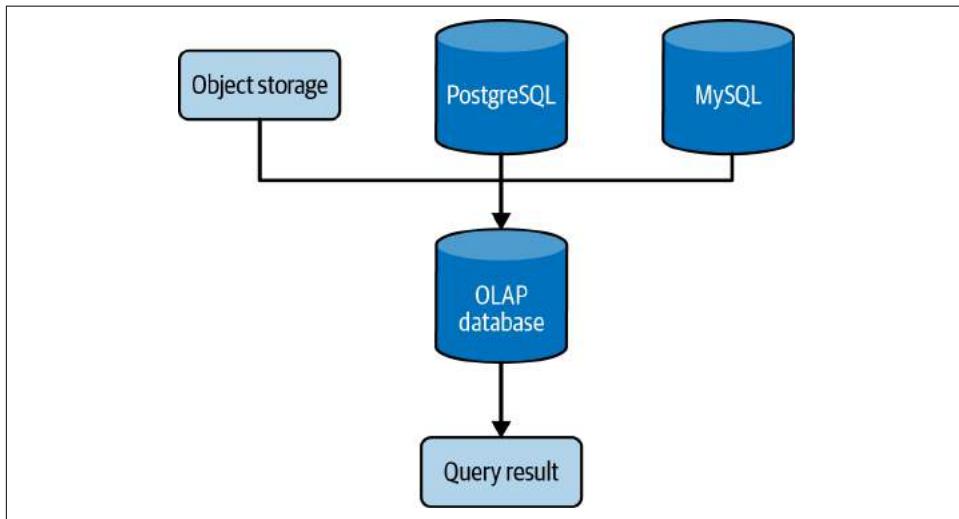


Figura 8-19. Um banco de dados OLAP emite uma consulta federada que obtém dados do armazenamento de objetos, MySQL e PostgreSQL e retorna um resultado de consulta com os dados combinados

Como outro exemplo, o Snowflake oferece suporte à noção de tabelas externas definidas nos buckets do S3. Um local de dados externo e um formato de arquivo são definidos ao criar a tabela, mas os dados ainda não são inseridos na tabela. Quando a tabela externa é consultada, o Snowflake lê do S3 e processa os dados com base nos parâmetros definidos no momento da criação da tabela. Podemos até juntar dados do S3 a tabelas de banco de dados internas. Isso torna o Snowflake e bancos de dados semelhantes mais compatíveis com um ambiente de data lake.

Alguns sistemas OLAP podem converter consultas federadas em visualizações materializadas. Isso nos dá muito do desempenho de uma tabela nativa sem a necessidade de ingerir dados manualmente toda vez que a fonte externa muda. A visualização materializada é atualizada sempre que os dados externos são alterados.

virtualização de dados

virtualização de dados está intimamente relacionado a consultas federadas, mas isso geralmente envolve um sistema de processamento e consulta de dados que não armazena dados internamente. Neste momento, Trino (por exemplo, Starburst) e Presto são exemplos por excelência. Qualquer mecanismo de consulta/processamento que suporte tabelas externas pode servir como um mecanismo de virtualização de dados. As considerações mais importantes com a virtualização de dados são as fontes externas suportadas e o desempenho.

Um conceito intimamente relacionado é a noção de *pushdown de consulta*. Suponha que eu queira consultar dados do Snowflake, juntar dados de um banco de dados MySQL e filtrar os resultados. O pushdown de consulta visa mover o máximo de trabalho possível para os bancos de dados de origem. O mecanismo pode procurar maneiras de enviar predicados de filtragem para as consultas nos sistemas de origem. Isso serve a dois propósitos: primeiro, descarrega a computação da camada de virtualização, aproveitando o desempenho de consulta da fonte. Em segundo lugar, reduz potencialmente a quantidade de dados que devem ser enviados pela rede, um gargalo crítico para o desempenho da virtualização.

A virtualização de dados é uma boa solução para organizações com dados armazenados em várias fontes de dados. No entanto, a virtualização de dados não deve ser usada aleatoriamente. Por exemplo, virtualizar um banco de dados MySQL de produção não resolve o problema principal das consultas analíticas que afetam adversamente o sistema de produção - como o Trino não armazena dados internamente, ele extrai do MySQL toda vez que executa uma consulta.

Como alternativa, a virtualização de dados pode ser usada como um componente de ingestão de dados e pipelines de processamento. Por exemplo, Trino pode ser usado para selecionar a partir do MySQL uma vez por dia à meia-noite, quando a carga no sistema de produção é baixa. Os resultados podem ser salvos no S3 para consumo por transformações downstream e consultas diárias, protegendo o MySQL de consultas analíticas diretas.

A virtualização de dados pode ser vista como uma ferramenta que expande o data lake para muito mais fontes, abstraindo as barreiras usadas para isolar dados entre unidades organizacionais. Uma organização pode armazenar dados transformados e acessados com frequência no S3 e virtualizar o acesso entre várias partes da empresa. Isso se encaixa perfeitamente com a noção de *malha de dados* discutido em [Capítulo 3](#), em que equipes pequenas são responsáveis por preparar seus dados para análise e compartilhá-los com o restante da empresa; a virtualização pode servir como uma camada de acesso crítico para compartilhamento prático.

Transformações e processamento de streaming

Já discutimos o processamento de fluxo no contexto de consultas. A diferença entre transformações de streaming e consultas de streaming é sutil e merece mais explicações.

Fundamentos

As consultas de streaming são executadas dinamicamente para apresentar uma visão atual dos dados, conforme discutido anteriormente. *Transformações de streaming* visam preparar dados para consumo a jusante.

Por exemplo, uma equipe de engenharia de dados pode ter um fluxo de entrada transportando eventos de uma fonte de IoT. Esses eventos de IoT carregam um ID de dispositivo e dados de evento. Desejamos enriquecer dinamicamente esses eventos com outros metadados do dispositivo, que são armazenados em um banco de dados separado. O mecanismo de processamento de fluxo consulta um banco de dados separado contendo

esses metadados por ID do dispositivo, geram novos eventos com os dados adicionados e os transmitem para outro fluxo. Consultas ao vivo e métricas acionadas são executadas nesse fluxo enriquecido (consulte [Figura 8-20](#)).

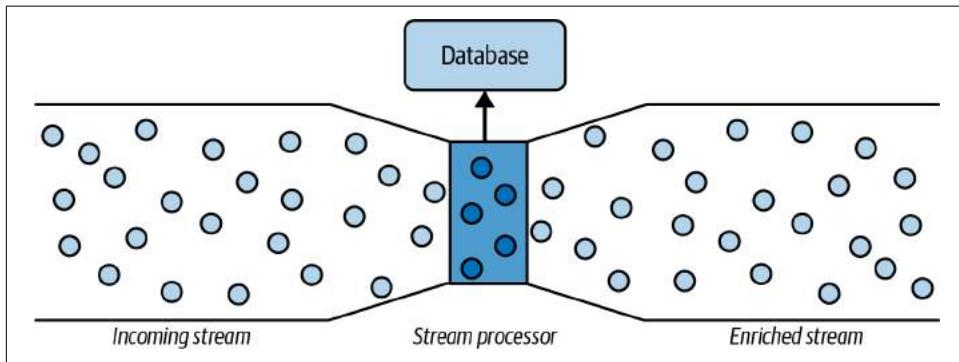


Figura 8-20. Um fluxo de entrada é transportado por uma plataforma de eventos de streaming e passado para um processador de fluxo

Transformações e consultas são contínuas

A linha entre transformações e consultas também é tênue no processamento em lote, mas as diferenças se tornam ainda mais sutis no domínio do streaming. Por exemplo, se calcularmos dinamicamente estatísticas de roll-up em janelas e, em seguida, enviarmos a saída para um fluxo de destino, isso é uma transformação ou uma consulta?

Talvez eventualmente adotemos uma nova terminologia para processamento de fluxo que represente melhor os casos de uso do mundo real. Por enquanto, faremos o possível com a terminologia que temos.

DAGs de streaming

Uma noção interessante intimamente relacionada ao enriquecimento de fluxo e junções é a *transmissão DAG*.¹⁹ Falamos sobre essa ideia pela primeira vez em nossa discussão sobre orquestração em [Capítulo 2](#). A orquestração é inherentemente um conceito de lote, mas se quiséssemos enriquecer, mesclar e dividir vários fluxos em tempo real?

Vejamos um exemplo simples em que o streaming de DAG seria útil. Suponha que queremos combinar dados de clickstream do site com dados de IoT. Isso nos permitirá obter uma visão unificada da atividade do usuário combinando eventos de IoT com cliques. Além disso, cada fluxo de dados precisa ser pré-processado em um formato padrão (consulte [Figura 8-21](#)).

19 Para obter uma aplicação detalhada do conceito de um DAG de streaming, consulte "Por que mudamos do Apache Kafka para Apache Pulsar" por Simba Khadder, blog StreamNative, 21 de abril de 2020, <https://oreil.ly/Rxfko>.

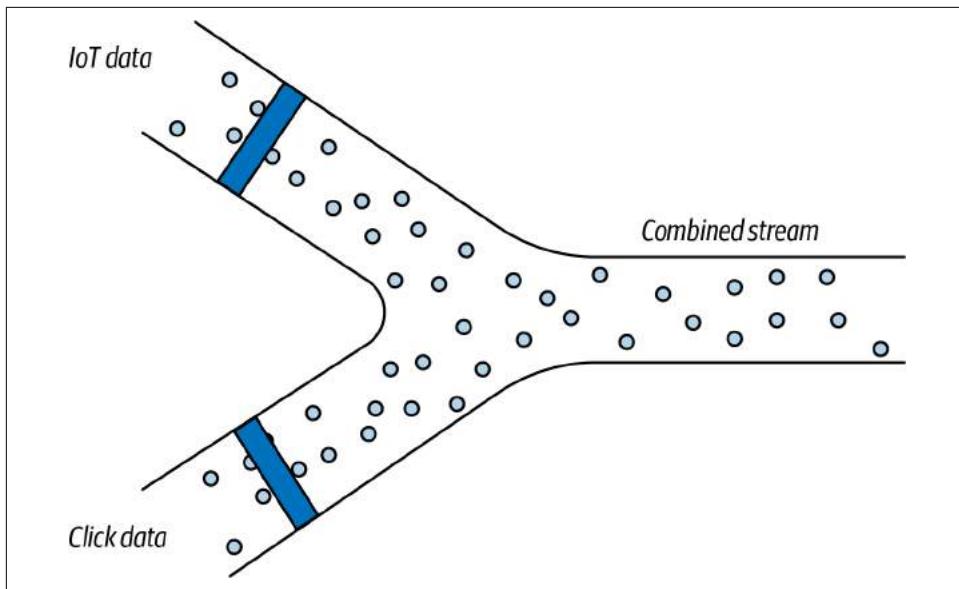


Figura 8-21. Um DAG de streaming simples

Há muito tempo isso é possível combinando uma loja de streaming (por exemplo, Kafka) com um processador de stream (por exemplo, Flink). A criação do DAG equivalia à construção de uma complexa máquina de Rube Goldberg, com vários tópicos e trabalhos de processamento conectados.

O Pulsar simplifica drasticamente esse processo, tratando os DAGs como uma abstração de streaming principal. Em vez de gerenciar fluxos em vários sistemas, os engenheiros podem definir seus DAGs de streaming como código dentro de um único sistema.

Microlote versus streaming verdadeiro

Uma longa batalha está em andamento entre as abordagens de micro-lote e streaming verdadeiro. Fundamentalmente, é importante entender seu caso de uso, os requisitos de desempenho e os recursos de desempenho da estrutura em questão.

O microlote é uma maneira de pegar uma estrutura orientada a lote e aplicá-la em uma situação de streaming. Um microlote pode ser executado a cada dois minutos a cada segundo. Algumas estruturas de microlote (por exemplo, Apache Spark Streaming) são projetadas para esse caso de uso e terão um bom desempenho com recursos alocados adequadamente em uma alta frequência de lote. (Na verdade, DBAs e engenheiros há muito usam microlote com bancos de dados mais tradicionais; isso geralmente leva a desempenho e consumo de recursos horríveis.)

Os verdadeiros sistemas de streaming (por exemplo, Beam e Flink) são projetados para processar um evento por vez. No entanto, isso vem com uma sobrecarga significativa. Além disso, é importante observar que, mesmo nesses verdadeiros sistemas de streaming, muitos processos ainda ocorrerão em lotes. A

processo de enriquecimento básico que adiciona dados a eventos individuais pode entregar um evento por vez com baixa latência. No entanto, uma métrica acionada no Windows pode ser executada a cada poucos segundos, a cada poucos minutos, etc.

Quando você está usando janelas e gatilhos (portanto, processamento em lote), qual é a frequência da janela? Qual é a latência aceitável? Se você estiver coletando métricas de vendas da Black Friday publicadas a cada poucos minutos, os micro-lotes provavelmente funcionarão, desde que você defina uma frequência de micro-lote apropriada. Por outro lado, se sua equipe de operações estiver computando métricas a cada segundo para detectar ataques DDoS, o verdadeiro streaming pode ser adequado.

Quando você deve usar um sobre o outro? Francamente, não há uma resposta universal. O termo *microlotetem* sido frequentemente usado para descartar tecnologias concorrentes, mas pode funcionar muito bem para o seu caso de uso e pode ser superior em muitos aspectos, dependendo de suas necessidades. Se sua equipe já tiver experiência em Spark, você poderá criar uma solução de streaming Spark (*microlote*) extremamente rápida.

Não há substituto para experiência de domínio e testes do mundo real. converse com especialistas que podem apresentar uma opinião imparcial. Você também pode testar facilmente as alternativas ativando testes na infraestrutura de nuvem. Além disso, fique atento a benchmarks espúrios fornecidos pelos fornecedores. Os fornecedores são notórios por selecionar benchmarks e criar exemplos artificiais que não correspondem à realidade (lembre-se de nossa conversa sobre benchmarks em [Capítulo 4](#)). Freqüentemente, os fornecedores mostram grandes vantagens em seus resultados de benchmark, mas não conseguem entregar no mundo real para o seu caso de uso.

Com quem você trabalhará

Consultas, transformações e modelagem impactam todas as partes interessadas em todo o ciclo de vida da engenharia de dados. O engenheiro de dados é responsável por várias coisas nesta fase do ciclo de vida. Do ponto de vista técnico, o engenheiro de dados projeta, constrói e mantém a integridade dos sistemas que consultam e transformam dados. O engenheiro de dados também implementa modelos de dados dentro deste sistema. Esta é a etapa mais “full-contact” onde seu foco é agregar o máximo de valor possível, tanto em termos de sistemas em funcionamento quanto em dados confiáveis e confiáveis.

Partes interessadas a montante

Quando se trata de transformações, as partes interessadas upstream podem ser divididas em duas grandes categorias: aquelas que controlam as definições de negócios e aquelas que controlam os sistemas que geram dados.

Ao interagir com as partes interessadas upstream sobre definições e lógica de negócios, você precisará conhecer as fontes de dados — o que são, como são usadas e a lógica e as definições de negócios envolvidas. Você trabalhará com os engenheiros responsáveis por esses sistemas de origem e as partes interessadas nos negócios que supervisionam os sistemas complementares

produtos e aplicativos. Um engenheiro de dados pode trabalhar junto com "o negócio" e as partes interessadas técnicas em um modelo de dados.

O engenheiro de dados precisa estar envolvido no projeto do modelo de dados e nas atualizações posteriores devido a mudanças na lógica de negócios ou novos processos. As transformações são bastante fáceis de fazer; basta escrever uma consulta e colocar os resultados em uma tabela ou visualização. Criá-los para que tenham bom desempenho e sejam valiosos para os negócios é outra questão. Sempre tenha em mente os requisitos e as expectativas dos negócios ao transformar dados.

As partes interessadas dos sistemas upstream querem ter certeza de que suas consultas e transformações impactam minimamente seus sistemas. Garanta a comunicação bidirecional sobre alterações nos modelos de dados (alterações de coluna e índice, por exemplo) nos sistemas de origem, pois podem afetar diretamente consultas, transformações e modelos de dados analíticos. Os engenheiros de dados devem saber sobre alterações de esquema, incluindo adição ou exclusão de campos, alterações de tipo de dados e qualquer outra coisa que possa afetar materialmente a capacidade de consultar e transformar dados.

Partes interessadas a jusante

As transformações são onde os dados começam a fornecer utilidade para as partes interessadas a jusante. Suas partes interessadas downstream incluem muitas pessoas, incluindo analistas de dados, cientistas de dados, engenheiros de ML e "o negócio". Colabore com eles para garantir que o modelo de dados e as transformações fornecidas sejam eficientes e úteis. Em termos de desempenho, as consultas devem ser executadas o mais rápido possível da maneira mais econômica. O que queremos dizer com útil? Analistas, cientistas de dados e engenheiros de ML devem ser capazes de consultar uma fonte de dados com a confiança de que os dados são da mais alta qualidade e integridade e podem ser integrados a seus fluxos de trabalho e produtos de dados. A empresa deve poder confiar que os dados transformados são precisos e açãoáveis.

Subcorrentes

O estágio de transformação é onde seus dados sofrem mutações e se transformam em algo útil para os negócios. Como há muitas partes móveis, as correntes subjacentes são especialmente críticas nesse estágio.

Segurança

Consultas e transformações combinam conjuntos de dados diferentes em novos conjuntos de dados. Quem tem acesso a este novo conjunto de dados? Se alguém tiver acesso a um conjunto de dados, continue a controlar quem tem acesso à coluna, linha e acesso em nível de célula de um conjunto de dados.

Esteja ciente dos vetores de ataque contra seu banco de dados no momento da consulta. Os privilégios de leitura/gravação no banco de dados devem ser rigorosamente monitorados e controlados. Consultar o acesso ao

banco de dados deve ser controlado da mesma forma que você normalmente controla o acesso aos sistemas e ambientes da sua organização.

Mantenha as credenciais ocultas; evite copiar e colar senhas, tokens de acesso ou outras credenciais em códigos ou arquivos não criptografados. É incrivelmente comum ver código em repositórios GitHub com nomes de usuário e senhas de banco de dados colados diretamente na base de código! Nem é preciso dizer, não compartilhe senhas com outros usuários. Finalmente, nunca permita que dados não seguros ou não criptografados atravessem a Internet pública.

Gestão de dados

Embora o gerenciamento de dados seja essencial no estágio do sistema de origem (e em todos os outros estágios do ciclo de vida da engenharia de dados), ele é especialmente crítico no estágio de transformação. A transformação cria inherentemente novos conjuntos de dados que precisam ser gerenciados. Assim como em outros estágios do ciclo de vida da engenharia de dados, é fundamental envolver todas as partes interessadas em modelos e transformações de dados e gerenciar suas expectativas. Além disso, certifique-se de que todos concordem com as convenções de nomenclatura que se alinham com as respectivas definições de negócios dos dados. As convenções de nomenclatura apropriadas devem ser refletidas em nomes de campo fáceis de entender. Os usuários também podem verificar um catálogo de dados para obter mais clareza sobre o que o campo significa quando foi criado, quem mantém o conjunto de dados e outras informações relevantes.

Contabilizar a precisão da definição é fundamental no estágio de transformação. A transformação segue a lógica de negócios esperada? Cada vez mais, a noção de uma camada semântica ou métrica independente das transformações está se tornando popular. Em vez de impor a lógica de negócios dentro da transformação em tempo de execução, por que não manter essas definições como um estágio independente antes de sua camada de transformação? Embora ainda seja cedo, espere ver camadas semânticas e métricas se tornando mais populares e comuns na engenharia e gerenciamento de dados.

Como as transformações envolvem dados mutantes, é fundamental garantir que os dados que você está usando estejam livres de defeitos e representem a verdade absoluta. Se o MDM é uma opção na sua empresa, prossiga com a sua implementação. Dimensões conformadas e outras transformações dependem do MDM para preservar a integridade original e a verdade básica dos dados. Se o MDM não for possível, trabalhe com as partes interessadas upstream que controlam os dados para garantir que todos os dados que você está transformando estejam corretos e em conformidade com a lógica de negócios acordada.

As transformações de dados tornam potencialmente difícil saber como um conjunto de dados foi derivado nas mesmas linhas. Em [Capítulo 6](#), discutimos catálogos de dados. À medida que transformamos os dados, *linhagem de dados* ferramentas tornam-se inestimáveis. As ferramentas de linhagem de dados ajudam tanto os engenheiros de dados, que devem entender as etapas de transformação anteriores à medida que criam novas transformações, quanto os analistas, que precisam entender de onde vieram os dados enquanto executam consultas e criam relatórios.

Finalmente, que impacto a conformidade regulatória tem em seu modelo de dados e transformações? Os dados dos campos confidenciais são mascarados ou ofuscados, se necessário? Você pode excluir dados em resposta a solicitações de exclusão? Seu rastreamento de linhagem de dados permite que você veja dados derivados de dados excluídos e reexecute transformações para remover dados downstream de fontes brutais?

DataOps

Com consultas e transformações, o DataOps tem duas áreas de atuação: dados e sistemas. Você precisa monitorar e ser alertado sobre alterações ou anomalias nessas áreas. O campo da observabilidade de dados está explodindo agora, com grande foco na confiabilidade dos dados. Existe até um cargo recente chamado *engenheiro de confiabilidade de dados*. Esta seção enfatiza a observabilidade e a integridade dos dados, com foco no estágio de consulta e transformação.

Vamos começar com o lado dos dados do DataOps. Quando você consulta dados, as entradas e saídas estão corretas? Como você sabe? Se esta consulta for salva em uma tabela, o esquema está correto? E quanto à forma dos dados e estatísticas relacionadas, como valores mínimos/máximos, contagens nulas e muito mais? Você deve executar testes de qualidade de dados nos conjuntos de dados de entrada e no conjunto de dados transformados, o que garantirá que os dados atendam às expectativas dos usuários upstream e downstream. Se houver um problema de qualidade de dados na transformação, você deverá ter a capacidade de sinalizar esse problema, reverter as alterações e investigar a causa raiz.

Agora vamos ver a parte Ops de DataOps. Como estão os sistemas? Monitore métricas como comprimento da fila de consulta, simultaneidade de consulta, uso de memória, utilização de armazenamento, latência de rede e E/S de disco. Use dados de métrica para identificar gargalos e consultas de desempenho insatisfatório que podem ser candidatos a refatoração e ajuste. Se a consulta estiver perfeita, você terá uma boa ideia de onde ajustar o próprio banco de dados (por exemplo, agrupando uma tabela para um desempenho de pesquisa mais rápido). Ou talvez seja necessário atualizar os recursos de computação do banco de dados. Os bancos de dados de nuvem e SaaS de hoje oferecem muita flexibilidade para atualizar (e baixar) seu sistema rapidamente. Adote uma abordagem baseada em dados e use suas métricas de observabilidade para identificar se você tem uma consulta ou um problema relacionado a sistemas.

A mudança para bancos de dados analíticos baseados em SaaS altera o perfil de custo do consumo de dados. Na época dos data warehouses locais, o sistema e as licenças eram adquiridos antecipadamente, sem custo adicional de uso. Enquanto os engenheiros de dados tradicionais se concentram na otimização de desempenho para extrair o máximo de utilidade de suas compras caras, os engenheiros de dados que trabalham com data warehouses em nuvem que cobram com base no consumo precisam se concentrar no gerenciamento e na otimização de custos. Esta é a prática de *FinOps* (ver [Capítulo 4](#)).

Arquitetura de dados

As regras gerais de uma boa arquitetura de dados em Capítulo 3 aplicam-se à fase de transformação. Construa sistemas robustos que podem processar e transformar dados sem implodir. Suas opções de ingestão e armazenamento afetarão diretamente a capacidade de sua arquitetura geral de realizar consultas e transformações confiáveis. Se a ingestão e o armazenamento forem apropriados para seus padrões de consulta e transformação, você estará em um ótimo lugar. Por outro lado, se suas consultas e transformações não funcionarem bem com seus sistemas upstream, você terá um mundo de dores.

Por exemplo, muitas vezes vemos equipes de dados usando os pipelines de dados e bancos de dados errados para o trabalho. Uma equipe de dados pode conectar um pipeline de dados em tempo real a um RDBMS ou Elasticsearch e usá-lo como seu data warehouse. Esses sistemas não são otimizados para consultas OLAP agregadas de alto volume e irão implodir sob essa carga de trabalho. Essa equipe de dados claramente não entendia como suas escolhas arquitetônicas afetariam o desempenho da consulta. Reserve um tempo para entender as compensações inerentes às suas escolhas de arquitetura; seja claro sobre como seu modelo de dados funcionará com os sistemas de ingestão e armazenamento e como as consultas serão executadas.

Orquestração

As equipes de dados geralmente gerenciam seus pipelines de transformação usando agendamentos simples baseados em tempo, por exemplo, cron jobs. Isso funciona razoavelmente bem no começo, mas se torna um pesadelo conforme os fluxos de trabalho ficam mais complicados. Use a orquestração para gerenciar pipelines complexos usando uma abordagem baseada em dependência. A orquestração também é a cola que nos permite montar pipelines que abrangem vários sistemas.

Engenharia de software

Ao escrever o código de transformação, você pode usar muitas linguagens – como SQL, Python e linguagens baseadas em JVM – plataformas que variam de data warehouses a clusters de computação distribuída e tudo mais. Cada linguagem e plataforma tem seus pontos fortes e peculiaridades, então você deve conhecer as melhores práticas de suas ferramentas. Por exemplo, você pode escrever transformações de dados em Python, desenvolvido por um sistema distribuído como Spark ou Dask. Ao executar uma transformação de dados, você está usando uma UDF quando uma função nativa pode funcionar muito melhor? Vimos casos em que UDFs lentos e mal escritos foram substituídos por um comando SQL integrado, com melhoria instantânea e drástica no desempenho.

A ascensão da engenharia analítica traz práticas de engenharia de software para usuários finais, com a noção de *análise como código*. Ferramentas de transformação de engenharia analítica como dbt explodiram em popularidade, dando aos analistas e cientistas de dados a capacidade de escrever transformações no banco de dados usando SQL, sem a intervenção direta de um DBA ou um engenheiro de dados. Nesse caso, o engenheiro de dados é responsável por configurar o código

repositório e pipeline de CI/CD usados pelos analistas e cientistas de dados. Esta é uma grande mudança na função de um engenheiro de dados, que historicamente construiria e gerenciaria a infraestrutura subjacente e criaria as transformações de dados. À medida que as ferramentas de dados reduzem as barreiras de entrada e se tornam mais democratizadas entre as equipes de dados, será interessante ver como os fluxos de trabalho das equipes de dados mudam.

Usando uma ferramenta de baixo código baseada em GUI, você obterá visualizações úteis do fluxo de trabalho da transformação. Você ainda precisa entender o que está acontecendo sob o capô. Essas ferramentas de transformação baseadas em GUI geralmente geram SQL ou alguma outra linguagem nos bastidores. Embora o objetivo de uma ferramenta de baixo código seja aliviar a necessidade de se envolver em detalhes de baixo nível, entender o código nos bastidores ajudará na depuração e na otimização do desempenho. Supor cegamente que a ferramenta está gerando código de alto desempenho é um erro.

Sugerimos que os engenheiros de dados prestem atenção especial às melhores práticas de engenharia de software no estágio de consulta e transformação. Embora seja tentador simplesmente lançar mais recursos de processamento em um conjunto de dados, saber como escrever um código limpo e de alto desempenho é uma abordagem muito melhor.

Conclusão

As transformações estão no centro dos pipelines de dados. É fundamental ter em mente o propósito das transformações. Em última análise, os engenheiros não são contratados para brincar com os brinquedos tecnológicos mais recentes, mas para atender seus clientes. As transformações são onde os dados agregam valor e ROI aos negócios.

Nossa opinião é que é possível adotar tecnologias de transformação empolgantes e atender as partes interessadas. [Capítulo 11](#) fala sobre *opilha de dados ao vivo*, basicamente reconfigurando a pilha de dados em torno da ingestão de dados de streaming e aproximando os fluxos de trabalho de transformação dos próprios aplicativos do sistema de origem. As equipes de engenharia que pensam nos dados em tempo real como a tecnologia pela tecnologia repetirão os erros da era do big data. Mas, na realidade, a maioria das organizações com as quais trabalhamos tem um caso de uso comercial que se beneficiaria com o streaming de dados. Identificar esses casos de uso e focar no valor antes de escolher tecnologias e sistemas complexos é fundamental.

À medida que avançamos para o estágio de serviço do ciclo de vida da engenharia de dados em [Capítulo 9](#), refletem sobre a tecnologia como uma ferramenta para atingir os objetivos organizacionais. Se você trabalha como engenheiro de dados, pense em como as melhorias nos sistemas de transformação podem ajudá-lo a atender melhor seus clientes finais. Se você está apenas embarcando no caminho da engenharia de dados, pense nos tipos de problemas de negócios que você está interessado em resolver com a tecnologia.

Recursos adicionais

- “Construindo um cofre de dados em tempo real no Snowflake” por Dmytro Yaroshenko e Kent Graziano
- “Construindo um Data Warehouse Escalável com o Data Vault 2.0” (Morgan Kaufmann) por Daniel Linstedt e Michael Olschimke
- “Construindo o Data Warehouse” (Wiley), Fábrica de Informações Corporativas, e O Esquema da Estrela Unificada (Publicações Técnicas) por WH (Bill) Inmon
- “Caching no Snowflake Data Warehouse” Página da Comunidade Snowflake
- “Data Warehouse: A Escolha de Inmon vs. Kimball” por Ian Abramson
- “O kit de ferramentas de armazenamento de dados” por Ralph Kimball e Margy Ross (Wiley)
- “Cofre de dados — uma visão geral” por John Ryan
- “Noções básicas de modelagem do Data Vault 2.0” por Kent Graziano
- Tutorial “Um guia detalhado sobre otimização de consultas SQL” por Megha
- “Diferença entre Kimball e Inmon” por manmeetjuneja5
- “Eventual vs. Forte Consistência em Bancos de Dados Distribuídos” por Saurabh.v
- “A Evolução da Fábrica de Informações Corporativas” por Bill Inmon
- Gavroshe EUA página web “DW 2.0”
 - do Google Cloud Documentação “Usando resultados de consulta em cache”
 - Holística “Não é possível combinar campos devido a problemas de fan-out?” página de perguntas frequentes
 - “Como funciona um mecanismo de banco de dados SQL,” por Dennis Pham
 - “Como as organizações devem estruturar seus dados?” por Michael Berk
 - “Inmon ou Kimball: qual abordagem é adequada para seu data warehouse?” por Sansu George
- Documento “Introdução à Modelagem de Cofre de Dados”, compilado por Kent Graziano e Dan Linstedt
- “Introdução ao Data Warehouse”, “Introdução à Modelagem Dimensional para Data Warehousing”, e “Introdução ao Data Vault para Data Warehousing” por Simon Kitching
- Grupo Kimball “Processo de projeto dimensional em quatro etapas”, “Dimensões Conformadas”, e “Técnicas de Modelagem Dimensional” páginas web
- Tópico do Reddit “Kimball vs. Inmon vs. Vault”
- “Modelagem de dados de streaming em tempo real?” Tópico do Stack Exchange
- “O novo paradigma 'Unified Star Schema' na revisão de modelagem de dados analíticos” por Andriy Zabavskyy

- OráculosTutorial “Mudando Lentamente Dimensões”
- ScienceDirect'sPágina web “Fábrica de Informações Corporativas”
- “Uma explicação simples de agregações simétricas ou 'Por que diabos meu SQL se parece com isso?'”por Lloyd Tabb
- “Modelagem de eventos de streaming”por Paul Stanton
- “Tipos de arquitetura de armazenamento de dados”por Amrita Fernando
- Patente dos EUA para “Método e Aparelho para Integração Funcional de Metadados”
- Zentut'sPágina da web “Bill Inmon Data Warehouse”

Servindo dados para análise, máquina Aprendizado e ETL reverso

Parabéns! Você atingiu o estágio final do ciclo de vida da engenharia de dados, fornecendo dados para casos de uso downstream (consulte [Figura 9-1](#)). Neste capítulo, você aprenderá sobre várias maneiras de fornecer dados para os três principais casos de uso que encontrará como engenheiro de dados. Primeiro, você fornecerá dados para análise e BI. Você preparará dados para uso em análises estatísticas, relatórios e painéis. Esta é a área mais tradicional de serviço de dados. Indiscutivelmente, é anterior à TI e aos bancos de dados, mas é mais importante do que nunca que as partes interessadas tenham visibilidade dos processos comerciais, organizacionais e financeiros.

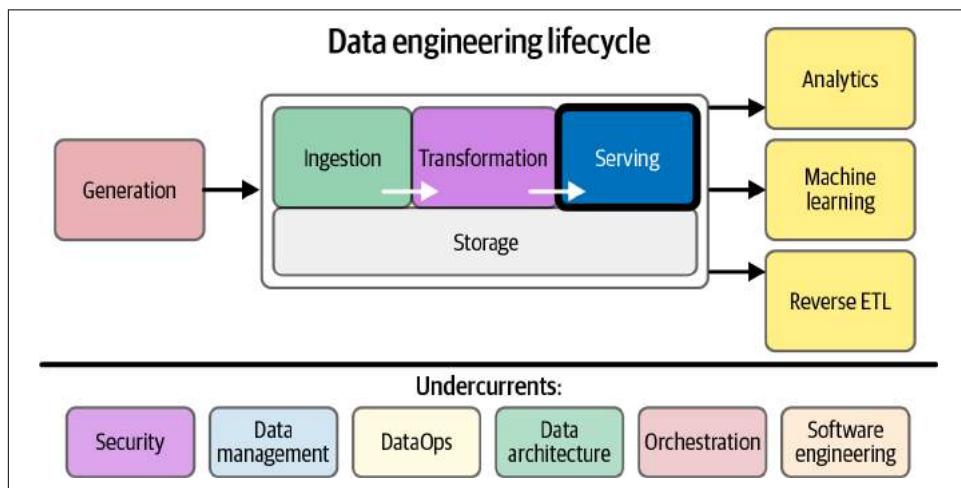


Figura 9-1. A veiculação fornece dados para casos de uso

Em segundo lugar, você fornecerá dados para aplicativos de ML. ML não é possível sem dados de alta qualidade, devidamente preparados. Os engenheiros de dados trabalham com cientistas de dados e engenheiros de ML para adquirir, transformar e fornecer os dados necessários para o treinamento do modelo.

Em terceiro lugar, você fornecerá dados por meio de ETL reverso.*ETL reverso* é o processo de enviar dados de volta às fontes de dados. Por exemplo, podemos adquirir dados de uma plataforma de tecnologia de anúncios, executar um processo estatístico nesses dados para determinar lances de custo por clique e, em seguida, alimentar esses dados de volta à plataforma de tecnologia de anúncios. O ETL reverso é altamente emaranhado com BI e ML.

Antes de entrarmos nessas três principais formas de fornecer dados, vamos examinar algumas considerações gerais.

Considerações Gerais para Servir Dados

Antes de nos aprofundarmos na exibição de dados, temos algumas considerações importantes. Em primeiro lugar está a confiança. As pessoas precisam confiar nos dados que você está fornecendo. Além disso, você precisa entender seus casos de uso e usuários, os produtos de dados que serão produzidos, como você servirá os dados (autoatendimento ou não), definições e lógica de dados e malha de dados. As considerações que discutiremos aqui são gerais e se aplicam a qualquer uma das três formas de fornecer dados. Entender essas considerações ajudará você a ser muito mais eficaz no atendimento a seus clientes de dados.

Confiar

Leva 20 anos para construir uma reputação e cinco minutos para arruiná-la. Se você pensar sobre isso, fará as coisas de maneira diferente.

-Warren Buffett¹

Acima de tudo, a confiança é a principal consideração no fornecimento de dados; os usuários finais precisam confiar nos dados que estão recebendo. A arquitetura de dados e a camada de serviço mais sofisticadas e sofisticadas são irrelevantes se os usuários finais não acreditarem que os dados são uma representação confiável de seus negócios. A perda de confiança costuma ser uma sentença de morte silenciosa para um projeto de dados, mesmo que o projeto não seja oficialmente cancelado até meses ou anos depois. O trabalho de um engenheiro de dados é fornecer os melhores dados possíveis, portanto, você deve garantir que seus produtos de dados sempre contenham dados confiáveis e de alta qualidade.

À medida que você aprender a fornecer dados ao longo deste capítulo, reforçaremos a ideia de inserir confiança em seus dados e discutiremos maneiras pragmáticas de fazer isso. Vemos muitos casos em que as equipes de dados estão obcecadas em enviar dados sem perguntar se as partes interessadas confiam neles em primeiro lugar. Muitas vezes, as partes interessadas perdem a confiança nos dados. Uma vez

1 Citado em Benjamin Snyder, "7 Insights from Legendary Investor Warren Buffett," CNBC Faça, 1 de Maio, 2017, <https://oreil.ly/QEqF9>.

a confiança se foi, recuperá-la é incrivelmente difícil. Isso inevitavelmente leva o negócio a não atingir todo o seu potencial, com dados e equipes de dados perdendo credibilidade (e possivelmente sendo dissolvidos).

Para obter a qualidade dos dados e construir a confiança das partes interessadas, utilize os processos de validação e observabilidade de dados, em conjunto com a inspeção visual e a confirmação da validade com as partes interessadas. *Data de validade* está analisando dados para garantir que eles representem com precisão informações financeiras, interações com clientes e vendas. *Observabilidade de dados* fornece uma visão contínua dos dados e processos de dados. Esses processos devem ser aplicados *durante todo o ciclo de vida da engenharia de dados* para obter um bom resultado quando chegarmos ao final. Discutiremos isso mais adiante em “[Subcorrentes](#)” na página 360.

Além de criar confiança na qualidade dos dados, cabe aos engenheiros criar confiança em seus SLAs e SLOs com seus usuários finais e partes interessadas upstream. Uma vez que os usuários dependem de dados para realizar processos de negócios, eles exigirão que os dados estejam consistentemente disponíveis e atualizados de acordo com os compromissos assumidos pelos engenheiros de dados. Dados de alta qualidade têm pouco valor se não estiverem disponíveis conforme o esperado na hora de tomar uma decisão comercial crítica. Observe que os SLAs e SLOs também podem assumir a forma de *contratos de dados* (ver [capítulo 5](#)), formal ou informalmente.

Conversamos sobre SLAs em [capítulo 5](#), mas vale a pena discuti-los novamente aqui. Os SLAs vêm em uma variedade de formas. Independentemente de sua forma, um SLA informa aos usuários o que esperar de seu produto de dados; é um contrato entre você e seus stakeholders. Um exemplo de um SLA pode ser: “Os dados estarão disponíveis de forma confiável e de alta qualidade”. Um SLO é uma parte fundamental de um SLA e descreve as maneiras pelas quais você medirá o desempenho em relação ao que concordou. Por exemplo, considerando o SLA de exemplo anterior, um SLO pode ser: “Nossos pipelines de dados para seu painel ou fluxo de trabalho de ML terão 99% de tempo de atividade, com 95% de dados livres de defeitos”. Certifique-se de que as expectativas sejam claras e que você tenha a capacidade de verificar se está operando dentro dos parâmetros de SLA e SLO acordados.

Não basta simplesmente concordar com um SLA. A comunicação contínua é uma característica central de um bom SLA. Você comunicou possíveis problemas que podem afetar suas expectativas de SLA ou SLO? Qual é o seu processo de correção e melhoria?

Confiança é tudo. Leva muito tempo para ganhar e é fácil perder.

Qual é o caso de uso e quem é o usuário?

O estágio de veiculação é sobre dados em ação. Mas o que é um *produtivo* uso de dados? Você precisa considerar duas coisas para responder a essa pergunta: qual é o caso de uso e quem é o usuário?

O caso de uso de dados vai muito além da visualização de relatórios e painéis. Os dados estão no seu melhor quando levam a *Ação*. Um executivo tomará uma decisão estratégica a partir de um relatório? Um usuário de um aplicativo móvel de entrega de comida receberá um cupom que o estimule a comprar nos próximos dois minutos? Os dados são freqüentemente usados em mais de um

caso de uso — por exemplo, para treinar um modelo de ML que faz pontuação de leads e preenche um CRM (ETL reverso). Dados de alta qualidade e alto impacto atraem inherentemente muitos casos de uso interessantes. Mas, ao buscar casos de uso, sempre pergunta: “O que *Ação* desses dados serão acionados e *Quem* estará realizando esta ação?”, com a pergunta de acompanhamento apropriada, “Esta ação pode ser automatizada?”

Sempre que possível, priorize os casos de uso com o ROI mais alto possível. Os engenheiros de dados adoram ficar obcecados com os detalhes técnicos da implementação dos sistemas que constroem, ignorando a questão básica do propósito. Os engenheiros querem fazer o que fazem de melhor: projetar coisas. Quando os engenheiros reconhecem a necessidade de se concentrar no valor e nos casos de uso, eles se tornam muito mais valiosos e eficazes em suas funções.

Ao iniciar um novo projeto de dados, trabalhar de trás para frente é útil. Embora seja tentador se concentrar nas ferramentas, recomendamos que você comece com o caso de uso e os usuários. Aqui estão algumas perguntas que você deve fazer a si mesmo ao começar:

- Quem usará os dados e como eles os usarão?
- O que as partes interessadas esperam?
- Como posso colaborar com as partes interessadas dos dados (por exemplo, cientistas de dados, analistas, usuários de negócios) para entender como os dados com os quais estou trabalhando serão usados?

Novamente, sempre aborde a engenharia de dados da perspectiva do usuário e seu caso de uso. Ao entender suas expectativas e objetivos, você pode trabalhar de trás para frente para criar produtos de dados incríveis com mais facilidade. Vamos dedicar um momento para expandir nossa discussão sobre um produto de dados.

Produtos de dados

Uma boa definição de produto de dados é um produto que facilita um objetivo final por meio do uso de dados.

— DJ Patil²

Os produtos de dados não são criados no vácuo. Como tantos outros processos organizacionais que discutimos, fazer produtos de dados é um esporte de contato total, envolvendo uma mistura de produtos e negócios junto com a tecnologia. É importante envolver as principais partes interessadas no desenvolvimento de um produto de dados. Na maioria das empresas, um engenheiro de dados está a algumas etapas de distância dos usuários finais de um produto de dados; um bom engenheiro de dados procurará entender completamente os resultados para usuários diretos, como analistas de dados e cientistas de dados ou clientes externos à empresa.

2 DJ Patil, “Data Jujitsu: The Art of Turning Data into Product,” *Radar O’Reilly*, 17 de julho de 2012, <https://oreil.ly/1YS9x>.

Ao criar um produto de dados, é útil pensar nas “tarefas a serem realizadas”.³ Um usuário “contrata” um produto para um “trabalho a ser feito”. Isso significa que você precisa saber o que o usuário deseja, ou seja, sua motivação para “contratar” seu produto. Um erro clássico de engenharia é simplesmente construir sem entender os requisitos, as necessidades do usuário final ou o ajuste do produto/mercado. Esse desastre acontece quando você cria produtos de dados que ninguém quer usar.

Um bom produto de dados tem ciclos de feedback positivos. Mais uso de um produto de dados gera dados mais úteis, que são usados para melhorar o produto de dados. Enxague e repita.

Ao criar um produto de dados, lembre-se destas considerações:

- Quando alguém usa o produto de dados, o que espera realizar? Com muita frequência, os produtos de dados são feitos sem uma compreensão clara do resultado esperado pelo usuário.
- O produto de dados atenderá a usuários internos ou externos? Em [Capítulo 2](#), discutimos a engenharia de dados interna e externa. Ao criar um produto de dados, saber se o cliente é interno ou externo afetará a maneira como os dados são atendidos.
- Quais são os resultados e o ROI do produto de dados que você está construindo?

Construir produtos de dados que as pessoas usarão e amarão é fundamental. Nada arruinará mais a adoção de um produto de dados do que utilidade indesejada e perda de confiança nas saídas de dados. Preste atenção à adoção e ao uso de produtos de dados e esteja disposto a fazer ajustes para deixar os usuários satisfeitos.

Autoatendimento ou não?

Como os usuários interagirão com seu produto de dados? Um diretor de negócios solicitará um relatório da equipe de dados ou esse diretor pode simplesmente criar o relatório? Os produtos de dados de autoatendimento — dando ao usuário a capacidade de criar produtos de dados por conta própria — têm sido uma aspiração comum dos usuários de dados por muitos anos. O que é melhor do que apenas dar ao usuário final a capacidade de criar diretamente relatórios, análises e modelos de ML?

Hoje, o BI de autoatendimento e a ciência de dados ainda são aspiracionais. Embora ocasionalmente vejamos empresas fazendo autoatendimento com dados com sucesso, isso é raro. Na maioria das vezes, as tentativas de dados de autoatendimento começam com grandes intenções, mas acabam falhando; os dados de autoatendimento são difíceis de implementar na prática. Assim, o analista ou cientista de dados

³ Clayton M. Christensen et al., “Know Your Customers’ ‘Jobs to Be Done’”, *Harvard Business Review*, setembro 2016, <https://oreil.ly/3uU4j>.

resta realizar o trabalho pesado de fornecer relatórios ad hoc e manter painéis.

Por que os dados de autoatendimento são tão difíceis? A resposta é sutil, mas geralmente envolve a compreensão do usuário final. Se o usuário for um executivo que precisa entender como o negócio está indo, essa pessoa provavelmente deseja apenas um painel predefinido de métricas claras e açãoáveis. O executivo provavelmente ignorará qualquer ferramenta de autoatendimento para criar visualizações de dados personalizadas. Se os relatórios provocarem mais perguntas, eles podem ter analistas à sua disposição para prosseguir com uma investigação mais profunda. Por outro lado, um usuário que é analista pode já estar buscando análise de autoatendimento por meio de ferramentas mais poderosas, como SQL. A análise de autoatendimento por meio de uma camada de BI não é útil. As mesmas considerações se aplicam à ciência de dados. Embora conceder ML de autoatendimento a "cientistas de dados cidadãos" tenha sido uma meta de muitos fornecedores de ML automatizados, a adoção ainda é incipiente pelos mesmos motivos da análise de autoatendimento. Nesses dois casos extremos, um produto de dados de autoatendimento é uma ferramenta errada para o trabalho.

Projetos de dados de autoatendimento bem-sucedidos se resumem a ter o público certo. Identifique os usuários de autoatendimento e o "trabalho" que desejam fazer. O que eles estão tentando realizar usando um produto de dados de autoatendimento em vez de fazer parceria com um analista de dados para fazer o trabalho? Um grupo de executivos com experiência em dados forma um público ideal para o autoatendimento; eles provavelmente querem dividir e dividir os dados sem precisar tirar o pó de suas habilidades de SQL lânguidas. Os líderes de negócios dispostos a investir tempo para aprender habilidades de dados por meio de uma iniciativa da empresa e um programa de treinamento também podem obter um valor significativo do autoatendimento.

Determine como você fornecerá dados a esse grupo. Quais são os requisitos de tempo para novos dados? O que acontece se eles inevitavelmente quiserem mais dados ou mudarem o escopo do que é exigido do autoatendimento? Mais dados geralmente significam mais perguntas, o que requer mais dados. Você precisará antecipar as crescentes necessidades de seus usuários de autoatendimento. Você também precisa entender o equilíbrio delicado entre flexibilidade e proteções que ajudarão seu público a encontrar valor e insights sem resultados incorretos e confusos.

Definições e lógica de dados

Conforme discutimos enfaticamente, a utilidade dos dados em uma organização deriva, em última análise, de sua correção e confiabilidade. Criticamente, a exatidão dos dados vai além da reprodução fiel dos valores de eventos dos sistemas de origem. A exatidão dos dados também abrange definições e lógica de dados adequadas; eles devem ser incorporados aos dados em todos os estágios do ciclo de vida, desde sistemas de origem até pipelines de dados, ferramentas de BI e muito mais.

Definição de dados refere-se ao significado dos dados como eles são entendidos em toda a organização. Por exemplo, *cliente* tem um significado preciso dentro de uma empresa e

departamentos. Quando a definição de um cliente varia, isso deve ser documentado e disponibilizado a todos que usam os dados.

Lógica de dados estipula fórmulas para derivar métricas de dados – digamos, vendas brutas ou valor vitalício do cliente. A lógica de dados adequada deve codificar definições de dados e detalhes de cálculos estatísticos. Para calcular as métricas de rotatividade de clientes, precisaríamos de uma definição: quem é um cliente? Para calcular o lucro líquido, precisaríamos de um conjunto de regras lógicas para determinar quais despesas deduzir da receita bruta.

Freqüentemente, vemos definições e lógica de dados tomadas como certas, muitas vezes repassadas pela organização na forma de conhecimento institucional. *conhecimento institucional* assume vida própria, muitas vezes à custa de anedotas que substituem percepções, decisões e ações baseadas em dados. Em vez disso, declarar formalmente as definições e a lógica dos dados em um catálogo de dados e nos sistemas do ciclo de vida da engenharia de dados ajuda muito a garantir a correção, consistência e confiabilidade dos dados.

As definições de dados podem ser fornecidas de várias maneiras, às vezes explicitamente, mas principalmente de forma implícita. Por *implícito*, queremos dizer que sempre que você fornece dados para uma consulta, um painel ou um modelo de ML, os dados e as métricas derivadas são apresentados de forma consistente e correta. Ao escrever uma consulta SQL, você assume implicitamente que as entradas para essa consulta estão corretas, incluindo a lógica e as definições do pipeline upstream. É aqui que a modelagem de dados (descrita em [Capítulo 8](#)) é incrivelmente útil para capturar definições de dados e lógica de uma forma comprehensível e utilizável por vários usuários finais.

Usando uma camada semântica, você consolida as definições de negócios e a lógica de maneira reutilizável. Escreva uma vez, use em qualquer lugar. Esse paradigma é uma abordagem orientada a objetos para métricas, cálculos e lógica. Teremos mais a dizer em “[Camadas Semânticas e Métricas](#)” na [página 355](#).

Malha de dados

A malha de dados será cada vez mais considerada ao servir dados. A malha de dados muda fundamentalmente a forma como os dados são servidos dentro de uma organização. Em vez de equipes de dados isoladas atendendo a seus constituintes internos, cada equipe de domínio assume dois aspectos de serviço de dados ponto a ponto descentralizado.

Primeiro, as equipes são responsáveis por fornecer dados *para outras equipes* preparando-o para o consumo. Os dados devem ser bons para uso em aplicativos de dados, painéis, análises e ferramentas de BI em toda a organização. Em segundo lugar, cada equipe potencialmente executa seus painéis e análises para *self-service*. As equipes consomem dados de toda a organização com base nas necessidades específicas de seu domínio. Os dados consumidos de outras equipes também podem entrar no software projetado por uma equipe de domínio por meio de análises incorporadas ou um recurso de ML.

Isso muda drasticamente os detalhes e a estrutura do atendimento. Introduzimos o conceito de malha de dados em [Capítulo 3](#). Agora que cobrimos algumas considerações gerais para fornecer dados, vamos examinar a primeira área principal: análise.

Análise

O primeiro caso de uso de serviço de dados que você provavelmente encontrará é *análise*, que é descobrir, explorar, identificar e tornar visíveis os principais insights e padrões nos dados. A análise tem muitos aspectos. Como prática, a análise é realizada usando métodos estatísticos, relatórios, ferramentas de BI e muito mais. Como engenheiro de dados, conhecer os vários tipos e técnicas de análise é fundamental para realizar seu trabalho. Esta seção visa mostrar como você fornecerá dados para análises e apresenta alguns pontos a serem considerados para ajudar seus analistas a obter sucesso.

Antes mesmo de fornecer dados para análise, a primeira coisa que você precisa fazer (o que deve soar familiar depois de ler a seção anterior) é identificar o caso de uso final. O usuário está olhando para as tendências históricas? Os usuários devem ser notificados imediatamente automaticamente sobre uma anomalia, como um alerta de fraude? Alguém está consumindo um painel em tempo real em um aplicativo móvel? Esses exemplos destacam as diferenças entre análise de negócios (geralmente BI), análise operacional e análise incorporada. Cada uma dessas categorias de análise tem objetivos diferentes e requisitos de veiculação exclusivos. Vejamos como você fornecerá dados para esses tipos de análise.

Analista de negócios

Analista de negócios usa dados históricos e atuais para tomar decisões estratégicas e açãoáveis. Os tipos de decisões tendem a levar em consideração tendências de longo prazo e geralmente envolvem uma mistura de análise estatística e de tendências, juntamente com conhecimento de domínio e julgamento humano. A análise de negócios é tanto uma arte quanto uma ciência.

A análise de negócios geralmente se enquadra em algumas grandes áreas - painéis, relatórios e análises ad hoc. Um analista de negócios pode se concentrar em uma ou todas essas categorias. Vamos ver rapidamente as diferenças entre essas práticas e ferramentas relacionadas. Compreender o fluxo de trabalho de um analista ajudará você, o engenheiro de dados, a entender como fornecer dados.

Após mostrado de forma concisa aos tomadores de decisão como uma organização está se saindo em relação a um punhado de métricas principais, como vendas e retenção de clientes. Essas métricas principais são apresentadas como visualizações (por exemplo, gráficos ou mapas de calor), estatísticas resumidas ou até mesmo um único número. Isso é semelhante ao painel de um carro, que fornece uma leitura única das coisas críticas que você precisa saber ao dirigir um veículo. Uma organização pode ter mais de um painel, com executivos de nível C usando um painel abrangente e seus subordinados diretos usando painéis com suas métricas específicas, KPIs ou objetivos e principais resultados (OKRs). Os analistas ajudam a criar e manter esses painéis. Uma vez que as partes interessadas do negócio adotam e confiam em um

Dashboard, o analista geralmente responde a solicitações para examinar um possível problema com uma métrica ou adicionar uma nova métrica ao painel. Atualmente, você pode usar plataformas de BI para criar painéis, como Tableau, Looker, Sisense, Power BI ou Apache Superset/Preset.

Os analistas são frequentemente encarregados pelas partes interessadas do negócio de criar um relatório. O objetivo de um relatório é usar dados para direcionar insights e ações. Um analista que trabalha em uma empresa de varejo on-line é solicitado a investigar quais fatores estão gerando uma taxa de retorno maior do que o esperado para shorts de corrida femininos. O analista executa algumas consultas SQL no data warehouse, agrupa os códigos de devolução que os clientes fornecem como o motivo de sua devolução e descobre que o tecido do short de corrida é de qualidade inferior, muitas vezes desgastado em poucos usos. As partes interessadas, como fabricação e controle de qualidade, são notificadas dessas descobertas. Além disso, as descobertas são resumidas em um relatório e distribuídas na mesma ferramenta de BI onde reside o painel.

O analista foi solicitado a investigar um possível problema e retornar com insights. Isso representa um exemplo de análise *ad hoc*. Os relatórios geralmente começam como solicitações ad hoc. Se os resultados da análise ad hoc forem impactantes, eles geralmente acabam em um relatório ou painel. As tecnologias usadas para relatórios e análises ad hoc são semelhantes aos painéis, mas podem incluir Excel, Python, notebooks baseados em R, consultas SQL e muito mais.

Bons analistas se envolvem constantemente com os negócios e mergulham nos dados para responder a perguntas e descobrir tendências e insights ocultos e contra-intuitivos. Eles também trabalham com engenheiros de dados para fornecer feedback sobre qualidade de dados, problemas de confiabilidade e solicitações de novos conjuntos de dados. O engenheiro de dados é responsável por abordar esse feedback e fornecer novos conjuntos de dados para uso do analista.

Voltando ao exemplo dos shorts de corrida, suponha que depois de comunicar suas descobertas, os analistas saibam que a manufatura pode fornecer a eles vários detalhes da cadeia de suprimentos com relação aos materiais usados nos shorts de corrida. Os engenheiros de dados realizam um projeto para inserir esses dados no data warehouse. Uma vez que os dados da cadeia de suprimentos estejam presentes, os analistas podem correlacionar números de série de roupas específicas com o fornecedor do tecido usado no item. Eles descobrem que a maioria das falhas está ligada a um de seus três fornecedores e a fábrica para de usar tecido desse fornecedor.

Os dados para análise de negócios são frequentemente servidos em modo de lote de um data warehouse ou data lake. Isso varia muito entre empresas, departamentos e até equipes de dados dentro das empresas. Novos dados podem estar disponíveis a cada segundo, a cada minuto, a cada 30 minutos, todos os dias ou uma vez por semana. A frequência dos lotes pode variar por vários motivos. Uma coisa importante a observar é que os engenheiros que trabalham em problemas analíticos devem considerar várias aplicações potenciais de dados - atuais e futuras. É comum ter frequências de atualização de dados mistas para atender os casos de uso de forma adequada, mas lembre-se de que a frequência de ingestão define um teto no downstream

frequência. Se existirem aplicativos de streaming para os dados, eles devem ser ingeridos como um stream, mesmo que algumas etapas de processamento e exibição de downstream sejam tratadas em lotes.

Obviamente, os engenheiros de dados devem abordar várias considerações técnicas de back-end ao fornecer análises de negócios. Algumas ferramentas de BI armazenam dados em uma camada de armazenamento interno. Outras ferramentas executam consultas em seu data lake ou data warehouse. Isso é vantajoso porque você pode aproveitar ao máximo o poder do seu banco de dados OLAP. Como discutimos em capítulos anteriores, a desvantagem é o custo, o controle de acesso e a latência.

Análise Operacional

Se a análise de negócios é sobre o uso de dados para descobrir insights açãoáveis, a análise operacional usa dados para obteração *imediata*:

Análise operacional versus análise de negócios =
ação imediata versus insights açãoáveis

A grande diferença entre análise operacional e análise de negócios é *tempo*. Os dados usados na análise de negócios têm uma visão mais ampla da questão em consideração. É bom saber atualizações atualizadas, mas não afetarão materialmente a qualidade ou o resultado. A análise operacional é exatamente o oposto, pois as atualizações em tempo real podem ser impactantes na resolução de um problema quando ele ocorre.

Um exemplo de análise operacional é o monitoramento de aplicativos em tempo real. Muitas equipes de engenharia de software desejam saber como está o desempenho de seus aplicativos; se surgirem problemas, eles querem ser notificados imediatamente. A equipe de engenharia pode ter um painel (consulte, por exemplo, [Figura 9-2](#)) que mostra as principais métricas, como solicitações por segundo, I/O do banco de dados ou qualquer outra métrica importante. Certas condições podem acionar eventos de dimensionamento, adicionando mais capacidade se os servidores estiverem sobrecarregados. Se certos limites forem violados, o sistema de monitoramento também pode enviar alertas por mensagem de texto, bate-papo em grupo e e-mail.

Análise de Negócios e Operações

A linha entre análise de negócios e operacional começou a se confundir. À medida que os dados de streaming e de baixa latência se tornam mais difundidos, é natural aplicar abordagens operacionais aos problemas de análise de negócios; além de monitorar o desempenho do site na Black Friday, um varejista online também pode analisar e apresentar vendas, receitas e o impacto de campanhas publicitárias em tempo real.

As arquiteturas de dados serão alteradas para se adequarem a um mundo onde você pode ter seus dados quentes e quentes em um só lugar. A pergunta central que você deve sempre fazer a si mesmo e aos seus stakeholders é esta: se você tem dados de streaming, o que você está

vai fazer com isso? Que ação você deve tomar? A ação correta cria impacto e valor. Dados em tempo real sem ação são uma distração implacável.

A longo prazo, prevemos que o streaming suplantará o lote. Os produtos de dados nos próximos 10 anos provavelmente serão streaming primeiro, com a capacidade de combinar dados históricos perfeitamente. Após a coleta em tempo real, os dados ainda podem ser consumidos e processados em lotes conforme necessário.



Figura 9-2. Um painel de análise operacional mostrando algumas métricas importantes do Google Compute Engine

Vamos voltar mais uma vez ao nosso exemplo de shorts de corrida. O uso de análises para descobrir tecidos ruins na cadeia de suprimentos foi um grande sucesso; líderes de negócios e engenheiros de dados querem encontrar mais oportunidades para utilizar dados para melhorar a qualidade do produto. Os engenheiros de dados sugerem a implantação de análises em tempo real na fábrica. A usina já utiliza

uma variedade de máquinas capazes de transmitir dados em tempo real. Além disso, a fábrica possui câmeras que gravam vídeos na linha de produção. No momento, os técnicos assistem às filmagens em tempo real, procuram itens defeituosos e alertam os responsáveis pela linha quando percebem um alto índice de empecilhos aparecendo nos itens.

Os engenheiros de dados percebem que podem usar uma ferramenta de visão de máquina em nuvem pronta para uso para identificar defeitos em tempo real automaticamente. Os dados de defeitos são vinculados a números de série de itens específicos e transmitidos. A partir daqui, um processo de análise em tempo real pode vincular itens defeituosos a eventos de streaming de máquinas mais adiante na linha de montagem.

Usando essa abordagem, os analistas de chão de fábrica descobrem que a qualidade do estoque de tecido bruto varia significativamente de caixa para caixa. Quando o sistema de monitoramento mostra uma alta taxa de defeitos, os funcionários da linha podem remover a caixa defeituosa e devolvê-la ao fornecedor.

Vendo o sucesso deste projeto de melhoria de qualidade, o fornecedor decide adotar processos de controle de qualidade semelhantes. Os engenheiros de dados do varejista trabalham com o fornecedor para implantar suas análises de dados em tempo real, melhorando drasticamente a qualidade de seu estoque de tecidos.

Análise incorporada

Enquanto as análises de negócios e operacionais são focadas internamente, uma tendência recente é a análise externa ou incorporada. Com tantos dados alimentando aplicativos, as empresas fornecem cada vez mais análises aos usuários finais. Estes são normalmente referidos como *aplicativos de dados*, geralmente com painéis analíticos incorporados no próprio aplicativo. Também conhecido como *análise incorporada*, esses painéis voltados para o usuário final fornecem aos usuários as principais métricas sobre seu relacionamento com o aplicativo.

Um termostato inteligente possui um aplicativo móvel que mostra a temperatura em tempo real e métricas de consumo de energia atualizadas, permitindo que o usuário crie uma programação de aquecimento ou resfriamento com melhor eficiência energética. Em outro exemplo, uma plataforma de comércio eletrônico terceirizada fornece aos vendedores um painel em tempo real sobre vendas, estoque e devoluções. O vendedor tem a opção de usar essas informações para oferecer ofertas aos clientes quase em tempo real. Em ambos os casos, um aplicativo permite que os usuários tomem decisões em tempo real (manual ou automaticamente) com base nos dados.

O cenário da análise incorporada está crescendo como uma bola de neve e esperamos que tais aplicativos de dados se tornem cada vez mais difundidos nos próximos anos. Como engenheiro de dados, você provavelmente não está criando o front-end analítico incorporado, já que os desenvolvedores de aplicativos lidam com isso. Como você é responsável pelos bancos de dados que atendem à análise incorporada, você precisará entender os requisitos de velocidade e latência para a análise incorporada.

O desempenho para análises incorporadas abrange três problemas. Primeiro, os usuários do aplicativo não são tão tolerantes com o processamento em lote pouco frequente quanto os analistas internos da empresa; os usuários de uma plataforma SaaS de recrutamento podem esperar ver uma mudança em suas estatísticas assim que fizerem o upload de um novo currículo. Os usuários querem baixa latência de dados. Em segundo lugar, os usuários de aplicativos de dados esperam desempenho da consulta. Quando eles ajustam os parâmetros em um painel analítico, eles querem ver os resultados atualizados aparecerem em segundos. Em terceiro lugar, os aplicativos de dados geralmente devem oferecer suporte a taxas de consulta extremamente altas em muitos painéis e vários clientes. Alto simultaneidade é crítico.

O Google e outros grandes players no espaço de aplicativos de dados desenvolveram tecnologias exóticas para lidar com esses desafios. Para novas startups, o padrão é usar bancos de dados transacionais convencionais para aplicativos de dados. À medida que suas bases de clientes se expandem, elas superam sua arquitetura inicial. Eles têm acesso a uma nova geração de bancos de dados que combinam alto desempenho – consultas rápidas, alta simultaneidade e atualizações quase em tempo real – com relativa facilidade de uso (por exemplo, análise baseada em SQL).

Aprendizado de máquina

A segunda área principal para fornecer dados é o aprendizado de máquina. O ML é cada vez mais comum, então presumimos que você esteja pelo menos familiarizado com o conceito. Com o surgimento da engenharia de ML (quase um universo paralelo à engenharia de dados), você pode se perguntar onde um engenheiro de dados se encaixa na imagem.

Reconhecidamente, a fronteira entre ML, ciência de dados, engenharia de dados e engenharia de ML é cada vez mais confusa, e essa fronteira varia drasticamente entre as organizações. Em algumas organizações, os engenheiros de ML assumem o processamento de dados para aplicativos de ML logo após a coleta de dados ou podem até formar uma organização de dados totalmente separada e paralela que lida com todo o ciclo de vida de todos os aplicativos de ML. Os engenheiros de dados lidam com todo o processamento de dados em outras configurações e, em seguida, repassam os dados aos engenheiros de ML para treinamento do modelo. Os engenheiros de dados podem até lidar com algumas tarefas extremamente específicas de ML, como caracterização de dados.

Voltemos ao nosso exemplo de qualidade para controle de shorts de corrida produzidos por um varejista online. Suponha que os dados de streaming tenham sido implementados na fábrica que faz o estoque de tecido bruto para os shorts. Os cientistas de dados descobriram que a qualidade do tecido fabricado é suscetível às características do poliéster bruto de entrada, temperatura, umidade e vários parâmetros ajustáveis do tear que tece o tecido. Os cientistas de dados desenvolvem um modelo básico para otimizar os parâmetros do chicote. Os engenheiros de ML automatizam o treinamento do modelo e configuram um processo para ajustar automaticamente o tear com base nos parâmetros de entrada. Os engenheiros de dados e ML trabalham juntos para projetar um pipeline de caracterização, e os engenheiros de dados implementam e mantêm o pipeline.

O que um engenheiro de dados deve saber sobre ML

Antes de discutirmos a exibição de dados para ML, você pode se perguntar quanto ML precisa saber como engenheiro de dados. ML é um tópico incrivelmente vasto e não tentaremos ensinar a você o campo; inúmeros livros e cursos estão disponíveis para aprender ML.

Embora um engenheiro de dados não precise ter um conhecimento profundo de ML, ajuda muito conhecer os fundamentos de como o ML clássico funciona e os fundamentos do aprendizado profundo. Conhecer os fundamentos do ML ajudará você a trabalhar ao lado de cientistas de dados na criação de produtos de dados.

Aqui estão algumas áreas de ML com as quais achamos que um engenheiro de dados deve estar familiarizado:

- A diferença entre aprendizado supervisionado, não supervisionado e semissupervisionado.
- A diferença entre técnicas de classificação e regressão.
- As várias técnicas para lidar com dados de séries temporais. Isso inclui análise de séries temporais, bem como previsões de séries temporais.
- Quando usar as técnicas "clássicas" (regressão logística, aprendizado baseado em árvore, máquinas de vetor de suporte) versus aprendizado profundo. Constantemente vemos cientistas de dados pularem imediatamente para o aprendizado profundo quando é um exagero. Como engenheiro de dados, seu conhecimento básico de ML pode ajudá-lo a identificar se uma técnica de ML é adequada e dimensionar os dados que você precisará fornecer.
- Quando você usaria aprendizado de máquina automatizado (AutoML) em vez de criar um modelo de ML? Quais são os trade-offs com cada abordagem em relação aos dados que estão sendo usados?
- Quais são as técnicas de manipulação de dados usadas para dados estruturados e não estruturados?
- Todos os dados usados para ML são convertidos em números. Se você estiver fornecendo dados estruturados ou semiestruturados, certifique-se de que os dados possam ser convertidos adequadamente durante o processo de engenharia de recursos.
- Como codificar dados categóricos e as incorporações para vários tipos de dados.
- A diferença entre aprendizado em lote e online. Qual abordagem é adequada para o seu caso de uso?
- Como o ciclo de vida da engenharia de dados se relaciona com o ciclo de vida do ML em sua empresa? Você será responsável pela interface ou suporte a tecnologias de ML, como armazenamento de recursos ou observabilidade de ML?
- Saiba quando é adequado treinar localmente, em um cluster ou na periferia. Quando você usaria uma GPU em vez de uma CPU? O tipo de hardware que você usa depende muito do tipo de problema de ML que você está resolvendo, da técnica que está usando e do tamanho do seu conjunto de dados.

- Saber a diferença entre as aplicações de dados em lote e streaming em modelos de ML de treinamento.
Por exemplo, os dados em lote geralmente se ajustam bem ao treinamento de modelo off-line, enquanto os dados de streaming funcionam com o treinamento on-line.
- O que são **cascatas de dados** como eles podem afetar os modelos de ML?
- Os resultados são retornados em tempo real ou em lote? Por exemplo, um modelo de transcrição de fala em lote pode processar amostras de fala e retornar texto em lote após uma chamada de API. Um modelo de recomendação de produto pode precisar operar em tempo real enquanto o cliente interage com um site de varejo online.
- O uso de dados estruturados versus não estruturados. Podemos agrupar dados de clientes tabulares (estruturados) ou reconhecer imagens (não estruturadas) usando uma rede neural.

ML é um *grande* área de assunto, e este livro não ensinará esses tópicos ou mesmo generalidades de ML. Se você quiser saber mais sobre ML, sugerimos a leitura *Aprendizado de máquina prático com Scikit-Learn, Keras e TensorFlow* por Aurélien Géron (O'Reilly); inúmeros outros cursos e livros de ML estão disponíveis online. Como os livros e cursos on-line evoluem tão rapidamente, faça sua pesquisa sobre o que parece ser uma boa opção para você.

Formas de fornecer dados para análise e ML

Assim como na análise, os engenheiros de dados fornecem aos cientistas de dados e engenheiros de ML os dados de que precisam para realizar seus trabalhos. Colocamos o serviço de ML ao lado da análise porque os pipelines e processos são extremamente semelhantes. Há muitas maneiras de fornecer dados para análise e ML. Algumas formas comuns de fornecer esses dados incluem arquivos, bancos de dados, mecanismos de consulta e compartilhamento de dados. Vejamos brevemente cada um.

troca de arquivos

A troca de arquivos é onipresente no serviço de dados. Processamos dados e geramos arquivos para passar aos consumidores de dados.

Tenha em mente que um arquivo pode ser usado para muitas finalidades. Um cientista de dados pode carregar um arquivo de texto (dados não estruturados) de mensagens de clientes para analisar os sentimentos das reclamações dos clientes. Uma unidade de negócios pode receber dados de fatura de uma empresa parceira como uma coleção de CSVs (dados estruturados) e um analista deve realizar algumas análises estatísticas nesses arquivos. Ou, um fornecedor de dados pode fornecer a um varejista on-line imagens de produtos no site de um concorrente (dados não estruturados) para classificação automatizada usando visão computacional.

A maneira como você exibe os arquivos depende de vários fatores, como estes:

- Caso de uso — análise de negócios, análise operacional, análise incorporada
- Os processos de tratamento de dados do consumidor de dados
- O tamanho e o número de arquivos individuais armazenados
- Quem está acessando este arquivo
- Tipo de dados—estruturado, semiestruturado ou não estruturado

O segundo ponto é uma das principais considerações. Frequentemente, é necessário fornecer dados por meio de arquivos, em vez de compartilhamento de dados, porque o consumidor de dados não pode usar uma plataforma de compartilhamento.

O arquivo mais simples de servir é algo como enviar por e-mail um único arquivo do Excel. Este ainda é um fluxo de trabalho comum, mesmo em uma época em que os arquivos podem ser compartilhados de forma colaborativa. O problema com o envio de arquivos por e-mail é que cada destinatário obtém sua versão do arquivo. Se um destinatário editar o arquivo, essas edições serão específicas do arquivo desse usuário. Desvios entre arquivos são inevitáveis. E o que acontece se você não quiser mais que o destinatário tenha acesso ao arquivo? Se o arquivo for enviado por e-mail, você terá poucos recursos para recuperá-lo. Se você precisar de uma versão coerente e consistente de um arquivo, sugerimos o uso de uma plataforma de colaboração como Microsoft 365 ou Google Docs.

Claro, servir arquivos únicos é difícil de escalar, e suas necessidades acabarão superando o armazenamento simples de arquivos em nuvem. Você provavelmente se tornará um depósito de armazenamento de objetos se tiver um punhado de arquivos grandes ou um data lake se tiver um suprimento constante de arquivos. O armazenamento de objetos pode armazenar qualquer tipo de arquivo blob e é especialmente útil para arquivos semiestruturados ou não estruturados.

Observaremos que geralmente consideramos a troca de arquivos por meio do armazenamento de objetos (data lake) como “compartilhamento de dados” em vez de troca de arquivos, pois o processo pode ser significativamente mais escalável e simplificado do que a troca de arquivos ad hoc.

bancos de dados

Os bancos de dados são uma camada crítica no fornecimento de dados para análises e ML. Para esta discussão, manteremos nosso foco implícito em servir dados de bancos de dados OLAP (por exemplo, data warehouses e data lakes). No capítulo anterior, você aprendeu sobre como consultar bancos de dados. Servir dados envolve consultar um banco de dados e, em seguida, consumir esses resultados para um caso de uso. Um analista ou cientista de dados pode consultar um banco de dados usando um editor SQL e exportar esses resultados para um arquivo CSV para consumo por um aplicativo downstream ou analisar os resultados em um notebook (descrito em [“Servindo dados em notebooks” na página 356](#)).

Servir dados de um banco de dados traz uma variedade de benefícios. Um banco de dados impõe ordem e estrutura aos dados por meio do esquema; os bancos de dados podem oferecer informações detalhadas

controles de permissão em nível de tabela, coluna e linha, permitindo que administradores de banco de dados criem políticas de acesso complexas para várias funções; e os bancos de dados podem oferecer alto desempenho de serviço para consultas grandes e com uso intensivo de computação e alta simultaneidade de consulta.

Os sistemas de BI geralmente compartilham a carga de trabalho de processamento de dados com um banco de dados de origem, mas o limite entre o processamento nos dois sistemas varia. Por exemplo, um servidor Tableau executa uma consulta inicial para obter dados de um banco de dados e os armazena localmente. O slicing and dicing OLAP/BI básico (filtragem e agregação interativa) é executado diretamente no servidor a partir da cópia de dados local. Por outro lado, o Looker (e sistemas de BI modernos semelhantes) conta com um modelo computacional chamado *pushdown de consulta*; O Looker codifica a lógica de processamento de dados em uma linguagem especializada (LookML), combina isso com a entrada dinâmica do usuário para gerar consultas SQL, executa-as no banco de dados de origem e apresenta a saída. (Ver “[Camadas Semânticas e Métricas](#)” na [página 355](#).) Tanto o Tableau quanto o Looker têm várias opções de configuração para armazenar resultados em cache para reduzir a carga de processamento de consultas executadas com frequência.

Um cientista de dados pode se conectar a um banco de dados, extrair dados e executar engenharia e seleção de recursos. Esse conjunto de dados convertido é alimentado em um modelo de ML; o modelo off-line é treinado e produz resultados preditivos.

Os engenheiros de dados geralmente têm a tarefa de gerenciar a camada de serviço do banco de dados. Isso inclui o gerenciamento de desempenho e custos. Em bancos de dados que separam computação e armazenamento, esse é um problema de otimização um pouco mais util do que nos dias de infraestrutura local fixa. Por exemplo, agora é possível criar um novo cluster Spark ou armazém Snowflake para cada carga de trabalho analítica ou de ML.

Geralmente, é recomendável dividir os clusters pelos principais casos de uso, como ETL e servir para análise e ciência de dados. Freqüentemente, as equipes de dados optam por segmentar mais detalhadamente, atribuindo um depósito por área principal. Isso possibilita que equipes diferentes orçem seus custos de consulta sob a supervisão de uma equipe de engenharia de dados.

Além disso, lembre-se das três considerações de desempenho que discutimos em “[Análise incorporada](#)” na [página 358](#). Estes são latência de dados, desempenho de consulta e simultaneidade. Um sistema que pode ingerir diretamente de um stream pode diminuir a latência de dados. E muitas arquiteturas de banco de dados contam com SSD ou cache de memória para aprimorar o desempenho de consulta e a simultaneidade para atender aos casos de uso desafiadores inerentes à análise incorporada.

Cada vez mais, plataformas de dados como Snowflake e Databricks permitem que analistas e cientistas de dados operem em um único ambiente, fornecendo editores SQL e notebooks de ciência de dados sob o mesmo teto. Como a computação e o armazenamento são separados, os analistas e cientistas de dados podem consumir os dados subjacentes de várias maneiras sem interferir uns nos outros. Isso permitirá alto rendimento e entrega mais rápida de produtos de dados para as partes interessadas.

Sistemas de streaming

A análise de streaming é cada vez mais importante no domínio do serviço. Em um nível alto, entenda que esse tipo de serviço pode envolver *métricas emitidas*, que são diferentes das consultas tradicionais.

Além disso, vemos bancos de dados de análise operacional desempenhando um papel crescente nessa área (consulte “[Análise operacional](#)” na página 346). Esses bancos de dados permitem que as consultas sejam executadas em uma grande variedade de dados históricos, abrangendo dados atuais atualizados.

Essencialmente, eles combinam aspectos de bancos de dados OLAP com sistemas de processamento de fluxo. Cada vez mais, você trabalhará com sistemas de streaming para fornecer dados para análise e ML, portanto, familiarize-se com esse paradigma.

Você aprendeu sobre sistemas de streaming ao longo do livro. Para ter uma ideia de para onde está indo, leia sobre a pilha de dados ao vivo em [Capítulo 11](#).

Consultar Federação

Como você aprendeu em [Capítulo 8](#), a federação de consulta extrai dados de várias fontes, como data lakes, RDBMSs e data warehouses. A federação está se tornando mais popular à medida que os mecanismos de virtualização de consultas distribuídas ganham reconhecimento como formas de atender consultas sem passar pelo problema de centralizar dados em um sistema OLAP. Hoje, você pode encontrar opções de OSS como Trino e Presto e serviços gerenciados como Starburst. Algumas dessas ofertas se descrevem como formas de habilitar a malha de dados; o tempo dirá como isso se desenvolverá.

Ao fornecer dados para consultas federadas, você deve estar ciente de que o usuário final pode estar consultando vários sistemas — OLTP, OLAP, APIs, sistemas de arquivos, etc. ([Figura 9-3](#)). Em vez de fornecer dados de um único sistema, agora você está fornecendo dados de vários sistemas, cada um com seus padrões de uso, peculiaridades e nuances. Isso representa desafios para o fornecimento de dados. Se as consultas federadas tocarem em sistemas de origem de produção ao vivo, você deverá garantir que a consulta federada não consuma recursos excessivos na origem.

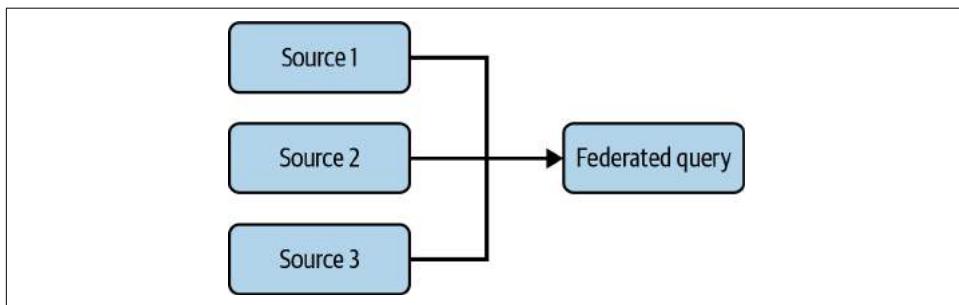


Figura 9-3. Uma consulta federada com três fontes de dados

Em nossa experiência, as consultas federadas são ideais quando você deseja flexibilidade na análise de dados ou quando os dados de origem precisam ser rigorosamente controlados. A federação permite consultas ad hoc para realizar análises exploratórias, combinando dados de vários sistemas sem a complexidade de configurar pipelines de dados ou ETL. Isso permitirá que você determine se o desempenho de uma consulta federada é suficiente para propósitos contínuos ou se você precisa configurar a ingestão em algumas ou todas as fontes de dados e centralizar os dados em um banco de dados OLAP ou data lake.

As consultas federadas também fornecem acesso somente leitura aos sistemas de origem, o que é ótimo quando você não deseja fornecer arquivos, acesso ao banco de dados ou despejos de dados. O usuário final lê apenas a versão dos dados que deve acessar e nada mais. A federação de consulta é uma ótima opção para explorar situações em que o acesso e a conformidade são críticos.

Compartilhamento de dados

[capítulo 5](#) inclui uma extensa discussão sobre compartilhamento de dados. Qualquer troca de dados entre organizações ou unidades dentro de uma organização maior pode ser vista como compartilhamento de dados. Ainda assim, queremos dizer especificamente o compartilhamento por meio de sistemas de armazenamento massivamente multitenant em um ambiente de nuvem. O compartilhamento de dados geralmente transforma o serviço de dados em um problema de segurança e controle de acesso.

As consultas reais agora são tratadas pelos consumidores de dados (analistas e cientistas de dados) em vez dos engenheiros que fornecem os dados. Seja servindo dados em uma malha de dados dentro de uma organização, fornecendo dados ao público ou servindo a empresas parceiras, o compartilhamento de dados é um modelo de serviço atraente. O compartilhamento de dados é cada vez mais um recurso central das principais plataformas de dados, como Snowflake, Redshift e BigQuery, permitindo que as empresas compartilhem dados com segurança entre si.

Camadas semânticas e métricas

Quando os engenheiros de dados pensam em servir, eles naturalmente tendem a gravitar em torno das tecnologias de processamento e armazenamento de dados, ou seja, você usará o Spark ou um data warehouse na nuvem? Seus dados são armazenados em armazenamento de objetos ou em cache em uma frota de SSDs? Mas poderosos mecanismos de processamento que fornecem resultados de consulta rápidos em vastos conjuntos de dados não contribuem inherentemente para análises de negócios de qualidade. Quando alimentados com dados de baixa qualidade ou consultas de baixa qualidade, mecanismos de consulta poderosos retornam rapidamente resultados ruins.

Onde a qualidade dos dados se concentra nas características dos próprios dados e em várias técnicas para filtrar ou melhorar dados ruins, a qualidade da consulta é uma questão de construir uma consulta com lógica apropriada que retorne respostas precisas para questões de negócios. Escrever consultas e relatórios ETL de alta qualidade é um trabalho detalhado e demorado. Várias ferramentas podem ajudar a automatizar esse processo, facilitando a consistência, a manutenção e a melhoria contínua.

Fundamentalmente, um *camada de métricas* é uma ferramenta para manter e computar a lógica de negócios.⁴ (A *camada semântica* é extremamente similar conceitualmente, se *BI* sem *cabeça* é outro termo intimamente relacionado.) Essa camada pode residir em uma ferramenta de BI ou em um software que cria consultas de transformação. Dois exemplos concretos são Looker e Data Build Tool (dbt).

Por exemplo, o LookML da Looker permite que os usuários definam uma lógica de negócios complexa e virtual. Relatórios e painéis apontam para LookML específico para métricas de computação. O Looker permite que os usuários definam métricas padrão e as referenciem em muitas consultas downstream; isso visa resolver o problema tradicional de repetição e inconsistência em scripts ETL tradicionais. O Looker usa o LookML para gerar consultas SQL, que são enviadas para o banco de dados. Os resultados podem ser mantidos no servidor Looker ou no próprio banco de dados para grandes conjuntos de resultados.

O dbt permite que os usuários definam fluxos de dados SQL complexos que abrangem muitas consultas e definições padrão de métricas de negócios, muito parecido com o Looker. Ao contrário do Looker, o dbt é executado exclusivamente na camada de transformação, embora isso possa incluir o envio de consultas para exibições que são computadas no momento da consulta. Enquanto o Looker se concentra em fornecer consultas e relatórios, o dbt pode servir como uma ferramenta robusta de orquestração de pipeline de dados para engenheiros de análise.

Acreditamos que as ferramentas da camada de métricas se tornarão mais populares com adoção mais ampla e mais participantes, bem como avançarão em direção ao aplicativo. As ferramentas da camada de métricas ajudam a resolver uma questão central na análise que tem atormentado as organizações desde que as pessoas analisaram os dados: “Esses números estão corretos?” Muitos novos participantes estão no espaço ao lado dos que mencionamos.

Servindo dados em notebooks

Os cientistas de dados costumam usar notebooks em seu trabalho diário. Seja explorando dados, recursos de engenharia ou treinando um modelo, o cientista de dados provavelmente usará um notebook. Até o momento, a plataforma de notebook mais popular é o Jupyter Notebook, juntamente com sua iteração de próxima geração, o JupyterLab. O Jupyter é de código aberto e pode ser hospedado localmente em um laptop, em um servidor ou por meio de vários serviços gerenciados em nuvem. *Jupyter*, *Julia*, *Python* e *R*—os dois últimos são populares para aplicativos de ciência de dados, especialmente notebooks. Independentemente do idioma usado, a primeira coisa que você precisa considerar é como os dados podem ser acessados de um notebook.

Os cientistas de dados se conectarão programaticamente a uma fonte de dados, como uma API, um banco de dados, um data warehouse ou um data lake (Figura 9-4). Em um notebook, todas as conexões

⁴ Benn Stancil, “A peça que faltava na moderna pilha de dados,” *benn.substack*, 22 de abril de 2021, <https://oreil.ly/wQyPb>.

⁵ Srinivas Kadamatli, “Understanding the Superset Semantic Layer”, blog Preset, 21 de dezembro de 2021, <https://oreil.ly/6smWC>.

são criados usando as bibliotecas incorporadas ou importadas apropriadas para carregar um arquivo de um caminho de arquivo, conectar-se a um terminal de API ou fazer uma conexão ODBC com um banco de dados. Uma conexão remota pode exigir as credenciais e privilégios corretos para estabelecer uma conexão. Uma vez conectado, um usuário pode precisar do acesso correto às tabelas (e linhas/colunas) ou arquivos armazenados no armazenamento de objetos. O engenheiro de dados frequentemente ajudará o cientista de dados a encontrar os dados corretos e, em seguida, garantir que eles tenham as permissões corretas para acessar as linhas e colunas necessárias.

Vejamos um fluxo de trabalho incrivelmente comum para cientistas de dados: executar um notebook local e carregar dados em um dataframe do pandas.*pandas* é uma biblioteca Python predominante usada para manipulação e análise de dados e é comumente usada para carregar dados (digamos, um arquivo CSV) em um notebook Jupyter. Quando os pandas carregam um conjunto de dados, ele armazena esse conjunto de dados na memória.

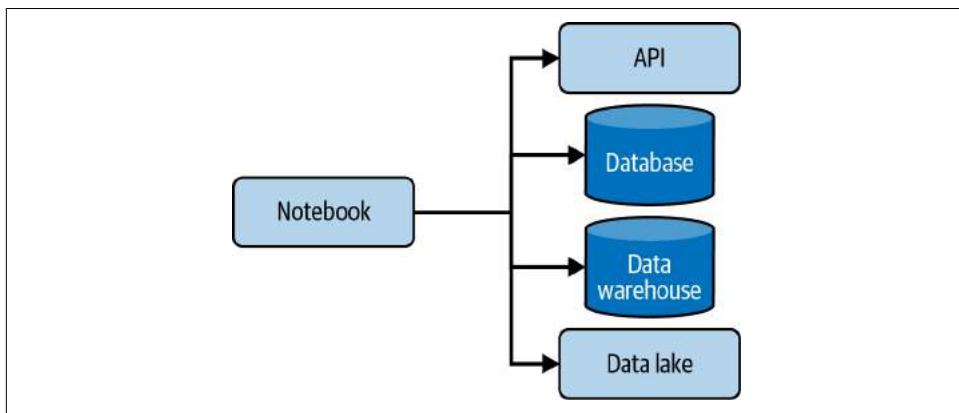


Figura 9-4. Um notebook pode receber dados de várias fontes, como armazenamento de objetos ou banco de dados, data warehouse ou data lake

Manipulação de credenciais

Credenciais manipuladas incorretamente em notebooks e código de ciência de dados são um grande risco de segurança; vemos constantemente credenciais maltratadas neste domínio. É comum incorporar credenciais diretamente no código, onde geralmente vazam para repositórios de controle de versão. As credenciais também são frequentemente repassadas por meio de mensagens e e-mail.

Incentivamos os engenheiros de dados a auditar as práticas de segurança da ciência de dados e trabalhar de forma colaborativa nas melhorias. Os cientistas de dados são altamente receptivos a essas conversas se tiverem alternativas. Os engenheiros de dados devem definir padrões para lidar com credenciais. As credenciais nunca devem ser incorporadas ao código; idealmente, os cientistas de dados usam gerenciadores de credenciais ou ferramentas CLI para gerenciar o acesso.

O que acontece quando o tamanho do conjunto de dados excede a memória disponível da máquina local? Isso inevitavelmente acontece devido à memória limitada de laptops e estações de trabalho: interrompe um projeto de ciência de dados no meio do caminho. É hora de considerar opções mais escaláveis. Primeiro, mude para um notebook baseado em nuvem, onde o armazenamento e a memória subjacentes do notebook podem ser dimensionados de forma flexível. Ao superar essa opção, observe os sistemas de execução distribuídos; opções populares baseadas em Python incluem Dask, Ray e Spark. Se uma oferta completa de gerenciamento em nuvem parecer atraente, considere configurar um fluxo de trabalho de ciência de dados usando Amazon SageMaker, Google Cloud Vertex AI ou Microsoft Azure Machine Learning. Por fim, as opções de fluxo de trabalho de ML de ponta a ponta de código aberto, como Kubeflow e MLflow, facilitam o escalonamento de cargas de trabalho de ML em Kubernetes e Spark, respectivamente.

Engenheiros de dados e engenheiros de ML desempenham um papel fundamental ao facilitar a mudança para uma infraestrutura de nuvem escalonável. A divisão exata do trabalho depende muito dos detalhes de sua organização. Eles devem assumir a liderança na configuração da infraestrutura de nuvem, supervisionando o gerenciamento de ambientes e treinando cientistas de dados em ferramentas baseadas em nuvem.

Os ambientes de nuvem exigem trabalho operacional significativo, como gerenciamento de versões e atualizações, controle de acesso e manutenção de SLAs. Assim como em outros trabalhos operacionais, uma recompensa significativa pode ocorrer quando as “operações de ciência de dados” são bem executadas.

Os notebooks podem até se tornar parte da ciência de dados de produção; notebooks são amplamente implantados na Netflix. Esta é uma abordagem interessante com vantagens e compensações. Os notebooks produzidos permitem que os cientistas de dados coloquem seu trabalho em produção muito mais rapidamente, mas também são inherentemente uma forma de produção abaixo do padrão. A alternativa é fazer com que o ML e os engenheiros de dados convertam os notebooks para uso em produção, sobrecarregando significativamente essas equipes. Um híbrido dessas abordagens pode ser ideal, com notebooks usados para produção “leve” e um processo de produção completo para projetos de alto valor.

ETL reverso

Hoje, *ETL reverso* é uma palavra da moda que descreve o fornecimento de dados carregando-os de um banco de dados OLAP de volta para um sistema de origem. Dito isso, qualquer engenheiro de dados que trabalhou no campo por mais de alguns anos provavelmente fez alguma variação de ETL reverso. O ETL reverso cresceu em popularidade no final dos anos 2010/início dos anos 2020 e é cada vez mais reconhecido como uma responsabilidade formal da engenharia de dados.

Um engenheiro de dados pode extrair clientes e solicitar dados de um CRM e armazená-los em um data warehouse. Esses dados são usados para treinar um modelo de pontuação de lead, cujos resultados são retornados ao data warehouse. A equipe de vendas da sua empresa deseja ter acesso a esses leads pontuados para tentar gerar mais vendas. Você tem algumas opções para obter os resultados de

este modelo de pontuação de leads nas mãos da equipe de vendas. Você pode colocar os resultados em um painel para que eles visualizem. Ou você pode enviar os resultados por e-mail para eles como um arquivo do Excel.

O desafio dessas abordagens é que elas não estão conectadas ao CRM, onde um vendedor faz seu trabalho. Por que não apenas colocar os leads pontuados de volta no CRM? Como mencionamos, produtos de dados bem-sucedidos reduzem o atrito com o usuário final. Nesse caso, o usuário final é a equipe de vendas.

Usar o ETL reverso e carregar os leads pontuados de volta no CRM é a abordagem mais fácil e melhor para esse produto de dados. O ETL reverso pega os dados processados do lado de saída do ciclo de vida da engenharia de dados e os alimenta de volta nos sistemas de origem ([Figura 9-5](#)).



Em vez de ETL reverso, nós, os autores, o chamamos meio de brincadeira **carga e transformação bidirecional (BLT)**. O termo *ETL reverse* não descreve com precisão o que está acontecendo neste processo. Independentemente disso, o termo ficou preso na imaginação popular e na imprensa, então vamos usá-lo ao longo do livro. Mais amplamente, se o termo *ETL reverse* por aí é uma incógnita, mas a prática de carregar dados de sistemas OLAP de volta para os sistemas de origem continuará sendo importante.

Como você começa a servir dados com ETL reverso? Embora você possa implementar sua solução de ETL reverso, muitas opções de ETL reverso prontas para uso estão disponíveis. Sugerimos o uso de código aberto ou um serviço comercial gerenciado. Dito isso, o espaço ETL reverso está mudando com extrema rapidez. Nenhum vencedor claro surgiu e muitos produtos ETL reversos serão absorvidos pelas principais nuvens ou outros fornecedores de produtos de dados. Escolha cuidadosamente.

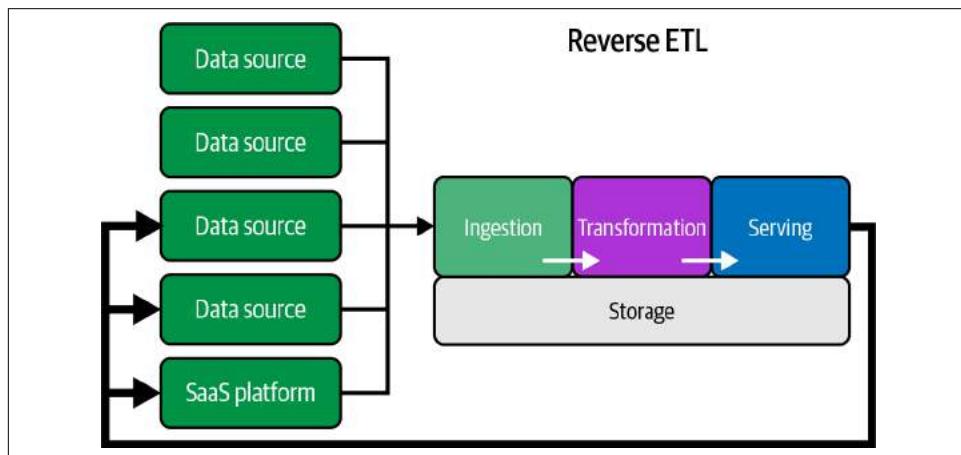


Figura 9-5. ETL reverso

Temos algumas palavras de advertência sobre ETL reverso. O ETL reverso cria inherentemente loops de feedback. Por exemplo, imagine que baixamos os dados do Google Ads, usamos um modelo para calcular novos lances, carregamos os lances de volta no Google Ads e iniciamos o processo novamente. Suponha que, devido a um erro em seu modelo de lance, a tendência dos lances seja cada vez maior e seus anúncios recebam cada vez mais cliques. Você pode rapidamente desperdiçar grandes quantias de dinheiro! Tenha cuidado e construa monitoramento e grades de proteção.

Com quem você trabalhará

Conforme discutimos, no estágio de serviço, um engenheiro de dados fará a interface com muitas partes interessadas. Estes incluem (mas não estão limitados a) o seguinte:

- Analistas de dados
- Cientistas de dados
- Engenheiros de MLOps/ML
- O negócio - partes interessadas, gerentes e executivos não relacionados a dados ou técnicos

Como lembrete, o engenheiro de dados opera em *um apoio/papel* para essas partes interessadas e não é necessariamente responsável pelos usos finais dos dados. Por exemplo, um engenheiro de dados fornece os dados para um relatório que os analistas interpretam, mas o engenheiro de dados não é responsável por essas interpretações. Em vez disso, o engenheiro de dados é responsável por produzir produtos de dados da mais alta qualidade possível.

Um engenheiro de dados deve estar ciente dos ciclos de feedback entre o ciclo de vida da engenharia de dados e o uso mais amplo dos dados quando estiverem nas mãos das partes interessadas. Os dados raramente são estáticos, e o mundo externo influenciará os dados que são ingeridos, servidos, reingeridos e reservados.

Uma grande consideração para os engenheiros de dados no estágio de serviço do ciclo de vida é a separação de deveres e preocupações. Se você estiver em uma empresa em estágio inicial, o engenheiro de dados também pode ser um engenheiro de ML ou cientista de dados; isso não é sustentável. À medida que a empresa cresce, você precisa estabelecer uma divisão clara de tarefas com outros membros da equipe de dados.

Adotar uma malha de dados reorganiza drasticamente as responsabilidades da equipe, e cada equipe de domínio assume aspectos de atendimento. Para que uma malha de dados seja bem-sucedida, cada equipe deve trabalhar efetivamente em suas responsabilidades de serviço de dados, e as equipes também devem colaborar efetivamente para garantir o sucesso organizacional.

Subcorrentes

As correntes ocultas chegam ao fim com o saque. Lembre-se de que o ciclo de vida da engenharia de dados é apenas isso: um ciclo de vida. O que vai, volta. Vemos muitos

instâncias em que a exibição de dados destaca algo perdido no início do ciclo de vida. Esteja sempre atento a como as correntes ocultas podem ajudá-lo a identificar maneiras de melhorar os produtos de dados.

Gostamos de dizer: "Os dados são um assassino silencioso", e as correntes ocultas vêm à tona no estágio de saque. A exibição é sua última chance de garantir que seus dados estejam em ótima forma antes de chegarem às mãos dos usuários finais.

Segurança

Os mesmos princípios de segurança se aplicam ao compartilhar dados com pessoas ou sistemas. Frequentemente vemos dados compartilhados indiscriminadamente, com pouco ou nenhum controle de acesso ou pensamento sobre para que os dados serão usados. Este é um grande erro que pode ter resultados catastróficos, como violação de dados e as multas resultantes, má imprensa e perda de empregos. Leve a segurança a sério, especialmente nesta fase do ciclo de vida. De todos os estágios do ciclo de vida, o serviço apresenta a maior superfície de segurança.

Como sempre, exerce o princípio do menor privilégio para pessoas e sistemas e forneça apenas o acesso necessário para o objetivo em questão e o trabalho a ser realizado. De quais dados um executivo precisa em comparação com um analista ou cientista de dados? E quanto a um pipeline de ML ou processo ETL reverso? Esses usuários e destinos têm diferentes necessidades de dados e o acesso deve ser fornecido de acordo. Evite dar permissões de carta branca para tudo e todos.

O fornecimento de dados geralmente é somente leitura, a menos que uma pessoa ou processo precise atualizar os dados no sistema do qual são consultados. As pessoas devem receber acesso somente leitura a bancos de dados e conjuntos de dados específicos, a menos que sua função exija algo mais avançado, como acesso de gravação ou atualização. Isso pode ser feito combinando grupos de usuários com determinadas funções do IAM (ou seja, grupo de analistas, grupo de cientistas de dados) ou funções personalizadas do IAM, se isso fizer sentido. Para sistemas, forneça contas de serviço e funções de maneira semelhante. Para usuários e sistemas, restrinja o acesso aos campos, linhas, colunas e células de um conjunto de dados, se necessário. Os controles de acesso devem ser o mais refinados possível e revogados quando o acesso não for mais necessário.

Os controles de acesso são críticos ao fornecer dados em um ambiente multilocatário. Certifique-se de que os usuários possam acessar apenas *de/lesdados* e nada mais. Uma boa abordagem é mediar o acesso por meio de exibições filtradas, aliviando assim os riscos de segurança inerentes ao compartilhamento de acesso a uma tabela comum. Outra sugestão é usar o compartilhamento de dados em seus fluxos de trabalho, o que permite controles granulares somente leitura entre você e as pessoas que consomem seus dados.

Verifique com que frequência os produtos de dados são usados e se faz sentido parar de compartilhar determinados produtos de dados. É extremamente comum para um executivo solicitar urgentemente a um analista para criar um relatório, apenas para que esse relatório rapidamente não seja utilizado. Se

produtos de dados não forem usados, pergunte aos usuários se eles ainda são necessários. Se não, elimine o produto de dados. Isso significa uma vulnerabilidade de segurança a menos flutuando.

Por fim, você deve ver o controle de acesso e a segurança não como impedimentos para servir, mas como facilitadores importantes. Estamos cientes de muitos casos em que sistemas de dados complexos e avançados foram criados, podendo ter um impacto significativo em uma empresa. Como a segurança não foi implementada corretamente, poucas pessoas tiveram permissão para acessar os dados, por isso definhou. O controle de acesso refinado e robusto significa que análises de dados e ML mais interessantes podem ser feitas enquanto ainda protege a empresa e seus clientes.

Gestão de dados

Você vem incorporando o gerenciamento de dados ao longo do ciclo de vida da engenharia de dados, e o impacto de seus esforços logo se tornará aparente à medida que as pessoas usarem seus produtos de dados. No estágio de veiculação, você se preocupa principalmente em garantir que as pessoas possam acessar dados confiáveis e de alta qualidade.

Como mencionamos no início deste capítulo, a confiança é talvez a variável mais crítica no serviço de dados. Se as pessoas confiarem em seus dados, elas os usarão; dados não confiáveis não serão usados. Certifique-se de tornar a confiança e a melhoria dos dados um processo ativo, fornecendo loops de feedback. À medida que os usuários interagem com os dados, eles podem relatar problemas e solicitar melhorias. Comunique-seativamente com seus usuários à medida que as alterações são feitas.

De quais dados as pessoas precisam para realizar seus trabalhos? Especialmente com questões regulatórias e de conformidade pesando sobre as equipes de dados, dar às pessoas acesso aos dados brutos – mesmo com campos e linhas limitados – representa um problema de rastreamento de dados até uma entidade, como uma pessoa ou um grupo de pessoas. Felizmente, os avanços na ofuscação de dados permitem que você forneça dados sintéticos, embaralhados ou anonimizados aos usuários finais. Esses conjuntos de dados “falsos” devem permitir que um analista ou cientista de dados obtenha o sinal necessário dos dados, mas de uma forma que dificulte a identificação de informações protegidas. Embora este não seja um processo perfeito - com esforço suficiente, muitos conjuntos de dados podem ser anonimizados ou submetidos a engenharia reversa - pelo menos reduz o risco de vazamento de dados.

Além disso, incorpore camadas semânticas e métricas em sua camada de serviço, juntamente com uma modelagem de dados rigorosa que expressa adequadamente a lógica e as definições de negócios. Isso fornece uma única fonte de verdade, seja para análise, ML, ETL reverso ou outros usos de serviço.

DataOps

As etapas que você executa no gerenciamento de dados - qualidade, governança e segurança dos dados - são monitoradas no DataOps. Essencialmente, DataOps operacionaliza o gerenciamento de dados. A seguir estão algumas coisas para monitorar:

- Saúde dos dados e tempo de inatividade dos dados
- Latência dos sistemas que atendem aos dados — painéis, bancos de dados, etc.
- Qualidade dos dados
- Segurança e acesso a dados e sistemas
- Versões de dados e modelo sendo veiculadas
- Tempo de atividade para atingir um SLO

Uma variedade de novas ferramentas surgiram para abordar vários aspectos de monitoramento. Por exemplo, muitas ferramentas populares de observabilidade de dados visam minimizar *tempo de inatividade de dados* e maximizar a qualidade dos dados. As ferramentas de observabilidade podem passar de dados para ML, suportando monitoramento de modelos e desempenho do modelo. O monitoramento de DevOps mais convencional também é crítico para DataOps — por exemplo, você precisa monitorar se as conexões estão estáveis entre armazenamento, transformação e serviço.

Como em cada estágio do ciclo de vida da engenharia de dados, controle de versão do código e operacionalize a implantação. Isso se aplica a código analítico, código de lógica de dados, scripts de ML e trabalhos de orquestração. Use vários estágios de implantação (desenvolvimento, teste, produção) para relatórios e modelos.

Arquitetura de dados

Os dados de serviço devem ter as mesmas considerações arquitetônicas que outros estágios do ciclo de vida da engenharia de dados. No estágio de servir, os ciclos de feedback devem ser rápidos e estreitos. Os usuários devem poder acessar os dados de que precisam o mais rápido possível quando precisam.

Os cientistas de dados são conhecidos por fazer a maior parte do desenvolvimento em suas máquinas locais. Conforme discutido anteriormente, incentive-os a migrar esses fluxos de trabalho para sistemas comuns em um ambiente de nuvem, onde as equipes de dados podem colaborar em ambientes de desenvolvimento, teste e produção e criar arquiteturas de produção adequadas. Facilite seus analistas e cientistas de dados com ferramentas de suporte para publicar insights de dados com pouco ônus.

Orquestração

O serviço de dados é o último estágio do ciclo de vida da engenharia de dados. Como o atendimento está a jusante de tantos processos, é uma área de sobreposição extremamente complexa. A orquestração não é simplesmente uma forma de organizar e automatizar o trabalho complexo, mas um meio de coordenar o fluxo de dados entre as equipes para que os dados sejam disponibilizados aos consumidores no tempo prometido.

A propriedade da orquestração é uma decisão organizacional fundamental. A orquestração será centralizada ou descentralizada? Uma abordagem descentralizada permite que pequenas equipes gerenciem

seus fluxos de dados, mas pode aumentar a carga de coordenação entre equipes. Em vez de simplesmente gerenciar fluxos dentro de um único sistema, acionando diretamente a conclusão de DAGs ou tarefas pertencentes a outras equipes, as equipes devem passar mensagens ou consultas entre os sistemas.

Uma abordagem centralizada significa que o trabalho é mais fácil de coordenar, mas também deve existir um controle significativo para proteger um único ativo de produção. Por exemplo, um DAG mal escrito pode interromper o Airflow. A abordagem centralizada significaria reduzir os processos de dados e servir em toda a organização. O gerenciamento centralizado da orquestração requer altos padrões, testes automatizados de DAGs e controle de acesso.

Se a orquestração for centralizada, quem será o dono? Quando uma empresa tem uma equipe de DataOps, a orquestração geralmente chega aqui. Muitas vezes, uma equipe envolvida em servir é um ajuste natural porque tem uma visão bastante holística de todos os estágios do ciclo de vida da engenharia de dados. Podem ser DBAs, engenheiros analíticos, engenheiros de dados ou engenheiros de ML. Os engenheiros de ML coordenam processos complexos de treinamento de modelo, mas podem ou não querer adicionar a complexidade operacional de gerenciar a orquestração a uma lista já lotada de responsabilidades.

Engenharia de software

Comparado a alguns anos atrás, servir dados tornou-se mais simples. A necessidade de escrever código foi drasticamente simplificada. Os dados também se tornaram mais voltados para o código, com a proliferação de estruturas de código aberto focadas em simplificar o fornecimento de dados. Existem muitas maneiras de fornecer dados aos usuários finais, e o foco de um engenheiro de dados deve estar em saber como esses sistemas funcionam e como os dados são entregues.

Apesar da simplicidade de servir dados, se o código estiver envolvido, um engenheiro de dados ainda deve entender como funcionam as principais interfaces de serviço. Por exemplo, um engenheiro de dados pode precisar traduzir o código que um cientista de dados está executando localmente em um notebook e convertê-lo em um relatório ou modelo básico de ML para operar.

Outra área em que os engenheiros de dados serão úteis é entender o impacto de como o código e as consultas serão executados nos sistemas de armazenamento. Os analistas podem gerar SQL de várias maneiras programáticas, incluindo LookML, Jinja via dbt, várias ferramentas de mapeamento relacional de objeto (ORM) e camadas de métricas. Quando essas camadas programáticas compilarem para SQL, como será o desempenho desse SQL? Um engenheiro de dados pode sugerir otimizações em que o código SQL pode não funcionar tão bem quanto o SQL manuscrito.

A ascensão da análise e ML IaC significa que a função de escrever código está se movendo para a construção de sistemas que oferecem suporte a cientistas e analistas de dados. Os engenheiros de dados podem ser responsáveis por configurar os pipelines de CI/CD e criar processos para sua equipe de dados. Eles também fariam bem em treinar e apoiar sua equipe de dados no uso

a infraestrutura de dados/MLOps que eles construíram para que essas equipes de dados possam ser o mais autossuficientes possível.

Para análises incorporadas, os engenheiros de dados podem precisar trabalhar com desenvolvedores de aplicativos para garantir que as consultas sejam retornadas de forma rápida e econômica. O desenvolvedor do aplicativo controlará o código front-end com o qual os usuários lidam. O engenheiro de dados está lá para garantir que os desenvolvedores recebam as cargas corretas conforme solicitado.

Conclusão

O ciclo de vida da engenharia de dados tem um final lógico no estágio de veiculação. Como em todos os ciclos de vida, ocorre um ciclo de feedback ([Figura 9-6](#)). Você deve ver o estágio de serviço como uma chance de aprender o que está funcionando e o que pode ser melhorado. Ouça as partes interessadas. Se eles levantarem questões - e inevitavelmente o farão - tente não se ofender. Em vez disso, use isso como uma oportunidade para melhorar o que você construiu.

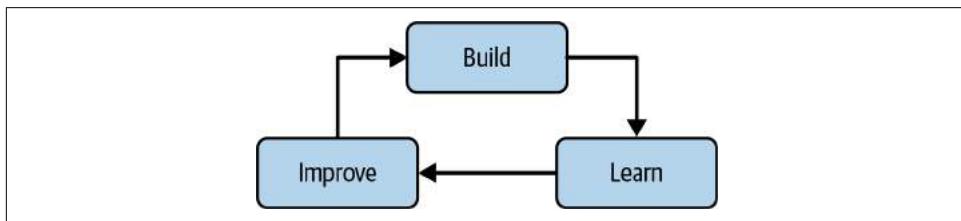


Figura 9-6. Construir, aprender, melhorar

Um bom engenheiro de dados está sempre aberto a novos comentários e constantemente encontra maneiras de melhorar seu ofício. Agora que fizemos uma jornada pelo ciclo de vida da engenharia de dados, você sabe como projetar, arquitetar, construir, manter e melhorar seus sistemas e produtos de engenharia de dados. Vamos voltar nossa atenção para [Parte III](#) do livro, onde abordaremos alguns aspectos da engenharia de dados sobre os quais somos constantemente questionados e que, francamente, merecem mais atenção.

Recursos adicionais

- “[Dados como um produto versus produtos de dados: quais são as diferenças?](#)” por Xavier Gumara Rigol
- “[Data Jujitsu: A Arte de Transformar Dados em Produtos](#)” por DJ Patil
- [Malha de dados](#) por Zhamak Dehghani (O'Reilly)
- “[Princípios de malha de dados e arquitetura lógica](#)” por Zhamak Dehghani
- “[Projetando produtos de dados](#)” por Seth O'Regan
- “[A evolução dos produtos de dados](#)” e “[O que é ciência de dados](#)” por Mike Loukides

- Forrester'sArtigo do blog “Self-Service Business Intelligence: Dissolvendo as Barreiras para Soluções Criativas de Suporte à Decisão”
- “Fundamentos do aprendizado de máquina de autoatendimento”por Paramita (Guha) Ghosh
- “O futuro do BI é sem cabeça”por ZD
- “Como construir ótimos produtos de dados”por Emily Glassberg Sands
- “Como estruturar uma equipe de análise de dados”por Niall Napier
- “Conheça as ‘tarefas a serem realizadas’ de seus clientes”por Clayton M. Christensen e outros.
- “A peça que faltava na pilha de dados moderna”e“Por que o autoatendimento ainda é um problema?”por Benn Stancil
- “Análise de autoatendimento” no Glossário do Gartner
- Dados Ternários“O que vem a seguir para bancos de dados analíticos? com Jordan Tigani (Mãe-Pato)” vídeo
- “Entendendo a camada semântica do superconjunto”por Srini Kadamatli
- “O que o BI de autoatendimento moderno e a análise de dados realmente significam?”por Harry Dix
- “O que é análise operacional (e como isso está mudando a forma como trabalhamos com dados)?”por Sylvain Giuliani
- “O que é análise voltada para o usuário?”por Chinmon Soman

PARTE III

**Segurança, privacidade e o futuro da
Engenharia de Dados**

Segurança e privacidade

Agora que você aprendeu sobre o ciclo de vida da engenharia de dados, gostaríamos de reiterar a importância da segurança e compartilhar algumas práticas simples que você pode incorporar em seu fluxo de trabalho diário. A segurança é vital para a prática da engenharia de dados. Isso deveria ser incrivelmente óbvio, mas estamos constantemente surpresos com a frequência com que os engenheiros de dados veem a segurança como uma reflexão tardia. Acreditamos que a segurança é a primeira coisa em que um engenheiro de dados precisa pensar em todos os aspectos de seu trabalho e em todas as etapas do ciclo de vida da engenharia de dados. Você lida com dados, informações e acessos confidenciais diariamente. Sua organização, clientes e parceiros de negócios esperam que esses ativos valiosos sejam tratados com o máximo cuidado e preocupação. Uma violação de segurança ou um vazamento de dados pode deixar sua empresa na água;

A segurança é um ingrediente chave para a privacidade. A privacidade tem sido fundamental para a confiança no espaço corporativo de tecnologia da informação; os engenheiros lidam direta ou indiretamente com dados relacionados à vida privada das pessoas. Isso inclui informações financeiras, dados sobre comunicações privadas (e-mails, mensagens de texto, telefonemas), histórico médico, registros educacionais e histórico de trabalho. Uma empresa que vazou essas informações ou fez mau uso delas pode se tornar uma pária quando a violação vier à tona.

Cada vez mais, a privacidade é uma questão de importância legal significativa. Por exemplo, o Family Educational Rights and Privacy Act (FERPA) entrou em vigor nos EUA na década de 1970; a Lei de Portabilidade e Responsabilidade de Seguro Saúde (HIPAA) surgiu na década de 1990; O GDPR foi aprovado na Europa em meados da década de 2010. Vários projetos de lei de privacidade baseados nos EUA foram aprovados ou serão aprovados em breve. Esta é apenas uma pequena amostra dos estatutos relacionados à privacidade (e acreditamos ser apenas o começo). Ainda assim, as penalidades pela violação de qualquer uma dessas leis podem ser significativas, até mesmo devastadoras, para uma empresa. E como os sistemas de dados estão integrados na estrutura da educação, saúde e negócios, os engenheiros de dados lidam com dados confidenciais relacionados a cada uma dessas leis.

As responsabilidades exatas de segurança e privacidade de um engenheiro de dados variam significativamente entre as organizações. Em uma pequena startup, um engenheiro de dados pode ter dupla função como engenheiro de segurança de dados. Uma grande empresa de tecnologia terá exércitos de engenheiros de segurança e pesquisadores de segurança. Mesmo nessa situação, os engenheiros de dados geralmente conseguem identificar práticas de segurança e vulnerabilidades de tecnologia em suas próprias equipes e sistemas que podem relatar e mitigar em colaboração com pessoal de segurança dedicado.

Como a segurança e a privacidade são essenciais para a engenharia de dados (sendo a segurança uma subcorrente), queremos dedicar mais tempo abordando a segurança e a privacidade. Neste capítulo, apresentamos algumas coisas que os engenheiros de dados devem considerar em relação à segurança, particularmente em pessoas, processos e tecnologia (nessa ordem). Esta não é uma lista completa, mas apresenta as principais coisas que gostaríamos que melhorassem com base em nossa experiência.

Pessoas

O elo mais fraco em segurança e privacidade é você. A segurança geralmente é comprometida no nível humano, portanto, comporte-se como se você fosse sempre um alvo. Um bot ou ator humano está tentando se infiltrar em suas credenciais e informações confidenciais a qualquer momento. Esta é a nossa realidade e não vai desaparecer. Assuma uma postura defensiva com tudo o que você faz online e offline. Exercite o poder do pensamento negativo e seja sempre paranóico.

O poder do pensamento negativo

Em um mundo obcecado com o pensamento positivo, o pensamento negativo é desagradável. No entanto, o cirurgião americano Atul Gawande escreveu um artigo de opinião em 2007 no *New York Times* precisamente sobre este assunto. Sua tese central é que o pensamento positivo pode nos cegar para a possibilidade de ataques terroristas ou emergências médicas e impedir a preparação. O pensamento negativo nos permite considerar cenários desastrosos e agir para evitá-los.

Os engenheiros de dados devem pensar ativamente nos cenários de utilização de dados e coletar dados confidenciais somente se houver uma necessidade real de downstream. A melhor maneira de proteger dados privados e confidenciais é evitar a ingestão desses dados em primeiro lugar.

Os engenheiros de dados devem pensar nos cenários de ataque e vazamento com qualquer canal de dados ou sistema de armazenamento que utilizam. Ao decidir sobre estratégias de segurança, certifique-se de que sua abordagem ofereça segurança adequada e não apenas a ilusão de segurança.

Seja sempre paranóico

Sempre tenha cuidado quando alguém pedir suas credenciais. Em caso de dúvida - e você sempre deve estar em dúvida extrema quando solicitado a fornecer credenciais, espere e obtenha uma segunda opinião de seus colegas de trabalho e amigos. Confirme com outras pessoas que o pedido é realmente legítimo. Um bate-papo rápido ou uma ligação telefônica é mais barato do que um ataque de ransomware desencadeado por meio de um clique em um e-mail. Não confie em ninguém.

pelo valor nominal quando solicitadas credenciais, dados confidenciais ou informações confidenciais, inclusive de seus colegas de trabalho.

Você também é a primeira linha de defesa no respeito à privacidade e à ética. Você se sente desconfortável com os dados confidenciais que foi encarregado de coletar? Você tem dúvidas éticas sobre a forma como os dados estão sendo tratados em um projeto? Exponha suas preocupações aos colegas e à liderança. Certifique-se de que seu trabalho seja legalmente compatível e ético.

Processos

Quando as pessoas seguem processos de segurança regulares, a segurança se torna parte do trabalho. Torne a segurança um hábito, pratique regularmente segurança real, pratique o princípio do menor privilégio e entenda o modelo de responsabilidade compartilhada na nuvem.

Teatro de segurança versus hábito de segurança

Com nossos clientes corporativos, vemos um foco generalizado em conformidade (com regras internas, leis, recomendações de órgãos normativos), mas não atenção suficiente para cenários potencialmente ruins. Infelizmente, isso cria uma ilusão de segurança, mas muitas vezes deixa lacunas que seriam evidentes com alguns minutos de reflexão.

A segurança precisa ser simples e eficaz o suficiente para se tornar habitual em toda a organização. Estamos impressionados com o número de empresas com políticas de segurança nas centenas de páginas que ninguém lê, a revisão anual da política de segurança que as pessoas esquecem imediatamente, tudo marcando uma caixa para uma auditoria de segurança. Este é o teatro de segurança, onde a segurança é feita na carta de conformidade (SOC-2, ISO 27001 e relacionadas) sem real compromisso.

Em vez disso, busque o espírito de segurança genuína e habitual; incorpore uma mentalidade de segurança em sua cultura. A segurança não precisa ser complicada. Por exemplo, em nossa empresa, realizamos treinamento de segurança e revisão de políticas pelo menos uma vez por mês para incorporar isso ao DNA de nossa equipe e atualizar uns aos outros sobre as práticas de segurança que podemos melhorar. A segurança não deve ser uma reflexão tardia para sua equipe de dados. Todos são responsáveis e têm um papel a desempenhar. Deve ser a prioridade para você e todos os outros com quem você trabalha.

Segurança ativa

Voltando à ideia do pensamento negativo, *segurança ativa* envolve pensar e pesquisar ameaças de segurança em um mundo dinâmico e em mudança. Em vez de simplesmente implantar ataques de phishing simulados programados, você pode adotar uma postura de segurança ativa pesquisando ataques de phishing bem-sucedidos e pensando nas vulnerabilidades de segurança de sua organização. Em vez de simplesmente adotar uma lista de verificação de conformidade padrão, você pode pensar nas vulnerabilidades internas específicas da sua organização e nos incentivos que os funcionários podem ter para vazar ou fazer uso indevido de informações privadas.

Temos mais a dizer sobre segurança ativa em “[Tecnologia](#)” na página 374.

O Princípio do Menor Privilégio

O *princípio do menor privilégio* significa que uma pessoa ou sistema deve receber apenas os privilégios e dados necessários para concluir a tarefa em questão e nada mais.

Frequentemente, vemos um antipadrão na nuvem: um usuário regular recebe acesso administrativo a tudo, quando essa pessoa pode precisar de apenas algumas funções do IAM para realizar seu trabalho. Dar a alguém carta branca para acesso administrativo é um grande erro e nunca deveria acontecer de acordo com o princípio do menor privilégio.

Em vez disso, forneça ao usuário (ou grupo ao qual ele pertence) as funções do IAM de que ele precisa quando ele precisa. Quando essas funções não forem mais necessárias, retire-as. A mesma regra se aplica às contas de serviço. Trate humanos e máquinas da mesma forma: dê a eles apenas os privilégios e os dados de que precisam para realizar seus trabalhos e apenas durante o período de tempo necessário.

Claro, o princípio do menor privilégio também é crítico para a privacidade. Seus usuários e clientes esperam que as pessoas vejam seus dados confidenciais somente quando necessário. Certifique-se de que este é o caso. Implemente controles de acesso em nível de coluna, linha e célula em torno de dados confidenciais; considere mascarar PII e outros dados confidenciais e crie exibições que contenham apenas as informações que o visualizador precisa acessar. Alguns dados devem ser mantidos, mas devem ser acessados apenas em caso de emergência. Coloque esses dados atrás de um processo de vídro quebrado: os usuários podem acessá-lo somente após passar por um processo de aprovação de emergência para corrigir um problema, consultar informações históricas críticas etc. O acesso é revogado imediatamente após a conclusão do trabalho.

Responsabilidade Compartilhada na Nuvem

A segurança é uma responsabilidade compartilhada na nuvem. O fornecedor de nuvem é responsável por garantir a segurança física de seu data center e hardware. Ao mesmo tempo, você é responsável pela segurança dos aplicativos e sistemas que cria e mantém na nuvem. A maioria das violações de segurança na nuvem continua sendo causada pelos usuários finais, não pela nuvem. As violações ocorrem devido a configurações incorretas não intencionais, erros, descuidos e negligéncia.

Sempre faça backup de seus dados

Os dados desaparecem. Às vezes, é um disco rígido ou servidor morto; em outros casos, alguém pode excluir acidentalmente um banco de dados ou um depósito de armazenamento de objetos. Um ator mal-intencionado também pode bloquear dados. Os ataques de ransomware são comuns atualmente. Algumas seguradoras estão reduzindo os pagamentos no caso de um ataque, deixando você no gancho tanto para recuperar seus dados quanto para pagar o malfeitor que os mantém como reféns. Você precisa fazer backup de seus dados regularmente, tanto para recuperação de desastres quanto para continuidade de

operações comerciais, se uma versão de seus dados for comprometida em um ataque de ransomware. Além disso, teste a restauração de seus backups de dados regularmente.

O backup de dados não se enquadra estritamente nas práticas de segurança e privacidade; vai sob o título maior de *prevenção de desastres*, mas é adjacente à segurança, especialmente na era dos ataques de ransomware.

Um Exemplo de Política de Segurança

Esta seção apresenta um exemplo de política de segurança referente a credenciais, dispositivos e informações confidenciais. Observe que não complicamos demais as coisas; em vez disso, damos às pessoas uma pequena lista de ações práticas que podem tomar imediatamente.

Exemplo de política de segurança

Proteja suas credenciais

Proteja suas credenciais a todo custo. Aqui estão algumas regras básicas para credenciais:

- Use um logon único (SSO) para tudo. Evite senhas sempre que possível e use SSO como padrão.
- Use autenticação multifator com SSO.
- Não compartilhe senhas ou credenciais. Isso inclui senhas e credenciais do cliente. Em caso de dúvida, consulte a pessoa a quem você se reporta. Se essa pessoa estiver em dúvida, continue pesquisando até encontrar uma resposta.
- Cuidado com chamadas de phishing e scam. Nunca dê suas senhas. (Novamente, priorize o SSO.)
- Desative ou exclua credenciais antigas. De preferência o último.
- Não coloque suas credenciais em código. Trate os segredos como configuração e nunca os comprometa com o controle de versão. Use um gerenciador de segredos sempre que possível.
- Sempre exerça o princípio do menor privilégio. Nunca dê mais acesso do que o necessário para fazer o trabalho. Isso se aplica a todas as credenciais e privilégios na nuvem e no local.

Proteja seus dispositivos

- Use o gerenciamento de dispositivos para todos os dispositivos usados pelos funcionários. Se um funcionário sair da empresa ou seu dispositivo for perdido, o dispositivo poderá ser apagado remotamente.
- Use a autenticação multifator para todos os dispositivos.
- Entre no seu dispositivo usando as credenciais de e-mail da sua empresa.
- Todas as políticas que abrangem credenciais e comportamento se aplicam ao(s) seu(s) dispositivo(s).

- Trate seu dispositivo como uma extensão de si mesmo. Não perca de vista o(s) dispositivo(s) atribuído(s).
- Ao compartilhar a tela, saiba exatamente o que você está compartilhando para proteger informações e comunicações confidenciais. Compartilhe apenas documentos individuais, guias do navegador ou janelas e evite compartilhar toda a área de trabalho. Compartilhe apenas o que for necessário para transmitir seu ponto de vista.
- Use o modo “não perturbe” nas videochamadas; isso evita que mensagens apareçam durante chamadas ou gravações.

Política de atualização de software

- Reinicie seu navegador da Web quando vir um alerta de atualização.
- Execute pequenas atualizações do sistema operacional em dispositivos pessoais e da empresa.
- A empresa identificará as principais atualizações críticas do sistema operacional e fornecerá orientação.
- Não use a versão beta de um sistema operacional.
- Aguarde uma ou duas semanas para novos lançamentos de versões principais do sistema operacional.

Estes são alguns exemplos básicos de como a segurança pode ser simples e eficaz. Com base no perfil de segurança da sua empresa, pode ser necessário adicionar mais requisitos para as pessoas seguirem. E, novamente, lembre-se sempre de que as pessoas são seu elo mais fraco em segurança.

Tecnologia

Depois de abordar a segurança com pessoas e processos, é hora de ver como você aproveita a tecnologia para proteger seus sistemas e ativos de dados. A seguir estão algumas áreas importantes que você deve priorizar.

Sistemas de correção e atualização

O software fica obsoleto e as vulnerabilidades de segurança são constantemente descobertas. Para evitar expor uma falha de segurança em uma versão mais antiga das ferramentas que você está usando, sempre corrija e atualize os sistemas operacionais e o software à medida que novas atualizações forem disponibilizadas. Felizmente, muitos SaaS e serviços gerenciados em nuvem executam automaticamente atualizações e outras manutenções sem a sua intervenção. Para atualizar seu próprio código e dependências, automatize as compilações ou defina alertas sobre lançamentos e vulnerabilidades para que você possa ser solicitado a executar as atualizações manualmente.

Criptografia

A criptografia não é uma bala mágica. Isso fará pouco para protegê-lo no caso de um *humano* violação de segurança que concede acesso às credenciais. A criptografia é um requisito básico para qualquer organização que respeite a segurança e a privacidade. Ele irá protegê-lo de ataques básicos, como interceptação de tráfego de rede.

Vamos examinar separadamente a criptografia em repouso e em trânsito.

Criptografia em repouso

Certifique-se de que seus dados estejam criptografados quando estiverem em repouso (em um dispositivo de armazenamento). Os laptops da sua empresa devem ter criptografia de disco completo habilitada para proteger os dados se um dispositivo for roubado. Implemente a criptografia do lado do servidor para todos os dados armazenados em servidores, sistemas de arquivos, bancos de dados e armazenamento de objetos na nuvem. Todos os backups de dados para fins de arquivamento também devem ser criptografados. Por fim, incorpore a criptografia no nível do aplicativo, quando aplicável.

Criptografia sobre o fio

A criptografia na rede agora é o padrão para os protocolos atuais. Por exemplo, HTTPS geralmente é necessário para APIs de nuvem modernas. Os engenheiros de dados devem sempre estar cientes de como as chaves são tratadas; o manuseio inadequado de chaves é uma fonte significativa de vazamentos de dados. Além disso, o HTTPS não faz nada para proteger os dados se as permissões do bucket forem deixadas abertas ao público, outra causa de vários escândalos de dados na última década.

Os engenheiros também devem estar cientes das limitações de segurança dos protocolos mais antigos. Por exemplo, o FTP simplesmente não é seguro em uma rede pública. Embora isso possa não parecer um problema quando os dados já são públicos, o FTP é vulnerável a ataques man-in-the-middle, por meio dos quais um invasor intercepta os dados baixados e os altera antes que cheguem ao cliente. É melhor simplesmente evitar o FTP.

Certifique-se de que tudo esteja criptografado na rede, mesmo com protocolos legados. Em caso de dúvida, use tecnologia robusta com criptografia integrada.

Registro, monitoramento e alerta

Hackers e malfeitos normalmente não anunciam que estão se infiltrando em seus sistemas. A maioria das empresas não descobre incidentes de segurança até bem depois do fato. Parte do DataOps é observar, detectar e alertar sobre incidentes. Como engenheiro de dados, você deve configurar monitoramento, registro e alerta automatizados para estar ciente de eventos peculiares quando eles ocorrerem em seus sistemas. Se possível, configure a detecção automática de anomalias.

Aqui estão algumas áreas que você deve monitorar:

Acesso

Quem está acessando o quê, quando e de onde? Que novos acessos foram concedidos? Existem padrões estranhos com seus usuários atuais que podem indicar que suas contas estão comprometidas, como tentar acessar sistemas que eles normalmente não acessam ou aos quais não deveriam ter acesso? Você vê novos usuários não reconhecidos acessando seu sistema? Certifique-se de verificar regularmente os logs de acesso, usuários e suas funções para garantir que tudo esteja OK.

Recursos

Monitore seu disco, CPU, memória e E/S em busca de padrões que pareçam fora do comum. Seus recursos mudaram repentinamente? Em caso afirmativo, isso pode indicar uma violação de segurança.

Cobrança

Especialmente com SaaS e serviços gerenciados em nuvem, você precisa supervisionar os custos. Configure alertas de orçamento para garantir que seus gastos estejam dentro das expectativas. Se ocorrer um pico inesperado em seu faturamento, isso pode indicar que alguém ou algo está utilizando seus recursos para fins maliciosos.

Excesso de permissões

Cada vez mais, os fornecedores estão fornecendo ferramentas que monitoram as permissões que são *não utilizadas* por um usuário ou conta de serviço durante algum tempo. Essas ferramentas geralmente podem ser configuradas para alertar automaticamente um administrador ou remover permissões após um tempo decorrido especificado.

Por exemplo, suponha que um determinado analista não tenha acessado o Redshift por seis meses. Essas permissões podem ser removidas, fechando uma possível brecha de segurança. Se o analista precisar acessar o Redshift no futuro, ele poderá inserir um ticket para restaurar as permissões.

É melhor combinar essas áreas em seu monitoramento para obter uma visão transversal de seus recursos, acesso e perfil de cobrança. Sugerimos a criação de um painel para que todos da equipe de dados visualizem o monitoramento e recebam alertas quando algo parecer fora do comum. Junte isso a um plano eficaz de resposta a incidentes para gerenciar violações de segurança quando elas ocorrerem e execute o plano regularmente para estar preparado.

Acesso à rede

Muitas vezes vemos engenheiros de dados fazendo coisas bem loucas em relação ao acesso à rede. Em vários casos, vimos baldes do Amazon S3 disponíveis publicamente contendo muitos dados confidenciais. Também testemunhamos instâncias do Amazon EC2 com acesso SSH de entrada aberto para todo o mundo para 0.0.0.0/0 (todos os IPs) ou bancos de dados com acesso aberto a todas as solicitações de entrada pela Internet pública. Estes são apenas alguns exemplos de práticas de segurança de rede terríveis.

Em princípio, a segurança da rede deve ser deixada para os especialistas em segurança da sua empresa. (Na prática, você pode precisar assumir uma responsabilidade significativa pela segurança da rede em uma pequena empresa.) Como engenheiro de dados, você encontrará bancos de dados, armazenamento de objetos e servidores com tanta frequência que deve pelo menos estar ciente das medidas simples que pode tomar, para garantir que você esteja de acordo com as boas práticas de acesso à rede. Entenda quais IPs e portas estão abertos, para quem e por quê. Permita os endereços IP de entrada dos sistemas e usuários que acessarão essas portas (também conhecidos como IPs de lista branca) e evite abrir conexões amplamente por qualquer motivo. Ao acessar a nuvem ou uma ferramenta SaaS, use uma conexão criptografada. Por exemplo, não use um site não criptografado de uma cafeteria.

Além disso, embora este livro tenha se concentrado quase inteiramente na execução de cargas de trabalho na nuvem, adicionamos uma breve observação aqui sobre a hospedagem de servidores locais. Lembre-se que em [Capítulo 3](#), discutimos a diferença entre um perímetro reforçado e segurança de confiança zero. A nuvem geralmente está mais próxima da segurança de confiança zero – cada ação requer autenticação. Acreditamos que a nuvem é uma opção mais segura para a maioria das organizações porque impõe práticas de confiança zero e permite que as empresas aproveitem o exército de engenheiros de segurança empregados pelas nuvens públicas.

No entanto, às vezes, a segurança reforçada do perímetro ainda faz sentido; encontramos algum consolo no conhecimento de que os silos de mísseis nucleares são isolados (não conectados a nenhuma rede). Os servidores air-gapped são o melhor exemplo de um perímetro de segurança reforçado. Lembre-se de que, mesmo no local, os servidores sem ar são vulneráveis a falhas de segurança humana.

Segurança para engenharia de dados de baixo nível

Para engenheiros que trabalham com sistemas de armazenamento e processamento de dados, é fundamental considerar as implicações de segurança de cada elemento. Qualquer biblioteca de software, sistema de armazenamento ou nó de computação é uma vulnerabilidade de segurança em potencial. Uma falha em uma biblioteca de registro obscura pode permitir que os invasores ignorem os controles de acesso ou a criptografia. Mesmo arquiteturas de CPU e microcódigo representam vulnerabilidades potenciais; dados sensíveis podem ser vulneráveis quando estão em repouso na memória ou em um cache da CPU. Nenhum elo da cadeia pode ser dado como certo.

É claro que este livro é principalmente sobre engenharia de dados de alto nível — juntando ferramentas para lidar com todo o ciclo de vida. Portanto, deixaremos que você explore os detalhes técnicos sangrentos.

Pesquisa de segurança interna

Discutimos a ideia de *segurança ativa* em “[Processos](#)” na [página 371](#). Também recomendamos fortemente a adoção de uma *segurança ativa* abordagem da tecnologia. Especificamente, isso significa que todo funcionário de tecnologia deve pensar nos problemas de segurança.

Por que isso é importante? Cada contribuidor de tecnologia desenvolve um domínio de especialização técnica. Mesmo que sua empresa empregue um exército de pesquisadores de segurança, os engenheiros de dados se familiarizarão intimamente com sistemas de dados específicos e serviços de nuvem em sua área de atuação. Os especialistas em uma determinada tecnologia estão bem posicionados para identificar brechas de segurança nessa tecnologia.

Incentive todos os engenheiros de dados a se envolverem ativamente na segurança. Quando identificam possíveis riscos de segurança em seus sistemas, eles devem pensar nas mitigações e assumir um papel ativo na implantação delas.

Conclusão

A segurança precisa ser um hábito mental e de ação; trate dados como sua carteira ou smartphone. Embora você provavelmente não seja o responsável pela segurança de sua empresa, conhecer as práticas básicas de segurança e manter a segurança em mente ajudará a reduzir o risco de violações de segurança de dados em sua organização.

Recursos adicionais

- *Construindo Sistemas Seguros e Confiáveis* por Heather Adkins e outros. (O'Reilly)
- Publicações do Open Web Application Security Project (OWASP)
- *Segurança prática na nuvem* por Chris Dotson (O'Reilly)

O futuro da engenharia de dados

Este livro surgiu do reconhecimento dos autores de que as mudanças de velocidade de dobra no campo criaram uma lacuna de conhecimento significativa para engenheiros de dados existentes, pessoas interessadas em seguir uma carreira em engenharia de dados, gerentes de tecnologia e executivos que desejam entender melhor como os dados engenharia se encaixa em suas empresas. Quando começamos a pensar em como organizar este livro, recebemos algumas críticas de amigos que perguntavam: “Como você ousa escrever sobre um campo que está mudando tão rapidamente?!” De muitas maneiras, eles estão certos. Certamente parece que o campo da engenharia de dados – e, na verdade, todos os dados relacionados – está mudando diariamente. Peneirando o ruído e encontrando o sinal de *o que é improvável que mude* foi uma das partes mais desafiadoras de organizar e escrever este livro.

Neste livro, focamos em grandes ideias que achamos que serão úteis nos próximos anos — daí a continuidade do ciclo de vida da engenharia de dados e suas correntes ocultas. A ordem das operações e os nomes das melhores práticas e tecnologias podem mudar, mas os estágios principais do ciclo de vida provavelmente permanecerão intactos por muitos anos. Estamos cientes de que a tecnologia continua a mudar em um ritmo exaustivo; trabalhar no setor de tecnologia em nossa era atual pode parecer uma montanha-russa ou talvez uma sala de espelhos.

Vários anos atrás, a engenharia de dados nem existia como um campo ou cargo. Agora você está lendo um livro chamado *Fundamentos da Engenharia de Dados!* Você aprendeu tudo sobre os fundamentos da engenharia de dados — seu ciclo de vida, subcorrentes, tecnologias e práticas recomendadas. Você deve estar se perguntando: o que vem a seguir na engenharia de dados? Embora ninguém possa prever o futuro, temos uma boa perspectiva sobre o passado, o presente e as tendências atuais. Tivemos a sorte de observar a gênese e a evolução da engenharia de dados de um lugar na primeira fila. Este capítulo final apresenta nossos pensamentos sobre o futuro, incluindo observações de desenvolvimentos em andamento e especulações futuras selvagens.

O ciclo de vida da engenharia de dados não vai acabar

Embora a ciência de dados tenha recebido a maior parte da atenção nos últimos anos, a engenharia de dados está amadurecendo rapidamente em um campo distinto e visível. É uma das carreiras que mais crescem em tecnologia, sem sinais de perda de força. À medida que as empresas percebem que primeiro precisam construir uma base de dados antes de passar para coisas “mais sensuais”, como IA e ML, a engenharia de dados continuará crescendo em popularidade e importância. Esse progresso gira em torno do ciclo de vida da engenharia de dados.

Alguns questionam se ferramentas e práticas cada vez mais simples levarão ao desaparecimento dos engenheiros de dados. Esse pensamento é superficial, preguiçoso e míope. À medida que as organizações aproveitam os dados de novas maneiras, novas fundações, sistemas e fluxos de trabalho serão necessários para atender a essas necessidades. Os engenheiros de dados estão no centro do projeto, arquitetura, construção e manutenção desses sistemas. Se as ferramentas se tornarem mais fáceis de usar, os engenheiros de dados subirão na cadeia de valor para se concentrar no trabalho de nível superior. O ciclo de vida da engenharia de dados não vai desaparecer tão cedo.

O declínio da complexidade e o surgimento de ferramentas de dados fáceis de usar

Ferramentas simplificadas e fáceis de usar continuam a reduzir a barreira de entrada para a engenharia de dados. Isso é ótimo, especialmente considerando a escassez de engenheiros de dados que discutimos. A tendência para a simplicidade continuará. A engenharia de dados não depende de uma tecnologia ou tamanho de dados específicos. Também não é apenas para grandes empresas. Nos anos 2000, a implantação de tecnologias de “big data” exigia uma equipe grande e bolsos cheios. A ascensão dos serviços gerenciados por SaaS removeu em grande parte a complexidade de entender os vários sistemas de “big data”. A engenharia de dados agora é algo que *todas* empresas podem fazer.

Big data é uma vítima de seu extraordinário sucesso. Por exemplo, Google BigQuery, um descendente de GFS e MapReduce, pode consultar petabytes de dados. Antes reservada para uso interno no Google, essa tecnologia insanamente poderosa agora está disponível para qualquer pessoa com uma conta GCP. Os usuários simplesmente pagam pelos dados que armazenam e consultam, em vez de ter que construir uma enorme pilha de infraestrutura. Snowflake, Amazon EMR e muitas outras soluções de dados em nuvem hiperescaláveis competem no espaço e oferecem recursos semelhantes.

A nuvem é responsável por uma mudança significativa no uso de ferramentas de código aberto. Mesmo no início de 2010, o uso de código aberto normalmente envolvia baixar o código e configurá-lo você mesmo. Atualmente, muitas ferramentas de dados de código aberto estão disponíveis como serviços de nuvem gerenciados que competem diretamente com serviços proprietários. O Linux está disponível pré-configurado e instalado em instâncias de servidor em todas as principais nuvens. Plataformas sem servidor como AWS Lambda e Google Cloud Functions permitem que você implante

aplicativos orientados a eventos em minutos, usando linguagens convencionais como Python, Java e Go rodando no Linux nos bastidores. Os engenheiros que desejam usar o Apache Airflow podem adotar o Cloud Composer do Google ou o serviço Airflow gerenciado da AWS. O Kubernetes gerenciado nos permite construir arquiteturas de microsserviço altamente escaláveis. E assim por diante.

Isso muda fundamentalmente a conversa em torno do código-fonte aberto. Em muitos casos, o código aberto gerenciado é tão fácil de usar quanto seus concorrentes de serviços proprietários. As empresas com necessidades altamente especializadas também podem implantar o código aberto gerenciado e, posteriormente, migrar para o código aberto autogerenciado, caso precisem personalizar o código subjacente.

Outra tendência significativa é o crescimento da popularidade dos conectores de dados disponíveis no mercado (no momento da redação deste artigo, os populares incluem Fivetran e Airbyte). Os engenheiros de dados tradicionalmente gastam muito tempo e recursos construindo e mantendo o encanamento para se conectar a fontes de dados externas. A nova geração de conectores gerenciados é altamente atraente, mesmo para engenheiros altamente técnicos, pois eles começam a reconhecer o valor de recapturar tempo e largura de banda mental para outros projetos. Os conectores de API serão um problema terceirizado para que os engenheiros de dados possam se concentrar nos problemas exclusivos que impulsionam seus negócios.

A interseção da competição acirrada no espaço de ferramentas de dados com um número crescente de engenheiros de dados significa que as ferramentas de dados continuarão diminuindo em complexidade enquanto adicionam ainda mais funcionalidade e recursos. Essa simplificação só aumentará a prática da engenharia de dados, pois mais e mais empresas encontram oportunidades para descobrir valor nos dados.

O sistema operacional de dados em escala de nuvem e a interoperabilidade aprimorada

Vamos revisar brevemente alguns dos funcionamentos internos dos sistemas operacionais (dispositivo único) e, em seguida, vincular isso aos dados e à nuvem. Esteja você usando um smartphone, um laptop, um servidor de aplicativos ou um termostato inteligente, esses dispositivos contam com um sistema operacional para fornecer serviços essenciais e orquestrar tarefas e processos. Por exemplo, posso ver cerca de 300 processos em execução no MacBook Pro em que estou digitando. Entre outras coisas, vejo serviços como WindowServer (responsável por fornecer janelas em uma interface gráfica) e CoreAudio (encarregado de fornecer recursos de áudio de baixo nível).

Quando executo um aplicativo nesta máquina, ele não acessa diretamente o hardware gráfico e de som. Em vez disso, ele envia comandos aos serviços do sistema operacional para desenhar janelas e reproduzir som. Esses comandos são emitidos para APIs padrão; uma especificação informa aos desenvolvedores de software como se comunicar com os serviços do sistema operacional. O sistema operacional *orquestra* um processo de inicialização para fornecer esses serviços, iniciando cada serviço na ordem correta com base nas dependências entre eles; isso também

mantém os serviços monitorando-os e reiniciando-os na ordem correta em caso de falha.

Agora vamos voltar aos dados na nuvem. Os serviços de dados simplificados que mencionamos ao longo deste livro (por exemplo, Google Cloud BigQuery, Azure Blob Storage, Snowflake e AWS Lambda) se assemelham a serviços de sistema operacional, mas em uma escala muito maior, executando em muitas máquinas em vez de em um único servidor.

Agora que esses serviços simplificados estão disponíveis, a próxima fronteira de evolução para essa noção de sistema operacional de dados em nuvem acontecerá em um nível mais alto de abstração. Benn Stancil pediu o surgimento de APIs de dados padronizados para a construção de pipelines de dados e aplicativos de dados.¹ Prevemos que a engenharia de dados se unirá gradualmente em torno de um punhado de padrões de interoperabilidade de dados. O armazenamento de objetos na nuvem crescerá em importância como uma camada de interface em lote entre vários serviços de dados. Os formatos de arquivo de nova geração (como Parquet e Avro) já estão assumindo o papel de intercâmbio de dados em nuvem, melhorando significativamente a terrível interoperabilidade do CSV e o baixo desempenho do JSON bruto.

Outro ingrediente crítico de um ecossistema de API de dados é um catálogo de metadados que descreve esquemas e hierarquias de dados. Atualmente, essa função é amplamente preenchida pelo legado Hive Metastore. Esperamos que novos entrantes surjam para ocupar seu lugar. Os metadados desempenharão um papel crucial na interoperabilidade de dados, tanto em aplicativos e sistemas quanto em nuvens e redes, impulsionando a automação e a simplificação.

Também veremos melhorias significativas no scaffolding que gerencia os serviços de dados em nuvem. O Apache Airflow emergiu como a primeira plataforma de orquestração de dados verdadeiramente orientada para a nuvem, mas estamos à beira de uma melhoria significativa. O Airflow crescerá em recursos, com base em sua massiva participação. Novos participantes, como Dagster e Prefect, competirão reconstruindo a arquitetura de orquestração desde o início.

Esta próxima geração de plataformas de orquestração de dados contará com integração de dados aprimorada e conscientização de dados. As plataformas de orquestração se integrarão à catalogação e linhagem de dados, tornando-se significativamente mais conscientes dos dados no processo. Além disso, as plataformas de orquestração criarião recursos de IaC (semelhantes ao Terraform) e recursos de implantação de código (como GitHub Actions e Jenkins). Isso permitirá que os engenheiros codifiquem um pipeline e o passem para a plataforma de orquestração para criar, testar, implantar e monitorar automaticamente. Os engenheiros poderão escrever especificações de infraestrutura diretamente em seus pipelines; infraestrutura e serviços ausentes (por exemplo, bancos de dados Snowflake, clusters Databricks e fluxos do Amazon Kinesis) serão implantados na primeira vez em que o pipeline for executado.

¹ Benn Stancil, "The Data OS," *benn.substack*, 3 de setembro de 2021, <https://oreil.ly/HetE9>.

Também veremos melhorias significativas no domínio *dados ao vivo*—por exemplo, pipelines de streaming e bancos de dados capazes de ingerir e consultar dados de streaming. No passado, construir um DAG de streaming era um processo extremamente complexo com uma alta carga operacional contínua (consulte [Capítulo 8](#)). Ferramentas como o Apache Pulsar apontam o caminho para um futuro em que os DAGs de streaming podem ser implantados com transformações complexas usando código relativamente simples. Já vimos o surgimento de processadores de stream gerenciados (como Amazon Kinesis Data Analytics e Google Cloud Dataflow), mas veremos uma nova geração de ferramentas de orquestração para gerenciar esses serviços, unindo-os e monitorando-os. Discutimos dados ao vivo em “[A pilha de dados ao vivo](#)” na [página 385](#).

O que essa abstração aprimorada significa para os engenheiros de dados? Como já argumentamos neste capítulo, o papel do engenheiro de dados não desaparecerá, mas evoluirá significativamente. Em comparação, estruturas e sistemas operacionais móveis mais sofisticados não eliminaram os desenvolvedores de aplicativos móveis. Em vez disso, os desenvolvedores de aplicativos móveis agora podem se concentrar na criação de aplicativos mais sofisticados e de melhor qualidade. Esperamos desenvolvimentos semelhantes para a engenharia de dados, pois o paradigma do sistema operacional de dados em escala de nuvem aumenta a interoperabilidade e a simplicidade em vários aplicativos e sistemas.

Engenharia de Dados “Empresarial”

A crescente simplificação das ferramentas de dados e o surgimento e documentação das melhores práticas significa que a engenharia de dados se tornará mais “empresarial”.² Isso fará com que muitos leitores se estremeçam violentamente. O termo *empreendimento*, para alguns, evoca pesadelos kafkianos de comitês sem rosto vestidos com camisas azuis excessivamente engomadas e calças cáqui, burocracia sem fim e projetos de desenvolvimento gerenciados em cascata com cronogramas constantemente atrasados e orçamentos crescentes. Resumindo, alguns de vocês leem “empresa” e imaginam um lugar sem alma onde a inovação vai morrer.

Felizmente, não é disso que estamos falando; estamos nos referindo a alguns dos *bom* coisas que empresas maiores fazem com dados – gerenciamento, operações, governança e outras coisas “chatas”. Atualmente, estamos vivendo a era de ouro das ferramentas de gerenciamento de dados “empresariais”. Tecnologias e práticas outrora reservadas para organizações gigantes estão se infiltrando. As partes antes difíceis de big data e streaming de dados agora foram amplamente abstraídas, com o foco mudando para facilidade de uso, interoperabilidade e outros refinamentos.

² Ben Rogojan, “Três especialistas em engenharia de dados compartilham suas ideias sobre o rumo dos dados”, *Melhorar Programação*, 27 de maio de 2021, <https://oreil.ly/IsY4W>.

Isso permite que os engenheiros de dados que trabalham em novas ferramentas encontrem oportunidades nas abstrações de gerenciamento de dados, DataOps e todas as outras subcorrentes da engenharia de dados. Os engenheiros de dados se tornarão “empresários”. Falando nisso...

Títulos e responsabilidades irão se transformar...

Embora o ciclo de vida da engenharia de dados não vá a lugar nenhum tão cedo, os limites entre engenharia de software, engenharia de dados, ciência de dados e engenharia de ML estão cada vez mais confusos. Na verdade, assim como os autores, muitos cientistas de dados são transformados em engenheiros de dados por meio de um processo orgânico; encarregados de fazer “ciência de dados”, mas sem as ferramentas para realizar seu trabalho, eles assumem a tarefa de projetar e construir sistemas para atender ao ciclo de vida da engenharia de dados.

À medida que a simplicidade sobe na pilha, os cientistas de dados gastam uma fatia menor de seu tempo reunindo e processando dados. Mas essa tendência se estenderá além dos cientistas de dados. A simplificação também significa que os engenheiros de dados gastarão menos tempo em tarefas de baixo nível no ciclo de vida da engenharia de dados (gerenciamento de servidores, configuração etc.) e a engenharia de dados “corporativa” se tornará mais predominante.

À medida que os dados se tornam mais integrados nos processos de todos os negócios, novas funções surgirão no domínio dos dados e dos algoritmos. Uma possibilidade é uma função que fica entre a engenharia de ML e a engenharia de dados. À medida que os conjuntos de ferramentas de ML se tornam mais fáceis de usar e os serviços de ML em nuvem gerenciados crescem em recursos, o ML está mudando da exploração ad hoc e do desenvolvimento de modelo para se tornar uma disciplina operacional.

Este novo engenheiro focado em ML que atravessa essa divisão conterá algoritmos, técnicas de ML, otimização de modelo, monitoramento de modelo e monitoramento de dados. No entanto, sua função principal será criar ou utilizar os sistemas que treinam modelos automaticamente, monitoram o desempenho e operacionalizam todo o processo de ML para tipos de modelo bem compreendidos. Eles também monitorarão pipelines e qualidade de dados, sobrepondo-se ao domínio atual da engenharia de dados. Os engenheiros de ML se tornarão mais especializados para trabalhar em tipos de modelo mais próximos da pesquisa e menos compreendidos.

Outra área em que os títulos podem se transformar é na interseção da engenharia de software e da engenharia de dados. Aplicativos de dados, que combinam aplicativos de software tradicionais com análises, impulsionarão essa tendência. Os engenheiros de software precisarão ter uma compreensão muito mais profunda da engenharia de dados. Eles desenvolverão experiência em coisas como streaming, pipelines de dados, modelagem de dados e qualidade de dados. Iremos além da abordagem de “jogar por cima do muro” que agora é difundida. Os engenheiros de dados serão integrados às equipes de desenvolvimento de aplicativos e os desenvolvedores de software adquirirão habilidades de engenharia de dados. Os limites existentes entre sistemas de back-end de aplicativos e ferramentas de engenharia de dados também serão reduzidos, com integração profunda por meio de streaming e arquiteturas orientadas a eventos.

Indo além da pilha de dados moderna, em direção à pilha de dados ao vivo

Seremos francos: a pilha de dados moderna (MDS) não é tão moderna. Aplaudimos o MDS por trazer uma grande seleção de poderosas ferramentas de dados para as massas, reduzindo os preços e capacitando os analistas de dados a assumir o controle de sua pilha de dados. A ascensão de ELT, armazenamentos de dados em nuvem e a abstração de pipelines de dados SaaS certamente mudaram o jogo para muitas empresas, abrindo novos poderes para BI, análise e ciência de dados.

Dito isto, o MDS é basicamente uma reformulação de antigas práticas de armazenamento de dados usando tecnologias modernas de nuvem e SaaS; como o MDS é construído em torno do paradigma de armazenamento de dados em nuvem, ele tem algumas limitações sérias quando comparado ao potencial de aplicativos de dados em tempo real de última geração. Do nosso ponto de vista, o mundo está indo além do uso de análises internas e ciência de dados baseadas em data warehouse, para capacitar negócios e aplicativos inteiros em tempo real com bancos de dados em tempo real de última geração.

O que está impulsionando essa evolução? Em muitos casos, a análise (BI e análise operacional) será substituída pela automação. Atualmente, a maioria dos painéis e relatórios respondem a perguntas sobre *o que e quando*. Pergunte a si mesmo: "Se estou perguntando a um *o que ou quando* pergunta, que ação devo tomar a seguir?" Se a ação for repetitiva, é candidata à automação. Por que consultar um relatório para determinar se deve agir quando você pode, em vez disso, automatizar a ação com base nos eventos conforme eles ocorrem?

E vai muito além disso. Por que usar um produto como TikTok, Uber, Google ou DoorDash parece mágica? Embora pareça a você um clique de um botão para assistir a um pequeno vídeo, pedir uma carona ou uma refeição ou encontrar um resultado de pesquisa, muita coisa está acontecendo nos bastidores. Esses produtos são exemplos de verdadeiros aplicativos de dados em tempo real, fornecendo as ações necessárias com o clique de um botão enquanto executam processamento de dados extremamente sofisticado e ML nos bastidores com latência minúscula. Atualmente, esse nível de sofisticação está trancado atrás de tecnologias personalizadas em grandes empresas de tecnologia, mas essa sofisticação e poder estão se tornando democratizados, semelhante à maneira como o MDS trouxe armazenamentos de dados e pipelines em escala de nuvem para as massas. O mundo dos dados logo estará "ao vivo".

A pilha de dados ao vivo

Essa democratização das tecnologias de tempo real nos levará ao sucessor do MDS: *o pilha de dados ao vivo* em breve estará acessível e difundido. A pilha de dados ao vivo, representada em **Figura 11-1**, fundirá análise em tempo real e ML em aplicativos usando tecnologias de streaming, cobrindo todo o ciclo de vida dos dados, desde sistemas de origem de aplicativos até processamento de dados para ML e vice-versa.

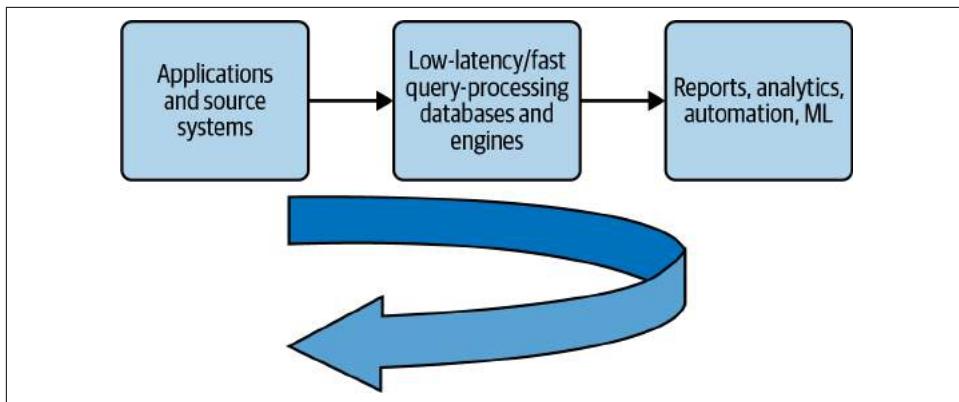


Figura 11-1. Na pilha de dados ao vivo, os dados e a inteligência se movem em tempo real entre o aplicativo e os sistemas de suporte

Assim como o MDS aproveitou a nuvem e trouxe tecnologias locais de data warehouse e pipeline para as massas, a pilha de dados ao vivo usa tecnologias de aplicativos de dados em tempo real usadas em empresas de tecnologia de elite e as disponibiliza para empresas de todos os tamanhos como ofertas baseadas em nuvem fáceis de usar. Isso abrirá um novo mundo de possibilidades para criar experiências de usuário e valor comercial ainda melhores.

Pipelines de streaming e bancos de dados analíticos em tempo real

O MDS se limita a técnicas de lote que tratam os dados como limitados. Em contraste, os aplicativos de dados em tempo real tratam os dados como um fluxo contínuo e ilimitado. Pipelines de streaming e bancos de dados analíticos em tempo real são as duas principais tecnologias que facilitarão a mudança do MDS para a pilha de dados ao vivo. Embora essas tecnologias já existam há algum tempo, os serviços de nuvem gerenciados em rápido amadurecimento os verão implantados de forma muito mais ampla.

As tecnologias de streaming continuarão a ter um crescimento extremo no futuro previsível. Isso acontecerá em conjunto com um foco mais claro no utilitário de negócios de streaming de dados. Até o presente, os sistemas de streaming têm sido frequentemente tratados como uma novidade cara ou um canal estúpido para obter dados de A para B. No futuro, o streaming transformará radicalmente a tecnologia organizacional e os processos de negócios; arquitetos e engenheiros de dados assumirão a liderança nessas mudanças fundamentais.

Bancos de dados analíticos em tempo real permitem ingestão rápida e consultas de subsegundos nesses dados. Esses dados podem ser enriquecidos ou combinados com conjuntos de dados históricos. Quando combinado com um pipeline de streaming e automação, ou painel capaz de análises em tempo real, um novo nível de possibilidades se abre. Você não está mais limitado por processos ELT lentos, atualizações de 15 minutos ou outras partes lentas. Os dados se movem em um fluxo contínuo. À medida que a ingestão de streaming se torna mais prevalente, a ingestão de lote será cada vez menos comum. Por que criar um lote

gargalo no início do seu pipeline de dados? Por fim, veremos a ingestão de lotes da mesma forma que vemos agora os modems dial-up.

Em conjunto com o aumento dos fluxos, esperamos um momento de volta ao futuro para as transformações de dados. Mudaremos de ELT — em transformações de banco de dados — para algo que se pareça mais com ETL. Nós nos referimos provisoriamente a isso como *transmitir, transformar e carregar*(STL). Em um contexto de streaming, a extração é um processo contínuo e contínuo.

Obviamente, as transformações em lote não desaparecerão totalmente. O lote ainda será muito útil para treinamento de modelos, relatórios trimestrais e muito mais. Mas a transformação do streaming se tornará a norma.

Embora o data warehouse e o data lake sejam ótimos para armazenar grandes quantidades de dados e realizar consultas ad hoc, eles não são tão bem otimizados para ingestão de dados de baixa latência ou consultas em dados que se movem rapidamente. A pilha de dados ao vivo será alimentada por bancos de dados OLAP criados especificamente para streaming. Hoje, bancos de dados como Druid, ClickHouse, Rockset e Firebolt estão liderando o caminho para capacitar o back-end da próxima geração de aplicativos de dados. Esperamos que as tecnologias de streaming continuem a evoluir rapidamente e que novas tecnologias proliferem.

Outra área que achamos que está pronta para a disruptão é a modelagem de dados, onde não houve inovação séria desde o início dos anos 2000. As técnicas tradicionais de modelagem de dados orientada a lotes que você aprendeu em [Capítulo 8](#)não são adequados para streaming de dados. Novas técnicas de modelagem de dados ocorrerão não dentro do data warehouse, mas nos sistemas que geram os dados. Esperamos que a modelagem de dados envolva alguma noção de uma camada de definições upstream, incluindo semântica, métricas, linhagem e definições de dados (consulte [Capítulo 9](#))—começando onde os dados são gerados no aplicativo. A modelagem também acontecerá em todos os estágios à medida que os dados fluem e evoluem ao longo de todo o ciclo de vida.

A fusão de dados com aplicativos

Esperamos que a próxima revolução seja a fusão das camadas de aplicativos e dados. No momento, os aplicativos ficam em uma área e o MDS fica em outra. Para piorar a situação, os dados são criados sem levar em conta como serão usados para análise. Conseqüentemente, muita fita adesiva é necessária para fazer os sistemas se comunicarem. Essa colcha de retalhos e configuração em silos é desajeitada e desajeitada.

Em breve, as pilhas de aplicativos serão pilhas de dados e vice-versa. Os aplicativos integrarão automação e tomada de decisão em tempo real, alimentados pelos pipelines de streaming e ML. O ciclo de vida da engenharia de dados não mudará necessariamente, mas o tempo entre os estágios do ciclo de vida diminuirá drasticamente. Muita inovação ocorrerá em novas tecnologias e práticas que irão melhorar a experiência de engenharia da pilha de dados ao vivo. Preste atenção às tecnologias de banco de dados emergentes projetadas para atender à combinação de casos de uso de OLTP e OLAP; As lojas de recursos também podem desempenhar um papel semelhante para casos de uso de ML.

O feedback apertado entre aplicativos e ML

Outra área que nos entusiasma é a fusão de aplicativos e ML. Hoje, aplicativos e ML são sistemas separados, como aplicativos e análises. Engenheiros de software fazem suas coisas aqui, cientistas de dados e engenheiros de ML fazem suas coisas lá.

O ML é adequado para cenários em que os dados são gerados em uma taxa e volume tão altos que os humanos não podem processá-los manualmente. À medida que os tamanhos e a velocidade dos dados aumentam, isso se aplica a todos os cenários. Altos volumes de dados em movimento rápido, juntamente com fluxos de trabalho e ações sofisticados, são candidatos ao ML. À medida que os ciclos de feedback de dados ficam mais curtos, esperamos que a maioria dos aplicativos integre o ML. À medida que os dados se movem mais rapidamente, o ciclo de feedback entre os aplicativos e o ML ficará mais estreito. Os aplicativos na pilha de dados ao vivo são inteligentes e capazes de se adaptar em tempo real às mudanças nos dados. Isso cria um ciclo de aplicativos cada vez mais inteligentes e aumenta o valor comercial.

Dados da Matéria Escura e o Surgimento de... Planilhas?!

Falamos sobre dados que se movem rapidamente e como os ciclos de feedback diminuirão à medida que aplicativos, dados e ML trabalharem mais juntos. Esta seção pode parecer estranha, mas precisamos abordar algo que é amplamente ignorado no mundo dos dados de hoje, especialmente pelos engenheiros.

Qual é a plataforma de dados mais usada? É a humilde planilha. Dependendo das estimativas que você leu, a base de usuários de planilhas está entre 700 milhões e 2 bilhões de pessoas. As planilhas são a matéria escura do mundo dos dados. Boa parte da análise de dados é executada em planilhas e nunca chega aos sofisticados sistemas de dados que descrevemos neste livro. Em muitas organizações, as planilhas lidam com relatórios financeiros, análises da cadeia de suprimentos e até CRM.

No fundo, o que é uma planilha? *A planilha* é um aplicativo de dados interativo que oferece suporte a análises complexas. Ao contrário de ferramentas puramente baseadas em código, como pandas (Python Data Analysis Library), as planilhas são acessíveis a todo um espectro de usuários, desde aqueles que apenas sabem como abrir arquivos e examinar relatórios até usuários avançados que podem criar scripts sofisticados de processamento de dados processuais. Até agora, as ferramentas de BI falharam em trazer interatividade comparável aos bancos de dados. Os usuários que interagem com a IU normalmente estão limitados a fatiar e dividir dados dentro de certas proteções, não análises programáveis de uso geral.

Prevemos que surgirá uma nova classe de ferramentas que combina os recursos analíticos interativos de uma planilha com o poder de back-end dos sistemas OLAP em nuvem. De fato, alguns candidatos já estão concorrendo. O vencedor final nesta categoria de produto pode continuar a usar paradigmas de planilhas ou pode definir idiomas de interface totalmente novos para interagir com dados.

Conclusão

Obrigado por se juntar a nós nesta jornada pela engenharia de dados! Atravessamos a boa arquitetura, os estágios do ciclo de vida da engenharia de dados e as práticas recomendadas de segurança. Discutimos estratégias para escolher tecnologias em um momento em que nosso campo continua a mudar em um ritmo extraordinário. Neste capítulo, apresentamos nossa especulação selvagem sobre o futuro próximo e intermediário.

Alguns aspectos do nosso prognóstico assentam numa base relativamente segura. A simplificação das ferramentas gerenciadas e a ascensão da engenharia de dados “empresariais” ocorreram dia a dia enquanto escrevíamos este livro. Outras previsões são de natureza muito mais especulativa; vemos indícios de uma emergência *pilha de dados ao vivo*, mas isso implica uma mudança de paradigma significativa para engenheiros individuais e para as organizações que os empregam. Talvez a tendência de dados em tempo real pare mais uma vez, com a maioria das empresas continuando a se concentrar no processamento básico em lote. Certamente, existem outras tendências que falhamos completamente em identificar. A evolução da tecnologia envolve interações complexas de tecnologia e cultura. Ambos são imprevisíveis.

A engenharia de dados é um tópico vasto; Embora não tenhamos podido entrar em profundidade técnica em áreas individuais, esperamos ter conseguido criar uma espécie de guia de viagem que ajudará os atuais engenheiros de dados, futuros engenheiros de dados e aqueles que trabalham adjacentes ao campo a encontrar seu caminho um domínio que está em fluxo. Aconselhamo-lo a continuar a exploração por conta própria. Ao descobrir tópicos e ideias interessantes neste livro, continue a conversa como parte de uma comunidade. Identifique especialistas de domínio que podem ajudá-lo a descobrir os pontos fortes e as armadilhas das tecnologias e práticas da moda. Leia extensivamente os últimos livros, postagens de blog e artigos. Participe de encontros e ouça palestras. Faça perguntas e compartilhe sua própria experiência. Fique de olho nos anúncios dos fornecedores para ficar a par dos desenvolvimentos mais recentes,

Através deste processo, você pode escolher a tecnologia. Em seguida, você precisará adotar tecnologia e desenvolver expertise, talvez como colaborador individual, talvez dentro de sua equipe como líder, talvez em toda uma organização de tecnologia. Ao fazer isso, não perca de vista os objetivos maiores da engenharia de dados. Concentre-se no ciclo de vida, em atender seus clientes – internos e externos – em seu negócio, em atender e em seus objetivos maiores.

Em relação ao futuro, muitos de vocês desempenharão um papel na determinação do que vem a seguir. As tendências tecnológicas são definidas não apenas por aqueles que criam a tecnologia subjacente, mas também por aqueles que a adotam e fazem bom uso dela. ferramenta de sucesso *usaré* tão crítico quanto uma ferramenta *criação*. Encontre oportunidades para aplicar tecnologia em tempo real que melhorará a experiência do usuário, criará valor e definirá tipos totalmente novos de aplicativos. É este tipo de aplicação prática que materializará *o dados ao vivo*

pilha como um novo padrão da indústria; ou talvez alguma outra nova tendência tecnológica que não conseguimos identificar ganhe o dia.

Finalmente, desejamos-lhe uma carreira emocionante! Escolhemos trabalhar com engenharia de dados, consultar e escrever este livro não apenas porque estava na moda, mas porque era fascinante. Esperamos ter conseguido transmitir-vos um pouco da alegria que sentimos ao trabalhar nesta área.

Serialização e Compressão

Detalhes técnicos

Os engenheiros de dados que trabalham na nuvem geralmente estão livres das complexidades do gerenciamento de sistemas de armazenamento de objetos. Ainda assim, eles precisam entender os detalhes dos formatos de serialização e desserialização. Como mencionamos em [Capítulo 6](#)sobre armazenamento de ingredientes brutos, algoritmos de serialização e compressão andam de mãos dadas.

Formatos de serialização

Muitos algoritmos e formatos de serialização estão disponíveis para engenheiros de dados. Embora a abundância de opções seja uma fonte significativa de problemas na engenharia de dados, elas também são uma grande oportunidade para melhorias de desempenho. Às vezes, vimos o desempenho do trabalho melhorar em um fator de 100 simplesmente alternando da serialização CSV para Parquet. À medida que os dados passam por um pipeline, os engenheiros também gerenciam a reserialização — a conversão de um formato para outro. Às vezes, os engenheiros de dados não têm escolha a não ser aceitar os dados em uma forma antiga e desagradável; eles devem projetar processos para desserializar esse formato e lidar com exceções e, em seguida, limpar e converter dados para processamento e consumo downstream consistentes e rápidos.

Serialização baseada em linha

Como o próprio nome sugere, *serialização baseada em linha* organiza os dados por linha. O formato CSV é um formato arquetípico baseado em linha. Para dados semiestruturados (objetos de dados que suportam aninhamento e variação de esquema), a serialização orientada a linha envolve o armazenamento de cada objeto como uma unidade.

CSV: o padrão fora do padrão

Discutimos o CSV em [Capítulo 7](#). CSV é um formato de serialização que os engenheiros de dados adoram odiar. O termo CSV é essencialmente um resumo para texto delimitado, mas há flexibilidade nas convenções de escape, aspas, delimitador e muito mais.

Os engenheiros de dados devem evitar o uso de arquivos CSV em pipelines porque eles são altamente propensos a erros e oferecem baixo desempenho. Os engenheiros geralmente precisam usar formato CSV para trocar dados com sistemas e processos de negócios fora de seu controle. CSV é um formato comum para arquivamento de dados. Se você usar CSV para arquivamento, inclua uma descrição técnica completa da configuração de serialização de seus arquivos para que futuros consumidores possam ingerir os dados.

XML

Extensible Markup Language (XML) era popular quando o HTML e a Internet eram novos, mas agora é visto como um legado; geralmente é lento para desserializar e serializar para aplicativos de engenharia de dados. XML é outro padrão com o qual os engenheiros de dados geralmente são forçados a interagir enquanto trocam dados com sistemas e software legados. JSON substituiu amplamente o XML para serialização de objeto de texto simples.

JSON e JSONL

JavaScript Object Notation (JSON) surgiu como o novo padrão para troca de dados por APIs e também se tornou um formato extremamente popular para armazenamento de dados. No contexto dos bancos de dados, a popularidade do JSON cresceu rapidamente com o surgimento do MongoDB e de outros armazenamentos de documentos. Bancos de dados como Snowflake, BigQuery e SQL Server também oferecem amplo suporte nativo, facilitando a troca de dados entre aplicativos, APIs e sistemas de banco de dados.

JSON Lines (JSONL) é uma versão especializada do JSON para armazenar dados semiestruturados em massa em arquivos. JSONL armazena uma sequência de objetos JSON, com objetos delimitados por quebras de linha. Do nosso ponto de vista, o JSONL é um formato extremamente útil para armazenar dados logo após serem ingeridos da API ou dos aplicativos. No entanto, muitos formatos colunares oferecem um desempenho significativamente melhor. Considere mudar para outro formato para estágios de pipeline intermediários e veiculação.

Avro

Avro é um formato de dados orientado a linha projetado para RPCs e serialização de dados. Avro codifica dados em um formato binário, com metadados de esquema especificados em JSON. O Avro é popular no ecossistema Hadoop e também é suportado por várias ferramentas de dados em nuvem.

Serialização Colunar

Os formatos de serialização que discutimos até agora são orientados a linha. Os dados são codificados como relações completas (CSV) ou documentos (XML e JSON) e são gravados em arquivos sequencialmente.

Com a *serialização colunar*, a organização de dados é essencialmente dinamizada pelo armazenamento de cada coluna em seu próprio conjunto de arquivos. Uma vantagem óbvia do armazenamento colunar é que ele nos permite ler dados de apenas um subconjunto de campos, em vez de ler linhas completas de uma só vez. Esse é um cenário comum em aplicativos analíticos e pode reduzir drasticamente a quantidade de dados que devem ser verificados para executar uma consulta.

Armazenar dados como colunas também coloca valores semelhantes próximos uns dos outros, permitindo-nos codificar dados colunares com eficiência. Uma técnica comum envolve procurar valores repetidos e tokenizá-los, um método de compactação simples, mas altamente eficiente, para colunas com grande número de repetições.

Mesmo quando as colunas não contêm um grande número de valores repetidos, elas podem apresentar alta redundância. Suponha que organizamos as mensagens de suporte ao cliente em uma única coluna de dados. É provável que vejamos os mesmos temas e palavrões repetidamente nessas mensagens, permitindo que os algoritmos de compactação de dados obtenham uma alta proporção. Por esse motivo, o armazenamento colunar geralmente é combinado com compactação, permitindo maximizar os recursos de largura de banda do disco e da rede.

O armazenamento colunar e a compactação também apresentam algumas desvantagens. Não podemos acessar facilmente registros de dados individuais; devemos reconstruir registros lendo dados de vários arquivos de colunas. As atualizações de registros também são desafiadoras. Para alterar um campo em um registro, devemos descompactar o arquivo da coluna, modificá-lo, recompactá-lo e gravá-lo de volta no armazenamento. Para evitar reescrever colunas inteiras em cada atualização, as colunas são divididas em muitos arquivos, geralmente usando estratégias de particionamento e agrupamento que organizam os dados de acordo com os padrões de consulta e atualização da tabela. Mesmo assim, a sobrecarga para atualizar uma única linha é terrível. Os bancos de dados colunares são péssimos para cargas de trabalho transacionais, portanto, os bancos de dados transacionais geralmente utilizam alguma forma de armazenamento orientado a linhas ou registros.

Parquet

O Parquet armazena dados em um formato colunar e foi projetado para obter excelente desempenho de leitura e gravação em um ambiente de data lake. O Parquet resolve alguns problemas que frequentemente atormentam os engenheiros de dados. Os dados codificados em parquet são construídos em informações de esquema e oferecem suporte nativo a dados aninhados, ao contrário do CSV. Além disso, o Parquet é portátil; enquanto bancos de dados como BigQuery e Snowflake serializam dados em formatos colunares proprietários e oferecem excelente desempenho de consulta em dados armazenados internamente, ocorre um grande impacto no desempenho ao interoperar com ferramentas externas. Os dados devem ser

deserializado, reserializado em um formato trocável e exportado para usar ferramentas de data lake, como Spark e Presto. Arquivos parquet em um data lake podem ser uma opção superior para data warehouses em nuvem proprietários em um ambiente de ferramenta poliglota.

O formato parquet é usado com vários algoritmos de compactação; algoritmos de compressão otimizados para velocidade, como o Snappy (discutido posteriormente neste apêndice) são especialmente populares.

ORC

Optimized Row Columnar (ORC) é um formato de armazenamento colunar semelhante ao Parquet. ORC era muito popular para uso com o Apache Hive; embora ainda seja amplamente usado, geralmente o vemos muito menos do que o Apache Parquet e tem um pouco menos de suporte nas ferramentas modernas do ecossistema de nuvem. Por exemplo, o Snowflake e o BigQuery oferecem suporte à importação e exportação de arquivos Parquet; embora possam ler arquivos ORC, nenhuma ferramenta pode exportar para ORC.

Apache Arrow ou serialização na memória

Quando introduzimos a serialização como um ingrediente bruto de armazenamento no início deste capítulo, mencionamos que o software pode armazenar dados em objetos complexos dispersos na memória e conectados por ponteiros, ou estruturas mais organizadas e densamente compactadas, como Fortran e matrizes C. Geralmente, as estruturas de dados densamente compactadas na memória eram limitadas a tipos simples (por exemplo, INT64) ou estruturas de dados de largura fixa (por exemplo, strings de largura fixa). Estruturas mais complexas (por exemplo, documentos JSON) não podiam ser densamente armazenadas na memória e requeriam serialização para armazenamento e transferência entre sistemas.

A ideia de [Apache Arrow](#) é repensar a serialização utilizando um formato de dados binários que seja adequado tanto para processamento na memória quanto para exportação. Isso nos permite evitar a sobrecarga de serialização e desserialização; simplesmente usamos o mesmo formato para processamento na memória, exportação pela rede e armazenamento de longo prazo. O Arrow depende do armazenamento colunar, onde cada coluna basicamente obtém seus próprios blocos de memória. Para dados aninhados, usamos uma técnica chamada *trituração*, que mapeia cada local no esquema de documentos JSON em uma coluna separada.

Essa técnica significa que podemos armazenar um arquivo de dados no disco, trocá-lo diretamente no espaço de endereço do programa usando a memória virtual e começar a executar uma consulta nos dados sem sobrecarga de desserialização. Na verdade, podemos trocar pedaços do arquivo na memória enquanto o examinamos e, em seguida, trocá-los de volta para evitar a falta de memória para grandes conjuntos de dados.

1 Dejan Simic, "Apache Arrow: Leia DataFrame com Memória Zero," *Rumo à ciência de dados*, 25 de junho de 2020, <https://oreil.ly/TDAdY>.

Uma dor de cabeça óbvia com essa abordagem é que diferentes linguagens de programação serializam dados de maneiras diferentes. Para resolver esse problema, o Arrow Project criou bibliotecas de software para uma variedade de linguagens de programação (incluindo C, Go, Java, JavaScript, MATLAB, Python, R e Rust) que permitem que essas linguagens interoperem com os dados do Arrow na memória. Em alguns casos, essas bibliotecas usam uma interface entre a linguagem escolhida e o código de baixo nível em outra linguagem (por exemplo, C) para ler e escrever do Arrow. Isso permite alta interoperabilidade entre idiomas sem sobrecarga extra de serialização. Por exemplo, um programa Scala pode usar a biblioteca Java para escrever dados de seta e depois passá-los como uma mensagem para um programa Python.

A Arrow está vendo uma aceitação rápida com uma variedade de estruturas populares, como o Apache Spark. A Arrow também expandiu um novo produto de data warehouse; [Dremio](#) é um mecanismo de consulta e data warehouse construído em torno da serialização Arrow para oferecer suporte a consultas rápidas.

Serialização Híbrida

Nós usamos o termo *serialização híbrida* para se referir a tecnologias que combinam várias técnicas de serialização ou integram a serialização com camadas de abstração adicionais, como gerenciamento de esquema. Citamos como exemplos o Apache Hudi e o Apache Iceberg.

Hudi

Hudi apoia *Hadoop Update Delete Incremental*. Essa tecnologia de gerenciamento de tabelas combina várias técnicas de serialização para permitir o desempenho do banco de dados colunar para consultas analíticas, além de oferecer suporte a atualizações atômicas e transacionais. Um aplicativo Hudi típico é uma tabela que é atualizada a partir de um fluxo CDC de um banco de dados de aplicativo transacional. O fluxo é capturado em um formato de serialização orientado a linha, enquanto a maior parte da tabela é mantida em um formato colunar. Uma consulta é executada em arquivos colunares e orientados a linhas para retornar resultados para o estado atual da tabela. Periodicamente, é executado um processo de reempacotamento que combina os arquivos de linha e colunares em arquivos colunares atualizados para maximizar a eficiência da consulta.

Iceberg

Assim como o Hudi, o Iceberg é uma tecnologia de gerenciamento de tabelas. O Iceberg pode rastrear todos os arquivos que compõem uma tabela. Ele também pode rastrear arquivos em cada instantâneo de tabela ao longo do tempo, permitindo a viagem no tempo da tabela em um ambiente de data lake. O Iceberg oferece suporte à evolução do esquema e pode facilmente gerenciar tabelas em uma escala de petabytes.

Mecanismos de armazenamento de banco de dados

Para completar a discussão sobre serialização, discutimos brevemente os mecanismos de armazenamento de banco de dados. Todos os bancos de dados possuem um mecanismo de armazenamento subjacente; muitos não expõem seus mecanismos de armazenamento como uma abstração separada (por exemplo, BigQuery, Snowflake). Alguns (notavelmente, MySQL) suportam mecanismos de armazenamento totalmente conectáveis. Outros (por exemplo, SQL Server)

oferecem as principais opções de configuração do mecanismo de armazenamento (coluna versus armazenamento baseado em linha) que afetam drasticamente o comportamento do banco de dados.

Normalmente, o mecanismo de armazenamento é uma camada de software separada do mecanismo de consulta. O mecanismo de armazenamento gerencia todos os aspectos de como os dados são armazenados em um disco, incluindo serialização, organização física dos dados e índices.

Os mecanismos de armazenamento tiveram inovações significativas nas décadas de 2000 e 2010. Enquanto os mecanismos de armazenamento no passado eram otimizados para acesso direto a discos giratórios, os mecanismos de armazenamento modernos são muito mais otimizados para suportar as características de desempenho dos SSDs. Os mecanismos de armazenamento também oferecem suporte aprimorado para tipos e estruturas de dados modernos, como cadeias de caracteres de comprimento variável, matrizes e dados aninhados.

Outra mudança importante nos mecanismos de armazenamento é uma mudança para o armazenamento colunar para aplicativos analíticos e de data warehouse. SQL Server, PostgreSQL e MySQL oferecem suporte de armazenamento colunar robusto.

Compressão: gzip, bzip2, Snappy, etc.

A matemática por trás dos algoritmos de compactação é complexa, mas a ideia básica é fácil de entender: os algoritmos de compactação procuram redundância e repetição nos dados e, em seguida, recodificam os dados para reduzir a redundância. Quando queremos ler os dados brutos, *descomprimi*r invertendo o algoritmo e colocando a redundância de volta.

Por exemplo, você notou que certas palavras aparecem repetidamente ao ler este livro. Executando algumas análises rápidas no texto, você pode identificar as palavras que ocorrem com mais frequência e criar tokens abreviados para essas palavras. Para compactar, você substituiria palavras comuns por seus tokens; para descompactar, você substituiria os tokens por suas respectivas palavras.

Talvez possamos usar essa técnica ingênua para obter uma taxa de compressão de 2:1 ou mais. Algoritmos de compressão utilizam técnicas matemáticas mais sofisticadas para identificar e remover redundância; eles geralmente podem obter taxas de compactação de 10:1 em dados de texto.

Note que estamos falando sobre *algoritmos de compressão sem perdas*. A descompactação de dados codificados com um algoritmo sem perdas recupera uma cópia exata bit a bit dos dados originais. *Algoritmos de compressão com perdas* para áudio, imagens e vídeo visam a fidelidade sensorial; a descompressão recupera algo que soa ou se parece com o original, mas não é uma cópia exata. Os engenheiros de dados podem lidar com algoritmos de compactação com perdas em pipelines de processamento de mídia, mas não na serialização para análises, onde a fidelidade exata dos dados é necessária.

Os mecanismos de compactação tradicionais, como gzip e bzip2, compactam dados de texto extremamente bem; eles são frequentemente aplicados a JSON, JSONL, XML, CSV e outros formatos de dados baseados em texto. Os engenheiros criaram uma nova geração de algoritmos de compactação que priorizam a velocidade e a eficiência da CPU em relação à taxa de compactação nos últimos anos. Os principais exemplos são Snappy, Zstandard, LZFSE e LZ4. Esses algoritmos são frequentemente usados para compactar dados em data lakes ou bancos de dados colunares para otimizar o desempenho de consultas rápidas.

rede em nuvem

Este apêndice discute alguns fatores que os engenheiros de dados devem considerar sobre a rede na nuvem. Os engenheiros de dados frequentemente encontram redes em suas carreiras e muitas vezes as ignoram, apesar de sua importância.

Topologia de rede em nuvem

Atopologia de rede em nuvem descreve como vários componentes na nuvem são organizados e conectados, como serviços em nuvem, redes, locais (zonas, regiões) e muito mais. Os engenheiros de dados devem sempre saber como a topologia da rede em nuvem afetará a conectividade nos sistemas de dados que eles constroem. Microsoft Azure, Google Cloud Platform (GCP) e Amazon Web Services (AWS) usam hierarquias de recursos notavelmente semelhantes de zonas e regiões de disponibilidade. No momento da redação deste artigo, o GCP adicionou uma camada adicional, discutida em “[Rede específica do GCP e redundância multirregional](#)” na página 401.

Cobranças de saída de dados

[Capítulo 4](#) discute a economia da nuvem e como os custos reais do provedor não determinam necessariamente o preço da nuvem. Em relação à rede, as nuvens permitem o tráfego de entrada gratuitamente, mas cobram pelo tráfego de saída para a Internet. O tráfego de saída não é inherentemente mais barato, mas as nuvens usam esse método para criar um fosso em torno de seus serviços e aumentar a aderência dos dados armazenados, uma prática que tem sido amplamente criticada. Observe que as cobranças de saída de dados também podem ser aplicadas a dados que passam entre zonas de disponibilidade e regiões dentro de uma nuvem.

1 Matthew Prince e Nitin Rao, “Egregious Egress da AWS,” *O blog da Cloudflare*, 23 de julho de 2021, <https://oreil.ly/NZqKa>.

Zonas de Disponibilidade

Ozona de disponibilidadeé a menor unidade de topologia de rede que as nuvens públicas tornam visíveis aos clientes ([Figura B-1](#)). Embora uma zona possa consistir em vários datacenters, os clientes da nuvem não podem controlar o posicionamento de recursos nesse nível.

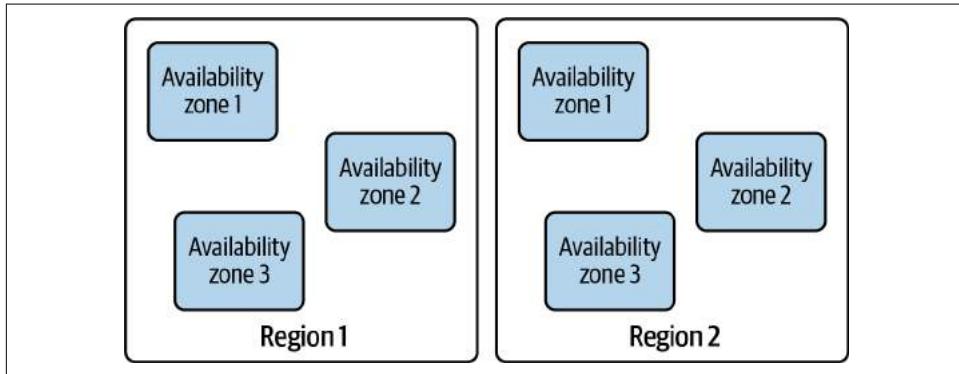


Figura B-1. Zonas de disponibilidade em duas regiões separadas

Geralmente, as nuvens suportam sua maior largura de banda de rede e menor latência entre sistemas e serviços dentro de uma zona. Cargas de trabalho de dados de alta taxa de transferência devem ser executadas em clusters localizados em uma única zona por motivos de desempenho e custo. Por exemplo, um cluster efêmero do Amazon EMR geralmente deve ficar em uma única zona de disponibilidade.

Além disso, o tráfego de rede enviado para VMs dentro de uma zona é gratuito, mas com uma ressalva significativa: o tráfego deve ser enviado para endereços IP privados. As principais nuvens utilizam redes virtuais conhecidas como *nuvens privadas virtuais*(VPC). As máquinas virtuais têm endereços IP privados dentro da VPC. Eles também podem receber endereços IP públicos para se comunicar com o mundo externo e receber tráfego da Internet, mas as comunicações usando endereços IP externos podem incorrer em cobranças de saída de dados.

Regiões

A *região*é uma coleção de duas ou mais zonas de disponibilidade. Os data centers requerem muitos recursos para funcionar (energia elétrica, água, etc.). Os recursos de zonas de disponibilidade separadas são independentes para que uma queda de energia local não derrube várias zonas de disponibilidade. Os engenheiros podem criar uma infraestrutura altamente resiliente e separada, mesmo dentro de uma única região, executando servidores em várias zonas ou criando processos automatizados de failover entre zonas.

A oferta de várias regiões permite que os engenheiros coloquem recursos próximos a qualquer um de seus usuários. Fechar significa que os usuários podem obter um bom desempenho da rede ao se conectar aos serviços, minimizando a distância física ao longo do caminho da rede e um número mínimo de

saltos através de roteadores. Tanto a distância física quanto os saltos de rede podem aumentar a latência e diminuir o desempenho. Os principais provedores de nuvem continuam a adicionar novas regiões.

Em geral, as regiões suportam redes rápidas e de baixa latência entre as zonas; o desempenho de rede entre zonas será pior do que dentro de uma única zona e incorrerá em cobranças de saída de dados nominais entre VMs. A movimentação de dados de rede entre regiões é ainda mais lenta e pode incorrer em taxas de saída mais altas.

Em geral, o armazenamento de objetos é um recurso regional. Alguns dados podem passar entre as zonas para chegar a uma máquina virtual, mas isso é principalmente invisível para os clientes da nuvem e não há cobrança direta de rede para isso. (Claro, os clientes ainda são responsáveis pelos custos de acesso ao objeto.)

Apesar do design com redundância geográfica das regiões, muitas das principais falhas de serviços em nuvem afetaram regiões inteiras, um exemplo de *defalha correlacionada*. Os engenheiros geralmente implantam código e configuração em regiões inteiras; as falhas regionais que observamos geralmente resultaram de problemas de código ou configuração que ocorrem no nível regional.

Rede específica do GCP e redundância multirregional

O GCP oferece várias abstrações exclusivas das quais os engenheiros devem estar cientes se trabalharem nessa nuvem. O primeiro é o *multirregional*, uma camada na hierarquia de recursos; uma multirregião contém várias regiões. As multirregiões atuais são EUA (centros de dados nos Estados Unidos), UE (centros de dados nos estados membros da União Europeia) e ÁSIA.

Vários recursos do GCP são compatíveis com várias regiões, incluindo Cloud Storage e BigQuery. Os dados são armazenados em várias zonas dentro da multirregião de maneira geograficamente redundante para que permaneçam disponíveis no caso de uma falha regional. O armazenamento multirregional também foi projetado para fornecer dados com eficiência aos usuários dentro da multirregião sem configurar processos de replicação complexos entre as regiões. Além disso, não há taxas de saída de dados para VMs em uma multirregião para acessar dados do Cloud Storage na mesma multirregião.

Os clientes de nuvem podem configurar a infraestrutura multirregional na AWS ou no Azure. No caso de bancos de dados ou armazenamento de objetos, isso envolve a duplicação de dados entre regiões para aumentar a redundância e colocar os dados mais próximos dos usuários.

O Google também possui essencialmente mais recursos de rede em escala global do que outros provedores de nuvem, algo que oferece a seus clientes como *rede de nível premium*. A rede de nível premium permite que o tráfego entre zonas e regiões passe inteiramente por redes de propriedade do Google sem passar pela Internet pública.

Conexões de rede diretas com as nuvens

Cada nuvem pública principal oferece opções de conectividade aprimoradas, permitindo que os clientes integrem suas redes diretamente a uma região de nuvem ou VPC. Por exemplo, Amazônia

oferece o AWS Direct Connect. Além de fornecer maior largura de banda e menor latência, essas opções de conexão geralmente oferecem grandes descontos nas cobranças de saída de dados. Em um cenário típico nos EUA, as cobranças de saída da AWS caem de 9 centavos por gigabyte na Internet pública para 2 centavos por gigabyte na conexão direta.

CDN

redes de entrega de conteúdo(CDNs) podem oferecer melhorias drásticas de desempenho e descontos para a entrega de ativos de dados ao público ou clientes. Os provedores de nuvem oferecem opções de CDN e muitos outros provedores, como Cloudflare. Os CDNs funcionam melhor ao fornecer os mesmos dados repetidamente, mas certifique-se de ler as letras miúdas. Lembre-se de que os CDNs não funcionam em todos os lugares e alguns países podem bloquear o tráfego da Internet e a entrega do CDN.

O futuro das taxas de saída de dados

As taxas de saída de dados são um impedimento significativo para a interoperabilidade, compartilhamento de dados e movimentação de dados para a nuvem. No momento, as taxas de saída de dados são um fosso projetado para impedir que clientes de nuvem pública saiam ou implantem em várias nuvens.

Mas sinais interessantes indicam que a mudança pode estar no horizonte. Em particular, o anúncio da Zoom em 2020, perto do início da pandemia de COVID-19, de que escolheu a Oracle como seu provedor de infraestrutura de nuvem chamou a atenção de muitos observadores de nuvem.² Como a Oracle ganhou esse importante contrato de nuvem para infraestrutura crítica de trabalho remoto contra os pesos pesados da nuvem? O especialista da AWS, Corey Quinn, oferece uma resposta razoavelmente direta.³ Por seu cálculo de fundo de envelope, as taxas mensais de saída de dados da AWS da Zoom chegariam a mais de US\$ 11 milhões pelo preço de tabela; O da Oracle custaria menos de US\$ 2 milhões.

Suspeitamos que GCP, AWS ou Azure anunciarão cortes significativos nas taxas de saída nos próximos anos, levando a uma mudança radical no modelo de negócios da nuvem. Também é perfeitamente possível que as taxas de saída desapareçam, semelhante a como os minutos de telefone celular limitados e caros desapareceram décadas atrás.

2 Mark Haranas e Steven Burke, "Oracle Bests Cloud Rivals to Win Blockbuster Cloud Deal", CRN, 28 de abril de 2020,<https://oreil.ly/LkqOj>.

3 Corey Quinn, "Por que a Zoom escolheu a Oracle Cloud em vez da AWS e talvez você também devesse", semana passada na AWS, 28 de abril de 2020,<https://oreil.ly/Lx5uu>.

Índice

Símbolos

1NF (primeira forma normal),[291](#) 2NF
(segunda forma normal),[291](#) 3NF
(terceira forma normal),[291](#)

A

abstração,[22](#)
políticas de acesso,[376](#)
responsabilidade,[55](#)
precisão,[55](#)
ACID (atomicidade, consistência, isolamento e durabilidade) transações,[103,158](#)
segurança ativa,[371,377](#) análise ad hoc,[345](#) arquitetura ágil,[81](#)

agilidade,[77](#)
pesquisadores de IA,[27](#)
fluxo de ar DAG,[320](#)
projeto de fluxo de ar,[150](#)
alertando,[375](#)
Amazon Elastic Block Store (EBS),[203](#)
Amazon EMR,[221](#)
dados analógicos,[156](#)
análise
 como código,[333](#)
 OLTPs e,[159](#)
 servindo dados para,[344-349](#)
 variações de,[44](#)
Flecha Apache,[394](#)
estrutura do Apache Beam,[106](#)
Druid Apache,[222](#)
Apache faísca,[221](#)
APIs (consulte interfaces de programa de aplicativo)

arquitetura de aplicativos,[72](#)(ver também arquivo de dados arquitetura; arquiteturas monolíticas; arquiteturas técnicas)
banhos de dados de aplicativos,[157-159](#) interfaces de programação de aplicativos (APIs),
[157,174-176,254](#)
níveis de arquitetura,[90](#)
armazenamento de arquivo,[197](#)
densidade de área,[192](#)
otimização assimétrica,[148](#)
ingestão assíncrona de dados,[238](#)
transações atômicas,[157-158](#)
atomicidade, consistência, isolamento e durabilidade (ACID) transações,[103,158](#) metadados gerados automaticamente,[52](#) políticas de ciclo de vida automatizadas,[225](#) gerenciamento automático do ciclo de vida dos dados,[226](#) automação,[61](#)

disponibilidade,[79,89,221](#) zonas de disponibilidade,[195,400](#) Avro,[392](#)
Estrutura bem arquitetada da AWS,[77](#)

B
árvore B,[166](#)
cópias de segurança,[372](#)
arquitetura básica,[81](#)
bash,[20](#)
basicamente disponível, soft-state, eventual consistência
 tenência (BASE),[198](#)
ingestão de dados em lote,[40,106,244-247](#)
transformações de dados em lote
 associações de transmissão,[310](#)
 lógica de negócios e dados derivados,[320](#)

disputa de dados,[319](#)
junções distribuídas,[310](#)
ETL, ELT e pipelines de dados,[311](#)
principais considerações para,[43](#)
MapReduce,[322](#)
cache de memória,[322](#)
atualizações de esquema,[318](#)
embaralhar junções de hash,[311](#) em faísca,[320](#)
SQL e ferramentas de transformação baseadas em código,
312
versus transformações de streaming,[310](#)
padrões de atualização,[315](#)
registros em lote,[161](#)
tamanho do batch,[247](#)
captura de dados de alteração orientada a lote,[252](#)
benchmarks,[147](#)
Mandato da API de Bezos,[82](#)
engenheiros de dados grandes,[8](#) era
dos grandes dados,[7,148,380](#)
armazenamento de blocos,[202-205](#)
tecnologias blockchain,[129](#) blocos
(HDFS),[211](#)
dados limitados,[236](#)
sistema Boyce-Codd,[294](#)
mesas de ponte,[305](#)
associações de transmissão,[310](#)
projetos brownfield,[96](#)
alertas de orçamento,[376](#)
construir,[22](#)
construir versus comprar
conselhos sobre a seleção,[138](#) vantagens de comprar,
[132](#) software livre,[133-137](#) jardins murados
proprietários,[137-138](#) adoção de tecnologia nas
empresas,[133](#) sistemas de armazenamento de dados
em massa,[191](#) ingestão de dados em rajada,[240](#) analista
de negócios,[346](#) arquitetura de negócios,[72](#) armazéns de
dados de negócios,[6](#) inteligência de negócios (BI),[45,344](#)
lógica de negócios,[43,320](#) metadados de negócios,[53,55](#)
partes interessadas do negócio,[111](#) valor do negócio,
aumentando,[127](#)

(veja também otimização de custos e negócios
valor)

C

executivos de nível C,[28](#)
hierarquia de cache,[197](#)
cache,[197,211,221,281](#) CAO-2s (diretores de
algoritmos),[30](#) CAOs (chefes de análise),[29](#)
despesas de capital (capex),[118](#) engenharia de
culto à carga,[116](#) CDC (consulte captura de dados
alterados) CDOs (diretores de dados principais),[29](#)
CEOs (diretores executivos),[28](#) captura de dados
alterados (CDC),[37,159,252-254,](#)
281

diretores de algoritmos (CAO-2s),[30](#)
Chief Analytics Officers (CAOs),[29](#)
diretores de dados (CDOs),[29](#) diretores
executivos (CEOs),[28](#) diretores de
informação (CIOs),[29](#) diretores de
operações (COOs),[29](#) diretores de
tecnologia (CTOs),[29](#) CIOs (diretores de
informação),[29](#) clonagem, cópia zero,[223](#) armazéns de dados na nuvem,[100](#),
216 rede em nuvem

topologia de rede em nuvem,[399-402](#)
redes de distribuição de conteúdo,[402](#)
futuro das taxas de saída de dados,[402](#)
nuvem de serviços de nuvens,[129](#) repatriação
na nuvem,[130-132](#) serviços na nuvem

adotando a abordagem cloud-first,[xiv](#) adotando
componentes comuns,[78](#) conselhos sobre a seleção,[129](#)
economia da nuvem,[125-127](#) sistema
operacional de dados em escala de nuvem,[381-383](#)
considerações para a escolha de tecnologias,

124-125
custos de,[xv](#)
descentralizado,[129](#)
conexões de rede diretas para,[401](#) sistemas
de arquivos,[201](#)
nuvem híbrida,[127](#)
gestão de gastos e recursos,[86](#)
motivação para usar,[120](#) implantação
multicloud,[128](#) suporte multilocação,[94](#)
ofertas de nuvem proprietárias,[138](#)
responsabilidade pela segurança,[85,372](#)

- separação de computação de armazenamento,
220-223
- armazenamento de bloco virtualizado em nuvem,
203 Cloudflare,**131-132** agrupamento,**213**
- ferramentas de transformação baseadas em
código,**312** dados frios,**39,224** armazém frio,**225**
- arquitetura colaborativa**,**81**
coleções,**169**
colocação,**220**
sistema de armazenamento de arquivos Colossus,**222**
serialização colunar,**213,393-395** colunas,**167,213,290**
formato de valores separados por vírgula (CSV),**392**
arquitetura de comando e controle,**81,85** software
comercial de código aberto (COSS),**135** comete,**278**
- expressões de tabela comuns (CTEs),**277** software de
código aberto gerenciado pela comunidade,
134
completude,**55**
conformidade,**226,265** componentes, escolhendo,**78**
visualizações materializadas que podem ser compostas,**324**
algoritmos de compressão,**196,396** computação,
separando-se do armazenamento,**220-223** modelos de
dados conceituais,**289** simultaneidade,**349**
- dimensão conformada,**301** consistência,
158,166,171,208 consumidores (de um
stream),**163,250** fuga de contêiner,**144**
- plataformas de contêineres,**144-145** redes de
entrega de conteúdo (CDNs),**402** captura de dados
de mudança contínua,**253,281** COOs (diretores de
operações principais),**29** copiar na gravação (COW),
317 falha correlacionada,**212**
- COSS (software comercial de código aberto),**135**
custo
economia da nuvem,**125**
argumentos de repatriação na nuvem,**130** de
serviços em nuvem,xv estrutura de custos de
dados,**86**
custos de saída de dados,**128,131,399,402** de
migração de dados,**261** despesas de
armazenamento de dados,**226** custos diretos,**118**
- de padrão monolítico distribuído,**142**
custos indiretos,**118**
supervisionando,**376**
de servidores em execução,**145**
de abordagem sem servidor,**143,147**
custo total de propriedade,**118**
custo de oportunidade total de propriedade,**119**
comparações de custos,**148**
otimização de custos e valor comercial,**118-120**
FinOps,**120**
importância de,**118**
custo total de propriedade,**118**
custo de oportunidade total de propriedade,
119 COW (cópia na gravação),**317**
criar, ler, atualizar e excluir (CRUD),**162,**
166
swaps de incumprimento de crédito,**125**
CRUD (criar, ler, atualizar e excluir),**162,**
166
formato CSV (valores separados por vírgula),**392**
CTEs (expressões de tabela comuns),**277** CTOs
(diretores de tecnologia),**29** maldição da
familiaridade,**126**
- D**
DAGs (gráficos acíclicos direcionados),**64,320,327**
DAMA (Data Management Association Inter-
nacional),**50**
dados obscuros,**102**
painéis,**344**
dados (ver também estágio de geração; sistemas de origem)
dados analógicos,**156**
fazendo backup,**372**
limitado versus ilimitado,**236**
dados frios,**39,224**
combinando fluxos com outros dados,**286**
estrutura de custos de,**86** excluindo,**316**
- dados digitais,**156**
durabilidade e disponibilidade de,**221**
grão de,**290**
dados quentes,**39,224**
interno e externo,**25** tipos
de,**241**
dados atrasados,**248**
dados mornos,**39,224**
dados de pré-junção,**276**
usos produtivos de,**339** dados
de autoatendimento,**341**

- forma de,²⁴¹
tamanho de,²⁴¹
estruturada, não estruturada e semiestruturada.
 turado,¹⁵⁷
fontes de terceiros para,¹⁷⁷
dados ilimitados,¹⁰⁶
dados quentes,²²⁴
frequência de acesso a dados,³⁹
responsabilidade de dados,⁵⁵
analistas de dados,²⁷
aplicativos de dados,^{159,348}
arquitetos de dados,^{25,81,111}
arquitetura de dados
 benefícios de uma boa arquitetura de dados,⁷¹ exemplos e tipos de,⁹⁸⁻¹¹⁰
 arquitetura para IoT,¹⁰⁶
 lagos de dados,¹⁰¹
 malha de dados,¹⁰⁹
 armazéns de dados,⁹⁸
 Modelo de fluxo de dados e lote unificado e transmissão,¹⁰⁵
 arquitetura kappa,¹⁰⁵
 arquitetura lambda,¹⁰⁴
 pilhas de dados modernas,¹⁰³ outros exemplos,¹¹⁰
 tendência à convergência,¹⁰² impacto no armazenamento de dados,²³⁰
 impacto em consultas, transformações e modelagem,³³³
 impacto na exibição de dados,³⁶³ impacto nos sistemas de origem,¹⁸⁵ impacto na seleção de tecnologia,¹⁵⁰ principais conceitos de arquitetura,⁸⁷⁻⁹⁸
 projetos brownfield versus projetos greenfield,⁹⁶ considerações para arquitetura de dados,⁹⁴
 sistemas distribuídos, escalabilidade e projetando para o fracasso,⁸⁸
 domínios e serviços,⁸⁷ arquitetura orientada a eventos,⁹⁵ principal objetivo das arquiteturas,⁸⁷
 acoplamento apertado versus solto,⁹⁰
 acesso do usuário,⁹⁴
 princípios da boa arquitetura de dados,⁷⁷⁻⁸⁷
 estar sempre arquitetando,⁸¹
 arquiteto para escalabilidade,⁸⁰
 arquitetura é liderança,⁸⁰
 estrutura da AWS,⁷⁷
 construir sistemas fracamente acoplados,⁸² escolha componentes comuns com sabedoria,⁷⁸
- abraçar FinOps,⁸⁵
princípios do Google Cloud,⁷⁷
tomar decisões reversíveis,⁸³
pilares de,⁷⁸
 planejar para o fracasso,⁷⁹ priorizar a segurança,⁸⁴ reconhecendo uma boa arquitetura de dados,⁷⁶ papel dos engenheiros de dados em,⁶⁴
 como subconjunto da arquitetura corporativa,⁷²
 versus ferramentas,¹¹⁵
 definição de trabalho de,⁷¹⁻⁷⁶
localização do bloco de dados,²²²
violação de dados,⁸⁵
Ferramenta de Criação de Dados (dbt),³⁵⁶ catálogos de dados,²¹⁸
conectores de dados,³⁸¹
contratos de dados,^{182,339}
linguagem de controle de dados (DCL),²⁷⁴
linguagem de definição de dados (DDL),²⁷³
definições de dados,³⁴²
custos de saída de dados,^{128,131,399,402}
engenharia de dados
 abordagem para aprender,^{xiii,XVI}
 maturidade de dados e,¹³⁻¹⁷
 complexidade declinante,³⁸⁰
 definição de termo,³⁻⁴ evolução do campo,⁶⁻¹¹ futuro de
 ferramentas de gerenciamento de nível empresarial,³⁸³ evolução de títulos e responsabilidades,³⁸⁴
 fusão de dados com aplicativos,³⁸⁷ melhor interoperabilidade,³⁸¹⁻³⁸³ mover em direção a pilhas de dados ao vivo,³⁸⁵⁻³⁸⁶
 surgingimento de planilhas,³⁸⁸ canais de transmissão,³⁸⁶
 feedback apertado entre aplicações e aprendizado de máquina,³⁸⁸
 objetivos de aprendizagem do livro,^{xv}
 armazenamentos de objetos para,²⁰⁷ pré-requisitos para aprender,^{xv}
 línguas primárias e secundárias usadas em,²⁰⁻²¹
 relação com a ciência de dados,¹¹⁻¹²
 segurança para baixo nível,³⁷⁷
 habilidades e atividades,¹³ abstrações de armazenamento,²¹⁵⁻²¹⁷ público-alvo do livro,^{xiv}

ciclo de vida da engenharia de dados (consulte também undercur-
aluéis)

definição de termo,**xiv**

futuro de,**379**

estágio de geração,**35-37** estágio de
ingestão,**39-42** relação com o ciclo de vida
dos dados,**34** estágio de dados de serviço,
44-48 estágios de,**5,19,33** estágio de
armazenamento,**38-39** estágio de
transformação,**43-44** engenheiros de dados

histórico e habilidades de,**17** engenheiros de dados
grandes,**8** liderança empresarial e,**28-31**
responsabilidades comerciais de,**18** engenharia de
culto à carga,**116** continuum de funções para,**21**
versus arquitetos de dados,**64** engenheiros de ciclo
de vida de dados,**10** definição de termo,**4** projetar
arquiteturas de dados,**111** evolução em engenheiros
de ciclo de vida de dados,**33** novas arquiteturas e
desenvolvimentos,**110** outras funções de gestão e,**31**
gerentes de produto e,**30** gerentes de projeto e,**30**
como engenheiros de segurança,**85**

responsabilidades técnicas de,**19-21,155**

dentro das organizações,**22-28** ética de dados
e privacidade,**59,85,265** caracterização de dados,
44

geração de dados (ver estágio de geração)

gravidade dos dados,**127**

ingestão de dados (ver estágio de ingestão)

integração de dados,**58,234** data

lakehouses,**103,216** lagos de dados,**101,216,**

312 latência de dados,**349**

ciclo de vida dos dados,**34**

engenheiros de ciclo de vida de dados,**10**

gerenciamento do ciclo de vida dos dados,**58**

linhagem de dados,**57**

ferramentas de linhagem de dados,

331 lógica de dados,**343**

gestão de dados

responsabilidade de dados,

55 Gestão de dados,**51**

integração de dados,**58**

gerenciamento do ciclo de vida dos dados,**58**

linhagem de dados,**57**

modelagem e design de dados,**57**

qualidade de dados,**55**

definição de termo,

50 descoberta,**52**

ética e privacidade,**59,85**

facetas de,**51**

impacto na ingestão de dados,**264** impacto

no armazenamento de dados,**228**

impacto em consultas, transformações e

modelagem,**331**

impacto na exibição de dados,**362** impacto

nos sistemas de origem,**184** impacto na
seleção de tecnologia,**149** gerenciamento de
dados mestre,**56** metadados,**52-54**

Associação Internacional de Gerenciamento de Dados (DAMA),**50**

Corpo de conhecimento de gerenciamento de dados
(DMBOK),**50,75**

Data Management Maturity (DMM),**13-17**

linguagem de manipulação de dados (DML),**273**

mercados de dados,**176**

data marts,**101**

maturidade de dados,**13-17** malha

de dados,**94,109,326,343** migração

de dados,**247**

modelagem de dados

alternativas para,**307**

resultados de negócios e,**288** modelos
conceituais, lógicos e físicos,
289

considerações para o sucesso,**289**

definição de termo,**287**

obter insights de negócios por meio de,**57**

exemplos de,**287**

futuro de,**387**

normalização,**290**

o propósito de,**288**

partes interessadas de,**329**

técnicas para dados analíticos em lote,
294-306

combinando,**294**

Cofre de Dados,**301**

tabelas de dimensões,**298**

tabelas de fatos,**298**

hubs (cofre de dados),**302**

Inmon,**295**

- Kimball,297**
- tabelas de links (cofre de dados),[303](#)
 - modelagem de dados de streaming,[307](#)
 - satélites (cofre de dados),[304](#) esquema estelar,[301](#)
 - amplas tabelas desnormalizadas,[305](#)
 - observabilidade de dados,[339](#)
 - Desenvolvimento Orientado à Observabilidade de Dados (DODD),[58,62](#)
 - orquestração de dados,[64,68](#) canais de dados,[234,386](#) plataformas de dados,[103,217,381](#) produtores e consumidores de dados,[24](#) produtos de dados,[340](#)
 - qualidade de dados,[55](#)
 - engenheiros de confiabilidade de dados,[332](#) retenção de dados,[223,225](#)
 - esquemas de dados,[37](#)
 - Hierarquia de necessidades da ciência de dados,[11](#)
 - cientistas de dados,[26](#)
 - segurança de dados,[49](#)
 - compartilhamento de dados,[176,219,262,355](#) pilhas de dados,[10,103,385-386](#) partes interessadas de dados,[182](#)
 - ciclo de vida de armazenamento de dados (consulte também estágio de armazenamento)
 - conformidade,[226](#)
 - custo,[226](#)
 - retenção de dados,[225](#)
 - dados quentes, mornos e frios,[223](#)
 - considerações de nível de armazenamento, [224](#) tempo,[226](#)
 - valor,[225](#)
 - pântanos de dados,[102](#)
 - tecnologias de dados
 - arquitetura versus ferramentas,[115](#) considerações para escolher avaliação comparativa,[147](#) construir versus comprar,[132-139](#) otimização de custos e valor comercial,[118-120](#)
 - tecnologia imutável versus transitória gies,[120-123](#)
 - impacto de subcorrentes,[149-151](#)
 - interoperabilidade,[117](#)
 - localização,[123-132](#)
 - monólito versus modular,[139-143](#)
 - visão geral de,[116](#)
 - sem servidor versus servidores,[143-147](#)
 - velocidade para o mercado,[117](#)
 - tamanho e capacidade da equipe,[116](#)
 - variedade de opções disponíveis,[115](#)
 - quando selecionar,[115](#) data de validade,[339](#)
 - valor de dados,[44](#)
 - Cofres de dados,[301](#)
 - virtualização de dados,[325](#)
 - armazéns de dados,[6,98-101,215,295](#)
 - disputa de dados,[319](#)
 - metadados de linhagem de dados,[54](#) testes de qualidade de dados,[267](#) registros do banco de dados,[161](#)
 - sistemas de gerenciamento de banco de dados (DBMSs),[166](#) replicação de banco de dados,[253](#) mecanismos de armazenamento de banco de dados,[395](#)
 - bancos de dados
 - compromete-se a,[278](#)
 - conectando-se diretamente a,[251](#)
 - registros mortos em,[280](#) exportação de arquivo de,[257](#)
 - considerações importantes para a compreensão, [166](#)
 - versus mecanismos de consulta,[272](#) bancos de dados analíticos em tempo real,[386](#) servindo dados via,[352](#) suporte a transações,[278](#) modelo de fluxo de dados,[56,106](#) DataOps
 - adotando hábitos culturais de,[60](#)
 - automação,[61](#)
 - elementos técnicos essenciais de,[61](#)
 - objetivos de,[59](#)
 - como alta prioridade,[63](#) impacto na ingestão de dados,[266](#) impacto no armazenamento de dados,[229](#)
 - impacto em consultas, transformações e modelagem,[332](#)
 - impacto na exibição de dados,[362](#) impacto nos sistemas de origem,[184](#) impacto na seleção de tecnologia,[149](#) resposta a incidentes,[63](#)
 - observabilidade e monitoramento,[62](#) relação com a manufatura enxuta,[60](#) conceito de portal de dados,[53](#)
 - datastrophes,[267](#)
 - DBMSs (sistemas de gerenciamento de banco de dados),[166](#)
 - dbt (ferramenta de construção de dados),[356](#) DCL (linguagem de controle de dados),[274](#) DDL (linguagem de definição de dados),[273](#)

registros de banco de dados mortos,[280](#) filas de mensagens mortas,[249](#) computação descentralizada,[129](#) tomada de decisão, eliminando decisões irreversíveis

Sões,[83](#)

descompressão,[396](#)
dissociação,[83,90](#)
postura defensiva,[370](#)
eliminação,[316](#)
desnormalização,[213,291,305](#)
dados derivados,[320](#)
desserialização,[239](#)
dispositivos,[106](#)
engenheiros DevOps,[26](#)
padrão de atualização diferencial,[246](#)
dados digitais,[156](#)
tabelas de dimensões,[297-301](#)
custos diretos,[118](#)
gráficos acíclicos direcionados (DAGs),[64,320,327](#)
prevenção de desastres,[373](#) descoberta,[52](#)

velocidade de transferência de disco,[192](#) junções distribuídas,[310](#)
padrão de monólito distribuído,[142](#)
armazenamento distribuído,[198](#)
Sistemas distribuídos,[88](#)
DMBOK (Data Management Body of Knowledge),[50,75](#)
DML (linguagem de manipulação de dados),[273](#) DMM (Maturidade de Gerenciamento de Dados),[13](#)
armazenamento de documentos,[169](#)
documentos,[169](#)
DODD (Data Observability Driven Development),[58,62](#)
acoplamento de domínio,[92](#)
domínios,[87](#)
não se repita (DRY),[290](#) partes interessadas a jusante,[26](#)
Dropbox,[131-132](#) durabilidade,[158,221,240](#) RAM dinâmica (DRAM),[194,221](#)

E

EA (arquitetura corporativa),[72-75](#) EABOK (Livro de Arquitetura Empresarial de Conhecimento),[73,81](#)
EBS (Amazon Elastic Block Store),[203](#)
computação de ponta,[129](#)
arestas (em um gráfico),[172](#)

eficiência,[176](#)

sistemas elásticos,[80,88](#)
intercâmbio eletrônico de dados (EDI),[257](#)
ELT (extrair, carregar e transformar),[99,246,312](#)
análise incorporada,[45,348](#) métricas emitidas,[354](#)

criptografia,[375](#)

enriquecimento,[286](#)
arquitetura corporativa (EA),[72-75](#) Livro de Conhecimento de Arquitetura Corporativa (EABOK),[73,81](#)
efemeridade,[220](#)
Manipulação de erros,[249](#)
ética,[59,85,265](#)
ETL (consulte extrair, transformar, carregar processo) hora do evento,[165](#)
dados baseados em eventos,[174,248-250](#) arquitetura orientada a eventos,[95](#) sistemas orientados a eventos,[163](#) plataformas de transmissão de eventos,[164,177,179-181](#),

255

eventual consistência,[158,171,198,208](#)
explicar plano,[277](#)
Linguagem de marcação extensível (XML),[392](#)
dados externos,[26](#)
engenheiros de dados externos,[23](#)
extrato (ETL),[246](#)
extrair, carregar e transformar (ETL),[99,246,312](#) extrair, transformar, carregar (ETL) processo
transformações de dados em lote,[311](#)
armazéns de dados e,[99](#) versus ETL,[246](#)
modelos push versus pull de ingestão de dados,[42](#)
ETL reverso,[47,358-360](#)

F

tabelas de fatos,[297](#)
falha, planejamento para,[79,88](#)
Lei de Direitos Educacionais Familiares e Privacidade (FERPA),[369](#)
abordagem de captura de dados de mudança de seguidor rápido,[281](#) tolerância ao erro,[181](#)
caracterização,[44](#)
consultas federadas,[324,354](#)
FERPA (Direitos Educacionais Familiares e Privacidade Agir),[369](#)
Campos,[167,290](#)
FIFO (primeiro a entrar, primeiro a sair),[179](#)

troca de arquivos,[351](#)
armazenamento de arquivo,[199-201](#)
Protocolo de transferência de arquivos (FTP),[375](#)
exportação baseada em arquivo,[246](#)
arquivos e formatos de arquivo,[156,257-258,391-395](#)
sistemas de arquivos,[201,210](#) gestão financeira,[85](#) FinOps,[85-87,120](#)

primeiro a entrar, primeiro a sair (FIFO),[179](#) primeira forma normal (1NF),[291](#)
Cinco princípios para arquitetura nativa da nuvem,[77](#)
esquema fixo,[37](#)
janelas de tempo fixo,[284](#)
chaves estrangeiras,[167](#)
frequência,[237](#)
FTP (Protocolo de Transferência de Arquivos),[375](#)
instantâneos completos,[246](#)
varreduras de tabela completa,[277](#)
criptografia de disco completo,[375](#)

G

Ciclo de Hype da Gartner,[72](#)
estágio de geração
sistemas de origem e,[35-37](#)
fontes de dados,[156](#) discos de ouro,[56](#)
rede específica do Google Cloud Platform,[401](#)
Sistema de arquivos do Google (GFS),[211](#) governança,[51](#)
grão,[290](#)
banhos de dados gráficos,[172](#)
GraphQL,[175](#)
projetos verdes,[97](#)
gRPC,[176](#)

H

Hadoop Distributed File System (HDFS),[211, 220](#)
Hadoop Update Delete Incremental (Hudi),[395](#)
exclusão forçada,[316](#)
arquitetura Harvard,[195](#) BI sem cabeça,[356](#)
Portabilidade e prestação de contas de planos de saúde
Lei da Cidade (HIPAA),[369](#) escala horizontal,[89](#)
dados quentes,[39,224](#)
armazenamento quente,[224](#)

ponto de acesso,[181](#)
HTTPS (Protocolo de Transferência de Hipertexto Seguro),[375](#)
hubs (cofre de dados),[302](#)
Hudi (Atualização do Hadoop Excluir Incremental),[395](#)
violações de segurança humana,[370,375](#) metadados gerados por humanos,[52](#) nuvem híbrida,[127,130](#) armazenamento colunar híbrido,[214](#) armazenamento de objetos híbridos,[222](#) separação híbrida,[221](#)

serialização híbrida,[395](#)

Protocolo de transferência de hipertexto seguro (HTTPS),[375](#)

EU

IaaS (infraestrutura como serviço),[124](#) IaC (infraestrutura como código),[67](#) Iceberg,[395](#)
sistemas de mensagens idempotentes,[179](#) elemento comercial identificável,[302](#) tecnologias imutáveis,[121,130](#)
definições de dados implícitos,[343](#)
serialização na memória,[394](#) resposta a incidentes,[63](#)
atualizações incrementais,[246](#)
ofertas independentes,[137](#)
índices,[166,213](#) custos indiretos,[118](#)
infraestrutura como serviço (IaaS),[124](#)
infraestrutura como código (IaC),[67](#) estágio de ingestão
considerações de ingestão de lote,[244-247](#)
lote versus streaming,[40-42,106](#) desafios enfrentados,[40](#) alterar a captura de dados,[252](#) pipelines de dados e,[234](#) definição de ingestão de dados,[234](#) impacto de subcorrentes sobre,[263-268](#) gateways IoT e,[108](#)

principais considerações de engenharia para,[235-243](#)

dados limitados versus dados ilimitados,[236](#)
frequência de ingestão de dados,[237](#) visão geral de,[40,235](#) carga útil,[241](#)

confiabilidade e durabilidade,[240](#)

serialização e desserialização,[239](#)
ingestão síncrona versus assíncrona
ção,[238](#)

rendimento e escalabilidade,[239](#) consideração de ingestão de mensagem e stream
ções,[248-250](#)
modelos push versus pull,[42,244](#)
partes interessadas de,[262](#)
maneiras de ingerir dados,[250-262](#)
APIs,[254](#)
compartilhamento de dados,[262](#)
bancos de dados e exportação de arquivos,[257](#)
conexão direta com o banco de dados,[251](#)
intercâmbio eletrônico de dados,[257](#) conectores de dados gerenciados,[256](#) filas de mensagens e streaming de eventos
plataformas,[255](#)
mover dados com armazenamento de objetos,[257](#) SFTP e SCP,[259](#) interface de concha,[258](#)

protocolo SSH,[259](#)
dispositivos de transferência para migração de dados,[261](#)
interfaces web,[260](#)
Raspagem da web,[260](#)
webhooks,[259](#)
tempo de ingestão,[165](#)
modelo de dados Immon,[295](#) inserir exclusão,[316](#)
padrão somente de inserção,[162,316](#)
inserções,[247](#)
volumes de armazenamento de instâncias,[204](#) conhecimento institucional,[343](#) integração,[58,234](#),[295](#) dados internos,[25](#)
ingestão interna,[234](#) pesquisa de segurança interna,[377](#) engenheiros de dados internos,[23](#) Internet das Coisas (IoT),[106-109](#)
interoperabilidade,[58,117](#) gateways IoT,[107](#)
decisões irreversíveis,[83](#)
isolamento (ACID),[158](#)

J

Java,[20](#)
linguagens Java Virtual Machine,[20](#) Notação de objeto JavaScript (JSON),[392](#) juntar estratégia,[275](#)
se junta,[286](#)
Linhas JSON (JSONL),[392](#)

K

arquitetura kappa,[105,282](#) bancos de dados chave-valor,[169](#) carimbos de data/hora de valor-chave,[180](#) chaves,[180](#)
modelo de dados de Kimball,[297](#)

eu

casas do lago,[103](#)
arquitetura lambda,[104](#) dados atrasados,[248](#) latência,[161,192](#),[349](#) liderança, arquitetura como, 80 práticas enxutas,[60](#)

Gerenciamento do ciclo de vida,[58](#)
levante e mude,[126](#) cache de objetos leves,[211](#) máquinas virtuais leves,[144](#) linhagem,[57](#)

densidade linear (armazenamento magnético),[192](#)
tabelas de links (cofre de dados),[303](#) pilhas de dados ao vivo,[385-386](#) mesas ao vivo,[324](#)

carga (ETL),[246](#)

armazenamento em disco local,[200](#)
volumes de instância local,[204](#)
localização,[250](#)
análise de log,[173](#)
capturas de dados alterados com base em log,[253](#) árvores de mesclagem estruturadas em log (LSMs),[166](#) exploração madeireira,[375](#)
lógica,[343](#)
modelos de dados lógicos,[289](#)
Histórico,[160](#)
Observador,[356](#)
dados de pesquisa,[54,207](#)
pesquisas,[166](#)
comunicação fracamente acoplada,[83](#) sistemas fracamente acoplados,[82,90](#) algoritmos de compressão sem perdas,[396](#) algoritmos de compressão com perdas,[396](#) LSMs (árvores de mesclagem estruturadas em log),[166](#) dados mornos,[39,224](#)

M

aprendizado de máquina (ML),[46,349-351,388](#)
engenheiros de aprendizado de máquina,[27](#)
operações de aprendizado de máquina (MLOps),[28](#)

- discos magnéticos,[191-193](#)
conectores de dados gerenciados,
[256](#) MapReduce,[220,322](#)
bancos de dados de processamento massivamente paralelo (MPP),
[6,99](#)
gerenciamento de dados mestre (MDM),[56](#)
visualizações materializadas,[324](#) tempo máximo de
retenção de mensagem,[249](#) MDSs (pilhas de dados
modernas),[10,103,385](#) Dados de medição,[174](#)
- memcached,[211](#)
cache de memória,[322](#)
sistemas de armazenamento baseados em memória,
[211](#) padrão de mesclagem,[317](#)
filas de mensagens,[163,177-179,255](#)
mensagens
noções básicas de,[163](#)
entregue fora de ordem,[248](#)
tratamento de erros e filas de mensagens mortas,[249](#)
dados atrasados,[248](#) recuperando da história,[249](#)
tamanho de,[249](#)
contra fluxos,[255](#)
metadados,[52-54,243](#)
camadas de métricas,[355](#)
abordagem de micro-lote,[328](#)
microparticionamento,[214](#)
arquitetura de microsserviços,[93](#)
migração,[247](#)
ML (aprendizado de máquina),[46,349-351,388](#)
MLOps (operações de aprendizado de máquina),
[28](#) modelagem e design,[57](#)
(veja também modelagem de
dados) padrões de modelagem,[166](#)
pilhas de dados modernas (MDSs),[10,103,](#)
[385](#) modularização,[83,139,140,142](#)
monitoramento,[62,78,375](#)
arquiteturas monolíticas,[92,139-140,142](#) Bancos de dados
MPP (processamento massivamente paralelo),
[6,99](#)
implantação multicloud,[128,130](#) camada
de rede multirregional,[401](#) suporte
multilocação,[46,94,176](#) armazenamento
multilocatário,[226](#) arquiteturas
multicamadas,[91](#) cache multicamadas,
[221](#)
- N**
arquiteturas de n camadas,[91](#)
- NAS (armazenamento conectado à rede),[201](#)
ingestão de dados quase em tempo real,[41,237](#)
Pensamento negativo,[370](#)
subconsultas aninhadas,[277](#)
segurança de rede,[376](#)
armazenamento conectado à rede (NAS),[201](#)
rede (consulte rede em nuvem) arquitetura von
Neumann,[195](#) Sistema de arquivos de nova
tecnologia (NTFS),[200](#) junção de nós,[310](#)
- nós,[172](#)
bancos de dados não relacionais (NoSQL),[168-174](#)
armazenamento de documentos,[169](#)
bancos de dados gráficos,[172](#)
história de,[168](#)
armazenamentos de valor-chave,[169](#)
versus bancos de dados relacionais,[168](#) bancos
de dados de pesquisa,[173](#)
bancos de dados de séries temporais,[173](#) bancos de dados de
coluna larga,[171](#) memória de acesso aleatório não volátil
(NVRAM),
[194](#)
formas normais,[291](#)
normalização,[290](#)
esquemas normalizados,[167](#)
NoSQL (não apenas SQL) (consulte dados não relacionais-
bases)
notebooks, servindo dados em,[356](#) NTFS (Sistema de
Arquivos de Nova Tecnologia),[200](#) NVRAM (memória de
acesso aleatório não volátil),
[194](#)
- O**
armazenamento de objetos,[200,205-211](#)
zonas de disponibilidade e,[206](#)
Benefícios de,[206,206](#)
para aplicações de engenharia de dados,[207](#)
definição de termo,[205](#) versus armazenamento
de arquivos,[200](#) versus disco local,[206](#) mover
dados com,[257](#)
consistência e controle de versão de objetos,[208](#)
pesquisa de objetos,[207](#)
sistemas de arquivos com armazenamento de objetos,
[210](#) popularidade de,[207](#)
escalabilidade de,[207](#)
classes e níveis de armazenamento,
[210](#) observabilidade,[62,78,339](#) conectores
de dados prontos para uso,[381](#)

pilhas de tecnologia no local,[123,126](#)
soluções de tecnologia de tamanho único,[79](#)
portas unidirecionais,[73,83](#)
sistemas de processamento analítico online (OLAP),
[159](#)
sistemas de processamento de transações on-line (OLTP),
[157](#)
Estrutura de arquitetura de grupo aberto, o
(TOGAF)
arquitetura de dados,[75](#)
arquitetura empresarial,[72](#)
software de código aberto (OSS),[133-137,381,388](#)
análise operacional,[45,346](#) arquitetura
operacional,[76](#) despesas operacionais (opex),[119](#)
metadados operacionais,[54](#) Coluna de linha
otimizada (ORC),[394](#) persistência opcional,[211](#)
orquestração

escolhendo os componentes com sabedoria,[78](#)
impacto na ingestão de dados,[268](#) impacto no
armazenamento de dados,[230](#)
impacto em consultas, transformações e
modelagem,[333](#)
impacto na exibição de dados,[363](#) impacto
nos sistemas de origem,[186](#) impacto na
seleção de tecnologia,[150](#) oleodutos e,[68](#)
processo de,[64](#)
características organizacionais,[82](#) arquitetura de
data warehouse organizacional,[98](#)
superarquitetura,[120](#)
a sobrecarga,[118](#)

P

PaaS (plataforma como serviço),[124](#) bancos
de dados de processamento paralelo,[6,99](#)
paranóia,[370](#)
Parquet,[393](#)
dependência parcial,[291](#)
chaves de partição,[180](#)
particionamento,[213](#)
remendos,[374](#)
cargas úteis,[241-243](#)
desempenho,[95](#)
permissões, monitoramento,[376](#)
persistência, opcional,[211](#) modelos
de dados físicos,[289](#) metadados de
pipeline,[54](#)

pipelines como código,[68](#)
Tabelas PIT (point-in-time),[305](#)
plataforma como serviço (PaaS),[124](#)
tarefas de encanamento,[175](#)
tabelas point-in-time (PIT),[305](#)
aplicações poliglotas,[141](#) dados
de pré-junção,[276](#)
rede de nível premium,[401](#) pré-
requisitos para livro,[xv](#) chaves
primárias,[167](#)
princípio do menor privilégio,[49,372](#) privacidade,[59,85](#)
,[265,369](#)(veja também segurança) tempo de processo,
[165](#)
motores de processamento, escolhendo,[78](#)
Tempo de processamento,[165](#)
gerentes de produto,[30](#)
gerentes de projeto,[30](#)
ofertas de nuvem proprietárias,[138](#)
jardins murados proprietários,[137-138](#)
poda,[278](#)
acesso público,[85](#)
editores,[163](#)
modelo pull de ingestão de dados,[42,244,250](#) push
modelo de ingestão de dados,[42,244,246,250](#) Pitão,
[20](#)

Q

qualidade,[55](#)
consultas
noções básicas de,[273-](#)
[274](#) execução de,[274](#)
consultas federadas,[324,354](#) melhorando o
desempenho de,[275-281](#) partes interessadas
de,[329](#)
em dados de streaming,[281-287](#) técnicas e
linguagens utilizadas para,[272](#) versus
transformações,[309](#) motores de consulta,[272](#)
otimizadores de consulta,[166,275](#)
desempenho da consulta,[349](#)
pushdown de consulta,[326,353](#)

R

RAID (matriz redundante de discos independentes),
[202](#)
memória de acesso aleatório (RAM),[194](#) RDBMSs (sistema de
gerenciamento de banco de dados relacional
tem),[35,157,167](#) bancos de dados
analíticos em tempo real,[386](#)

- ingestão de dados em tempo real,[41](#),[106](#),[237](#)
registros em tempo real,[161](#)
objetivo do ponto de recuperação (RPO),[79](#)
objetivo de tempo de recuperação (RTO),[79](#)
reduzir passo,[220](#)
redundância,[401](#)
matriz redundante de discos independentes (RAID),
[202](#)
metadados de referência,[54](#)
regiões,[400](#)
regulamentos,[226](#)
sistemas de gerenciamento de banco de dados relacional
(RDBMS),[35](#),[157](#),[167](#)
esquema relacional,[219](#)
relações (linhas),[167](#),[169](#)
confiabilidade,[79](#),[89](#),[240](#)
chamadas de procedimento remoto (RPCs),[176](#)
repatriamento,[130](#)
repetir,[212](#),[249](#)
relatórios,[345](#)
APIs de transferência de estado representacional (REST),[174](#)
resiliência,[181](#)
resolução,[161](#)
uso de recursos,[376](#)
desenvolvimento voltado para o currículo,[97](#),
[115](#) cache reverso,[197](#)
ETL reverso,[47](#),[358](#)-[360](#)
decisões reversíveis,[83](#)
latência rotacional,[192](#)
explosão de linha,[276](#)
serialização baseada em linha,
[391](#) linhas,[213](#),[290](#)
RPO (objetivo do ponto de recuperação),[79](#)
RTO (objetivo de tempo de recuperação),[79](#)
- S**
- Classe de armazenamento S3 Standard-Infrequent Access,
[210](#)
SaaS (software como serviço),[124](#) Sistemas SAN
(rede de área de armazenamento),[203](#) satélites
(cofre de dados),[304](#) escala,[20](#)
- escalabilidade
arquitetando para,[80](#)
Benefícios de,[88](#)
banhos de dados e,[166](#)
fase de ingestão e,[239](#) sistemas de
mensagens e,[179](#) armazenamento
de objetos e,[207](#)
- sistemas de pagamento conforme o uso,[86](#) separação de
computação de armazenamento,[220](#) varreduras, mesa cheia,
[277](#)
SCDs (dimensões que mudam lentamente),[300](#)
agendadores,[64](#)
mudanças de esquema,[264](#)
evolução do esquema,[248](#)
metadados do esquema,[54](#)
esquema na técnica de leitura,[219](#)
esquema na técnica de gravação,[219](#)
atualizações de esquema,[318](#)
opção sem esquema,[37](#)
esquemas
noções básicas de,[219](#)
definição de termo,[37](#)
detectar e lidar com mudanças em,[243](#) em
cargas úteis de dados,[242](#) registros para,[243](#)
em bancos de dados relacionais,[167](#)
esquema estelar,[301](#)
bancos de dados de pesquisa,[173](#)
segunda forma normal (2NF),[291](#)
cópia segura (SCP),[259](#) FTP seguro
(SFTP),[259](#) segurança
manipulação de credenciais em notebooks,
[357](#) atividades humanas e,[370](#)
impacto no ciclo de vida da engenharia de dados,[49](#)
impacto na ingestão de dados,[264](#) impacto no
armazenamento de dados,[228](#)
impacto em consultas, transformações e
modelagem,[330](#)
impacto na exibição de dados,[361](#)
impacto nos sistemas de origem,[183](#)
priorizando,[84](#)
processos e,[371](#)-[374](#) papel na
privacidade,[369](#) único versus
multilocatário,[95](#) tecnologia e,[374](#)
-[378](#) modelos de confiança zero,
[84](#)
políticas de segurança,[373](#)
dados de autoatendimento,[341](#)
camadas semânticas,[355](#)
dados semiestruturados,[157](#)
planos de sequenciamento,[81](#)
serialização,[195](#),[239](#) formatos de
serialização,[391](#)-[395](#)
 serialização colunar,[393](#)
 serialização híbrida,[395](#)

- melhorar o desempenho e,³⁹¹
serialização baseada em linha,³⁹¹
ofertas de nuvem sem servidor,^{143,146}
servidores, considerações ao usar,¹⁴⁵
acordos de nível de serviço (SLAs),¹⁸³
objetivos de nível de serviço (SLOs),¹⁸³
Serviços,⁸⁷
estágio de dados de serviço
análise,^{44,344-349} inteligência de negócios,^{45,344} análise incorporada,^{45,348} considerações gerais para,³³⁸⁻³⁴⁴ obter valor dos dados,⁴⁴
dispositivos IoT e,¹⁰⁸ aprendizado de máquina,^{46,349-351} suporte multilocação,⁴⁶ análise operacional,^{45,346} ETL reverso,^{47,358-360} partes interessadas de,³⁶⁰
- maneiras de servir dados,³⁵¹⁻³⁵⁸
janelas de sessão,²⁸³
sessãoização,²⁸⁴
SFTP (FTP seguro),²⁵⁹
forma,²⁴¹
arquiteturas de disco compartilhado,⁹²
modelo de segurança de responsabilidade compartilhada,^{84,372} arquiteturas sem compartilhamento,⁹² interface de concha,²⁵⁸
síndrome do objeto brilhante,¹¹⁵ embaralhar junções de hash,³¹¹ armazenamento de dados de máquina única,¹⁹⁸ inserções de linha única,³¹⁶
armazenamento de locatário único,²²⁶ arquiteturas de camada única,⁹⁰ engenheiros de confiabilidade do local (SREs),²⁶ tamanho,^{241,249}
- ingestão de lote baseada em tamanho,²⁴⁵
SLAs (contratos de nível de serviço),¹⁸³ janelas de correr,²⁸⁴
SLOs (objetivos de nível de serviço),¹⁸³
dimensões que mudam lentamente (SCDs),³⁰⁰
instantâneos,²⁴⁶
Microparticionamento floco de neve,²¹⁴
capital social e conhecimento,⁵³
exclusão suave,³¹⁶
software como serviço (SaaS),¹²⁴
Engenharia de software
código de processamento de dados principais,⁶⁶
impacto na ingestão de dados,²⁶⁸
- impacto no armazenamento de dados,²³⁰
impacto em consultas, transformações e modelagem,³³³
impacto na exibição de dados,³⁶⁴ impacto nos sistemas de origem,¹⁸⁷ impacto na seleção de tecnologia,¹⁵¹ infraestrutura como código (IaC),⁶⁷ estruturas de código aberto,⁶⁶ pipelines como código,⁶⁸ processamento de dados de streaming,⁶⁷ transição de codificação para quadros de dados,⁶⁶ engenheiros de software,²⁵
- unidades de estado sólido (SSDs),¹⁹³
sistemas de origem (ver também estágio de geração)
exemplo de banco de dados do aplicativo,³⁵
evitando quebras em pipelines e análises,³⁵
noções básicas de,¹⁵⁶⁻¹⁶⁵
APIs,¹⁵⁷
bancos de dados de aplicativos,¹⁵⁷ alterar o método de captura de dados,¹⁵⁹ registros do banco de dados,¹⁶¹
arquivos e dados não estruturados,¹⁵⁶
padrão somente de inserção,¹⁶² Histórico,¹⁶⁰
mensagens e transmissões,¹⁶³
sistema de processamento analítico online (OLAP) tems,¹⁵⁹
tipos de tempo,¹⁶⁴
definição de termo,³⁵
avaliando,³⁶⁻³⁷
impacto de subcorrentes sobre,¹⁸³⁻¹⁸⁷ Exemplo de enxame de IoT e fila de mensagens,³⁶ detalhes práticos,¹⁶⁵⁻¹⁸¹
APIs,¹⁷⁴
compartilhamento de dados,¹⁷⁶
banhos de dados,¹⁶⁶
filas de mensagens e streaming de eventos plataformas,¹⁷⁷
fontes de dados de terceiros,¹⁷⁷
partes interessadas de,¹⁸¹
aspectos únicos de,³⁷ API
Spark,^{314,320} velocidade para o mercado,¹¹⁷
gastos e recursos, gestão,⁸⁵ derrame para o disco,³²³ planilhas,³⁸⁸
- SQL (consulte Linguagem de consulta estruturada)
SREs (engenheiros de confiabilidade do site),²⁶

- SSDs (unidades de estado sólido),[193](#)
protocolo SSH,[259](#)
partes interessadas
Rio abaixo,[26](#)
fase de ingestão e,[262](#)
para consultas, transformações e modelagem,[329](#)
servindo dados e,[360](#)
sistemas de origem e,[181](#)
rio acima,[25](#)
trabalhando ao lado,[111](#)
esquema estelar,[301](#)
estado,[180](#)
STL (transmitir, transformar e carregar),[387](#) sistemas de rede de área de armazenamento (SAN),[203](#) estágio de armazenamento
grandes ideias e tendências em armazenamento,[218-227](#)
catálogos de dados,[218](#)
compartilhamento de dados,[219](#)
ciclo de vida de armazenamento de dados e retenção de dados,[223](#)
esquemas,[219](#)
separação de computação de armazenamento,[220](#) armazenamento de locatário único versus armazenamento de vários locatários,[226](#)
cache,[197](#)
abstrações de armazenamento de engenharia de dados,[215-217](#)
sistemas de armazenamento de dados,[197-214](#)
armazenamento de blocos,[202](#)
cache e armazenamento baseado em memória,[211](#)
eventual versus consistência forte,[198](#)
armazenamento de arquivo,[199](#)
Sistema de Arquivos Distribuídos Hadoop,[211](#) índices, particionamento e agrupamento,[213](#) armazenamento de objetos,[205](#)
máquina única versus distribuída,[198](#)
armazenamento de streaming,[212](#)
divisão de responsabilidades para,[227](#)
dispositivos IoT e,[108](#)
principais considerações de engenharia,[38](#) estágios do ciclo de vida englobados por,[190](#) elementos físicos e de software de,[190](#) ingredientes brutos de armazenamento de dados,[191-197](#)
compressão,[196](#)
discos magnéticos,[191](#)
rede e CPU,[195](#) memória de acesso aleatório,[194](#)
serialização,[195](#)
unidades de estado sólido,[193](#) selecionar sistemas de armazenamento,[39,78,189](#) noções básicas de solução de armazenamento,[38](#) entender a frequência de acesso aos dados,[39](#) partições de fluxo,[180](#)
transmitir, transformar e carregar (STL),[387](#)
arquitetura de armazenamento stream-to-batch,[217](#)
ingestão de dados de streaming,[40,237,248-250](#)
modelagem de dados de streaming,[307](#)
processamento de dados de streaming,[67](#) consultas de dados de streaming,[281-287](#) streaming de grafos acíclicos direcionados,[327](#) canais de transmissão,[386](#)
plataformas de transmissão,[163,354](#) consultas de streaming,[326](#)
transformações de streaming,[326-329](#)
fluxos
noções básicas de,[164](#)
combinando com outros dados,[286](#)
enriquecedor,[286](#)
associação de fluxo a fluxo,[286](#)
buffers de streaming,[287](#)
forte consistência,[198,208](#) Linguagem de consulta estruturada (SQL)
transformações de dados em lote,[312](#)
construção de fluxos de trabalho de dados complexos,[313](#) desenvolvimentos baseados em, 6 eficácia de, 20-21 focar em,[272](#)
otimização de estruturas de processamento,[314](#)
quando evitar,[313](#) orientação do assunto,[295](#)
ingestão de dados síncrona,[238](#) parte interessada dos sistemas,[182](#)
- T**
junções de tabela,[286](#)
tecnologias de gerenciamento de tabelas,[395](#)
mesas,[169](#)
arquitetura alvo,[81](#) público-alvo do livro,[xiv](#) TCO (custo total de propriedade),[118](#) TDD (desenvolvimento orientado a testes),[62](#) arquitetura técnica,[72,76](#)
acoplamento técnico,[92](#)
arquitetura de armazenamento de dados técnicos,[98](#)
metadados técnicos,[53](#)
tecnologias (ver tecnologias de dados)
tabelas temporárias,[277](#)

desenvolvimento orientado a testes (TDD),⁶²

Pesquisa de texto,¹⁷³

terceira forma normal (3NF),²⁹¹

fontes de dados de terceiros,¹⁷⁷

arquiteturas de três camadas,⁹¹ Taxa de transferência,²³⁹

camadas

arquiteturas multcamadas,⁹¹ rede de nível premium,⁴⁰¹ arquiteturas de camada única,⁹⁰ classes de armazenamento e,^{210,224} estruturar camadas de arquitetura,⁹⁰ acoplamento apertado,⁹⁰

tempo de vida (TTL),^{226,249} tempo, tipos de,

¹⁶⁴ ingestão de lote com intervalo de tempo,

²⁴⁵ bancos de dados de séries temporais,¹⁷³

pontualidade,⁵⁶

carimbos de data/hora,¹⁸⁰

TOCO (custo total de oportunidade de propriedade),
119

TOGAF (consulte Quadro de Arquitetura de Grupo Aberto-trabalho, O)

ferramentas, versus arquitetura,¹¹⁵
(ver também tecnologias)

tópicos,¹⁸⁰

custo total de propriedade (TCO),¹¹⁸

custo total de oportunidade de propriedade (TOCO),
119

produção sem toque,²⁶⁵ linguagem de controle de transação (TCL),²⁷⁴ bancos de dados transacionais,^{157,253} transações,^{158,278} aparelhos de transferência,²⁶¹ ELT de transformação na leitura,¹⁰⁰ estágio de transformação,⁴³ transformações

transformações em lote,³¹⁰⁻³²³

virtualização de dados,³²⁵ consultas federadas,³²⁴

visualizações materializadas,³²⁴

o propósito de,³⁰⁹

contra consultas,³⁰⁹

partes interessadas de,³²⁹

transformações e processamento de streaming,
326-329,387

Visualizações,³²³

dependência transitiva,²⁹¹

tecnologias transitórias,^{121,130}

truncar padrão de atualização,³¹⁶

confiar,³³⁸

TTL (tempo de vida),
226 janelas caindo,²⁸⁴

portas de duas vias,^{74,83} cientistas de dados tipos A e B,²²

Você

dados ilimitados,^{106,236}

correntes ocultas

considerações para a escolha de tecnologias,
149-151

arquitetura de dados,⁶⁴

gestão de dados,⁵⁰⁻⁵⁹

DataOps,⁵⁹⁻⁶⁴ exemplos de,^{5,49}

impacto no armazenamento de dados,²²⁸⁻²³⁰

impacto em consultas, transformações e

modelagem,³³⁰⁻³³⁴ impacto na

exibição de dados,³⁶⁰⁻³⁶⁵

orquestração,^{64,68}

papel na engenharia de dados,^{13,34}

segurança,⁴⁹

Engenharia de software,⁶⁶⁻⁶⁸

responsabilidades técnicas e,¹⁹

chaves primárias únicas,²⁹¹ dados não estruturados,¹⁵⁷

operações de atualização,^{247,374}

padrões de atualização,³¹⁵⁻³¹⁹

padrão de upsert,³¹⁷

partes interessadas a montante,²⁵

casos de uso, determinando,³³⁹

V

aspirar,²⁸⁰

validação,³³⁹

valor dos dados,⁴⁴

valores,¹⁸⁰

metadados da versão,²⁰⁹

sistemas de controle de versão, selecionando,⁷⁸

Visualizações,³²³

volatilidade,¹⁹⁴

arquitetura von Neumann,¹⁹⁵

C

dados quentes,²²⁴

gerenciamento de projetos em cascata,⁸¹

marcas d'água,²⁸⁵

interfaces web,[260](#)

Raspagem da web,[260](#)

webhooks,[176](#),[259](#)

ampas tabelas desnormalizadas,[305](#)

mesas largas,[305](#)

bancos de dados de coluna larga,[171](#)

métodos de janelas,[67](#),[106](#),[283-285](#) escrever

uma vez, ler nunca (WORN),[102](#)

X

XML (Extensible Markup Language),[392](#)

Z

clonagem de cópia zero,[223](#)

modelos de segurança de confiança zero,[84](#)

sobre os autores

joe reis é um nerd de dados voltado para negócios que trabalhou no setor de dados por 20 anos, com responsabilidades que vão desde modelagem estatística, previsão, aprendizado de máquina, engenharia de dados, arquitetura de dados e quase tudo mais. Joe é CEO e cofundador da Ternary Data, uma empresa de consultoria em arquitetura e engenharia de dados com sede em Salt Lake City, Utah. Além disso, ele é voluntário em vários grupos de tecnologia e leciona na Universidade de Utah. Nas horas vagas, Joe gosta de escalar rochas, produzir música eletrônica e levar os filhos em loucas aventuras.

Matt Housley é consultor de engenharia de dados e especialista em nuvem. Depois de algumas experiências iniciais de programação com Logo, Basic e montagem 6502, ele completou um doutorado em matemática na Universidade de Utah. Matt então começou a trabalhar em ciência de dados, eventualmente se especializando em engenharia de dados baseada em nuvem. Ele cofundou a Ternary Data com Joe Reis, onde aproveita sua experiência de ensino para treinar futuros engenheiros de dados e aconselhar equipes sobre arquitetura de dados robusta. Matt e Joe também pontificam sobre todos os dados sobre *O bate-papo de dados da manhã de segunda-feira*.

Colofão

O animal na capa da *Fundamentos da Engenharia de Dados* é o papagaio-de-orelha-branca (*Nystalus chacuru*).

Assim chamados por causa da notável mancha branca em suas orelhas, bem como por sua plumagem fofa, esses pássaros pequenos e rotundos são encontrados em uma ampla faixa da América do Sul central, onde habitam bordas de florestas e savanas.

Os papagaios-de-orelha-branca são caçadores de sentar e esperar, pousando em espaços abertos por longos períodos e alimentando-se oportunisticamente de insetos, lagartos e até pequenos mamíferos que se aproximam. Eles são mais freqüentemente encontrados sozinhos ou em pares e são pássaros relativamente quietos, vocalizando apenas raramente.

A União Internacional para a Conservação da Natureza listou o papagaio-de-orelha-branca como sendo de *menor preocupação*, devido, em parte, ao seu extenso alcance e população estável. Muitos dos animais nas capas da O'Reilly estão ameaçados de extinção; todos eles são importantes para o mundo.

A ilustração da capa é de Karen Montgomery, baseada em uma gravura antiga de *Shawzoologia geral*. As fontes da capa são Gilroy Semibold e Guardian Sans. A fonte do texto é Adobe Minion Pro; a fonte do cabeçalho é Adobe Myriad Condensed; e a fonte do código é Ubuntu Mono de Dalton Maag.



O'REILLY®

**Aprenda com especialistas.
Torne-se um você mesmo.**

Livros | Cursos online ao vivo Respostas
instantâneas | Eventos virtuais Vídeos |
Aprendizagem interativa

Comece em oreilly.com.